Hybrid metaheuristic algorithms for sum coloring and bandwidth coloring

Yan Jin

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

université angers

Thèse de Doctorat

Yan JIN

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

École doctorale : 503 (STIM)

Discipline : Informatique
Spécialité : Informatique
Unité de recherche : Laboratoire d'Études et de Recherches en Informatique d'Angers (LERIA)

Soutenue le 29 Mai 2015
Thèse n° : 1421

Hybrid metaheuristic algorithms for sum coloring and bandwidth coloring

Métaheuristiques hybrides pour la somme coloration et la coloration de bande passante

JURY

Rapporteurs :	**M. Alexandre CAMINADA**, Professeur, Université de Technologie de Belfort-Montbéliard
	M. Aziz MOUKRIM, Professeur, Université de Technologie de Compiègne
Examinateur :	**M. Jean-Charles BILLAUT**, Professeur, Ecole Polytechnique de l'Université de Tours
Directeur de thèse :	**M. Jin-Kao HAO**, Professeur, Université d'Angers
Co-encadrant de thèse :	**M. Jean-Philippe HAMIEZ**, Maître de Conférences, Université d'Angers

Contents

I

General introduction

Introduction

Context

The vertex coloring problem (VCP) is a special case of graph labeling in graph theory and one of the most challenging combinatorial optimization problems in discrete mathematics. Given an undirected graph, the VCP aims to find a minimum number of colors to color the vertices of a graph such that no two adjacent vertices receive the same color. The VCP is one of the 21 fundamental problems whose NP-hardness was proved in the early 1970s [Karp 1972]. The VCP is extensively studied not only for its theoretical intractability, but also for its real world applications in many domains, such as timetabling, scheduling, register allocation, train platforming, frequency assignment, and communication networks for instances. In recent decades, the VCP as well as its generalizations are receiving more and more attention in the literature, these generalizations allowing more applications to be embraced. This thesis focuses on three generalized vertex coloring problem, namely the minimum sum coloring problem (MSCP), the bandwidth coloring problem (BCP, also known as the restricted T-coloring problem) and the bandwidth multicoloring problem (BMCP, also called the restricted set T-coloring problem).

The objective of the MSCP is to find a legal k-coloring of a graph such that the sum of the colors assigned to the vertices is minimized (colors are represented by consecutive integers $1, 2 \ldots, k$). The MSCP was introduced in [Kubicka 1989] from the graph theory point of view and in [Supowit 1987] from the application perspective. In practice, the MSCP is notable in VLSI design, scheduling, and resource allocation, etc. Let us consider the scheduling application in the open shop problem as an example: There are n jobs $J_1, \ldots, J_n$ to be executed and each job J_i is composed of a set of t_i tasks. Suppose each job has to be run on a processor for one unit of time and the number of processors is not limited. Different tasks of the same job cannot be processed simultaneously and one processor cannot run two tasks at the same time. The objective is to minimize the total time to complete all the jobs. This scheduling problem is equivalent to the MSCP where vertices represent jobs and edges represent the conflicts between the tasks.

The BCP and BMCP are two other important generalizations of the vertex coloring problem. Given an undirected graph, the BCP consists in finding a k-coloring with the smallest value of k such that the absolute value of the difference between the colors of adjacent vertices is not less than the weight of associated edge. The BCP can be generalized as the BMCP where each vertex can be colored with more than one color. A legal bandwidth multicoloring must satisfy two constraints: (1) the absolute value of the difference between the colors of adjacent vertices is not less than the weight of associated edge; (2) the absolute value of the difference between any two distinct colors of a vertex is at least the weight of the loop edge of this vertex. The BMCP is to find a legal bandwidth multicoloring with k minimum. The BCP and BMCP are notable for their applicability to a number of important applications in particular in the area of frequency assignment in mobile networks [Aardal et al. 2007]. Given a number of radio transmitters, each transmitter must be assigned an operating frequency and frequencies assigned to two nearby transmitters must be separated by a given threshold to avoid interferences. The objective is to minimize the number of assigned frequencies. This is equivalent to the BCP where colors represent frequencies, vertices represent transmitters and edge weights correspond to the required separation of frequencies for two nearby transmitters.

Since these generalizations of the vertex coloring problem (MSCP, BCP and BMCP) are NP-hard problems [Johnson and Trick 1996, Johnson et al. 2002], no polynomial-time algorithm can solve or approximate them effectively unless $P = NP$. In practice, heuristics and metaheuristics are often used to obtain suboptimal solutions in acceptable computing time. Hence, this thesis is dedicated to design effective hybrid metaheuristic algorithms for solving the MSCP, BCP and BMCP.

Objectives

This thesis aims to study hybrid approaches for the minimum sum coloring problem, the bandwidth coloring problem, and the bandwidth multicoloring problem. This thesis challenge is broken down into the following key objectives:

- Developing effective memetic algorithms which follow the general framework combining the population-based evolutionary strategy and local optimization procedure.
- Designing an initial population generation technique by incorporating problem specific knowledge for generating diverse solutions of good quality to boost the effectiveness of the memetic algorithm.
- Designing effective crossover operators since the recombination is an important ingredient for hybrid memetic algorithms.
- Devising an updating rule to maintain a good quality and healthy diversity of the population.
- Developing a new hybrid search algorithm which explores the cooperative framework between an informed construction procedure and a local search procedure.
- Designing a learning technique and integrating learning information into the search process to reinforce the performance of the metaheuristic algorithm.

Contributions

The main contributions of this thesis are summarized below:

A memetic algorithm: For the minimum sum coloring problem, we proposed a memetic algorithm (MASC) based on a tabu search procedure with two neighborhoods and a multi-parent crossover operator. Experiments on a set of 77 well-known DIMACS and COLOR 2002–2004 benchmark instances show that the proposed algorithm achieves highly competitive results in comparison with five state-of-the-art algorithms. This work is published in *Computers & Operations Research* [Jin et al. 2014].

A hybrid search algorithm: The MSCP is further studied and a hybrid search algorithm (HSA) is proposed for computing the lower and upper bounds of this NP-hard problem. To create an initial population of good quality, a maximum independent set algorithm SBTS is used to generate in a step-by-step way k mutually disjoint independent sets. HSA relies on a joint use of two dedicated crossover operators to generate promising offspring solutions and an iterated double-phase tabu search procedure to improve offspring solutions. A distance-and-quality updating rule is used to maintain a healthy diversity in the population. Extensive experimental studies on 94 well-known DIMACS and COLOR 2002–2004 benchmark instances show that the proposed algorithm matches most of the current best known results. In particular, it can find better solutions for 51 instances (24 instances for the upper bounds and 27 instances for the lower bounds).

Moreover, we show for the first time a landscape analysis of the MSCP to shed lights on the behavior of the proposed algorithm. A paper describing SBTS is published in *Engineering Applications of Artificial Intelligence* [Jin and Hao 2015b] and this work is also detailed in [Jin and Hao 2015c].

An effective learning-based hybrid search algorithm: For the bandwidth coloring problem and the bandwidth multicoloring problem, we present a learning-based hybrid search (LHS). LHS combines a construction phase to progressively build feasible (partial) colorings and a local search phase to reestablish feasibility when an illegal partial solution is encountered. The construction phase relies on a learning-based guiding function to determine the next vertex for color assignment while the local search phase uses a tabu search procedure to repair coloring conflicts. Experiments on a set of 33 well-known benchmarks demonstrate that the proposed approach can match the best known solution for most benchmarks. In particular, LHS finds an improved solution for 14 instances. A paper describing this work is accepted in *IEEE Transactions on Systems, Man, and Cybernetics: Systems* [Jin and Hao 2015a].

Organization

The manuscript is organized in the following way:

– In the first and second chapter, we introduce the minimum sum coloring problem and the bandwidth (multi)coloring problem, then provide an overview of the most representative algorithms proposed in the literature as well as the benchmarks that are frequently used to evaluate the performance of algorithms for each problem.

– In the third chapter, we first present the general memetic framework of our MASC algorithm for solving the MSCP. Then, we describe the components in detail, including the population initialization, the crossover operator, the double-neighborhood tabu search procedure and the population updating. Finally, we evaluate our MASC algorithm on challenging benchmark instances and report experimental comparative results. Meanwhile, we investigate and analyze some key issues of the proposed memetic algorithm.

– In the fourth chapter, we describe the proposed HSA for computing upper bounds of the MSCP. Then, we explain the adjustments of the proposed algorithm to compute lower bounds. In the following, extensive computational results of HSA and comparisons with the state-of-the-art algorithms in the literature are reported to demonstrate the effectiveness of the proposed algorithm. Finally, we study the impact of the joint use of two crossover operators on the performance of HSA and analyze the landscape of the MSCP.

– In the fifth chapter, we first introduce a framework for solving the bandwidth coloring problem and the bandwidth multicoloring problem (LHS). Then, we present the components in detail, including the learning-based guiding function, the construction phase with forward checking and the tabu repair phase. Finally, we provide computational results of LHS on well-known benchmark instances and compare our approach with some best performing algorithms.

– In the last two chapters, we give a general conclusion of this thesis and propose some perspectives.

– In the appendix, we present the general swap-based multiple neighborhood tabu search for the maximum independent set problem. We first introduce the general procedure and then present the detailed intensification and diversification procedure. Finally, we provide computational results on well-known benchmarks to demonstrate the effectiveness of the proposed algorithm.

II

The minimum sum coloring problem and the bandwidth (multi)coloring problem: A state-of-the-art

1

The minimum sum coloring problem

1.1 Introduction

Given a graph G, a proper k-coloring of G is an assignment of k different colors $\{1, \ldots, k\}$ to the vertices of G such that two adjacent vertices receive two different colors. The Minimum Sum Coloring Problem (MSCP) is to find a proper k-coloring while minimizing the sum of the colors assigned to the vertices. To our knowledge, the MSCP was proposed by Kubicka [Kubicka 1989] in the field of graph theory and by Supowit [Supowit 1987] in VLSI design. Kubicka proposed the "chromatic sum" measure of a graph and proved its NP-hardness [Kubicka and Schwenk 1989], Supowit introduced the "optimum cost chromatic partition problem" [Supowit 1987], and Kroon et al. [Kroon et al. 1997] also proved the NP-hardness of the MSCP. However, polynomial time algorithms exist for some specific graphs, such as trees, unicyclic graphs, outplanar graphs, chain bipartite graphs and k-split graphs [Kroon et al. 1997, Kubicka 2005, Kubicka and Schwenk 1989, Salavatipour 2003].

The MSCP is closely related to the classical vertex coloring problem, which is notable for its practical applicability and theoretical intractability. The MSCP can find its applications in VLSI design, scheduling and resource allocation [Kroon et al. 1997, Malafiejski 2004, Sen et al. 1992] for instance. Due to the high computational complexity of the problem, no polynomial-time algorithm can solve or approximate it efficiently unless $P = NP$. In the past several decades, much efforts have been devoted to the development of various heuristics and metaheuristics, such as greedy algorithms [Li et al. 2009, Moukrim et al. 2010], tabu search [Bouziri and Jouini 2010], breakout local search [Benlic and Hao 2012], iterated local search [Helmar and Chiarandini 2011], ant colony [Douiri and Elbernoussi 2012], genetic and memetic algorithms [Douiri and Elbernoussi 2011, Jin et al. 2014, Jin and Hao 2015c, Kokosiński and Kwarciany 2007, Moukrim et al. 2013, Wang et al. 2013] as well as heuristics based on independent set extraction [Wu and Hao 2012; 2013b]. Since greedy algorithms are fast but usually give solutions of poor quality they are often used to build initial solutions for other heuristic procedures. Besides, a greedy heuristic based on independent set extraction is effective especially on large instances [Wu and Hao 2012]. While various local search heuristics have been proposed to solve the MSCP, their main differences rely on the search space, the neighborhood structures and the strategies used to escape from a local optima. Hybrid metaheuristics lead to excellent performances in terms of solution quality. They often use local search as an improvement strategy and the two- (or multi-) parent(s) crossover operator.

To the best of our knowledge, there is only one review in 2004 [Kubicka 2004] that reports the history of the MSCP and the developments of polynomial-time algorithms on specific graphs as well as the generalizations and the applications. In this chapter, we provide a detailed review of the different MSCP solution approaches proposed in the recent literature for general graphs. In particular, we carry out an in-depth analysis of the studied approaches and encourage further studies on this problem.

In the following sections, we first provide a general definition of the MSCP, followed by the introduction of the studied heuristics and metaheuristics. Then, we report the "theoretical" lower and upper bounds. Finally, the MSCP benchmark instances are presented.

1.2 Definitions and formulation

Given a simple undirected graph $G = (V, E)$ with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E \subset V \times V$, a proper k-coloring c of G is a mapping $c : V \rightarrow \{1, \ldots, k\}$ such that $c(v_i) \neq c(v_j), \forall\{v_i, v_j\} \in E$. A proper k-coloring c can also be defined as a partition of V into k independent sets $V_1, \ldots, V_k$ such that $\{v_i, v_j\} \in V_l \times V_l \ (l = 1, \ldots, k) \Rightarrow \{v_i, v_j\} \notin E$. The objective of the MSCP is to find a proper k-coloring c with a minimum sum of the colors assigned to all the vertices. The minimum sum of colors for the MSCP is called the chromatic sum of G, and is denoted by $\sum(G)$. Let $\mathcal{C}(G)$ be the set of all proper k-coloring of G and $f(c)$ is presented in Eq. (1.1)

$$f(c) = \sum_{i=1}^{n} c(v_i) \text{ or } f(c) = \sum_{l=1}^{k} l|V_l| \tag{1.1}$$

where $|V_l|$ the cardinality of V_l and $|V_1| \geq \ldots \geq |V_k|$, then:

$$\sum(G) = \min_{c \in \mathcal{C}(G)} f(c) \tag{1.2}$$

Note that integer k is not fixed but larger than or equal to the chromatic number $\chi(G)$ of G in the classical vertex coloring problem (VCP). From the left graph in Fig. 1.1, we can see $\chi(G)$ = 3 but $f(c)$ = 18, while in the right graph, k = 4 and $\sum(G)$ = 15. It could be observed that the sum of coloring may not be optimal when using $\chi(G)$ colors to color the graph, and using $k > \chi(G)$ colors to color the graph may lead to the minimum sum of coloring. Hence, the VCP algorithms cannot be adapted for solving the MSCP directly due to the difference of the objective function.

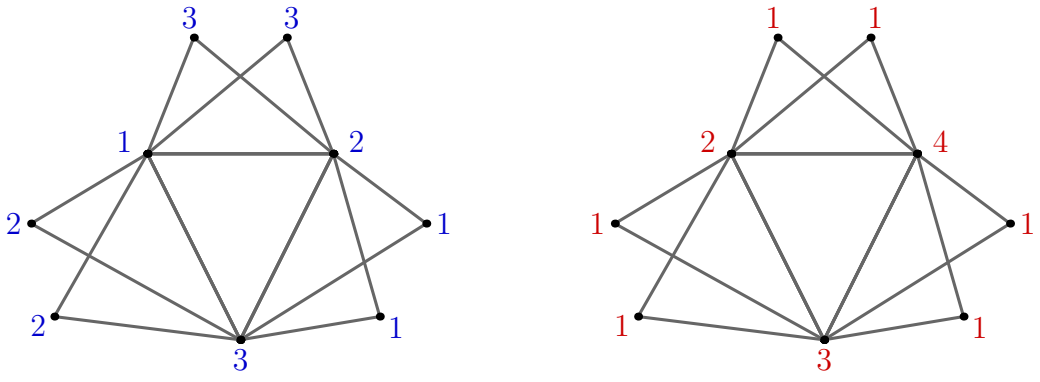


Figure 1.1: An illustrative example for the MSCP

Besides, the MSCP can also be formalized as a binary quadratic problem [Sen et al. 1992, Wang et al. 2013]:

$\min g(x) = \sum_{i=1}^{n} \sum_{l=1}^{k} l \cdot x_{il}$

subject to

$$\begin{aligned} &\sum_{l=1}^{k} x_{il} = 1, i \in \{1, \ldots, n\} \\ &x_{il} + x_{jl} \leq 1, \forall \{v_i, v_j\} \in E, l \in \{1, \ldots, k\} \\ &x_{il} \in \{0, 1\} \end{aligned} \tag{1.3}$$

where $x_{il} = 1$ if v_i is assigned color l (0 otherwise).

1.3 Heuristics and metaheuristics

As far as we know there is no exact algorithm especially designed for the MSCP except by applying CPLEX on the binary quadratic formulation of the MSCP [Wang et al. 2013]. Several efficient approximation algorithms are proposed for specific graphs, such as interval graphs, bipartite graphs, line graphs of trees, etc [Bar-Noy and Kortsarz 1998, Bonomo et al. 2011, Hajiabolhassan et al. 2000, Jansen 2000, Jiang and West 1999, Kroon et al. 1997, Kubicka et al. 1991, Malafiejski 2004, Salavatipour 2003]. However, a number of studies focus on heuristic and metaheuristic algorithms in the recent decades. In this section, we review the most representative and effective MSCP heuristics and metaheuristics, see also the summary in Table 1.1 that contains comments on performances.

1.3.1 Greedy algorithms

Greedy algorithms for the MSCP are fast, simple, and classical. Since they usually achieve poor quality results they are often integrated into other optimization approaches. For instances, they can be used to limit the search space in exact algorithms or to build an initial (pool of) solution(s) in heuristics and metaheuristics strategies.

Two families of improved greedy algorithms are proposed in [Li et al. 2009]: MDSAT(n) and MRLF(n). They are based on two well-known greedy coloring heuristics DSATUR [Brélaz 1979] and RLF [Leighton 1979]. The original DSATUR heuristic employs the saturation degree $dsat$ of a vertex as the selection criterion to dynamically determine the next vertex to color. MDSAT(n) improves DSATUR by considering the impact of coloring a vertex where the impact measure is based on the number of vertices whose $dsat$ would (not) be changed. The original RLF heuristic follows the partition perspective. It colors as many non-adjacent vertices as possible and continues the same coloring procedure with the next color. MRLF(n) improves RLF by considering the probability of using a new color with the target of minimizing the number of colors and makes the current color class as large as possible. The probability measure is based on the cardinality of a subset of uncolored vertices that could be colored with and without using a new color.

A more complicated greedy heuristic (EXSCOL) has been proposed in [Wu and Hao 2012]. It is based on independent sets extraction and is very effective for hard and large graphs. At each iteration, EXSCOL first identifies an independent set S as large as possible by using a tabu search procedure. Secondly, it searches as many independent sets as possible of size $|S|$ to build a pool of candidate independent sets. Then, EXSCOL determines a maximum number of disjoint independent sets by solving a maximum set packing problem. Finally, each extracted independent set is assigned to the smallest available color. The

above process is repeated until the graph becomes empty. Notice that there is no procedure to reconsider the extracted independent sets such that it is impossible for EXSCOL to attain an optimal solution once a "bad" independent set has been extracted.

1.3.2 Neighborhood search heuristics

Neighborhood search heuristics progressively modify a candidate solution by local transformations until a stop condition is reached. The essential components of a classical neighborhood search procedure are the evaluation function, the search space to be explored and the neighborhood structure. According to the neighborhood structure, we can classify the representative and effective MSCP algorithms into two categories: Single neighborhood search and multi-neighborhood search.

Single neighborhood search

The tabu search algorithm TS proposed in [Bouziri and Jouini 2010] starts from a random initial coloring. If there exists conflicting vertices, TS will choose a best move (according to the objective function) to change the color of a conflicting vertex. Otherwise, TS will choose a (non-conflicting) vertex and change its color at random. The above steps are repeated until a stopping criterion is satisfied. Note that TS explores feasible and infeasible regions of the coloring search space.

The breakout local search algorithm (BLS) described in [Benlic and Hao 2012] jointly uses two descent methods and adaptive perturbation strategies to escape from local optima. Like TS, it starts from a random initial solution c and a move consists in changing the color of a vertex. If c is not a proper coloring, BLS applies a first descent strategy to solve the conflicts (to attain a proper coloring). If c is proper, BLS applies another descent search to attain a local optimum (without any improving neighbor). The perturbation strategies differ according to the strength of the perturbation (strong or weak) and the type of moves (directed or random). The perturbation mechanism is adaptively chosen and depends on the number of times a local optimum is visited. Note that BLS also alternates between feasible and infeasible regions of the search space.

Multi-neighborhood search

The MDS(5)+LS algorithm [Helmar and Chiarandini 2011] applies an iterated multi-neighborhood search and also explores the feasible and infeasible regions of the search space. It first employs the "swap" neighborhood operator that moves a vertex v_i from color class V_l to $V_{l'}$ and then moves all its adjacent vertices $v_j \in V_{l'}(\{v_i, v_j\} \in E)$ to V_l. Note that the obtained solution is not necessarily a proper coloring. When no further improvement exists for the swap changes, the "one-move" neighborhood operator is employed. It simply changes the color of a vertex until no improvement can be obtained. At this moment, the local solution is proper. Then it assigns all the vertices with their smallest legal color and changes the color labels according to the sorted cardinality of the color classes V_l ($|V_1| \geq \ldots \geq |V_k|$). Afterwards, a random perturbation operator is applied which consists in moving some vertices from their current color class to another one at random.

1.3.3 Evolutionary algorithms

Different from neighborhood search strategies which are based on a single solution, evolutionary algorithms use a pool of solutions and try to find gradually better solutions by applying genetic operators (e.g., crossover, mutation,...) to solutions of the population.

The most popular evolutionary algorithms for the MSCP jointly use a recombination operator and a local search improvement to explore the search space. They include, for instance, the MASC and MA

Table 1.1: Main heuristics and metaheuristics for the MSCP

Algorithm name	Reference	Type of approach	Comments on performance
MDSAT(n) MRLF(n)	[Li et al. 2009]	Greedy algorithm	A family of improved greedy algorithms based on the well-known greedy coloring strategies DSATUR and RLF.
TS	[Bouziri and Jouini 2010]	Tabu search based on a single neighborhood	A very simple tabu search but the results are better than those of the greedy algorithms MDSAT(n) and MRLF(n).
MDS(5)+LS	[Helmar and Chiarandini 2011]	Neighborhood search based on a multi-neighborhood	An iterated multi-neighborhood search combined with a random perturbation procedure achieving better results than MDSAT(n), MRLF(n) and TS.
BLS	[Benlic and Hao 2012]	Neighborhood search based on a single neighborhood	A breakout local search combining a greedy descent strategy with an adaptive perturbation step. It performs well on the small DIMACS graphs.
EXSCOL	[Wu and Hao 2012]	Greedy algorithm	A complicated greedy algorithm, based on independent sets extraction, which is quite effective for large graphs.
MA	[Moukrim et al. 2013]	Evolutionary algorithm	A genetic algorithm with a two-parents crossover operator combined with a local search based on an hill climbing and a “destroy and repair” procedures. Results are competitive with MASC.
MASC	[Jin et al. 2014]	Evolutionary algorithm	A memetic algorithm based on a double-neighborhood tabu search and a multi-parent crossover operator. Most results are better than those of the local search heuristics.
HSA	[Jin and Hao 2015c]	Evolutionary algorithm	A hybrid search algorithm based on a jointly use of two crossover operators and an iterated double-phase tabu search procedure. The lower and upper bounds obtained by the HSA are highly competitive with the best known results in the literature.

memetic algorithms [Jin et al. 2014, Moukrim et al. 2013], the HSA hybrid search algorithm [Jin and Hao 2015c] and the BQP-PR evolutionary algorithm [Wang et al. 2013]. BQP-PR relies on a binary quadratic programming formulation of the problem and combines a path relinking approach with a tabu search procedure. Besides, the parallel genetic algorithm PGA [Kokosiński and Kwarciany 2007] employs assignment and partition crossovers, first-fit mutation, and proportional selection without any local search improvement.

MA is a hybrid genetic algorithm [Moukrim et al. 2013] that focuses on the feasible search space. It includes a two-parents crossover operator (an adaptive variant of the well-known GPX crossover originally proposed for the classical vertex coloring problem[Galinier and Hao 1999]) and a local search based on a hill-climbing and a "destroy and repair" procedures. During the local search phase, the hill-climbing procedure is first applied to improve the current solution. The "destroy and repair" procedure is then used to escape from the local optimum. It randomly removes some vertices and re-inserts each of them into its largest available color class while keeping the solution proper. If there is no such a color class, the vertex is moved to a new color class. MA employs the above two procedures alternately until no further improvement can be obtained.

In this thesis, we introduce two new population-based heuristics. In Chapter 3, we present the MASC memetic algorithm [Jin et al. 2014] which is based on a multi-parent crossover operator (MGPX) and a double-neighborhood tabu search procedure. MGPX is another variant of the GPX crossover [Galinier and Hao 1999]. It builds the color classes of the offspring (which is always a proper coloring) one by one and transmits entire color classes as large as possible until all vertices of the offspring are colored. Besides, the tabu search procedure applies two different and complementary neighborhoods N_1 and N_2 in a token-ring way to explore the search space. N_1 is an "exchange" operator which swaps some vertices from two color classes. N_2 is a "one-move" operator: It changes the color of a single vertex. MASC only explores the feasible search space of the MSCP since N_1 and N_2 include all the proper colorings that can be obtained from the current candidate solution by applying the "exchange" or "one-move" operator.

In Chapter 4, we present the HSA algorithm which is a hybrid search algorithm [Jin and Hao 2015c] that alternates between feasible and infeasible regions of the search space. HSA relies on a double-crossover recombination method and an iterated double-phase tabu search procedure. The recombination method jointly uses a diversification-guided crossover and a grouping-guided crossover to generate promising offspring solutions. During the double-phase tabu search procedure, it first identifies if the given solution c is a proper coloring. If c is proper, the first tabu search is called to improve its sum of colors. Otherwise, another tabu search is called for conflict resolution to attain a proper coloring which is further improved by the first tabu search according to the objective function. The double-phase tabu search only employs a "one-move" operator that changes the color of a single vertex.

1.4 Bounds

1.4.1 Theoretical bounds

For any undirected simple graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, the chromatic number $\chi(G)$ is the smallest number of colors needed to color the vertices of G such that a proper k-coloring is obtained and the chromatic sum $\sum(G)$ is the minimum sum of the colors assigned to all the vertices among all proper k-colorings of G. In this section, we list the theoretical lower and upper bounds of the MSCP according to [Kokosiński and Kwarciany 2007, Moukrim et al. 2013, Thomassen et al. 1989]:

$$\begin{aligned} &\sum(G) \leq n+m \\ &\lceil\sqrt{8m}\rceil \leq \sum(G) \leq \lfloor\frac{3(m+1)}{2}\rfloor \\ &n+\frac{\chi(G)(\chi(G)-1)}{2} \leq \sum(G) \leq \lfloor\frac{n(\chi(G)+1)}{2}\rfloor \end{aligned} \tag{1.4}$$

From Eq. (1.4), one easily observes that the best theoretical lower and upper bounds available for the MSCP are respectively $LB_t = \max\{\lceil\sqrt{8m}\rceil, n+\frac{\chi(G)(\chi(G)-1)}{2}\}$ and $UB_t = \min\{n+m, \lfloor\frac{3(m+1)}{2}\rfloor, \lfloor\frac{n(\chi(G)+1)}{2}\rfloor\}$.

1.4.2 Computational bounds

Recall that the MSCP is to find a proper k-coloring while minimizing the sum of the colors assigned to the vertices. The quantity of Eq. (1.2) gives a "computational" upper bound for the MSCP.

Let $G' = (V, E')$ $(E' \subseteq E)$ be any partial graph of $G = (V, E)$, $\sum(G')$ is a lower bound of $\sum(G)$ since any proper coloring of G must be a proper coloring of G': $\sum(G) \geq \sum(G')$.

Partial graphs considered in the literature to estimate the "computational" lower bound f_{LB} include bipartite graphs (trees and paths) [Kroon et al. 1997, Garey and Johnson 2002] and cliques, the decomposition into cliques providing better bounds [Moukrim et al. 2010]. Let $c = \{S_1, S_2, \ldots, S_k\}$ be a clique decomposition of G, then the quantity from Eq. (1.5) gives a "computational" lower bound for the MSCP.

To obtain a clique decomposition, one popular approach is to find a proper coloring of the complementary graph $\bar{G}$ of G [Helmar and Chiarandini 2011, Jin and Hao 2015c, Moukrim et al. 2013, Wu and Hao 2013b], since each color class (independent set) of $\bar{G}$ is a clique of G.

$$f_{LB}(c) = \sum_{l=1}^{k} \frac{|S_l|(|S_l|+1)}{2} \tag{1.5}$$

1.5 Benchmark

We consider a set of 94 frequently used instances in order to evaluate the performance of MSCP algorithms, 58 of which were part of the COLOR 2002–2004 workshops, the 36 others come from the DIMACS challenge. This benchmark is available online from `http://mat.gsia.cmu.edu/COLOR02`. Compared to the well-known DIMACS instances, the COLOR 2002-2004 ones are relatively easy except the four "wap" large graphs. These instances refer to various topologies and densities, which can be classified into the 14 following types:

- Twelve classical random graphs (DSJC$n.d$, $n \in \{125, 250, 500, 1\,000\}$, $d \in \{1, 5, 9\}$);
- Three geometric graphs (DSJR500.d, $d \in \{1c, 1, 5\}$);
- Six flat graphs (flat300_χ_0 with $\chi \in \{20, 26, 28\}$ and flat1000_χ_0 with $\chi \in \{50, 60, 76\}$);
- Twelve Leighton graphs (le450_χa, le450_χb, le450_χc, le450_χd, $\chi \in \{5, 15, 25\}$);
- Four latin square graph (latin_sqr_10 and qg.orderχ, $\chi \in \{30, 40, 50\}$);
- Two very large random graphs (C2000.5 and C4000.5);
- Fourteen graphs based on register allocation (fpsol2.i.a, inithx.i.a, zeroin.i.a, mulsol.i.b, $a \in \{1, 2, 3\}$ and $b \in \{1, 2, 3, 4, 5\}$);
- Two graphs from the scheduling area (school1 and school1_nsh);

Table 1.2: Main characteristics of the MSCP benchmark (94 instances)

Graph G	n	m	d	$\chi(G)$	LB_t	UB_t	Graph G	n	m	d	$\chi(G)$	LB_t	UB_t
myciel3	11	20	0.36	4	17	27	zeroin.i.1	211	4100	0.19	49	1387	4311
myciel4	23	71	0.28	5	33	69	zeroin.i.2	211	3541	0.16	30	646	3270
myciel5	47	236	0.22	6	62	164	zeroin.i.3	206	3540	0.17	30	641	3193
myciel6	95	755	0.17	7	116	380	wap05	905	43081	0.11	50	2130	23077
myciel7	191	2360	0.13	8	219	859	wap06	947	43571	0.10	40	1727	19413
anna	138	493	0.05	11	193	631	wap07	1809	103368	0.06	≤ 46	2844	42511
david	87	406	0.11	11	142	493	wap08	1870	104176	0.06	≤ 47	2951	44880
huck	74	301	0.11	11	129	375	qg.order30	900	26100	0.06	30	1335	13950
jean	80	254	0.08	10	125	334	qg.order40	1600	62400	0.05	40	2380	32800
homer	561	1628	0.01	13	639	2189	qg.order60	3600	212400	0.03	60	5370	109800
queen5.5	25	160	0.53	5	36	75	DSJC125.1	125	736	0.09	5	135	375
queen6.6	36	290	0.46	7	57	144	DSJC125.5	125	3891	0.50	17	261	1125
queen7.7	49	476	0.40	7	70	196	DSJC125.9	125	6961	0.90	44	1071	2812
queen8.8	64	728	0.36	9	100	320	DSJC250.1	250	3218	0.10	≤ 8	278	1125
queen8.12	96	1368	0.30	12	162	624	DSJC250.5	250	15668	0.50	≤ 28	628	3625
queen9.9	81	1056	0.33	10	126	445	DSJC250.9	250	27897	0.90	≤ 72	2806	9125
queen10.10	100	1470	0.30	11	155	600	DSJC500.1	500	12458	0.10	≤ 12	566	3250
queen11.11	121	1980	0.27	11	178	726	DSJC500.5	500	62624	0.50	≤ 47	1581	12000
queen12.12	144	2596	0.25	12	210	936	DSJC500.9	500	112437	0.90	≤ 126	8375	31750
queen13.13	169	3328	0.23	13	247	1183	DSJC1000.1	1000	49629	0.10	≤ 20	1190	10500
queen14.14	196	4186	0.22	14	287	1470	DSJC1000.5	1000	249826	0.50	≤ 82	4321	41500
queen15.15	225	5180	0.21	15	330	1800	DSJC1000.9	1000	449449	0.90	≤ 222	25531	111500
queen16.16	256	6320	0.19	16	376	2176	DSJR500.1	500	3555	0.03	12	566	3250
school1	385	19095	0.26	14	476	2887	DSJR500.1c	500	121275	0.97	84	3986	21250
school1-nsh	352	14612	0.24	14	443	2640	DSJR500.5	500	58862	0.47	122	7881	30750
games120	120	638	0.09	9	156	600	flat300_20_0	300	21375	0.48	20	490	3150
miles250	128	387	0.05	8	156	515	flat300_26_0	300	21633	0.48	26	625	4050
miles500	128	1170	0.14	20	318	1298	flat300_28_0	300	21695	0.48	28	678	4350
miles750	128	2113	0.26	31	593	2048	flat1000_50_0	1000	245000	0.49	50	2225	25500
miles1000	128	3216	0.40	42	989	2752	flat1000_60_0	1000	245830	0.49	60	2770	30500
miles1500	128	5198	0.64	73	2756	4736	flat1000_76_0	1000	246708	0.49	76	3850	38500
fpsol2.i.1	496	11654	0.09	65	2576	12150	le450_5a	450	5714	0.06	5	460	1350
fpsol2.i.2	451	8691	0.09	30	886	6990	le450_5b	450	5734	0.06	5	460	1350
fpsol2.i.3	425	8688	0.10	30	860	6587	le450_5c	450	9803	0.10	5	460	1350
mug88_1	88	146	0.04	4	94	220	le450_5d	450	9757	0.10	5	460	1350
mug88_25	88	146	0.04	4	94	220	le450_15a	450	8168	0.08	15	555	3600
mug100_1	100	166	0.03	4	106	250	le450_15b	450	8169	0.08	15	555	3600
mug100_25	100	166	0.03	4	106	250	le450_15c	450	16680	0.17	15	555	3600
2-Insert_3	37	72	0.11	4	43	92	le450_15d	450	16750	0.17	15	555	3600
3-Insert_3	56	110	0.07	4	62	140	le450_25a	450	8260	0.08	25	750	5850
inithx.i.1	864	18707	0.05	54	2295	19571	le450_25b	450	8263	0.08	25	750	5850
inithx.i.2	645	13979	0.07	31	1110	10320	le450_25c	450	17343	0.17	25	750	5850
inithx.i.3	621	13969	0.07	31	1086	9936	le450_25d	450	17425	0.17	25	750	5850
mulsol.i.1	197	3925	0.20	49	1373	4122	latin_sqr_10	900	307350	0.76	≤ 97	5556	44100
mulsol.i.2	188	3885	0.22	31	653	3008	C2000.5	2000	999836	0.50	≤ 145	12585	147000
mulsol.i.3	184	3916	0.23	31	649	2944	C4000.5	4000	4000268	0.50	≤ 259	37670	522000
mulsol.i.4	185	3946	0.23	31	650	2960							
mulsol.i.5	186	3973	0.23	31	651	2976							

- Twenty four graphs from the Donald Knuth's Stanford GraphBase (miles$n$ with $n \in \{250, 500, 750, 1000, 1500\}$, anna, david, huck, jean, homer, games120, queen8.12, and queen$a.a$, $a \in \{5, \ldots, 16\}$);
- Five graphs based on the Mycielski transformation (myciel$a$, $a \in \{3, 4, 5, 6, 7\}$);
- Four graphs that have a hard-to-find four clique embedded (mug$n$_a, $n \in \{88, 100\}, a \in \{1, 25\}$);
- Two "insertion" graphs (2-Insert_3 and 3-Insert_3);
- Four graphs from real-life optical network design problems (wap05, wap06, wap07, and wap08).

Table 1.2 gives the detailed characteristics of the benchmark. Columns 1–5 and 8–12 present the number n of vertices, the number m of edges, the density $d = 2m/n(n-1)$ and the chromatic number $\chi(G)$ of each graph. Columns 6–7 and 13–14 indicate the best theoretical lower and upper bounds (LB_t and UB_t respectively). Note that, since the chromatic number $\chi(G)$ of some difficult graphs are still unknown, we use the minimum k for which a k-coloring has been reported for G instead of $\chi(G)$ to compute UB_t and LB_t using the $\min$ / $\max$ equations introduced in Section 1.4.1.

2

The bandwidth (multi)coloring problem

2.1 Introduction

The Bandwidth Coloring Problem (BCP) and the Bandwidth MultiColoring Problem (BMCP) are two other important generalizations of the vertex coloring problem. The BCP is known as the restricted T-coloring problem and the BMCP is also known as the restricted set T-coloring problem [Hale 1980, Roberts 1991]. The T-coloring and the set T-coloring problems have been introduced by Hale in connection with the channel assignment problem in communications [Hale 1980]. Tresman presents a comprehensive survey of the literature on T-coloring [Tesman 1989] and later Roberts summarizes the basic results of the T-coloring problem and provides some connected problems and their variations [Roberts 1991]. The studies on heuristic and metaheuristic methods for T-coloring and set T-coloring include the Dsatur algorithm [Costa 1993], a generic tabu search algorithm [Dorne and Hao 1999], two hybrid evolutionary approaches combining an ACO algorithm and a tabu search [Aicha et al. 2010] for instance.

The BCP and BMCP are notable for their applicability to a number of important applications in particular in the area of frequency assignment in mobile networks. There is extensive literature on this application, including exact approaches, heuristics and metaheuristics [Aardal et al. 2007, Allen et al. 2002, Bernardo et al. 2010, Castelino et al. 1996, Gamst 1986, Hale 1980, Hao et al. 1998, Salcedo-Sanz et al. 2004, Walser 1996, Wang et al. 2008, Zoellner and Beall 1977]. Aardal *et al.* provides a comprehensive survey of the frequency assignment problem [Aardal et al. 2007], which introduces a broad description of the practical settings in which frequency assignment is applied and a classification of the different models and formulations in the literature.

Since the BCP and BMCP are NP-hard problems, much efforts have been devoted to the development of various heuristics and metaheuritics, such as neighborhood search algorithms based on a single solution [Lim et al. 2005, Bui and Nguyen 2006, Lai and Lü 2013, Prestwich 2008, Jin and Hao 2015a] and evolutionary algorithms based on a pool of solutions [Malaguti and Toth 2008, Lai et al. 2014, Dorne and Hao 1995, Hao and Dorne 1996]. More methods can be found in the 2008 special issue of Discrete Applied Mathematics Johnson et al. [2008] dedicated to computational methods for graph coloring and its generalizations. This chapter thus aims to provide a detailed review of different BCP and BMCP solution approaches proposed in the recent literature.

In the following sections, we first provide a general definition of the BCP and BMCP, followed by the introduction of the studied heuristic and metaheuristic algorithms. Finally, the benchmark instances which are frequently used the BCP and the BMCP algorithms are presented.

2.2 Definitions and formulation

Given an undirected graph $G = (V, E)$ with vertex set $V = \{v_1, \ldots, v_n\}$, edge set $E \subset V \times V$ and edge weight $d(i, j)$ for each edge $\{v_i, v_j\} \in E$ ($d(i, j)$ can also be considered as a distance between two adjacent vertices v_i and v_j), a legal bandwidth coloring is a function $c : V \rightarrow \{1, 2, ..., k\}$ such that the absolute difference between $c(v_i)$ and $c(v_j)$ of an edge $\{v_i, v_j\} \in E$ is at least $d(i, j)$, i.e.,

$$|c(v_i) - c(v_j)| \geqslant d(i, j),\ \forall \{v_i, v_j\} \in E \tag{2.1}$$

The bandwidth coloring problem is to find a legal bandwidth coloring of G with k minimum. The problem of k-BCP corresponds to BCP with k being fixed where one seeks a legal bandwidth coloring with k colors.

The BCP can be generalized as the bandwidth multicoloring problem where each vertex v_i receives a subset $S(i) \subset \{1, 2, ..., k\}$ of $p(i)$ different colors. A legal bandwidth multicoloring must satisfy two distance constraints: The absolute difference between any member of $S(i)$ and $S(j)$ is at least $d(i, j)$ for each edge $\{v_i, v_j\} \in E$ and the absolute difference between two distinct values in $S(i)$ is at least $d(i, i)$ for each vertex ($d(i, i)$ is the color separation distance for vertex v_i), i.e.,

$$\begin{aligned} |x - y| \geqslant d(i, j), \forall \{v_i, v_j\} \in E,\ \forall x \in S(i), y \in S(j) \\ |a - b| \geqslant d(i, i), \forall v_i \in V, \forall a, b \in S(i) \end{aligned} \tag{2.2}$$

The BMCP is to find a legal bandwidth multicoloring of G with k minimum.

Besides, the integer liner programming for the BCP and the BMCP are proposed in [Malaguti 2009]. The BCP is formulated as follows:

$minimize \quad k$

subject to

$$\begin{array}{lll} (a) & k \geq y_u u, & u \in U \\ (b) & \sum_{u \in U} x_{iu} = 1, & i \in V \\ (c) & x_{iu} + x_{jl} \leq 1, & (i, j) \in E, u \in U, l \in \{u - d(i, j) + 1, \ldots, u + d(i, j) - 1\} \\ (d) & x_{iu} \leq y_u, & i \in V, u \in U \\ (e) & x_{iu} \in \{0, 1\}, & i \in V, u \in U \\ (f) & y_u \in \{0, 1\}, & u \in U \end{array} \tag{2.3}$$

where $U = \{1, \ldots, \bar{u}\}$ ($\bar{u}$ is an upper bound on the number of colors needed to color the graph) is the set of available colors, $x_{iu} = 1 (i \in V, u \in U)$ if v_i is assigned color u (0 otherwise), and the $y_u = 1$ if color u is used (0 otherwise).

The objective function is to minimize the maximum color used with constraint (a). Constraint (b) imposes each vertex v_i to receive only one color. Constraint (c) indicates that the absolute value of the difference between the colors assigned to vertices v_i and v_j must be at least equal to $d(i, j)$. Constraint (d)

assures that if a vertex v_i receives a color u, the color u is used. Constraints (e) and (f) impose the variables to be binary.

The formulation of the BMCP is given by the following binary program [Malaguti and Toth 2010].

$$minimize \quad k$$

subject to

$$\begin{array}{lll}
(a) & k \geq y_u u, & u \in U \\
(b) & \sum_{u \in U} x_{iu} = w_i, & i \in V \\
(c) & x_{iu} + x_{jl} \leq 1, & (i,j) \in E, u \in U, l \in \{u - d(i,j) + 1, \ldots, u + d(i,j) - 1\} \\
(d) & x_{iu} \leq y_u, & i \in V, u \in U \\
(e) & x_{iu} \in \{0,1\}, & i \in V, u \in U \\
(f) & y_u \in \{0,1\}, & u \in U
\end{array} \tag{2.4}$$

where w_i is the number of different colors for each vertex of the graph. The above constraints have the same meanings as the constraints for the BCP except constraint (b). It imposes that each vertex receives the specified number of colors.

2.3 Heuristics and metaheuristics

2.3.1 Neighborhood search heuristics

Neighborhood search heuristic is one of the most effective methods for solving the BCP and BMCP. The search space, the evaluation function and the neighborhood function form a search strategy for a neighborhood search algorithm. According to the search space, we can classify the representative and effective BCP and BMCP algorithms into two categories: Complete coloring strategy and partial coloring strategy.

Complete coloring strategy

The squeaky wheel optimization with tabu search heuristic (SWO+TS) [Lim et al. 2005] combines three important components: Greedy techniques, squeaky wheel optimization (SWO) and tabu search. SWO employs a traditional construct-analyze-prioritize cycle. The constructor procedure uses a greedy algorithm to build the solution which assigns colors to the vertices greedily based on the vertex sequence (the order of the vertices). The analyzer procedure identifies the vertices whose color exceeds a given target (they are called "trouble makers") and those vertices are given a blame value. The prioritizer procedure modifies the vertex sequence according to the blame value. SWO is run for a number of iterations and then the best solution is submitted to the tabu search procedure. The tabu search procedure applies a "swap" move that exchanges two vertices in the vertex sequence of a solution to generate a set of neighborhood solutions.

The agent-based algorithm ABGC [Bui and Nguyen 2006] jointly uses an iterated greedy algorithm, an ants repair phase and a local optimization procedure. It starts from an initial coloring c generated by an iterated greedy algorithm and applies a colony of ants to resolve the conflicts in c. Note that the ants are distributed among the vertices based on the conflicts at the vertices. If c is proper, ABGC applies a local optimization algorithm to further improve this coloring and the number of colors is decreased by 1 when c is improved. Otherwise, ABGC increases the number of colors by 1. ABGC is not an ant colony optimization

algorithm (ACO) [Dorigo et al. 1999] since each ant only colors a subset of the vertices of the graph and the ants do not use pheromone to communicate with each other.

The multistart iterated tabu search (MITS) [Lai and Lü 2013] integrates an iterated tabu search algorithm (ITS) with a multistart method. Starting from a random coloring, MITS applies an ITS procedure to improve this solution. The evaluation function calculates the degree of constraint violations. ITS applies tabu search to explore the search space by applying a "one-move" neighborhood that changes a conflicting vertex v_i from its original color class V_l to another color class $V_{l'}$. When the tabu search procedure terminates, ITS employs a perturbation operator to jump out of the local optimum and then calls the tabu search again. ITS stops when the current best solution cannot be improved within a given maximum number of iterations. MITS combines this ITS procedure with a multistart technique and repeats this combination until a legal k-coloring is found or a timeout limit is reached.

Partial coloring strategy

The forward checking colouration neighborhood search (FCNS) [Prestwich 2008] is based on a partial coloring strategy and combines a local search with a constraint propagation algorithm. FCNS enhances the *impasse* appraoch [Prestwich 2002] by adding the constraint programming technique of forward checking such that some colors can be pruned during the search process. FCNS uses a *domain* to include the available colors of a vertex. FCNS selects uncolored vertices heuristically according to the *domain* cardinality to extend the partial solution until reaching a *dead-end*, i.e., when the *domain* of a vertex is empty. Afterwards, FCNS removes one or more colored vertices in a heuristic way to resolve the conflicts. This procedure is repeated until all the vertices are colored or a terminate condition is met.

The adaptive memory programming (AMP) [Marti et al. 2010] combines a memory-based construction heuristic and a tabu search procedure. AMP constructs the partial coloring solution by selecting a vertex with its lowest evaluation value in the candidate list which incorporates frequency information until all the vertices are colored. Then, AMP uncolors some vertices whose color is larger than the given target, employs a tabu search procedure to randomly select a vertex to color and uncolor all the adjacent vertices so as to avoid violating the distance constraints. Notice that the color for the selected vertex is chosen based on the consideration that it minimizes the sum of edge weights incident with uncolored vertices. The constructive procedure is invoked once the tabu search procedure is terminated. This procedure is repeated until all the vertices are colored or a stop condition is met.

In Chapter 5 of this thesis, we introduce a learning-based hybrid search (LHS) [Jin and Hao 2015a]. It combines a construction phase to progressively build feasible partial colorings and a local search phase to reestablish feasibility when an illegal partial solution is encountered. LHS repeats two phases: A coloring construction phase and a repair phase. Starting from an empty solution, the coloring construction phase selects an uncolored vertex and tries to assign it an available color according to a learning-based guiding function. This phase extends a proper partial solution until no available color is possible for the selected vertex (a *dead-end* vertex). Then, the repair phase is invoked after assigning a random color to this *dead-end* vertex. Obviously, the current partial coloring is an illegal coloring owing to the *dead-end* vertex violates the distance constraints. The repair phase takes the current partial coloring as an input and applies a tabu search to resolve the conflicts. The tabu search procedure is terminated when a legal partial solution is found or a given target iteration is reached. If the former occurs, LHS switches back to the coloring construction phase. Otherwise, LHS drops the current partial solution and restarts a new round of "construction-repair" process by updating the guiding function to learn from this failure. LHS repeats the above process until a target number of maximum tries is reached or a complete legal k-coloring is obtained.

Table 2.1: Main heuristic and metaheuristic approaches for the BCP and the BMCP

Algorithm name	Reference	Type of approach	Comments on performance
SWO+TS	[Lim et al. 2005]	Neighborhood search based on a complete coloring strategy	A neighborhood search combining the greedy techniques, squeaky wheel optimization and the tabu search procedure.
ABGC	[Bui and Nguyen 2006]	Neighborhood search based on a complete coloring strategy	A local search applies a number of collaborative ants to attain a complete coloring which is further improved by a local optimization algorithm.
EA	[Malaguti and Toth 2008]	Evolutionary algorithm	An evolutionary algorithm combining an effective tabu search algorithm with population management procedures, which performs well for the BCP benchmark as well as the BMCP benchmark.
FCNS	[Prestwich 2008]	Neighborhood search based on a partial coloring strategy	Combination of a local search and a constraint propagation algorithm, which is effective for the BCP benchmark.
AMP	[Marti et al. 2010]	Neighborhood search based on a partial coloring strategy	A constructive heuristic that combines a memory-based construction approach and a tabu search procedure.
MITS	[Lai and Lü 2013]	Neighborhood search based on a complete coloring strategy	An iterated tabu search algorithm combined with a multistart method, which is quite effective for the BMCP benchmark.
PR	[Lai et al. 2014]	Evolutionary algorithm	A memetic algorithm that combines a population based path relinking method and a tabu search based local search procedure, some results are improved when compared to the MITS.
LHS	[Jin and Hao 2015a]	Neighborhood search based on a partial coloring strategy	A constructive algorithm that combines a construction phase to progressively build feasible partial colorings and a local search phase to reestablish feasibility when an illegal partial solution is encountered. Results are highly competitive with the best known results in the literature.

2.3.2 Evolutionary algorithms

Different from neighborhood search algorithms, evolutionary algorithms use a pool of solutions and need to balance the quality and the distance among these solutions.

The evolutionary approach (EA) presented in [Malaguti and Toth 2008] constructs an initial solution using the DSATUR algorithm [Brélaz 1979] and combines an effective tabu search algorithm with a population management procedure. Malaguti and Toth propose six different crossover operators and experimentally show that the *distance-crossover* is the best performing among these crossovers. The *distance-crossover* transmits all "tight distance" pairs from one parent P_1 to the offspring and keeps the same color assignment. The "tight distance" pair is two adjacent vertices whose absolute value of the difference between the colors is equal to the weight of the associated edge. Then, it transmits "tight distance" pairs from the other parent P_2 while keeping the same color assignment if this pair does not violate the distance constraints with respect to the colored vertices of the offspring. For the remaining uncolored vertices, it uses a greedy algorithm to obtain a partial proper k-coloring. The tabu search procedure explores partial k colorings where all the distance constraints are satisfied. It is based on an adaption of the *impasse* neighborhood [Prestwich 2002]. The evaluation function measures the total distances of the set of uncolored vertices. A neighborhood solution is obtained by assigning a color to a vertex and uncoloring its adjacent vertices to avoid breaking the distance constraints.

The path relinking algorithm (PR) [Lai et al. 2014] combines a population based relinking method and a tabu search procedure. From a scratch, an initial population includes different solutions which are randomly generated and optimized by a tabu search procedure [Lai and Lü 2013]. Then, PR selects a solution pair at random and builds paths from the initiating solution to the guiding solution in order to generate two offspring solutions. Afterwards, PR employs the tabu search procedure to improve the offspring solution and a population updating procedure to determine whether this improved solution should be inserted into the population and which solution in the population should be replaced.

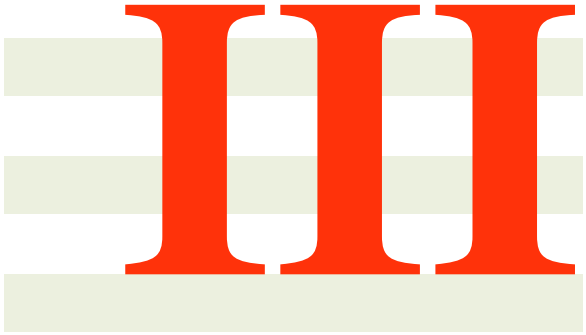
2.4 Benchmark

In order to evaluate the performance of BCP and BMCP algorithms, a data set of 66 well-known benchmark instances have been frequently used. The first set of benchmarks is composed of 33 BCP instances [Johnson et al. 2002]. These instances belong to three types: GEOM*n*, GEOM*na* and GEOM*nb* (where n denotes the number of vertices of the graph). The first type refers to sparse graphs, while the two other types correspond to dense graphs. The second set of 33 BMCP instances is transformed from the BCP instances.

Table 2.2 gives the detailed characteristics of the BCP and BMCP instances. Column 2–7 and column 8–13 present the number of vertices (n), the number of edges (m), the density of the graph (d), the average degree of a vertex (deg_{ave}), the minimum degree of a vertex (deg_{min}) and the maximum degree of a vertex (deg_{max}) respectively.

Table 2.2: Main characteristics of the BCP and the BMCP benchmark (66 instances)

	BCP						BMCP					
Instance	n	m	d	deg_{ave}	deg_{min}	deg_{max}	n	m	d	deg_{ave}	deg_{min}	deg_{max}
GEOM20	20	20	0.1053	2.00	0	4	118	1048	0.1518	17.76	4	27
GEOM20a	20	37	0.1947	3.70	0	7	100	1327	0.2681	26.54	0	44
GEOM20b	20	32	0.1684	3.20	1	6	40	132	0.1692	6.60	2	11
GEOM30	30	50	0.1149	3.33	1	6	143	1419	0.1398	19.85	5	35
GEOM30a	30	81	0.1862	5.40	2	10	171	3288	0.2262	38.45	12	63
GEOM30b	30	81	0.1862	5.40	2	10	69	447	0.1905	12.96	6	21
GEOM40	40	78	0.1000	3.90	0	6	220	3074	0.1276	27.95	1	41
GEOM40a	40	146	0.1872	7.30	3	12	203	4473	0.2182	44.07	20	67
GEOM40b	40	157	0.2013	7.85	2	13	84	743	0.2131	17.69	3	27
GEOM50	50	127	0.1037	5.10	0	9	285	4935	0.1219	34.63	0	66
GEOM50a	50	238	0.1943	9.52	4	16	302	9649	0.2123	63.90	22	115
GEOM50b	50	249	0.2033	9.96	3	17	104	1140	0.2128	21.92	7	35
GEOM60	60	185	0.1045	6.17	1	10	315	6174	0.1248	39.20	7	59
GEOM60a	60	339	0.1915	11.30	6	18	362	13105	0.2005	72.40	33	124
GEOM60b	60	366	0.2068	12.20	3	20	127	1785	0.2231	28.10	7	44
GEOM70	70	267	0.1106	7.63	1	13	384	8584	0.1167	44.71	11	69
GEOM70a	70	459	0.1901	13.11	7	20	379	14821	0.2069	78.21	33	120
GEOM70b	70	488	0.202	13.94	4	24	148	2212	0.2033	29.89	10	53
GEOM80	80	349	0.1104	8.72	3	14	465	12927	0.1198	55.60	21	89
GEOM80a	80	612	0.1936	15.30	7	23	389	15545	0.2060	79.90	28	129
GEOM80b	80	663	0.2098	16.575	5	29	169	3028	0.2133	35.83	11	62
GEOM90	90	441	0.1101	9.80	5	15	530	16180	0.1154	61.05	27	96
GEOM90a	90	789	0.1970	17.53	8	25	454	20455	0.1989	90.11	37	136
GEOM90b	90	860	0.2147	19.11	6	34	184	3602	0.2139	39.15	12	64
GEOM100	100	547	0.1105	10.94	5	18	581	19829	0.1177	68.25	28	104
GEOM100a	100	992	0.2000	19.84	9	28	528	28496	0.2048	107.93	51	180
GEOM100b	100	1050	0.2121	21.00	5	37	200	4429	0.2226	44.29	11	72
GEOM110	110	638	0.1064	11.60	6	19	643	24799	0.1201	77.14	37	127
GEOM110a	110	1207	0.2013	21.94	12	32	602	38783	0.2144	128.85	62	177
GEOM110b	110	1256	0.2095	22.84	5	39	220	5163	0.2143	46.94	12	85
GEOM120	120	773	0.1083	12.88	6	21	680	27759	0.1202	81.64	33	126
GEOM120a	120	1434	0.2000	23.90	13	35	664	46429	0.2109	139.80	64	212
GEOM120b	120	1491	0.2088	24.85	5	43	235	5779	0.2102	49.18	11	81



Personal contributions

3

MASC: A Memetic Algorithm for minimum Sum Coloring

In this chapter, we present a memetic algorithm (MASC) for computing upper bounds of the minimum sum coloring problem (MSCP). The proposed MASC algorithm employs a tabu search procedure with two neighborhoods and a multi-parent crossover operator. Experiments on a set of 77 well-known DIMACS and COLOR 2002-2004 benchmark instances show that the proposed algorithm competes favorably with the current best performing algorithms for the MSCP. The content of this chapter is published in [Jin et al. 2014].

Contents

3.1 Introduction

This chapter is dedicated to the minimum sum coloring problem which is formally presented in Chapter 1. Recall that given an undirected graph $G = (V, E)$, the MSCP is to find a legal assignment of colors (represented by natural numbers) to each vertex of G such that the total sum of the colors assigned to the vertices

is minimized. We introduce a memetic algorithm MASC for the minimum sum coloring problem which relies on three key components. First, a double-neighborhood tabu search procedure (DNTS) is especially designed for the MSCP. DNTS is based on a token-ring application of two complementary neighborhoods to explore the search space and a perturbation strategy to escape from local optima. Second, a multi-parent crossover operator is used for solution recombination. Basically, it tries to transmit large color classes from the parents to the offspring. Finally, a population updating mechanism is devised to determine how the offspring solution is inserted into the population.

We evaluate the performance of MASC on 77 frequently used instances from DIMACS and COLOR 2002-2004 graph coloring competitions. The computational results show that MASC can match the best known results in the literature for most cases. In particular, it improves the previous best solution for 15 graphs for which an upper bound is known.

This chapter is organized as follows. Next section describes the general framework and the components of our MASC memetic algorithm, including the population initialization, the crossover operator and the double-neighborhood tabu search procedure. Detailed computational results and comparisons with five state-of-the-art algorithms are presented in Section 3.3. Before concluding, Section 3.4 investigates and analyzes three key issues of the proposed memetic algorithm.

3.2 Components of the MASC approach

A memetic algorithm is a population-based approach where the traditional mutation operator is replaced by a local search procedure [Moscato and Cotta 2003, Neri et al. 2012]. Memetic algorithms are among the most powerful paradigms for solving NP-hard combinatorial optimization problems. In particular, they have been successfully applied to the tightly related VCP [Galinier and Hao 1999, Lü and Hao 2010, Malaguti et al. 2008, Porumbel et al. 2010].

Our MASC algorithm follows the general principle for designing effective memetic algorithms for discrete optimization [Hao 2012] and is summarized in Algorithm 1. After population initialization, MASC repeats a series of generations (limited to $MaxGeneration$) to explore the search space which is defined by the set of all proper k-colorings (k is not a fixed value, Section 3.2.1). At each generation, two or more parents are selected at random (line 6) and used by the dedicated crossover operator to generate an offspring solution (line 7, Section 3.2.3). The offspring solution is then improved by a double neighborhood tabu search (line 8, 3.2.4). If the improved offspring has a better sum of colors, it is then used to update the current best solution found so far (lines 9-10). Finally, the population updating criterion decides whether the improved offspring will replace one existing individual of the population or not (line 12, Section 3.2.5).

3.2.1 Search space and evaluation function

The search space explored by MASC is the set $\mathcal{C}$ of all *proper* k-colorings of G (k is not fixed). For a given proper k-coloring c, its quality is directly assessed by the sum of colors $f(c) = \sum_{v \in V} c(v) = \sum_{l=1}^{k} l|V_l|$.

3.2.2 Initial population

Our algorithm begins with a population P of p feasible colorings. This population can be obtained by any graph coloring algorithm that is able to generate different proper colorings for a graph. In our case, we employ the well-known TABUCOL [Hertz and de Werra 1987], more precisely its improved version introduced in [Galinier and Hao 1999]. For a given graph G, TABUCOL tries to find a proper k-coloring

Algorithm 1 An overview of the MASC memetic algorithm for the MSCP

```
1: input: A graph G
2: output: The minimum sum coloring c_* and f(c_*) found
3: Population_Initialization(P, p) /* Population P has p solutions, Section 3.2.2 */
4: f_* ← min_{c∈P} f(c) /* f_* records the best objective value found so far */
5: for i ← 1 to MaxGeneration do
6:    P' ← Selection(P) /* Select 2 or more parents at random for crossover */
7:    o ← Crossover(P') /* Crossover to get an offspring solution, Section 3.2.3 */
8:    o ← DNTS(o) /* Improve o with the DNTS procedure, Section 3.2.4*/
9:    if f(o) < f_* then
10:      f_* ← f(o); c_* ← o
11:   end if
12:   Population_Updating(P, o) /* Section 3.2.5 */
13: end for
14: return f_*, c_*
```

where k is the best known result for the VCP, i.e., the smallest k for which a k-coloring is known in the literature. If TABUCOL cannot reach a proper k-coloring for the current k value, TABUCOL is restarted with k increased by 1 (this makes the task of finding a legal coloring easier). This process is repeated until a proper k-coloring is obtained. Each resulting k-coloring is then submitted to the dedicated DNTS procedure to improve its coloring sum (see Section 3.2.4). Each improved k-coloring is finally inserted into P if the coloring is not already present in P (discarded otherwise). This process is repeated until P is filled with p different k-colorings. Notice that the solutions generated by TABUCOL may take different k values due to the stochastic nature of TABUCOL. Also in Section 3.4.3, we provide a comparative study to show to which extent the solutions can be improved by the proposed MASC approach.

3.2.3 Crossover operator

The crossover operator is an important component in a population-based algorithm. It is used to generate one or more new offspring individuals to discover new promising search areas. MASC uses a multi-parent crossover operator, called MGPX, which is similar to the one introduced in [Hamiez and Hao 2002] as a variant of the well-known GPX crossover first proposed in [Galinier and Hao 1999] for the VCP (restricted to two parents). MGPX generates only one offspring solution o from α parents randomly chosen from P, where α varies from 2 to 4 according to n and the best k-coloring found for the VCP (see Eq. (3.1)).

$$\alpha = \begin{cases} 2, & \text{if } n/k < 5 \\ 3, & \text{if } 5 \leq n/k \leq 15 \\ 4, & \text{otherwise} \end{cases} \tag{3.1}$$

Motivations for these α values can be found in [Porumbel et al. 2010]. The dense graphs obviously need more colors k such that the average color class sizes become very low ($n/k < 5$, i.e., the classes become very small). In this case, it is better to use 2 parents in order to avoid excessive disruptions when blending the color classes. Inversely, the sparse graphs need fewer colors such that the average class sizes are very high ($n/k > 15$). In this case, we use more parents (4 in our case) in order to increase the probability of selecting and inheriting good classes from different parents. Besides, we choose 3 parents for the graphs between these two extreme situations.

MGPX is summarized in Algorithm 2. It builds the color classes of the o offspring one by one, transmitting as many vertices as possible from the parents at each step (for quality purpose) (lines 8-15). Once a parent has been used for transmitting an entire color class to o, the parent is not considered for $\lfloor \alpha/2 \rfloor$ steps

Algorithm 2 Pseudo-code of the MGPX combination operator

1: **input**: A set P' of α parents randomly chosen from P
2: **output**: An offspring o
3: $\nu \leftarrow 0$ /* Counts the number of colored vertices in o */
4: $\kappa \leftarrow 0$ /* Counts the number of colors used in o */
5: $\omega \leftarrow 0$ /* Counts the iterations */
6: Set forbidden length for each parent: $\tau(P'_i) = 0, i = 1, ..., \alpha$
7: **while** $\nu < n$ **do**
8: $\quad \kappa \leftarrow \kappa + 1$
9: $\quad$ Identify the non-forbidden parents: P'_s=$\{P'_i | \tau(P'_i) \leqslant \omega\}$
10: $\quad$ Find the maximum cardinality class V^j_* ($V^j_* \in P'_j$) from P'_s
11: $\quad \nu \leftarrow \nu + |V^j_*|$
12: $\quad$ **for all** $v \in V^j_*$ **do**
13: $\quad\quad o(v) \leftarrow \kappa$
14: $\quad\quad$ Remove v from α parents P'
15: $\quad$ **end for**
16: $\quad \tau(P'_j) \leftarrow \omega + \lfloor \alpha/2 \rfloor$ /* Forbid parent P'_j for $\lfloor \alpha/2 \rfloor$ steps */
17: $\quad \omega \leftarrow \omega + 1$
18: **end while**
19: **return** o

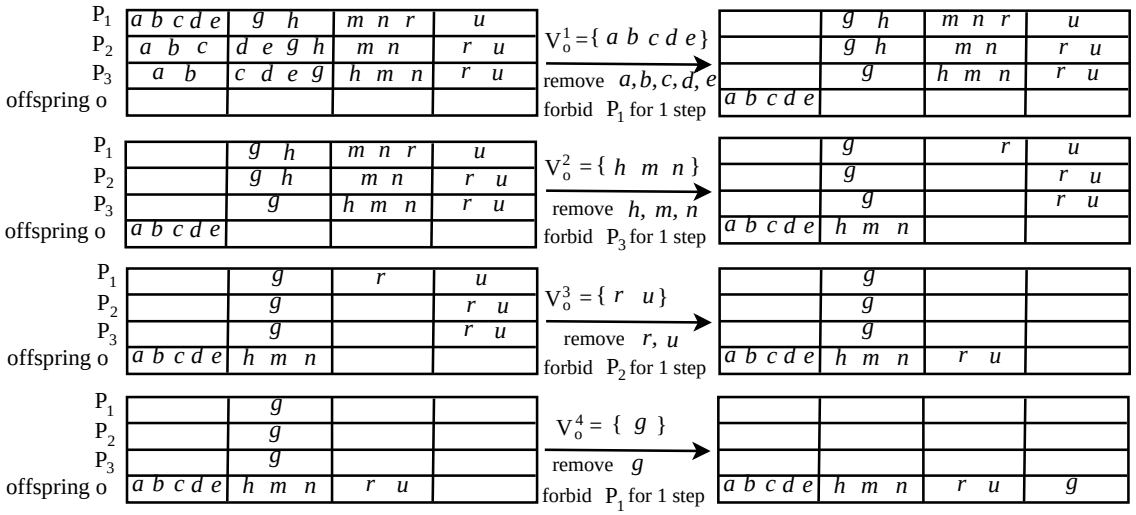


Figure 3.1: An illustrative example of the MGPX crossover

with the purpose of varying the origins of the color classes of o (line 16). This strategy avoids transmitting always from the same parent and introduces some diversity in o [Lü and Hao 2010]. Note that the offspring solution is always a proper k coloring while the number of colors used by the offspring can be higher than those of the considered parents.

Figure 3.1 illustrates the detailed operations of our MGPX crossover. In the example, there are 3 parents ($\alpha = 3$) with $k = 4$ colors, and 11 vertices $a, b, ..., u$. The forbidden length for each parent is initially set to 0 ($\tau(P'_i) = 0$) which means all three parents can be selected at the beginning of the MGPX operator. At the first step, the largest color class $\{a, b, c, d, e\}$ in parent P_1 is chosen to become the first class V^1_o of the offspring o. Then vertices a, b, c, d, e are removed from all three parents. Parent P_1 is forbidden for 1 step ($\lfloor \alpha/2 \rfloor = \lfloor 3/2 \rfloor = 1$). Similarly, we build the color class $V^2_o = \{h, m, n\}$ from parent P_3, $V^3_o = \{r, u\}$ from parent P_2 and $V^4_o = \{g\}$ from parent P_1 respectively. After four steps, all the vertices are assigned such that a complete offspring is constructed. One notices that in this example, the sum of colors in the offspring is better than or equal with its parents.

3.2.4 A double-neighborhood tabu search

Local optimization is another important element within a memetic algorithm. In our case, its role is to improve as far as possible the quality (i.e., the sum of colors) of a given solution returned by the MGPX crossover operator. This is achieved by a Double-Neighborhood Tabu Search (DNTS) procedure specifically designed for the MSCP (see Algorithm 3).

DNTS is based on tabu search [Glover and Laguna 1999] and uses two different and complementary neighborhoods N_1 and N_2 which are applied in a token-ring way [Di Gaspero and Schaerf 2006, Lü et al. 2011] to find good local optima (intensification) (lines 2-14). More precisely, we start our search with one neighborhood (lines 6-9) and when the search ends with its best local optimum, we switch to the other neighborhood to continue the search while using the last local optimum as the starting point (lines 10-13). When this second search terminates, we switch again to the first neighborhood and so on. DNTS continues the exploration of each neighborhood N_i $(i = 1, 2)$ until μ_i $(i = 1, 2)$ consecutive iterations fail to update the best solution found.

This neighborhood-based intensification phase terminates if the best local optimum is not updated for μ_ρ consecutive iterations (line 14). At this point, we enter into a diversification phase by triggering a perturbation to escape from the local optimum (line 15, Section 3.2.4). The DNTS procedure stops when a maximum number of iterations $MaxIters$ is met. We explain below the two neighborhoods, the tabu list management and the perturbation mechanism.

Algorithm 3 Pseudo-code of double-neighborhood tabu search for MSCP

1: **Input**: A k-coloring c of a graph G
2: **Output**: the best improved k-coloring
3: $c_* \leftarrow c$
4: **while** a stop condition is not met **do**
5: **repeat**
6: $c \leftarrow$ TS(N_1, c, μ_1) /* Tabu search with neighborhood N_1, Section 3.2.4 */
7: **if** $f(c) < f(c_*)$ **then**
8: $c^* \leftarrow c$
9: **end if**
10: $c \leftarrow$ TS(N_2, c, μ_2) /* Tabu search with neighborhood N_2, Section 3.2.4 */
11: **if** $f(c) < f(c_*)$ **then**
12: $c_* \leftarrow c$
13: **end if**
14: **until** c_* not improved for μ_ρ consecutive iterations
15: $c \leftarrow$ Perturbation(c_*) /* Search is stagnating, generate a new starting solution by perturbing the best k-coloring found so far, Section 3.2.4 */
16: **end while**

N_1: A neighborhood based on connected components

The first neighborhood N_1 can be described by the operator $Exchange(i, j)$. Given a proper k-coloring $c = \{V_1, \ldots, V_k\}$, operator $Exchange(i, j), (1 \leq i \neq j \leq k)$ swaps some vertices of a color class V_i against some connected vertices of another color class V_j. Formally, let $G_{i,j}(c)$ be the set of all connected components of more than one vertex in the subgraph of G induced by color classes V_i and V_j in a proper k-coloring c. For a k-coloring c, the size of neighborhood N_1 is bounded by $O(\frac{n}{2} \times \frac{k(k-1)}{2})$ (n is the number of vertices). In Figure 3.2 (left) for instance, $G_{i,j}(c)$ is composed of two graphs (say g_1 and g_2): g_1 is the subgraph induced by $\{v_2, v_3, v_6, v_7, v_8\}$ and g_2 is induced by $\{v_4, v_5, v_9\}$.

Neighborhood $N_1(c)$ is composed of the set $\mathcal{G}(c)$ of all possible elements in all the $G_{i,j}(c)$ sets: $\mathcal{G}(c) = \cup_{1 \leq i < j \leq k} G_{i,j}(c)$. In other words, $N_1(c)$ includes all the proper k-colorings that can be obtained from the

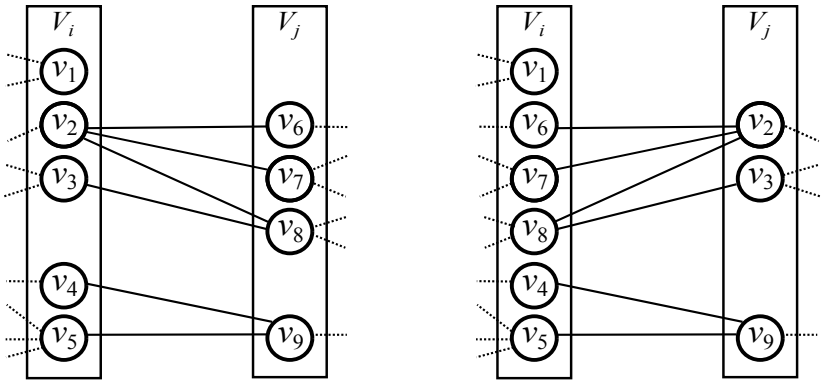


Figure 3.2: N_1: An illustrative example with two partial colorings (c and c' are restricted here to two V_i and V_j color classes)

current k-coloring c by exchanging the vertices of a connected component induced by color classes V_i and V_j. Figure 3.2 shows an example where by exchanging the two sets of vertices $\{v_2, v_3\}$ and $\{v_6, v_7, v_8\}$ of connected component g_1 of the current k-coloring c (left drawing), we obtain a neighboring k-coloring c' (right drawing).

N_2: A neighborhood based on one-vertex-move

The second neighborhood N_2 is conventional and is simpler than N_1. N_2 can be described by the operator $OneMove(v, i, j)$. Given a proper k-coloring $c = \{V_1, \ldots, V_k\}$, operator $OneMove(v, i, j), (1 \leq i \neq j \leq k)$ displaces one single vertex v of a color class V_i to another color class V_j such that the resulting k-coloring remains proper. For instance, from the current coloring c of the left drawing of Figure 3.2, moving vertex v_1 from V_i to V_j gives a neighboring solution. Neighborhood $N_2(c)$ is composed of all the possible proper k-colorings by applying $OneMove(v, i, j)$ to the current k-coloring c. Like neighborhood N_1, the solutions of this second neighborhood are also proper k-colorings. The size of the neighborhood N_2 is bounded by $O(n \times k)$. Moreover, the number of colors of the neighboring solutions remains the same as that of the current coloring.

Neighborhood examination and tabu list

DNTS applies these two neighborhoods N_1 and N_2 in a token-ring way [Di Gaspero and Schaerf 2006, Lü et al. 2011]. The alternation between N_1 and N_2 is triggered when the current neighborhood is exhausted, i.e., when the current best solution cannot be further improved for a fixed number of consecutive iterations.

As shown in Algorithm 3, at each iteration of our DNTS, a best neighboring solution is selected among all the allowed solutions (from N_1 or N_2) to replace the current solution. Precisely, for the neighborhood N_1 defined by the operator $Exchange(i, j)$, we first identify all the connected components in each pair of color classes for the current k-coloring. Theoretically, this step has a worst time complexity of $O(\frac{n^2}{2} \times \frac{k(k-1)}{2})$. But in practice, the time consuming is much lower since this operator is related to the density of the graph. Then we select the best connected components for exchange according to the objective function $f(c)$ (ties are broken at random). When a set of vertices of a color class V_i are exchanged with a set of vertices of another color class V_j, exchanges between V_i and V_j are forbidden for the next TT iterations (called tabu tenure). Finally, we only need to update the connected components in the pairs of color classes which contain class V_i or V_j. For the neighborhood N_2 defined by the $OneMove(v, i, j)$ operator, we go through all legal moves (there are $O(n \times k)$ of them) and select a best move for the $OneMove(v, i, j)$ operation. When

a vertex v of a color class V_i is displaced to another color class V_j, the vertex v is forbidden to go back to V_i for the next TT iterations.

The tabu list is introduced to avoid short-term cycles [Glover and Laguna 1999] and is updated after each iteration. The tabu tenure TT is determined simply by taking a random number from $\{0, \ldots k-1\}$. Moreover, a forbidden $Exchange$ or $OneMove$ operation is always accepted if it leads to a neighboring solution better than the best solution found so far (this is called aspiration according to the tabu search terminology).

The perturbation mechanism

In addition to the basic diversification mechanism of the tabu list, our DNTS algorithm applies a stronger diversification strategy based on perturbations to escape deep local optima. The perturbation is triggered when the current intensification phase cannot update the recorded best solution c_* for μ_ρ consecutive iterations (see line 15, Algorithm 3). In this case, the search is considered to be trapped in a deep local optimum and a strong diversification is needed to bring the search to a new search region. To achieve this, we apply the following perturbation technique to modify the recorded best solution c_* and then use this perturbed solution to initialize DNTS. Suppose c_* is composed of k different color classes and let V_l be the largest color class. We introduce an additional color class V_{k+1} and then move randomly one third of the vertices of V_l into V_{k+1}. In order to prevent the subsequent search from coming back to c_*, V_l and V_{k+1} are classified tabu and cannot take part of an $Exchange$ or a $OneMove$ operation for the next TT iterations (see Section 3.2.4).

3.2.5 Population updating

The management of the population usually controls and balances two important factors in population-based heuristics: Quality and diversity. Quality can naturally be measured here using the coloring sum function (f). The proper k-coloring c_i is better than c_j if $f(c_i) < f(c_j)$. We use the following distance H to estimate the diversity. Given two coloring c_i and c_j, $H_{i,j}$ is the number of vertices in c_i and c_j which have different colors: $H_{i,j} = |\{v \in V : c_i(v) \neq c_j(v)\}|$. A small $H_{i,j}$ value indicates a high similarity between c_i and c_j. H is also employed to measure how much diversity $H_{i,P}$ a particular k-coloring c_i contributes to the entire population P: $H_{i,P} = \min_{j \neq i} H_{i,j}$. Again, a small (large) $H_{i,P}$ value indicates that c_i adds a low (high) diversity to P.

In MASC, f and H are combined in a s "score" function which is used to decide whether an offspring solution o replaces an individual in the population P or not: $s(c_i) = f(c_i) + e^{0.08n/H_{i,P}}\ \forall c_i \in P$. Precisely we first add o into P and compute all $s(c_i)$. We then identify the worse configuration c_w (i.e., $s(c_w)$ is maximum). The replacement strategy applies the following rules:

Case 1 (c_w is not o): Remove c_w from P;

Case 2 (c_w is o): Remove the second worse individual from P with probability 0.2, and discard o otherwise.

Notice that unlike partition based distances [Porumbel et al. 2010], the distance used here does not consider explicitly the symmetry of solutions. We adopted this simpler distance for two practical reasons. First, given that the solutions are all improved by DNTS (tabu search), the population has generally a certain level of diversity. So the diversity control mechanism has a limited role. Second, the computation of a partition distance is much more expensive. The experimental results show that in the context of this work, the above distance seems sufficient for our MASC algorithm.

3.3 Experimental results

Our MASC approach was tested on a benchmark composed of 77 well-known graphs commonly used to report computational results for the MSCP: 39 are part of the COLOR 2002–2004 competitions and the 38 others are known as "DIMACS" instances. Most of these graphs are available on-line from http://mat.gsia.cmu.edu/COLOR04. The main characteristics of each graph appear in Tables 3.2 and 3.5, see columns 1–4 (COLOR 2002–2004 instances are at the top of Table 3.2 and DIMACS instances at the bottom): Name of the graph, order (n), size (m), and the best known sum f_b.

MASC is programmed in C++ and compiled using GNU gcc on a PC with 2.7 GHz CPU and 4 Gb RAM. Like many memetic algorithms, we use a small population of 10 individuals. The values of the other parameters were determined empirically, see Table 3.1. Notice that $MaxGenerations = 50$ is the stop condition that determines the running time of the algorithm. Given its stochastic nature, MASC is run 30 times with different seeds.

Table 3.1: Settings of parameters

Parameter	Section	Description	Value
μ_1	3.2.4	Maximum number of non-improving moves for TS using N_1	500
μ_2	3.2.4	Maximum number of non-improving moves for TS using N_2	1 000
μ_ρ	3.2.4	Maximum number of non-improving moves of TS for perturbation	4 000
$MaxIters$	3.2.4	Maximum iterations of TS procedure	10 000
$MaxGenerations$	3.2.5	Maximum number of generations	50

3.3.1 Computational results

Columns 6–10 in Table 3.2 present detailed computational results of our MASC algorithm: Best result obtained (f_*) with the number of required colors (k_*), success rate (SR, percentage of runs such that the sum of colors f_* of MASC is at least the current best known value f_b, i.e., $f_* \leq f_b$), average coloring sum (Avg.), standard deviation (σ) and average running time to reach f_* (t, in minutes). Column k shows the chromatic number or its best upper bound (i.e., the smallest number of colors for which a k-coloring is ever reported). The reported values are based on 30 independent runs (i.e., with different random seeds).

From Table 3.2, one observes that for the 39 COLOR 2002–2004 instances with known upper bounds (see top part of the table), MASC improves the best known upper bound for two instances (miles500 and homer) and equals the best known results for the other 37 graphs. Furthermore, MASC achieves robust results here since SR $= 30/30$ and $\sigma = 0.0$ for these graphs except two instances (homer and queen9.9). The average running time of MASC ranges from less than one second to about 13 minutes except for the homer instance.

For the set of 24 DIMACS instances (bottom part), the MASC algorithm improves the best known upper bound for 3 graphs (flat300_28_0, le450_15c, and le450_15d) and equals the best known results for 10 instances. Unfortunately, MASC was unable to reach the best known results for the other 11 graphs (see lines where SR $= 0/30$). The average running time is less than 76 minutes except for the DSJC500.5 and flat300_28_0 instances. Finally, we notice that the number of colors needed to ensure the best sum coloring (k_*) can be larger than the chromatic number or its best upper bound (k).

3.3.2 Comparisons with state-of-the-art algorithms

Table 3.3 compares MASC with 5 recent effective algorithms that cover the best known results for the considered benchmark: EXSCOL [Wu and Hao 2012], BLS [Benlic and Hao 2012], MA [Moukrim et al.

Table 3.2: Detailed computational results of MASC on the set of 39 COLOR 2002-2004 instances (upper part) and 24 DIMACS instances (bottom part)

Characteristics of the graphs				MASC					
Name	n	m	f_b	k	$f_*(k_*)$	SR	Avg.	σ	t
myciel3	11	20	21	4	21(4)	30/30	21.0	0.0	0.0
myciel4	23	71	45	5	45(5)	30/30	45.0	0.0	0.0
myciel5	47	236	93	6	93(6)	30/30	93.0	0.0	0.0
myciel6	95	755	189	7	189(7)	30/30	189.0	0.0	0.1
myciel7	191	2 360	381	8	381(8)	30/30	381.0	0.0	1.1
anna	138	493	276	11	276(11)	30/30	276.0	0.0	0.1
david	87	406	237	11	237(11)	30/30	237.0	0.0	0.1
huck	74	301	243	11	243(11)	30/30	243.0	0.0	0.0
jean	80	254	217	10	217(10)	30/30	217.0	0.0	0.0
homer	561	1 628	1 157	13	**1 155(13)**	1/30	1 158.5	1.7	63.9
queen5.5	25	160	75	5	75(5)	30/30	75.0	0.0	0.0
queen6.6	36	290	138	7	138(8)	30/30	138.0	0.0	1.1
queen7.7	49	476	196	7	196(7)	30/30	196.0	0.0	0.0
queen8.8	64	728	291	9	291(9)	30/30	291.0	0.0	12.8
queen9.9	81	1 056	409	10	409(10)	9/30	410.5	1.2	1.2
queen8.12	96	1 368	624	12	624(12)	30/30	624.0	0.0	0.0
games120	120	638	443	9	443(9)	30/30	443.0	0.0	0.5
miles250	128	387	325	8	325(8)	30/30	325.0	0.0	0.4
miles500	128	1 170	$\leq$ 708	20	**705(20)**	30/30	705.0	0.0	1.0
fpsol2.i.1	496	11 654	3 403	65	3 403(65)	30/30	3 403.0	0.0	8.7
fpsol2.i.2	451	8 691	1 668	30	1 668(30)	30/30	1 668.0	0.0	5.7
fpsol2.i.3	425	8 688	1 636	30	1 636(30)	30/30	1 636.0	0.0	7.0
mug88_1	88	146	178	4	178(4)	30/30	178.0	0.0	0.1
mug88_25	88	146	178	4	178(4)	30/30	178.0	0.0	0.2
mug100_1	100	166	202	4	202(4)	30/30	202.0	0.0	0.2
mug100_25	100	166	202	4	202(4)	30/30	202.0	0.0	0.3
2-Insertions_3	37	72	62	4	62(4)	30/30	62.0	0.0	0.0
3-Insertions_3	56	110	92	4	92(4)	30/30	92.0	0.0	0.0
inithx.i.1	864	18 707	3 676	54	3 676(54)	30/30	3 676.0	0.0	7.6
inithx.i.2	645	13 979	2 050	31	2 050(31)	30/30	2 050.0	0.0	4.4
inithx.i.3	621	13 969	1 986	31	1 986(31)	30/30	1 986.0	0.0	1.8
mulsol.i.1	197	3 925	1 957	49	1 957(49)	30/30	1 957.0	0.0	0.1
mulsol.i.2	188	3 885	1 191	31	1 191(31)	30/30	1 191.0	0.0	0.2
mulsol.i.3	184	3 916	1 187	31	1 187(31)	30/30	1 187.0	0.0	0.2
mulsol.i.4	185	3 946	1 189	31	1 189(31)	30/30	1 189.0	0.0	0.2
mulsol.i.5	186	3 973	1 160	31	1 160(31)	30/30	1 160.0	0.0	0.2
zeroin.i.1	211	4 100	1 822	49	1 822(49)	30/30	1 822.0	0.0	0.2
zeroin.i.2	211	3 541	1 004	30	1 004(30)	30/30	1 004.0	0.0	0.1
zeroin.i.3	206	3 540	998	30	998(30)	30/30	998.0	0.0	0.1
DSJC125.1	125	736	326	5	326(7)	20/30	326.6	0.9	4.4
DSJC125.5	125	3 891	1 012	17	1 012(18)	2/30	1 020.0	3.9	3.5
DSJC125.9	125	6 961	2 503	44	2 503(44)	12/30	2 508.0	5.6	1.9
DSJC250.1	250	3 218	973	8	974(9)	0/30	990.5	8.3	17.3
DSJC250.5	250	15 668	3 214	28	3 230(31)	0/30	3 253.7	14.3	23.1
DSJC250.9	250	27 897	8 277	72	8 280(74)	0/30	8 322.7	22.3	5.6
DSJC500.1	500	12 458	2850	12	2 940(14)	0/30	3 013.4	28.3	50.4
DSJC500.5	500	62 624	10 910	48	11 101(53)	0/30	11 303.5	73.9	202.5
DSJC500.9	500	112 437	29 912	126	29 994(126)	0/30	30059.1	31.6	90.9
flat300_20_0	300	21 375	3 150	20	3 150(20)	30/30	3 150.0	0.0	0.0
flat300_26_0	300	21 633	3 966	26	3 966(26)	30/30	3 966.0	0.0	0.8
flat300_28_0	300	21 695	$\leq$ 4 261	28	**4 238(30)**	1/30	4 313.4	22.3	309.7
le450_5a	450	5 714	1 350	5	1 350(5)	30/30	1 350.0	0.0	0.7
le450_5b	450	5 734	1 350	5	1 350(5)	30/30	1 350.0	0.0	0.4
le450_5c	450	9 803	1 350	5	1 350(5)	30/30	1 350.0	0.0	0.2
le450_5d	450	9 757	1 350	5	1 350(5)	30/30	1 350.0	0.0	0.5
le450_15a	450	8 168	2 632	15	2 706(19)	0/30	2 742.6	13.8	41.3
le450_15b	450	8 169	2 642	15	2 724(19)	0/30	2 756.2	14.8	40.3
le450_15c	450	16 680	$\leq$ 3 866	15	**3 491(16)**	30/30	3 491.0	0.0	45.3
le450_15d	450	16 750	$\leq$ 3 921	15	**3 506(17)**	30/30	3 511.8	3.6	59.8
le450_25a	450	8 260	3 153	25	3 166(27)	0/30	3 176.8	4.4	39.2
le450_25b	450	8 263	3 366	25	3 366(26)	1/30	3 375.1	3.4	40.3
le450_25c	450	17 343	4 515	25	4 700(31)	0/30	4 773.3	25.2	75.3
le450_25d	450	17 425	4 544	25	4 722(29)	0/30	4 805.7	27.4	63.4

2013], MDS5 [Helmar and Chiarandini 2011], and MRLF [Li et al. 2009]. No averaged value appears in the table for MA, MDS5 and MRLF since this information is not given in [Helmar and Chiarandini 2011, Li et al. 2009, Moukrim et al. 2013]. Furthermore, "–" marks signal that some instances were not tested by some approaches.

Since most reference algorithms give only results for a (small) subset of the considered benchmark, it is difficult to analyze the performance of these algorithms by statistical tests. Hence, we compare the performance between MASC and these reference algorithms one by one and summarize the comparisons in Table 3.4. The first column of Table 3.4 indicates the name of the reference heuristics, followed by the number $\#G$ of graphs tested by each algorithm and shown in Table 3.3. The last three columns give the number of times MASC reports a better, equal, or worse result compared to each reference algorithm.

From Table 3.4, it can be observed that MASC obtains absolutely no worse results than MDS5 and MRLF (see the last three lines). Furthermore, MASC gets better results than these algorithms for 9 and 16 instances respectively. Our algorithm is also quite competitive with EXSCOL, BLS and MA which are the most recent and effective methods since it obtains better or equivalent results for 28, 22 and 49 graphs respectively. MASC reaches worse results than EXSCOL, BLS and MA only for 8, 3 and 8 graphs respectively.

3.3.3 Experiments on large graphs

We turn now our attention to the performance of our MASC algorithm to color large graphs with at least 500 vertices. These large graphs are known to be quite difficult for almost all the existing sum coloring approaches except EXSCOL which dominates the other heuristics particularly on large graphs. We show a new experiment with MASC applied to color 17 large graphs. In this experiment, we run MASC 10 times on each graph under exactly the same condition as in Section 3.3.1. The only difference is that we use the solution of EXSCOL [1] as one of MASC's 10 initial solutions while the 9 other initial solutions are generated according to the procedure described in Section 3.2.2. With this experiment, we aimed to investigate two interesting questions. Is it possible for MASC to improve the results of the powerful EXSCOL algorithm? Does the initial population influence the performance of MASC? The computational outcomes of this experiment are provided in Table 3.5.

In Table 3.5, column 4 presents the best known result (f_b) in the literature, columns 5–6 present the best result (f_*) and the average coloring sum (Avg.) of EXSCOL and columns 7–11 present detailed computational results of our MASC algorithm: Best result obtained (f_*) with the number of required colors (k_*), average coloring sum (Avg.), standard deviation (σ), and average running time to reach f_* (t, in minutes). One notices that the values of columns 4 and 5 (f_b and f_*) are identical for EXSCOL except for the *qg.order60* instance.

Table 3.5 shows that with the help of its search mechanism, our MASC algorithm is able to further improve the best known results of 10 instances (entries in bold). This is remarkable given that very few existing approaches can even equal the previous best known results. Moreover, if we contrast the results of the three DSJC500.d graphs ($d = 1, 5, 9$) reported in Tables 3.2 and 3.5, it is clear that the initial population impacts directly MASC's outcomes. This indicates that the performance of MASC could be further improved by using a more powerful coloring algorithm to generate the initial solutions of its population.

1. Available at http://www.info.univ-angers.fr/pub/hao/exscol.html

Table 3.3: Comparisons of MASC with five state-of-the-art sum coloring algorithms

Graph		EXSCOL		BLS		MA	MDS5	MRLF	MASC	
Name	f_b	f_*	Avg.	f_*	Avg.	f_*	f_*	f_*	f_*	Avg.
myciel3	21	21	21.0	21	21.0	21	21	21	21	21.0
myciel4	45	45	45.0	45	45.0	45	45	45	45	45.0
myciel5	93	93	93.0	93	93.0	93	93	93	93	93.0
myciel6	189	189	189.0	189	196.6	189	189	189	189	189.0
myciel7	381	381	381.0	381	393.8	381	381	381	381	381.0
anna	276	283	283.2	276	276.0	276	276	277	276	276.0
david	237	237	238.1	237	237.0	237	237	241	237	237.0
huck	243	243	243.8	243	243.0	243	243	244	243	243.0
jean	217	217	217.3	217	217.0	217	217	217	217	217.0
homer	1 157	–	–	–	–	1 157	–	–	**1 155**	1 158.5
queen5.5	75	75	75.0	75	75.0	75	75	75	75	75.0
queen6.6	138	150	150.0	138	138.0	138	138	138	138	138.0
queen7.7	196	196	196.0	196	196.0	196	196	196	196	196.0
queen8.8	291	291	291.0	291	291.0	291	291	303	291	291.0
queen9.9	409	–	–	–	–	409	–	–	409	410.5
queen8.12	624	–	–	–	–	624	–	–	624	624.0
games120	443	443	447.9	443	443.0	443	443	446	443	443.0
miles250	325	328	333.0	327	328.8	325	325	334	325	325.0
miles500	$\leq$ 708	709	714.5	710	713.3	708	712	715	**705**	705.0
fpsol2.i.1	3 403	–	–	–	–	3 403	3 403	–	3 403	3 403.0
fpsol2.i.2	1 668	–	–	–	–	1 668	–	–	1 668	1 668.0
fpsol2.i.3	1 636	–	–	–	–	1 636	–	–	1 636	1 636.0
mug88_1	178	–	–	–	–	–	178	–	178	178.0
mug88_25	178	–	–	–	–	–	178	–	178	178.0
mug100_1	202	–	–	–	–	–	202	–	202	202.0
mug100_25	202	–	–	–	–	–	202	–	202	202.0
2-Insertions_3	62	–	–	–	–	–	62	–	62	62.0
3-Insertions_3	92	–	–	–	–	–	92	–	92	92.0
inithx.i.1	3 676	–	–	–	–	3 676	–	–	3 676	3 676.0
inithx.i.2	2 050	–	–	–	–	2 050	–	–	2 050	2 050.0
inithx.i.3	1 986	–	–	–	–	1 986	–	–	1 986	1 986.0
mulsol.i.1	1 957	–	–	–	–	1 957	–	–	1 957	1 957.0
mulsol.i.2	1 191	–	–	–	–	1 191	–	–	1 191	1 191.0
mulsol.i.3	1 187	–	–	–	–	1 187	–	–	1 187	1 187.0
mulsol.i.4	1 189	–	–	–	–	1 189	–	–	1 189	1 189.0
mulsol.i.5	1 160	–	–	–	–	1 160	–	–	1 160	1 160.0
zeroin.i.1	1 822	–	–	–	–	1 822	–	–	1 822	1 822.0
zeroin.i.2	1 004	–	–	–	–	1 004	1 004	–	1 004	1 004.0
zeroin.i.3	998	–	–	–	–	998	998	–	998	998.0
DSJC125.1	326	326	326.7	326	326.9	326	326	352	326	326.6
DSJC125.5	1 012	1 017	1 019.7	1 012	1 012.9	1 013	1 015	1 141	1 012	1 020.0
DSJC125.9	2 503	2 512	2 512.0	2 503	2 503.0	2 503	2 511	2 653	2 503	2 508.0
DSJC250.1	973	985	985.0	973	982.5	983	977	1 068	974	990.5
DSJC250.5	3 214	3 246	3 253.9	3 219	3 248.5	3 214	3 281	3 658	3 230	3 253.7
DSJC250.9	8 277	8 286	8 288.8	8 290	8 316.0	8 277	8 412	8 942	8 280	8 322.7
DSJC500.1	2 850	2 850	2 857.4	2 882	2 942.9	2 897	2 951	3 229	2 940	3 013.4
DSJC500.5	10 910	10 910	10 918.2	11 187	11 326.3	11 082	11 717	12 717	11 101	11 303.5
DSJC500.9	29 912	29 912	29 936.2	30 097	30 259.2	29 995	30 872	32 703	29 994	30 059.1
flat300_20_0	3 150	3 150	3 150.0	–	–	3 150	–	–	3 150	3 150.0
flat300_26_0	3 966	3 966	3 966.0	–	–	3 966	–	–	3 966	3 966.0
flat300_28_0	$\leq$ 4 261	4 282	4 286.1	–	–	4 261	–	–	**4 238**	4 313.4
le450_5a	1 350	–	–	–	–	1 350	–	–	1 350	1 350.0
le450_5b	1 350	–	–	–	–	1 350	–	–	1 350	1 350.0
le450_5c	1 350	–	–	–	–	1 350	–	–	1 350	1 350.0
le450_5d	1 350	–	–	–	–	1 350	–	–	1 350	1 350.0
le450_15a	2 632	2 632	2 641.9	–	–	2 681	–	–	2 706	2 742.6
le450_15b	2 642	2 642	2 643.4	–	–	2 690	–	–	2 724	2 756.2
le450_15c	$\leq$ 3 866	3 866	3 868.9	–	–	3 943	–	–	**3 491**	3 491.0
le450_15d	$\leq$ 3 921	3 921	3 928.5	–	–	3 926	–	–	**3 506**	3 511.8
le450_25a	3 153	3 153	3 159.4	–	–	3 178	–	–	3 166	3 176.8
le450_25b	3 366	3 366	3 371.9	–	–	3 379	–	–	3 366	3 375.1
le450_25c	4 515	4 515	4 525.4	–	–	4 648	–	–	4 700	4 773.3
le450_25d	4 544	4 544	4 550.0	–	–	4 696	–	–	4 722	4 805.7

Table 3.4: MASC vs. five state-of-the-art sum coloring algorithms

Competitor	$\#G$	Best results of MASC (f_*)		
		Better	Equal	Worse
EXSCOL [Wu and Hao 2012]	36	12	16	8
BLS [Benlic and Hao 2012]	25	5	17	3
MA [Moukrim et al. 2013]	57	10	39	8
MDS5 [Helmar and Chiarandini 2011]	34	9	25	0
MRLF [Li et al. 2009]	25	16	9	0

Table 3.5: Results of MASC on 17 large graphs with at least 500 vertices

Characteristics of the graphs				EXSCOL		MASC				
Name	n	m	f_b	f_*	Avg.	k	$f_*(k_*)$	Avg.	σ	t
DSJC500.1	500	12 458	2 850	2 850	2 857.4	12	**2 841(14)**	2 844.1	3.2	28.9
DSJC500.5	500	62 624	10 910	10 910	10 918.2	48	**10 897(51)**	10 905.8	4.6	73.3
DSJC500.9	500	112 437	29 912	29 912	29 936.2	126	**29 896(131)**	29 907.8	5.8	59.0
DSJC1000.1	1 000	49 629	9 003	9 003	9 017.9	20	**8 995(22)**	9 000.5	3.0	70.7
DSJC1000.5	1 000	249 826	37 598	37 598	37 673.8	83	**37 594(87)**	37 597.6	1.2	200.4
DSJC1000.9	1 000	449 449	103 464	103 464	103 531.0	223	103 464(231)	103 464.0	0.0	125.9
flat1000_50_0	1 000	245 000	25 500	25 500	25 500.0	50	25 500(50)	25 500.0	0.0	0.1
flat1000_60_0	1 000	245 830	30 100	30 100	30 100.0	60	30 100(60)	30 100.0	0.0	114.6
flat1000_76_0	1 000	246 708	37 167	37 167	37 213.2	82	37 167 (85)	37 167.0	0.0	1.1
latin_sqr_10	900	307 350	42 223	42 223	42 392.7	98	**41 444(100)**	41 481.5	19.1	101.2
wap05	905	43 081	13 680	13 680	13 718.4	50	**13 669(51)**	13 677.8	3.7	3.3
wap06	947	43 571	13 778	13 778	13 830.9	46	**13 776(48)**	13 777.8	0.6	4.1
wap07	1 809	103 368	28 629	28 629	28 663.8	46	**28 617(50)**	28 624.7	3.8	12.4
wap08	1 870	104 176	28 896	28 896	28 946.0	45	**28 885(50)**	28 890.9	3.2	15.1
qg.order30	900	26 100	13 950	13 950	13 950.0	30	13 950(30)	13 950.0	0.0	3.8
qg.order40	1 600	62 400	32 800	32 800	32 800.0	40	32 800(40)	32 800.0	0.0	11.8
qg.order60	3 600	212 400	109 800	110 925	110 993.0	60	109 800(60)	109 800.0	0.0	290.6

3.4 Analysis of MASC

In this section, we investigate the influence of three important ingredients of the proposed memetic algorithm, i.e., the multi-parent crossover operator, the combined neighborhood and the improvement of MASC over the initial population. Experiments were based on 16 selected graphs of different types, for which some reference algorithms cannot achieve the best known results. Hence, these selected instances can be considered to be difficult and representative.

3.4.1 Influence of the multi-parent crossover operator

For our memetic algorithm, it is relevant to evaluate the effectiveness of its crossover operator. To verify this, we carry out experiments on the 16 selected graphs and run both MASC (using the MGPX crossover) and DNTS (without MGPX) for 30 times (with the same parameter μ_1, μ_2, and μ_ρ settings as defined in Table 3.1). The DNTS (without MGPX) starts with a single solution which is generated for MASC. DNTS stops after a maximum number of 5×10^5 iterations in order to make sure that MASC and DNTS are given the same search effort. The results are given in Table 3.6.

From Table 3.6, one notices that DNTS equals and improves respectively 5 and 3 best known results while MASC equals and improves respectively 5 and 11 best known results. Furthermore, the last column t-test indicates whether the observed difference between MASC and DNTS is statistically significant when a 95% confidence t-test is performed in terms of the best result obtained (f_*). If MASC and DNTS achieve always the same results, t-test column is marked by '-'. The t-test indicates that MASC is statistically better than DNTS for 12 out of 16 cases except for the instances where DNTS can achieve the best known results (f_b). These comparative results provide clear evidences that the MGPX crossover operator plays an important role in the MASC algorithm.

Table 3.6: Comparative results of MASC and DNTS

Graph		MASC		DNTS		*t-test*
Name	f_b	f_*	Avg.	f_*	Avg.	
anna	276	*276*	276.0	*276*	276.0	-
queen6.6	138	*138*	138.0	*138*	138.0	-
miles250	325	*325*	325.0	*325*	325.0	-
miles500	$\leq$ 709	**705**	705.0	**705**	705.6	Y
DSJC125.1	326	*326*	326.6	*326*	328.6	Y
DSJC125.5	1 012	**1 012**	1 020.0	1 016	1 029.8	Y
DSJC125.9	2 503	**2 503**	2 508.0	2 506	2 530.1	Y
DSJC250.1	973	**974**	990.5	981	997.7	Y
DSJC250.5	3 219	**3 230**	3 253.7	3 234	3 301.7	Y
DSJC250.9	$\leq$ 8 286	**8 280**	8 322.7	8 321	8 381.9	Y
flat300_26_0	3 966	*3 966*	3 966.0	*3 966*	3 966.0	-
flat300_28_0	$\leq$ 4 282	**4 238**	4 313.4	4 303	4 406.3	Y
le450_15c	$\leq$ 3 866	**3 491**	3 491.0	**3 491**	3 492.1	Y
le450_15d	$\leq$ 3 921	**3 506**	3 511.8	**3 506**	3 515.0	Y
le450_25c	4 515	**4 700**	4 773.3	4 749	4 803.9	Y
le450_25d	4 544	**4 722**	4 805.7	4 784	4 835.3	Y

3.4.2 Influence of the neighborhood combination

The neighborhood is an important element that influences the local search procedure. Our proposed algorithm relies on two different neighborhoods: N_1 (neighborhood based on connected components) and N_2 (neighborhood based on the one-vertex-move) which are explored in a token-ring way (see Section 3.2.4). In this section, we investigate the interest of this combined use of the two neighborhoods. For this purpose, we carried out experiments on the 16 selected graphs to compare the original Double-Neighborhood Tabu Search (DNTS) with two variants which uses only one neighborhood N_1 or N_2. We use below TS_{N1} and TS_{N2} to denote these two variants. These three TS procedures (DNTS, TS_{N1} and TS_{N2}) are run under the same stop condition, i.e. limited to 5×10^5 iterations.

We run 30 times these TS procedures to solve each of the 16 selected graphs and report the computational outcomes (the best and average results) in Table 3.7. One easily observes that DNTS obtains better or equal results compared to TS_{N1} and TS_{N2} for all the instances in terms of the best known result (f_*) and the average result (Avg.). The t-test $t - test_{N_i}$ $(i = 1, 2)$ in the last two columns confirms that with a 95% confidence level DNTS is slightly or significantly better than TS_{N1} and TS_{N2}. This experiment demonstrates thus the advantage of the token-ring combination of the two neighborhoods compared to each individual neighborhood.

Table 3.7: Comparative results of the tabu search improvement method according to the neighborhood employed

Graph		DNTS		TS_{N2}		TS_{N1}		$t - test_{N_2}$	$t - test_{N_1}$
Name	f_b	f_*	Avg.	f_*	Avg.	f_*	Avg.		
anna	276	276	276.0	282	285.8	276	276.0	Y	-
queen6.6	138	138	138.0	138	138.4	138	138.0	Y	-
miles250	325	**325**	325.0	346	361.6	335	340.7	Y	Y
miles500	$\leq$ 709	**705**	705.6	722	736.0	719	730.9	Y	Y
DSJC125.1	326	**326**	328.6	334	340.8	329	334.0	Y	Y
DSJC125.5	1 012	**1016**	1029.8	1 031	1 045.1	1 020	1 031.8	Y	N
DSJC125.9	2 503	**2 506**	2 530.1	2 514	2 557.6	2 512	2 538.3	Y	N
DSJC250.1	973	**981**	997.7	1 004	1 021.3	1 022	1 039.9	Y	Y
DSJC250.5	3 219	**3 234**	3 301.7	3 271	3 323.9	3 260	3 306.5	Y	N
DSJC250.9	$\leq$ 8 286	**8 321**	8 381.9	8 347	8 405.6	8 318	8 387.5	Y	N
flat300_26_0	3 966	3 966	3 966.0	3 966	3 966.0	3 966	3 966.0	-	-
flat300_28_0	$\leq$ 4 282	**4 303**	4 406.3	4 347	4 427.8	4 332	4 435.5	N	Y
le450_15c	$\leq$ 3 866	**3 491**	3 492.5	3 503	3 517.2	3 508	3 551.8	Y	Y
le450_15d	$\leq$ 3 921	**3 506**	3 515.0	3 528	3 538.5	3 526	3 568.2	Y	Y
le450_25c	4 515	**4 749**	4 803.9	4 828	4 893.9	5 005	5 067.4	Y	Y
le450_25d	4 544	**4 784**	4 835.3	4 848	4 907.0	5 035	5 119.1	Y	Y

Table 3.8: Comparative results of MASC and TABUCOL

Graph		MASC		TABUCOL		*t-test*
Name	f_b	f_*	Avg.	f_*	Avg.	
anna	276	276	276.0	371	374.4	Y
queen6.6	138	138	138.0	141	141.0	-
miles250	325	325	325.0	429	436.6	Y
miles500	$\leq$ 709	705	705.0	1 040	1 066.0	Y
DSJC125.1	326	326	327.0	337	348.2	Y
DSJC125.5	1 012	1 012	1 021.5	1 028	1 047.0	Y
DSJC125.9	2 503	2 503	2 509.6	2 523	2 563.9	Y
DSJC250.1	973	985	993.8	1 022	1 059.1	Y
DSJC250.5	3 219	3 230	3 263.2	3 279	3 312.4	Y
DSJC250.9	$\leq$ 8 286	8 290	8 319.5	8 338	8 373.8	Y
flat300_26_0	3 966	3 966	3 966.0	*3 966*	3 966.0	-
flat300_28_0	$\leq$ 4 282	4 238	4 313.4	4 363	4 430.4	Y
le450_15c	$\leq$ 3 866	3 491	3 491.0	3 532	3 584.7	Y
le450_15d	$\leq$ 3 921	3 506	3 512.6	3 567	3 591.8	Y
le450_25c	4 515	4 743	4 798.3	5 384	5 448.6	Y
le450_25d	4 544	4 750	4 833.4	5 226	5 464.4	Y

3.4.3 Improvements of MASC over TABUCOL

Recall that the initial population is generated by the well-known graph coloring procedure TABUCOL. It is interesting to know to which extent our MASC procedure (which is specially designed for the Minimum Sum Coloring Problem) can improve the quality of solutions generated by TABUCOL in terms of sum of colors. For this purpose, we re-run 30 times our MASC procedure on the set of 16 selected graphs. Like for the previous experiments, we report in Table 3.8 the best and average objective value f_* both for TABUCOL (initial population) and MASC (final population). Given the stochastic nature of TABUCOL and MASC, some results reported in this experiment may be slightly different from those reported in Table 3.2.

From Table 3.8, one easily observes that MASC improves significantly the initial results generated by TABUCOL. Indeed, the best and average sums of colors achieved by MASC are systematically smaller (better) than those of TABUCOL for all the graphs except in one case (flat300_26_0) for which TABUCOL alone achieves already the best known result. Furthermore, the last column confirms with a 95% confidence level the significance of the improvements of MASC over the solutions provided by TABUCOL.

3.5 Conclusion

This chapter deals with the minimum sum coloring problem (MSCP), which is an important generalization of the classic vertex coloring problem (VCP). To approximate the MSCP, we proposed a memetic algorithm (MASC) which employs an effective tabu search procedure with a combination of two neighborhoods, a multi-parent crossover operator and a population updating mechanism to balance intensification and diversification.

We assessed the performance of MASC on 77 frequently used graphs from the DIMACS and COLOR 2002-2004 competitions. MASC can improve 15 best known upper bounds including 10 large and very hard graphs with at least 500 vertices while equaling 54 previous best results. Compared with five recent and effective algorithms which cover the best known results for the tested instances, our MASC algorithm remains quite competitive.

Furthermore, we investigated two important components of the proposed algorithm. The experiments demonstrate the relevance of the multi-parent crossover operator and the combined neighborhood for the overall performance of MASC. Finally, we showed the proposed MASC approach significantly improves the classical tabu search graph coloring approach TABUCOL for the minimum sum coloring problem.

4

HSA: Hybrid Search Algorithm for minimum sum coloring

In this chapter, we further study the minimum sum coloring problem (MSCP). We are interested in the computation of both lower and upper bounds of the MSCP and introduce an effective hybrid search algorithm (HSA) which improves on upper bounds of our previous MASC algorithm and also provides lower bounds. The proposed algorithm relies on a joint use of two dedicated crossover operators (to generate offspring solutions) and an iterated double-phase tabu search procedure (to improve offspring solutions). A distance-and-quality updating rule is used to maintain a healthy diversity in the population. We show extensive experimental results to demonstrate the effectiveness of the proposed algorithm and provide a first landscape analysis of the MSCP. Besides, the differences with our previous MASC algorithm are discussed in Section 4.2.6. This work is detailed in [Jin and Hao 2015c].

Contents

4.1 Introduction

In this chapter, we further study the minimum sum coloring problem which is formally introduced in Chapter 1 and we are interested in the computation of both lower and upper bounds of the MSCP. In order to approximate these two optimization problems, we propose an effective hybrid search algorithm (HSA) and summarize the main contributions of this work as follows.

- From the algorithm perspective, the HSA approach integrates several special features to ensure a high search efficiency. These include an original recombination mechanism to generate offspring solutions and an iterated double-phase tabu search procedure to ensure local optimization. The solution recombination mechanism combines a diversification-guided crossover operator and a grouping-guided crossover operator to create diversified and promising offspring solutions. The double-phase tabu search procedure is designed to handle both feasible and unfeasible solutions. A dedicated perturbation mechanism is also introduced to escape local optima. Finally, a population updating procedure is employed to maintain a healthy diversity and high-quality population.

- From the computational perspective, we evaluate the HSA approach on 94 well-known DIMACS and COLOR 2002-2004 benchmark instances. The computational results show that our HSA algorithm can achieve the best-known results for most of these benchmark instances established by several best performing algorithms. Moreover, HSA finds 51 improved best solutions (24 improved upper bounds and 27 improved lower bounds).

The rest of this chapter is organized as follows. Section 4.2 presents the proposed algorithm for computing upper bounds of the MSCP. Section 4.3 explains the adjustments of the proposed algorithm to compute lower bounds. Section 4.4 shows extensive computational results of HSA and comparisons with the state-of-the-art algorithms. Before concluding, Section 4.5 investigates and analyzes some key issues of the proposed algorithm.

4.2 Components of the HSA approach

The proposed hybrid search algorithm for the MSCP follows the general memetic framework which combines population-based evolutionary search and local optimization [Moscato and Cotta 2003, Neri et al. 2012]. Our HSA algorithm repeatedly alternates between the double-crossover procedure that generates new offspring solutions (Section 4.2.3) and the iterated double-phase tabu search procedure (IDTS) that optimizes the newly generated offspring solutions (Section 4.2.4). As soon as an offspring solution is improved by IDTS, the population is accordingly updated based on the solution quality and population diversity (Section 4.2.5).

The general scheme of our HSA algorithm for the MSCP is summarized in Algorithm 4. HSA starts with an initial population of solutions (line 3, see Sect. 4.2.2) and then repeats a number of generations until a stop condition is met (lines 5–15, in our case, a time limit is used as stop condition). At each generation, two solutions from the population are selected at random to serve as parent solutions (line 6). Then, the double-crossover recombination procedure is employed to create two offspring solutions (line 7) which are further improved by the iterated double-phase tabu search procedure (IDTS) (lines 9). Subsequently, the population updating rule decides whether the improved solution should be inserted into the population and which existing solution is to be replaced (lines 13). In the following subsections, we describe these basic components.

Algorithm 4 An overview of the HSA algorithm for the MSCP

Input: A graph G, population size p
Output: The best sum coloring c_* found and its sum of colors f_*
Population_Initialization(P, p) /* Generate p initial solutions, Sect. 4.2.2 */
$f_* \leftarrow \min_{c \in P} f(c)$ /* f_* records the best objective value found so far */
repeat
  $(P_1, P_2) \leftarrow$ Selection(P) /* Select at random parents for crossover */
  $(o_1, o_2) \leftarrow$ Double-Crossover(P_1, P_2) /* Use two crossover operators to generate two offspring o_1 and o_2, Sect. 4.2.3 */
  for each $i \in \{1, 2\}$ **do**
    $o \leftarrow$ IDTS(o_i) /* Improve o_i with the IDTS procedure, Sect.4.2.4*/
    if $f(o)$ is better than f_* **then**
      $f_* \leftarrow f(o)$; $c_* \leftarrow o$
    end if
    Population_Updating(P, o) /* Use offspring o to update the population, Sect. 4.2.5 */
  end for
until Stop condition is met
return f_*, c_*

4.2.1 Search space and evaluation function

A proper k-coloring satisfies the coloring constraint such that any two adjacent vertices $\{u, v\} \in E$ belong to two different color classes. A k-coloring is improper if the coloring constraint is violated. The search space of our HSA algorithm contains the set Ω of all possible partitions of V into k color classes including both the proper and improper k-colorings. Given a proper coloring, its objective value is given by the f function presented in Eq. (1.1). For two proper k-coloring solutions $c_1 \in \Omega$ and $c_2 \in \Omega$, c_1 is better than c_2 if and only if $f(c_1) < f(c_2)$. We discuss the evaluation of improper colorings in Section 4.2.4.

4.2.2 Initial population

The initial population of our HSA algorithm is composed of p proper k-colorings. To create an individual, we use the maximum independent set algorithm SBTS [Jin and Hao 2015b] to generate in a step-by-step way k (k is not fixed) mutually disjoint independent sets (The SBTS algorithm is described in the Appendix of this thesis). At each step, we apply SBTS to extract a maximal independent set V_i from the graph G and then remove from G the vertices of V_i and their incident edges. This procedure is repeated until the graph becomes empty. The resulting independent sets $\{V_1, \ldots, V_k\}$ form a proper k-coloring.

Since SBTS is a stochastic local search algorithm, each SBTS run generally leads to a different k-coloring. Each new k-coloring is inserted into the population P if it does not duplicate any existing individual of the population. Otherwise, this k-coloring is discarded and another new individual is generated. This process is repeated until the population is filled up with p individuals (i.e., proper k-colorings). These individuals are generally of good quality and serve as the inputs for the double-crossover recombination procedure.

4.2.3 A double-crossover recombination procedure

Recombination is an important ingredient for population-based memetic approaches. In HSA, we propose a double-crossover recombination procedure which jointly uses two different operators to generate suitable offspring solutions: The diversification-guided crossover operator (DGX) and the grouping-guided

crossover operator (GGX). At each generation of the HSA algorithm, HSA first randomly chooses two parents from the population which have not been selected to serve as parents in the previous generation, and then employs DGX and GGX to generate two offspring solutions respectively. Each offspring solution is finally submitted to the iterated double-phase tabu search procedure to improve its quality (minimizing its sum of colors).

Diversification-guided crossover

The diversification-guided crossover (DGX) aims to generate offspring solutions whose quality and diversity are both reasonably respected. Given two parent solutions (i.e., two proper k-colorings) $P_1 = \{V_1^1, \ldots, V_{k_1}^1\}$ ($|V_1^1| \geq \ldots \geq |V_{k_1}^1|$) and $P_2 = \{V_1^2, \ldots, V_{k_2}^2\}$ ($|V_1^2| \geq \ldots \geq |V_{k_2}^2|$). The offspring solution $o = \{V_1^o, \ldots, V_{k_o}^o\}$ is constructed as follows.

Step 1: Let $k_{min} = min\{k_1, k_2\}$ and $k_{max} = max\{k_1, k_2\}$. We first transmit the vertices that share the same colors in both parents such that they keep their colors in the offspring. Formally, we set $V_i^o = V_i^1 \cap V_i^2, i = 1, \ldots, k_{min}$.

Step 2: Let $U = V \setminus \bigcup_{i=1}^{k_{min}} V_i^o$ be the set of unassigned vertices in o. We pick randomly m (see below) vertices from U to form U_m. Then for each vertex $v \in U_m$, v conserves its color of parent P_1 in o, i.e., if $v \in V_i^1$ ($i = 1, \ldots, k_1$), v is added into V_i^o.

Step 3: Finally, for each remaining vertex u of $U \setminus U_m$, u conserves its color from parent P_2 in o, i.e., if $u \in V_j^2 (j = 1, \ldots, k_2)$, u is added into V_j^o.

The DGX operator uses m to control the relative importance of P_1 and P_2 with respect to the generated offspring solution o. In order to avoid a dominance of one parent over the other parent, we set $m = \lfloor \frac{1}{3}|U| \rfloor$ + $rand(\lfloor \frac{1}{3}|U| \rfloor)$ where $|U|$ is the cardinality of U and $rand(\mathcal{N})$ gives a random number in $\{1, \ldots, \mathcal{N}\}$. This value for m ensures that o is separated from either parent by at least $\lfloor \frac{1}{3}|U| \rfloor$ vertices.

The above steps have a time complexity of $O(n)$. Since we need to maintain the order $|V_1| \geq \ldots \geq |V_k|$ of the color classes of the offspring, the total complexity of each crossover operation is $O(n \times k)$.

Fig. 4.1 shows an illustrative example with 3 color classes and 10 vertices represented by $A, B, \ldots, J$. At step 1, the unique common vertex {A} of both parents is directly transmitted to the offspring solution. Then, the remaining vertices $\{B, C, D, E, F, G, H, I, J\}$ are collected in U and m is assumed to be 4. At step 2, we randomly choose $m = 4$ vertices from U (say $\{B, E, H, I\}$) and preserve them from parent P_1 into the offspring. Finally, the remaining unassigned vertices (i.e., $\{C, D, F, G, J\}$) are preserved from parent P_2 to complete the offspring solution.

One observes that the offspring solution generated by the DGX operator may be an improper k-coloring. If this happens, it is repaired by the iterated double-phase tabu search procedure described in Section 4.2.4.

Grouping-guided crossover

Unlike the previous DGX operator, the grouping-guided crossover (GGX) aims to transmit whole color classes from parents to offspring. Given two parent solutions (i.e., two proper k-colorings) $P_1 = \{V_1^1, \ldots, V_{k_1}^1\}$ ($|V_1^1| \geq \ldots \geq |V_{k_1}^1|$) and $P_2 = \{V_1^2, \ldots, V_{k_2}^2\}$ ($|V_1^2| \geq \ldots \geq |V_{k_2}^2|$). The offspring solution $o = \{V_1^o, \ldots, V_{k_o}^o\}$ is constructed in two steps.

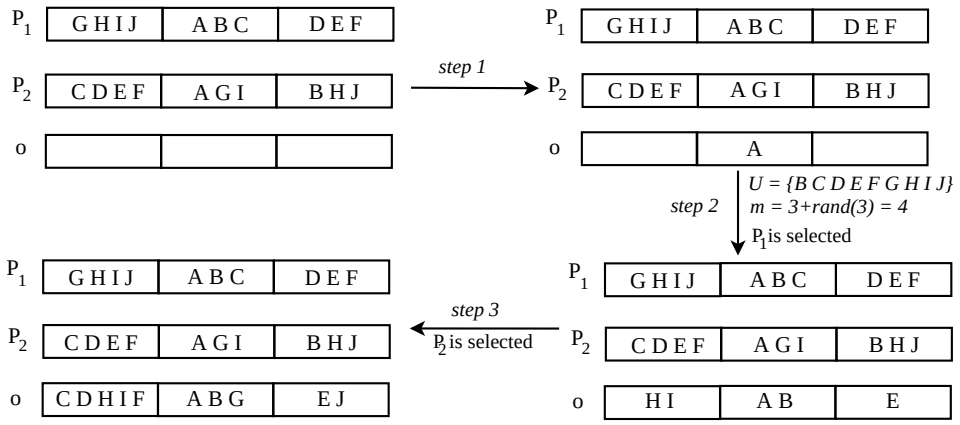


Figure 4.1: An illustrative example of the DGX crossover.

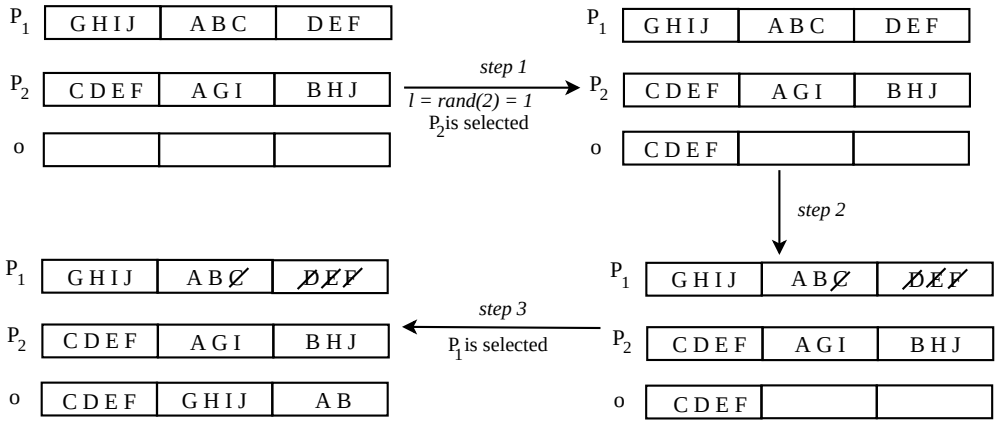


Figure 4.2: An illustrative example of the GGX crossover.

Step 1: We generate an integer $l = rand(\lceil \frac{2}{3} k_2 \rceil)$ and transmit the first l color classes from one parent (randomly selected, say P_2) to construct a partial offspring solution $o = \{V_1^o, \ldots, V_l^o\}$. We remove the vertices v ($v \in o$) from the other parent (P_1).

Step 2: We transmit the non-empty color classes of P_1 to form the $l+1, \ldots, k_o$ color classes of offspring o such that a complete offspring solution is constructed.

The value for l is based on the consideration that we wish to introduce some randomness when deciding the number of transmitted color classes while ensuring some distance between the offspring and each of its parent. An example of the GGX crossover is provided in Fig. 4.2 where l takes the value of 1. The time complexity of this GGX crossover is $O(n \times k)$.

Contrary to the DGX crossover, the GGX operator ensures that offspring solutions are always proper k-colorings. By conserving pertinent properties (color classes) of parent solutions, the offspring colorings are generally of good quality. In the shown example, the offspring even has a better quality than its parents even if this is not true in general.

4.2.4 An iterated double-phase tabu search procedure

Since the recombination procedure may lead to both feasible and unfeasible colorings, we devise an iterated double-phase tabu search (IDTS) able to repair unfeasible solutions while minimizing the sum of colors.

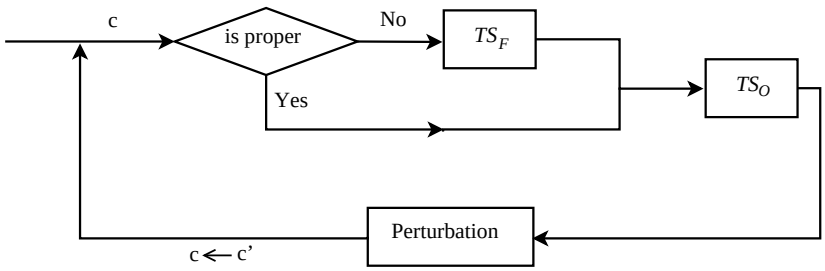


Figure 4.3: An illustration for the IDTS procedure.

The overall IDTS procedure is illustrated in Fig. 4.3. It uses a double-phase tabu search for intensified search and a perturbation mechanism for diversification. The intensification phase applies a tabu search procedure (denoted by TS_O) to improve the quality of a proper coloring according to the objective function and another tabu search procedure (denoted by TS_F) to reestablish the feasibility of an improper coloring. IDTS starts by checking whether a given solution c is a proper coloring. If this is the case, TS_O is called to improve its sum of colors. Otherwise, TS_F is applied for conflict-repairing to attain a proper coloring which is further improved by TS_O according to the objective function. Notice that, to repair an improper coloring, TS_F may increase the number k of used colors until obtaining a proper coloring. The perturbation mechanism is applied to escape local optima when TS_O stagnates, i.e., no improved solution is reached after μ_O consecutive iterations. The perturbed solution is submitted to the next round of the double-phase tabu search process until a maximum number of iterations fixed by $maxIter$ is reached.

A double-phase tabu search

The two tabu search procedures TS_O and TS_F follow the general principle of the tabu search methodology [Glover and Laguna 1999]. Both procedures iteratively visit a series of solutions following the given neighborhood (see below). At each iteration, a best neighboring solution is chosen (ties are broken randomly) to replace the current solution, even if the selected solution does not improve the current solution. To avoid cycling, a tabu list is used to avoid a visited solution to be re-visited during the next TT iterations (TT is called the tabu tenure). Nevertheless, a forbidden solution is always accepted if it is better than the best solution found so far (called aspiration criterion). The tabu search TS_F stops once a proper coloring is obtained and the TS_O procedure stops when the best solution cannot be improved within a given number of solution transitions.

Although both TS_F and TS_O employ the scheme of tabu search, they use different neighborhoods, evaluation functions and tabu tenures.

– **Neighborhood:** The neighborhood of the double-phase tabu search can be described by the "one-move" operator $mv(v, V_i, V_j)$ which displaces a vertex v from its original color class V_i to another color class V_j ($i \neq j$). Given a k-coloring $c = \{V_1, \ldots, V_k\}$, a vertex $v \in V_i$ is conflicting if v shares the same color with at least one adjacent vertex v', i.e., $\exists v' \in V_i$ and $v' \neq v$, $\{v', v\} \in E$. The TS_F procedure (to repair improper colorings) operates with conflicting vertices. At each iteration, it displaces a single conflicting vertex v. For a k-coloring c with nb_{conf} conflicting vertices, the size of this neighborhood is bounded by $O(nb_{conf} \times k)$. The TS_O procedure (to minimize the sum of colors) applies $mv(v, V_i, V_j)$ to move a vertex $v \in V_i$ to another color class V_j such that the resulting k-coloring remains proper ($\forall v' \in V_j, \{v', v\} \notin E$). Hence, the size of this neighborhood is bounded by $O(n \times k)$. In our implementation, we employ an incremental evaluation technique [Fleurent and Ferland 1996, Galinier and Hao 1999] to efficiently evaluate the whole neighborhood.

– **Evaluation function:** Both TS_F and TS_O scan their whole neighborhood to find a best neighboring solution to replace the current solution. The TS_F procedure evaluates all the neighbor solutions by considering both the variation in the number of conflicting vertices $\Delta conf(v, V_i, V_j)$ and the variation in the sum of colors $\Delta f(v, V_i, V_j)$ when applying the $mv(v, V_i, V_j)$ operator. The evaluation of each candidate move is given in Eq. (4.1) which is adopted from [Benlic and Hao 2012]. For two neighboring solutions s' and s'', s' is better than s'' if and only if $\Delta(s') < \Delta(s'')$.

$$\Delta = \Delta conf(v, V_i, V_j) \times \Delta f', \text{where}$$
$$\Delta f' = \begin{cases} abs(\Delta f(v, V_i, V_j)) + k + 1, \text{if } \Delta f(v, V_i, V_j) < 0 \\ k - \Delta f(v, V_i, V_j) + 1, \text{otherwise} \end{cases} \tag{4.1}$$

The TS_O procedure evaluates all the neighboring solutions only by considering the variation in the sum of colors $\Delta f(v, V_i, V_j)$ in terms of the objective function defined by Eq. (1.1).

– **Tabu tenure:** Once a "one-move" $mv(v, V_i, V_j)$ is performed to transform the incumbent coloring c, vertex v cannot be moved back to the color class V_i for the next TT iterations. The tabu tenure TT_F for TS_F is dynamically determined by $TT_F = rand(10) + 0.6 * nb_{conf}$ where $rand(10)$ takes a random number in $\{1, \ldots, 10\}$ and nb_{conf} is the number of conflicting vertices. The tabu tenure TT_O for TS_O is dynamically determined by $TT_O = rand(10) + 0.1 * f(c)$ where c is the incumbent coloring.

Perturbation mechanism

The above double-phase tabu search is generally able to attain solutions of good quality but can get stuck in local optima. To help the procedure to continue its search, we apply a perturbation mechanism to bring the search out of the problematic local optima. The perturbation consists in employing a $(1, r)$-swap ($r = 0, 1, 2, \ldots$) in the case that the current iterations $iter_{cur}\%100 \neq 0$, otherwise, replacing the current solution by the local optimal solution. The $(1, r)$-swap procedure is employed in the following way. First, we randomly choose a vertex v ($v \in V_i$) and a color class V_j ($V_j \neq V_i$). Then, we identify all the vertices v' in V_j which are adjacent to v ($\{v', v\} \in E$). Finally, we move v from V_i to V_j and move all the vertices v' from V_j to V_i. These vertices are forbidden to move back to their original color class for the next TT_P iterations ($TT_P = rand(10) + 0.1 * f(c)$). This perturbation mechanism may introduce conflicts to the current solution such that the search can transit between the feasible and infeasible regions. From this perturbed solution, the double-phase tabu search procedure is relaunched.

4.2.5 Population updating

In order to avoid premature convergence of the search process, the population updating procedure is critical for our hybrid search algorithm. The population updating rule decides whether and how an offspring solution, which is optimized by the IDTS procedure, should replace an existing individual of the population. Basically, our updating rule is based on both solution quality and distance between solutions in the population [Lü and Hao 2010, Porumbel et al. 2010, Sörensen and Sevaux 2006].

Definition 1. Distance between two individuals D_{ij}: Given two individuals $P_i = \{V_1^i, \ldots, V_{k_i}^i\}$ and $P_j = \{V_1^j, \ldots, V_{k_j}^j\}$, the distance between P_i and P_j is defined as the total number of the vertices whose corresponding colors are different in the two individuals, $D_{ij} = |\{v \in V : v \in V_{k_i}^i, v \in V_{k_j}^j, k_i \neq k_j\}|$.

The general scheme of our population updating strategy is described in Algorithm 5. In order to update the population, we calculate the distance D_{oi} between the offspring P_o and any existing solution of the population $P_i \in P$ ($i = 1, \ldots, p$), record the minimum distance D_{min}, and identify the closest individual

Algorithm 5 The population updating procedure

1: **Input**: Population $P = \{P_1, \ldots, P_p\}$ and offspring P_o
2: **Output**: The updated population P
3: $D_{min} \leftarrow +\infty$
4: **for** $i \in \{1, \ldots, p\}$ **do**
5: Calculate the distance D_{oi} between P_o and P_i
6: /*Identify the closest individual P_d with the minimum distance D_{min} to P_o*/
7: **if** $D_{oi} < D_{min}$ **then**
8: $D_{min} \leftarrow D_{oi}$
9: $P_d \leftarrow P_i$
10: **end if**
11: **end for**
12: Identify the worst individual P_w with the largest objective value in P
13: **if** $D_{min} \geq 0.1 \times n$ and $f(P_o) \leq f(P_w)$ **then**
14: Replace P_w with P_o: $P = P \bigcup \{P_o\} \backslash \{P_w\}$
15: **else**
16: **if** $f(P_o) \leq f(P_d)$ **then**
17: Replace P_d with P_o: $P = P \bigcup \{P_o\} \backslash \{P_d\}$
18: **else**
19: **if** $rand(0, 1) \leq 0.1$ **then**
20: Replace P_w with P_o: $P = P \bigcup \{P_o\} \backslash \{P_w\}$
21: **end if**
22: **end if**
23: **end if**

P_d with respect to P_o (lines 3–11). Meanwhile, the worst individual P_w with the largest objective value ($P_w \in P$) is also identified (line 12). If D_{min} is no smaller than $0.1 \times n$ and $f(P_o)$ is no worse than $f(P_w)$, the offspring P_o is inserted into the population and replaces the worst individual P_w (lines 13–14). Otherwise, if $f(P_o)$ is no larger than $f(P_d)$, P_o is close to P_d but has a better quality than P_d, then $f(P_o)$ is inserted into the population and replaces the closest individual P_d (lines 16–17). Otherwise, the worst individual P_w is replaced by P_o with a small probability of 0.1 (lines 19–20).

4.2.6 Discussions

In this section, we discuss the relation of our HSA algorithm with two previous algorithms (MASC) [Jin et al. 2014] and (MA) [Moukrim et al. 2013] for the MSCP. Indeed, all these algorithms follows the general memetic framework which combines population-based search and local optimization. However, our HSA algorithm distinguishes itself from these algorithms by its key components.

First, HSA employs a maximum independent set algorithm to generate proper initial solutions of high quality while MASC and MA use respectively the TABUCOL procedure [Hertz and de Werra 1987] and a greedy coloring heuristic [Li et al. 2009]. Second, for solution recombination, HSA uses two different crossover operators which can generate both feasible and infeasible solutions while MASC and MA allow only feasible solutions. Third, HSA applies an iterated two-phase tabu search procedure to make transitions between feasible and infeasible regions while MASC and MA only explore feasible solutions. Finally, HSA employs a more elaborated pool updating rule to decide whether an offspring solution should replace a worst (or closest) individual in the population. In MASC, this is achieved by a "scoring" function combining solution quality and distance while in MA only solution quality is considered.

As shown in Section 4.4 (experimental results), the proposed HSA algorithm equipped with its particular features shows a highly competitive performance for lower and upper bounds of the MSCP.

4.3 The lower bounds of the minimum sum coloring problem

As described in Chapter 1, we could try to find a partial graph of the original graph to calculate a lower bound for the MSCP and the original graph can be decomposed into partial graphs like trees, paths and cliques. Moreover, the clique decomposition provides better lower bounds than tree or path decomposition [Moukrim et al. 2010]. Let $c = \{S_1, S_2, \ldots, S_k\}$ be a clique decomposition of G, then the following quantity gives a lower bound for the MSCP:

$$f_{LB}(c) = \sum_{l=1}^{k} \frac{|S_l|(|S_l|+1)}{2} \tag{4.2}$$

To obtain a clique decomposition, one popular approach is to find a proper coloring of the complementary graph $\bar{G}$ of G [Helmar and Chiarandini 2011, Wu and Hao 2013b, Moukrim et al. 2013] since each color class of $\bar{G}$ is a clique in G.

We apply our HSA algorithm to color the complementary graph $\bar{G}$ in order to obtain lower bounds. For this purpose, we need to make the following adjustments to our HSA algorithm:

– To evaluate the colorings, we use the objective function defined by Eq. (4.2) instead of sum of colors. For the purpose of computing lower bounds, this objective function is to be maximized. For two proper k-coloring solutions c_1 and c_2 (of $\bar{G}$), c_1 is better than c_2 if and only if $f_{LB}(c_1) > f_{LB}(c_2)$.

– The evaluation function used in the double-phase tabu search needs to be adjusted. The TS_F procedure (for conflict repairing) applies the evaluation function Eq. (4.3) to evaluate a neighboring solution.

$$\Delta = \Delta conf(v, S_i, S_j) \times \Delta f'_{LB}, \text{where}$$
$$\Delta f'_{LB} = \begin{cases} \Delta f_{LB}(v, S_i, S_j) + k + 1, \text{if } \Delta conf(v, S_i, S_j) < 0 \\ k - \Delta f_{LB}(v, S_i, S_j) + 1, \text{otherwise} \end{cases} \tag{4.3}$$

where $\Delta conf(v, S_i, S_j)$ is the variation in the number of conflicting vertices and $\Delta f_{LB}(v, S_i, S_j)$ is the variation in the objective value of Eq. (4.2). For two neighboring solutions s' and s'', s' is better than s'' if and only if $\Delta(s') < \Delta(s'')$. The TS_O procedure evaluates the neighboring solutions by only considering the variation in terms of the objective function from Eq. (4.2).

– For a k-coloring $c = \{S_1, \ldots, S_k\}$, it is no more necessary to sort its color classes S_i $(i = 1, 2, \ldots, k)$, since the calculation of the objective value f_{LB} (Eq. (4.2)) does not sort the cardinality of S_l according to $|S_1| \geq \ldots \geq |S_k|$.

4.4 Experimental results

To evaluate the efficiency of our proposed HSA algorithm, we carry out experiments on the set of 94 COLOR 2002-2004 and DIMACS instances already introduced in Chapter 1.

4.4.1 Experimental protocol

Our HSA algorithm is coded in C++ and compiled using g++ with the '-O3' option on a cluster running Linux with a 2.83 GHz processor and 8 GB RAM. When we run the DIMACS machine benchmark pro-

gram [1] with g++ on our machine, we obtain the following results: 0.20 CPU seconds for graph r300.5, 1.23 CPU seconds for r400.5 and 4.68 CPU seconds for r500.5.

To obtain our computational results, each instance is solved 30 times independently with different random seeds. Each run is stopped when the processing time reaches a fixed timeout limit which is set to be 2 hours (which is a cutoff time frequently used in the literature). All the computational results are obtained with the parameter setting given in Table 4.1.

Table 4.1: Settings of parameters

Parameter	Sect.	Description	Value
μ_O	4.2.4	Maximum number of non-improving moves for TS_O	10
$maxIter$	4.2.4	Maximum number of iterations for perturbation	10^4
p	4.2.2	Population size	20

4.4.2 Computational results

This section is dedicated to an evaluation of HSA's performance for the upper and lower bounds of the MSCP on the 94 benchmark instances. Columns 1–3 in Table 4.2 present the characteristics of the tested graphs and columns f^b_{UB} and f^b_{LB} give the current best-known upper and lower bounds of the MSCP reported in the literature. Columns 6–8 present the detailed computational results of our HSA algorithm for the upper bound: The best upper bound f^*_{UB}, average upper bound $Avg.$ and average running time t to reach the best value for each of the 30 runs (in minutes). Columns 9–11 show the computational results for the lower bound: The best lower bound f^*_{LB}, average lower bound $Avg.$ and average running time t to reach the best value (in minutes).

Table 4.2: Detailed computational results of HSA on the set of 58 COLOR 2002-2004 instances and 36 DIMACS instances

Characteristics of the graphs					HSA			HSA		
Name	n	m	f^b_{UB}	f^b_{LB}	f^*_{UB}	$Avg.$	t	f^*_{LB}	$Avg.$	t
myciel3	11	20	21	16	21	21.0	0.0	16	16.0	0.0
myciel4	23	71	45	34	45	45.0	0.0	34	34.0	0.0
myciel5	47	236	93	70	93	93.0	0.0	70	70.0	0.0
myciel6	95	755	189	142	189	189.0	0.0	142	142.0	0.3
myciel7	191	2 360	381	286	381	381.0	0.0	286	286.0	2.4
anna	138	493	276	273	276	276.0	0.2	273	273.0	0.4
david	87	406	237	234	237	237.0	0.1	234	234.0	0.1
huck	74	301	243	243	243	243.0	0.0	243	243.0	0.0
jean	80	254	217	216	217	217.0	0.0	216	216.0	0.0
homer	561	1 628	1 155	1 129	**1 150**	1 151.8	47.8	1 129	1 129.0	16.6
queen5.5	25	160	75	75	75	75.0	0.0	75	75.0	0.0
queen6.6	36	290	138	126	138	138.0	0.0	126	126.0	0.0
queen7.7	49	476	196	196	196	196.0	0.0	196	196.0	0.0
queen8.8	64	728	291	288	291	291.0	0.1	288	288.0	0.0
queen8.12	96	1 368	624	624	624	624.0	0.0	624	624.0	0.0
queen9.9	81	1 056	409	405	409	409.0	0.5	405	405.0	0.0
queen10.10	100	1 470	553	550	553	553.6	29.6	550	550.0	0.0
queen11.11	121	1 980	733	726	733	734.4	30.1	726	726.0	0.0
queen12.12	144	2 596	944	936	**943**	947.0	41.1	936	936.0	0.0
queen13.13	169	3 328	1 192	1 183	**1 191**	1195.4	29.3	1 183	1183.0	0.0
queen14.14	196	4 186	1 482	1 470	1 482	1487.3	21.6	1 470	1 470.0	0.0
queen15.15	225	5 180	1 814	1 800	1 814	1820.1	25.3	1 800	1 800.0	0.0
queen16.16	256	6 320	2 197	2 176	**2 193**	2199.4	28.1	2 176	2 176.0	0.0
school1	385	19 095	2 674	2 345	2 674	2674.0	0.1	**2 439**	2 418.9	70.6
school1-nsh	352	14 612	2 392	2 106	2 392	2392.0	0.3	**2 176**	2 169.4	61.5
games120	120	638	443	442	443	443.0	0.3	442	442.0	0.0
miles250	128	387	325	318	325	325.0	1.4	318	318.0	0.2
miles500	128	1 170	705	686	705	705.8	20.3	686	686.0	0.0

Continued on next page

1. ftp://dimacs.rutgers.edu/pub/dsj/clique/

Continued from previous page

Characteristics of the graphs					HSA			HSA		
Name	n	m	f^b_{UB}	f^b_{LB}	f^*_{UB}	$Avg.$	t	f^*_{LB}	$Avg.$	t
miles750	128	2 113	1 173	1 145	1 173	1173.6	16.1	1 145	1 145.0	0.0
miles1000	128	3 216	1 679	1 623	**1 666**	1670.5	28.6	1 623	1 623.0	0.1
miles1500	128	5 198	3 354	3 239	3 354	3354.0	0.5	3 239	3 239.0	0.0
fpsol2.i.1	496	11 654	3 403	3 403	3 403	3 403.0	8.5	3 403	3 403.0	13.1
fpsol2.i.2	451	8 691	1 668	1 668	1 668	1 668.0	0.8	1 668	1 668.0	8.9
fpsol2.i.3	425	8 688	1 636	1 636	1 636	1 636.0	1.2	1 636	1 636.0	7.2
mug88_1	88	146	178	164	178	178.0	0.0	164	164.0	0.1
mug88_25	88	146	178	162	178	178.0	0.0	162	162.0	0.1
mug100_1	100	166	202	188	202	202.0	0.0	188	188.0	0.2
mug100_25	100	166	202	186	202	202.0	0.0	186	186.0	0.2
2-Insert_3	37	72	62	55	62	62.0	0.0	55	55.0	0.0
3-Insert_3	56	110	92	84	92	92.0	0.0	84	84.0	0.1
inithx.i.1	864	18 707	3 676	3 676	3 676	3 676.0	1.3	3 676	3 675.3	82.1
inithx.i.2	645	13 979	2 050	2 050	2 050	2 050.0	1.3	2 050	2 050.0	21.5
inithx.i.3	621	13 969	1 986	1 986	1 986	1 986.0	0.0	1 986	1 986.0	18.8
mulsol.i.1	197	3 925	1 957	1 957	1 957	1 957.0	0.2	1 957	1 957.0	0.5
mulsol.i.2	188	3 885	1 191	1 191	1 191	1 191.0	0.0	1 191	1 191.0	0.6
mulsol.i.3	184	3 916	1 187	1 187	1 187	1 187.0	0.0	1 187	1 187.0	0.5
mulsol.i.4	185	3 946	1 189	1 189	1 189	1 189.0	0.0	1 189	1 189.0	0.6
mulsol.i.5	186	3 973	1 160	1 160	1 160	1 160.0	0.0	1 160	1 160.0	0.2
zeroin.i.1	211	4 100	1 822	1 822	1 822	1 822.0	0.0	1 822	1 822.0	0.5
zeroin.i.2	211	3 541	1 004	1 004	1 004	1 004.0	0.0	1 004	1 004.0	1.1
zeroin.i.3	206	3 540	998	998	998	998.0	0.0	998	998.0	0.5
wap05	905	43 081	13 669	12 428	13 887	13 962.7	43.8	**12 449**	12 438.9	57.9
					13656	13 677.8	1 872.5			
wap06	947	43 571	13 776	12 393	14 028	14 090.6	46.0	**12 454**	12 431.6	53.2
					13773	13 777.6	621.3			
wap07	1 809	103 368	28 617	24 339	29 154	29261.1	4.4	**24 800**	24 783.6	72.6
wap08	1 870	104 176	28 885	24 791	29 460	29542.3	3.0	**25 283**	25 263.4	65.6
qg.order30	900	26 100	13 950	13 950	13 950	13 950.0	0.0	13 950	13 950.0	0.1
qg.order40	1 600	62 400	32 800	32 800	32 800	32 800.0	0.0	32 800	32 800.0	0.3
qg.order60	3 600	212 400	109 800	109 800	109 800	109 800.0	0.2	109 800	109 800.0	2.7
DSJC125.1	125	736	326	247	326	326.1	5.2	247	247.0	0.4
DSJC125.5	125	3 891	1 012	549	1 012	1012.2	10.1	549	548.5	34.0
DSJC125.9	125	6 961	2 503	1 689	2 503	2 503.0	0.3	**1 691**	1 691.0	18.8
DSJC250.1	250	3 218	973	569	**970**	980.4	30.7	**570**	569.2	49.0
DSJC250.5	250	15 668	3 214	1 280	**3 210**	3 235.6	47.1	**1 287**	1 271.6	65.6
DSJC250.9	250	27 897	8 277	4 279	8 277	8 277.2	24.6	**4 311**	4 279.4	58.1
DSJC500.1	500	12 458	2 841	1 250	2 848	2 867.1	82.6	1 250	1 243.4	62.0
					2836	28 36.0	1 997.9			
DSJC500.5	500	62 624	10 897	2 921	10 992	11 063.2	97.0	**2 923**	2 896.0	65.6
					10 886	10 891.5	4 919.3			
DSJC500.9	500	112 437	29 896	10 881	**29 886**	29910.4	95.4	**11 053**	10 950.1	68.6
					29 862	29 874.3	5 513.3			
DSJC1000.1	1 000	49 629	8 995	2 762	9 182	9 237.2	101.6	2 719	2 707.6	66.0
					8 991	8 996.5	5 604.4			
DSJC1000.5	1 000	249 826	37 594	6 708	38 520	37 597.6	33.5	6 582	6 541.3	44.6
					37 575	37 594.7	3 090.3			
DSJC1000.9	1 000	449 449	103 464	26 557	104 483	105 221.3	103.1	26 296	26 150.3	51.8
					103 445	103 463.3	211.2			
DSJR500.1	500	3 555	2 173	2 061	**2 156**	2 170.7	99.8	**2 069**	2 069.0	4.2
DSJR500.1c	500	121 275	16 311	15 025	**16 286**	16 286.0	21.7	**15 398**	15 212.4	65.0
DSJR500.5	500	58 862	25 630	22 728	**25 440**	25 684.1	97.6	**22 974**	22 656.7	32.0
flat300_20_0	300	21 375	3 150	1 524	3 150	3 150.0	0.0	**1 531**	1 518.2	75.1
flat300_26_0	300	21 633	3 966	1 536	3 966	3 966.0	0.4	**1 548**	1 530.3	70.2
flat300_28_0	300	21 695	4 238	1 541	4 260	4 290.0	49.7	**1 547**	1 536.5	62.5
flat1000_50_0	1 000	245 000	25 500	6 601	25 500	25 500.0	0.3	6 476	6 452.1	51.5
flat1000_60_0	1 000	245 830	30 100	6 640	30 100	30 100.0	2.7	6 491	6 466.5	46.2
flat1000_76_0	1 000	246 708	37 167	6 632	38 089	38 313.5	36.8	6 509	6 482.8	34.1
					37 164	37 165.9	2 237.0			
le450_5a	450	5 714	1 350	1 190	1 350	1 350.0	0.1	**1 193**	1 191.5	67.4
le450_5b	450	5 734	1 350	1 186	1 350	1 350.1	0.8	**1 189**	1 185.0	67.0
le450_5c	450	9 803	1 350	1 272	1 350	1 350.0	0.9	**1 278**	1 270.4	66.8
le450_5d	450	9 757	1 350	1 269	1 350	1 350.0	0.3	**1 282**	1 274.2	71.6
le450_15a	450	8 168	2 632	2 329	2 634	2 648.4	91.5	**2 331**	2 331.0	23.3
le450_15b	450	8 169	2 642	2 348	**2 632**	2 656.5	89.9	2 348	2 348.0	4.8
le450_15c	450	16 680	3 491	2 593	**3 487**	3 792.4	86.7	**2 610**	2 606.6	57.3

Continued on next page

Continued from previous page

Characteristics of the graphs					HSA			HSA		
Name	n	m	f^b_{UB}	f^b_{LB}	f^*_{UB}	$Avg.$	t	f^*_{LB}	$Avg.$	t
le450_15d	450	16 750	3 506	2 622	**3 505**	3 883.1	82.7	**2 628**	2 627.1	54.9
le450_25a	450	8 260	3 153	3 003	3 157	3 166.7	88.5	3 003	3 003.0	1.2
le450_25b	450	8 263	3 366	3 305	**3 365**	3 375.2	88.6	3 305	3 305.0	1.0
le450_25c	450	17 343	4 515	3 638	4 553	4 583.8	84.8	**3 657**	3 656.9	41.7
le450_25d	450	17 425	4 544	3 697	4 569	4 607.6	92.4	**3 698**	3 698.0	8.3
latin_sqr_10	900	307 350	41 444	40 950	41 492	41 672.8	98.3	40 950	40 950.0	0.0
C2000.5	2 000	999 836	132 515	15 091	139 141	139 676.0	21.4	14 498	14 442.9	24.2
					132 483	132 513.9	161.8			
C4000.5	4 000	4 000 268	473 234	33 033	513 457	514 639.0	75.3	31 525	31 413.3	66.5

From Table 4.2, one observes that HSA is able to improve a number of best lower and upper bounds reported in the literature (indicated in bold) within a time limit of 2 hours. Specifically, for the upper bounds, HSA can improve the best results for 15 instances and match the previous best values for 61 instances. For the lower bounds, HSA can improve the previous best known results for 27 instances and match the previous best results for 59 instances.

When we compare the upper bounds and the lower bounds, we observe large gaps for the DIMACS instances. However, there are 21 instances where the upper bounds are identical to the lower bounds (underlined). Hence, the optimality of these instances is proven by our computational results.

On the other hand, we observe that the HSA algorithm performs less well on some large DIMACS instances which are known to be very difficult for most MSCP algorithms. In order to see if HSA can improve its results on these instances, we carry out another experiment focusing on 14 large graphs with at least 500 vertices as follows. We use the solution of EXSCOL [Wu and Hao 2012] as one of the 20 initial solutions of the population and rerun HSA 30 times on each of the 14 graphs under the same test condition as before. Interestingly, the results of this experiment show that HSA can improve the best known upper bounds for 10 out of 14 graphs (bold italic entries in Table 4.2).

4.4.3 Comparisons with four state-of-the-art algorithms for the lower bounds

In order to further evaluate the proposed HSA algorithm, we compare its lower and upper bounds with those obtained from some of the best performing algorithms in the literature.

Table 4.3 gives the computational comparison for the lower bounds of our HSA algorithm with four state-of-the-art algorithms, which cover the best known lower bounds for all the tested graphs. These algorithms are respectively named RMDS(n) [Moukrim et al. 2010], MDS(5)+LS [Helmar and Chiarandini 2011], EXCLIQUE [Wu and Hao 2013b] and MA [Moukrim et al. 2013]. Notice that the results of RMDS(n) are extracted directly from [Helmar and Chiarandini 2011]. The experimental platforms used by the reference algorithms are as follows:

- RMDS(n) runs on an Intel Core i7 processor 2.93 GHz with 4 GB RAM and uses five heuristics.
- MDS(5)+LS runs on an Intel Core i7 processor 2.93 GHz with 4 GB RAM and uses a limit of 1 hour as the stop condition.
- EXCLIQUE runs on a 2.8 GHz computer with 2GB RAM and iteratively extracts maximum independent sets until the graph is empty.
- MA runs on an Intel Core 2 Duo T5450–1.66 GHz with 2 GB RAM and uses a limit of 2 hours as the stop condition.

Columns 1–2 in Table 4.3 present the tested graph and its best known lower bounds f^b_{LB} reported in the literature. The following 10 columns give the best results f^*_{LB} and the average results $Avg.$ of the four reference algorithms and our HSA algorithm respectively. The "–" marks for the references algorithms in

the table mean that the algorithms did not report results on the tested graphs. The italic entries in the table indicate that the reference algorithms fail to attain the best known results on the tested graphs. The last row in Table 4.3 also presents the number of cases where an algorithm can achieve the best known result ($Suc\#$) over the total number of the tested graphs ($Total\#$). Given the differences among the programming languages, compiler options and computers, we focus on solution quality. We mention that our time limit (2h) is the same as MA, similar to EXCLIQUE on small instances but shorter than EXCLIQUE on large graphs, and longer than RMDS(n) and MDS(5)+LS.

Table 4.3: Comparisons of HSA with four state-of-the-art sum coloring algorithms for the lower bounds of the MSCP on 94 graphs

Graph		RMDS(n)		MDS(5)+LS		EXCLIQUE		MA		HSA	
Name	f^b_{LB}	f^*_{LB}	Avg.	f^*_{LB}	Avg.	f^*_{LB}	Avg.	f^*_{LB}	Avg.	f^*_{LB}	Avg.
myciel3	16	16	–	16	–	16	16.0	16	16.0	16	16.0
myciel4	34	34	–	34	–	34	34.0	34	34.0	34	34.0
myciel5	70	70	–	70	–	70	70.0	70	70.0	70	70.0
myciel6	142	142	–	142	–	142	142.0	142	139.5	142	142.0
myciel7	286	286	–	286	–	286	286.0	286	277.5	286	286.0
anna	273	*272*	–	273	–	273	273.0	273	273.0	273	273.0
david	234	234	–	234	–	*229*	229.0	234	234.0	234	234.0
huck	243	243	–	243	–	243	243.0	243	243.0	243	243.0
jean	216	216	–	216	–	216	216.0	216	216.0	216	216.0
homer	1 129	–	–	–	–	–	–	1 129	1 129.0	1 129	1 129.0
queen5.5	75	75	–	75	–	75	75.0	75	75.0	75	75.0
queen6.6	126	126	–	126	–	126	126.0	126	126.0	126	126.0
queen7.7	196	196	–	196	–	196	196.0	196	196.0	196	196.0
queen8.8	288	288	–	288	–	288	288.0	288	288.0	288	288.0
queen8.12	624	–	–	–	–	–	–	624	624.0	624	624.0
queen9.9	405	–	–	–	–	–	–	405	405.0	405	405.0
queen10.10	550	–	–	–	–	–	–	550	550.0	550	550.0
queen11.11	726	–	–	–	–	–	–	726	726.0	726	726.0
queen12.12	936	–	–	–	–	–	–	936	936.0	936	936.0
queen13.13	1 183	–	–	–	–	–	–	1 183	1 183.0	1 183	1 183.0
queen14.14	1 470	–	–	–	–	–	–	1 470	1 470.0	1 470	1 470.0
queen15.15	1 800	–	–	–	–	–	–	1 800	1 800.0	1 800	1 800.0
queen16.16	2 176	–	–	–	–	–	–	2 176	2 176.0	2 176	2 176.0
school1	2 345	–	–	–	–	–	–	2 345	2 283.3	**2 439**	2 418.9
school1-nsh	2 106	–	–	–	–	–	–	2 106	2 064.6	**2 176**	2 169.4
games120	442	442	–	442	–	442	441.4	442	442.0	442	442.0
miles250	318	*316*	–	318	–	318	316.2	318	318.0	318	318.0
miles500	686	*677*	–	686	–	*677*	671.4	686	686.0	686	686.0
miles750	1 145	–	–	–	–	–	–	1 145	1 145.0	1 145	1 145.0
miles1000	1 623	–	–	–	–	–	–	1 623	1 623.0	1 623	1 623.0
miles1500	3 239	–	–	–	–	–	–	3 239	3 239.0	3 239	3 239.0
fpsol2.i.1	3 403	*3 402*	–	*3 151*	–	3 403	3 403.0	3 403	3 403.0	3 403	3 403.0
fpsol2.i.2	1 668	–	–	–	–	–	–	1 668	1 668.0	1 668	1 668.0
fpsol2.i.3	1 636	–	–	–	–	–	–	1 636	1 636.0	1 636	1 636.0
mug88_1	164	*163*	–	164	–	164	162.3	–	–	164	164.0
mug88_25	162	*161*	–	162	–	162	160.3	–	–	162	162.0
mug100_1	188	*186*	–	188	–	188	188.0	–	–	188	188.0
mug100_25	186	*183*	–	186	–	186	183.4	–	–	186	186.0
2-Insert_3	55	55	–	55	–	55	55.0	–	–	55	55.0
3-Insert_3	84	84	–	84	–	84	82.8	–	–	84	84.0
inithx.i.1	3 676	*3 581*	–	*3 486*	–	3 676	3 676.0	3 676	3 616.0	3 676	3 675.3
inithx.i.2	2 050	–	–	–	–	–	–	2 050	1 989.2	2 050	2 050.0
inithx.i.3	1 986	–	–	–	–	–	–	1 986	1 961.8	1 986	1 986.0
mulsol.i.1	1 957	–	–	–	–	–	–	1 957	1 957.0	1 957	1 957.0
mulsol.i.2	1 191	–	–	–	–	–	–	1 191	1 191.0	1 191	1 191.0
mulsol.i.3	1 187	–	–	–	–	–	–	1 187	1 187.0	1 187	1 187.0
mulsol.i.4	1 189	–	–	–	–	–	–	1 189	1 189.0	1 189	1 189.0
mulsol.i.5	1 160	–	–	–	–	–	–	1 160	1 160.0	1 160	1 160.0
zeroin.i.1	1 822	–	–	–	–	–	–	1 822	1 822.0	1 822	1 822.0
zeroin.i.2	1 004	1 004	–	1 004	–	1 004	1 004.0	1 004	1 002.1	1 004	1 004.0

Continued on next page

Continued from previous page

Graph		RMDS(n)		MDS(5)+LS		EXCLIQUE		MA		HSA	
Name	f^b_{LB}	f^*_{LB}	Avg.	f^*_{LB}	Avg.	f^*_{LB}	Avg.	f^*_{LB}	Avg.	f^*_{LB}	Avg.
zeroin.i.3	998	998	–	998	–	998	998.0	998	998.0	998	998.0
wap05	12 428	–	–	–	–	12 428	12 339.3	–	–	**12 449**	12 438.9
wap06	12 393	–	–	–	–	12 393	12 348.8	–	–	**12 454**	12 431.6
wap07	24 339	–	–	–	–	24 339	24 263.8	–	–	**24 800**	24 783.6
wap08	24 791	–	–	–	–	24 791	24 681.1	–	–	**25 283**	25 263.4
qg.order30	13 950	–	–	–	–	13 950	13 950.0	13 950	13 950.0	13 950	13 950.0
qg.order40	32 800	–	–	–	–	32 800	32 800.0	32 800	32 800.0	32 800	32 800.0
qg.order60	109 800	–	–	–	–	109 800	109 800.0	109 800	109 800.0	109 800	109 800.0
DSJC125.1	247	*238*	–	*238*	–	*246*	244.1	247	244.6	247	247.0
DSJC125.5	549	*504*	–	*493*	–	*536*	522.4	549	541.0	549	548.5
DSJC125.9	1 689	*1 600*	–	*1 621*	–	*1 664*	1 592.5	1 689	1 677.7	**1 691**	1 691.0
DSJC250.1	569	*537*	–	*521*	–	*567*	562.0	569	558.4	**570**	569.2
DSJC250.5	1 280	*1 150*	–	*1 128*	–	*1 270*	1 258.8	1 280	1 249.4	**1 287**	1 271.6
DSJC250.9	4 279	*3 972*	–	*3 779*	–	*4 179*	4 082.4	4 279	4 160.9	**4 311**	4 279.4
DSJC500.1	1 250	*1 163*	–	*1 143*	–	1 250	1 246.6	*1 241*	1 214.9	1 250	1 243.4
DSJC500.5	2 921	*2 616*	–	*2 565*	–	2 921	2 902.6	*2 868*	2 797.7	**2 923**	2 896.0
DSJC500.9	10 881	*10 074*	–	*9 731*	–	10 881	10 734.5	*10 759*	10 443.8	**11 053**	10 950.1
DSJC1000.1	2 762	*2 499*	–	*2 456*	–	2 762	2 758.6	*2 707*	2 651.2	*2 719*	2 707.6
DSJC1000.5	6 708	*5 787*	–	*5 660*	–	6 708	6 665.9	*6 534*	6 182.5	*6 582*	6 541.3
DSJC1000.9	26 557	*23 863*	–	*23 208*	–	26 557	26 300.3	*26 157*	24 572.0	*26 296*	26 150.3
DSJR500.1	2 061	–	–	–	–	–	–	2 061	2 052.9	**2 069**	2 069.0
DSJR500.1c	15 025	–	–	–	–	–	–	15 025	14 443.9	**15 398**	15 212.4
DSJR500.5	22 728	–	–	–	–	–	–	22 728	22 075.0	**22 974**	22 656.7
flat300_20_0	1 524	–	–	–	–	1 524	1 505.7	*1 515*	1 479.3	**1 531**	1 518.2
flat300_26_0	1 536	–	–	–	–	*1 525*	1 511.4	1 536	1 501.6	**1 548**	1 530.3
flat300_28_0	1 541	–	–	–	–	*1 532*	1 515.3	1 541	1 503.9	**1 547**	1 536.5
flat1000_50_0	6 601	–	–	–	–	6 601	6 571.8	*6 433*	6 121.5	*6 476*	6 452.1
flat1000_60_0	6 640	–	–	–	–	6 640	6 600.5	*6 402*	6 047.7	*6 491*	6 466.5
flat1000_76_0	6 632	–	–	–	–	6 632	6 583.2	*6 330*	6 074.6	*6 509*	6 482.8
le450_5a	1 190	–	–	–	–	–	–	1 190	1 171.5	**1 193**	1 191.5
le450_5b	1 186	–	–	–	–	–	–	1 186	1 166.5	**1 189**	1 185.0
le450_5c	1 272	–	–	–	–	–	–	1 272	1 242.3	**1 278**	1 270.4
le450_5d	1 269	–	–	–	–	–	–	1 269	1 245.2	**1 282**	1 274.2
le450_15a	2 329	–	–	–	–	2 329	2 313.7	2 329	2 324.3	**2 331**	2 331.0
le450_15b	2 348	–	–	–	–	*2 343*	2 315.7	2 348	2 335.0	2 348	2 348.0
le450_15c	2 593	–	–	–	–	*2 591*	2 545.3	2 593	2 569.1	**2 610**	2 606.6
le450_15d	2 622	–	–	–	–	*2 610*	2 572.4	2 622	2 587.2	**2 628**	2 627.1
le450_25a	3 003	–	–	–	–	*2 997*	2 964.4	3 003	3 000.4	3 003	3 003.0
le450_25b	3 305	–	–	–	–	3 305	3 304.1	3 305	3 304.1	3 305	3 305.0
le450_25c	3 638	–	–	–	–	*3 619*	3 597.1	3 638	3 617.0	**3 657**	3 656.9
le450_25d	3 697	–	–	–	–	*3 684*	3 627.4	3 697	3 683.2	**3 698**	3 698.0
latin_sqr_10	40 950	–	–	–	–	40 950	40 950.0	–	–	40 950	40 950.0
C2000.5	15 091	–	–	–	–	15 091	15 077.6	–	–	*14 498*	14 442.9
C4000.5	33 033	–	–	–	–	33 033	33 018.8	–	–	*31 525*	31 413.3
Suc#/Total#		17/38		24/38		46/62		71/81		86/94	

Table 4.3 discloses that RMDS(n), MDS(5)+LS, EXCLIQUE, MA and our HSA algorithm can match the best known results for 17/38 (i.e., 17 over 38 tested graphs), 24/38, 46/62, 71/81 and 86/94 graphs respectively. In particular, our HSA algorithm can improve 27 best known lower bounds (see bold entries).

Since each reference algorithm only reports results for a subset of the considered 94 graphs, we compare the performances between our HSA algorithm and the four reference algorithms one by one and summarize the comparisons for the lower bounds of the MSCP in Figure 4.4. The heights of bars in the figure represent the number of graphs and the three different bars indicate the results obtained by our HSA algorithm are better than, equal to, and worse than the results obtained by each reference algorithm respectively. From Figure 4.4, we can observe that HSA obtains improved lower bounds for 21, 14, 24 and 30 graphs, equal results for 17, 24, 30 and 51 graphs and worse results for 0, 0, 8 and 0 graphs compared to RMDS(n), MDS(5)+LS, EXCLIQUE and MA respectively.

Finally, since MDS(5)+LS uses a time limit of 1 hour, we rerun our HSA algorithm under this reduced time condition. We observe that HSA still obtains better lower bounds for 14 instances, equal lower bounds for 24 instances and no worse results compared to the MDS(5)+LS algorithm.

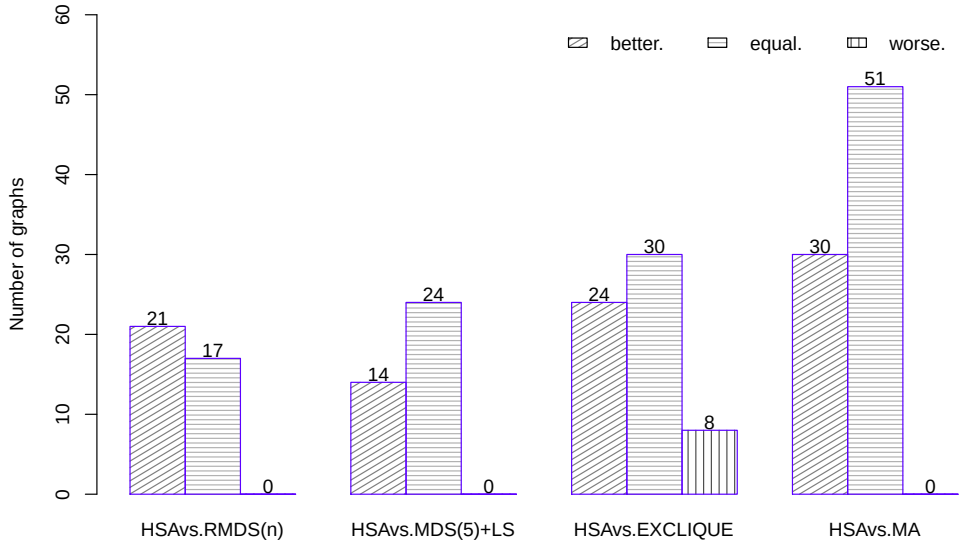


Figure 4.4: Comparisons of HSA and four reference algorithms for the lower bounds.

4.4.4 Comparisons with four state-of-the-art algorithms for the upper bounds

Table 4.4 summarizes the computational comparison for the upper bounds of our HSA algorithm with four very recent state-of-the-art algorithms, which cover the best known results for all the tested graphs. These algorithms are respectively named EXSCOL [Wu and Hao 2012], BLS [Benlic and Hao 2012], MASC [Jin et al. 2014] and MA [Moukrim et al. 2013]. The experimental platforms used by the reference algorithms are as follows:

- EXSCOL runs on a 2.8 GHz computer with 2GB RAM and iteratively extracts maximum independent sets until the graph is empty.
- BLS runs on a Xeon E5440 with 2.83 GHz with 2GB RAM and uses a timeout limit of 2 hours as the stop condition.
- MASC runs on a 2.7 GHz PC with 4GB RAM and uses 10^4 maximum iterations of TS procedure and 50 maximum generations.
- MA runs on an Intel Core 2 Duo T5450–1.66 GHz with 2 GB RAM and uses a timeout limit of 2 hours as the stop condition.

Table 4.4: Comparisons of HSA with four state-of-the-art sum coloring algorithms for the upper bounds of the MSCP on 94 graphs

Graph		EXSCOL		BLS		MASC		MA		HSA	
Name	f^b_{UB}	f^*_{UB}	$Avg.$	f^*_{UB}	$Avg.$	f^*_{UB}	$Avg.$	f^*_{UB}	$Avg.$	f^*_{UB}	$Avg.$
myciel3	21	21	21.0	21	21.0	21	21.0	21	21.0	21	21.0
myciel4	45	45	45.0	45	45.0	45	45.0	45	45.0	45	45.0
myciel5	93	93	93.0	93	93.0	93	93.0	93	93.0	93	93.0
myciel6	189	189	189.0	189	196.6	189	189.0	189	189.0	189	189.0
myciel7	381	381	381.0	381	393.8	381	381.0	381	381.0	381	381.0
anna	276	*283*	283.2	276	276.0	276	276.0	276	276.0	276	276.0
david	237	237	238.1	237	237.0	237	237.0	237	237.0	237	237.0
huck	243	243	243.8	243	243.0	243	243.0	243	243.0	243	243.0
jean	217	217	217.3	217	217.0	217	217.0	217	217.0	217	217.0

Continued on next page

Continued from previous page

Graph		EXSCOL		BLS		MASC		MA		HSA	
Name	f^b_{UB}	f^*_{UB}	Avg.	f^*_{UB}	Avg.	f^*_{UB}	Avg.	f^*_{UB}	Avg.	f^*_{UB}	Avg.
homer	1 155	–	–	–	–	1 155	1 158.5	*1 157*	1 481.9	**1 150**	1 151.8
queen5.5	75	75	75.0	75	75.0	75	75.0	75	75.0	75	75.0
queen6.6	138	*150*	150.0	138	138.0	138	138.0	138	138.0	138	138.0
queen7.7	196	196	196.0	196	196.0	196	196.0	196	196.0	196	196.0
queen8.8	291	291	291.0	291	291.0	291	291.0	291	291.0	291	291.0
queen8.12	624	–	–	–	–	624	624.0	624	624.0	624	624.0
queen9.9	409	–	–	–	–	409	410.5	409	411.9	409	409.0
queen10.10	553	–	–	–	–	–	–	553	555.2	553	553.6
queen11.11	733	–	–	–	–	–	–	733	735.4	733	734.4
queen12.12	944	–	–	–	–	–	–	944	948.7	**943**	947.0
queen13.13	1 192	–	–	–	–	–	–	1 192	1 197.0	**1 191**	1 195.4
queen14.14	1 482	–	–	–	–	–	–	1 482	1 490.8	1 482	1 487.3
queen15.15	1 814	–	–	–	–	–	–	1 814	1 823.0	1 814	1 820.1
queen16.16	2 197	–	–	–	–	–	–	2 197	2 205.9	**2 193**	2 199.4
school1	2 674	–	–	–	–	–	–	2 674	2 766.8	2 674	2 674.0
school1-nsh	2 392	–	–	–	–	–	–	2 392	2 477.1	2 392	2 392.0
games120	443	443	447.9	443	443.0	443	443.0	443	443.0	443	443.0
miles250	325	*328*	333.0	*327*	328.8	325	325.0	325	325.4	325	325.0
miles500	705	*709*	714.5	*710*	713.3	705	705.0	*708*	711.2	705	705.8
miles750	1 173	–	–	–	–	–	–	1 173	1 183.9	1 173	1 173.6
miles1000	1 679	–	–	–	–	–	–	1 679	1 697.3	**1 666**	1 670.5
miles1500	3 354	–	–	–	–	–	–	3 354	3 357.2	3 354	3 354.0
fpsol2.i.1	3 403	–	–	–	–	3 403	3 403.0	3 403	3 403.0	3 403	3 403.0
fpsol2.i.2	1 668	–	–	–	–	1 668	1 668.0	1 668	1 668.0	1 668	1 668.0
fpsol2.i.3	1 636	–	–	–	–	1 636	1 636.0	1 636	1 636.0	1 636	1 636.0
mug88_1	178	–	–	–	–	178	178.0	–	–	178	178.0
mug88_25	178	–	–	–	–	178	178.0	–	–	178	178.0
mug100_1	202	–	–	–	–	202	202.0	–	–	202	202.0
mug100_25	202	–	–	–	–	202	202.0	–	–	202	202.0
2-Insert_3	62	–	–	–	–	62	62.0	–	–	62	62.0
3-Insert_3	92	–	–	–	–	92	92.0	–	–	92	92.0
inithx.i.1	3 676	–	–	–	–	3 676	3 676.0	3 676	3 679.6	3 676	3 676.0
inithx.i.2	2 050	–	–	–	–	2 050	2 050.0	2 050	2 053.7	2 050	2 050.0
inithx.i.3	1 986	–	–	–	–	1 986	1 986.0	1 986	1 986.0	1 986	1 986.0
mulsol.i.1	1 957	–	–	–	–	1 957	1 957.0	1 957	1 957.0	1 957	1 957.0
mulsol.i.2	1 191	–	–	–	–	1 191	1 191.0	1 191	1 191.0	1 191	1 191.0
mulsol.i.3	1 187	–	–	–	–	1 187	1 187.0	1 187	1 187.0	1 187	1 187.0
mulsol.i.4	1 189	–	–	–	–	1 189	1 189.0	1 189	1 189.0	1 189	1 189.0
mulsol.i.5	1 160	–	–	–	–	1 160	1 160.0	1 160	1 160.0	1 160	1 160.0
zeroin.i.1	1 822	–	–	–	–	1 822	1 822.0	1 822	1 822.0	1 822	1 822.0
zeroin.i.2	1 004	–	–	–	–	1 004	1 004.0	1 004	1 004.0	1 004	1 004.0
zeroin.i.3	998	–	–	–	–	998	998.0	998	998.0	998	998.0
wap05	13 669	*13 680*	13 718.4	–	–	13 669	13 677.8	–	–	**13 656**	13 677.8
wap06	13 776	*13 778*	13 830.9	–	–	13 776	13 777.8	–	–	**13 773**	13 777.6
wap07	28 617	*28 629*	28 663.8	–	–	28 617	28 624.7	–	–	*29 154*	29 261.1
wap08	28 885	*28 896*	28 946.0	–	–	28 885	28 890.9	–	–	*29 460*	29 542.3
qg.order30	13 950	13 950	13 950.0	–	–	13 950	13 950.0	13 950	13 950.0	13 950	13 950.0
qg.order40	32 800	32 800	32 800.0	–	–	32 800	32 800.0	32 800	32 800.0	32 800	32 800.0
qg.order60	10 9800	*110 925*	110 993.0	–	–	10 9800	10 9800.0	10 9800	10 9800.0	10 9800	10 9800.0
DSJC125.1	326	326	326.7	326	326.9	326	326.6	326	327.3	326	326.1
DSJC125.5	1 012	*1 017*	1 019.7	1 012	1 012.9	1 012	1 020.0	*1 013*	1 018.5	1 012	1 012.2
DSJC125.9	2 503	*2 512*	2 512.0	2 503	2 503.0	2 503	2 508.0	2 503	2 519.0	2 503	2 503.0
DSJC250.1	973	*985*	985.0	973	982.5	*974*	990.5	*983*	995.8	**970**	980.4
DSJC250.5	3 214	*3 246*	3 253.9	*3 219*	3 248.5	*3 230*	3 253.7	3 214	3 285.5	**3 210**	3 235.6
DSJC250.9	8 277	*8 286*	8 288.8	*8 290*	8 316.0	*8 280*	8 322.7	8 277	8 348.8	8 277	8 277.2
DSJC500.1	2 841	*2 850*	2 857.4	*2 882*	2 942.9	2 841	2 844.1	*2 897*	2 990.5	**2 836**	2 836.0
DSJC500.5	10 897	*10 910*	10 918.2	*11 187*	11 326.3	10 897	10 905.8	*11 082*	11 398.3	**10 886**	10 891.5
DSJC500.9	29 896	*29 912*	29 936.2	*30 097*	30 259.2	29 896	29 907.8	*29 995*	30 361.9	**29 862**	29 874.3
DSJC1000.1	8 995	*9 003*	9 017.9	*9 520*	9 630.1	8 995	9 000.5	*9 188*	9 667.1	**8 991**	8 996.5
DSJC1000.5	37 594	*37 598*	37 673.8	*40 661*	41 002.6	37 594	37 597.6	*38 421*	40 260.9	**37 575**	37 594.7
DSJC1000.9	103 464	103 464	103 531.0	–	–	103 464	103 464.0	*105 234*	107 349.0	**103 445**	103 463.3
DSJR500.1	2 173	–	–	–	–	–	–	2 173	2 253.1	**2 156**	2 170.7
DSJR500.1c	16 311	–	–	–	–	–	–	16 311	16 408.5	**16 286**	16 286.0
DSJR500.5	25 630	–	–	–	–	–	–	25 630	26 978.0	**25 440**	25 684.1
flat300_20_0	3 150	3 150	3 150.0	–	–	3 150	3 150.0	3 150	3 150.0	3 150	3 150.0
flat300_26_0	3 966	3 966	3 966.0	–	–	3 966	3 966.0	3 966	3 966.0	3 966	3 966.0
flat300_28_0	4 238	*4 282*	4 286.1	–	–	4 238	4 313.4	*4 261*	4 389.4	*4 260*	4 290.0

Continued on next page

Continued from previous page

Graph		EXSCOL		BLS		MASC		MA		HSA	
Name	f^b_{UB}	f^*_{UB}	Avg.	f^*_{UB}	Avg.	f^*_{UB}	Avg.	f^*_{UB}	Avg.	f^*_{UB}	Avg.
flat1000_50_0	25 500	25 500	25 500.0	–	–	25 500	25 500.0	25 500	25 500.0	25 500	25 500.0
flat1000_60_0	30 100	30 100	30 100.0	–	–	30 100	30 100.0	30 100	30 100.0	30 100	30 100.0
flat1000_76_0	37 167	37 167	37 213.2	–	–	37 167	37 167.0	*38 213*	39 722.7	**37 164**	37 165.9
le450_5a	1 350	–	–	–	–	1 350	1 350.0	1 350	1 350.0	1 350	1 350.0
le450_5b	1 350	–	–	–	–	1 350	1 350.0	1 350	1 350.0	1 350	1 350.1
le450_5c	1 350	–	–	–	–	1 350	1 350.0	1 350	1 350.0	1 350	1 350.0
le450_5d	1 350	–	–	–	–	1 350	1 350.0	1 350	1 350.0	1 350	1 350.0
le450_15a	2 632	2 632	2 641.9	–	–	*2 706*	2 742.6	*2 681*	2 733.1	*2 634*	2 648.4
le450_15b	2 642	2 642	2 643.4	–	–	*2 724*	2 756.2	*2 690*	2 730.6	**2 632**	2 656.5
le450_15c	3 491	*3 866*	3 868.9	–	–	3 491	3 491.0	*3 943*	4 048.4	**3 487**	3 792.4
le450_15d	3 506	*3 921*	3 928.5	–	–	3 506	3 511.8	*3 926*	4 032.4	**3 505**	3 883.1
le450_25a	3 153	3 153	3 159.4	–	–	*3 166*	3 176.8	*3 178*	3 204.3	*3 157*	3 166.7
le450_25b	3 366	3 366	3 371.9	–	–	3 366	3 375.1	*3 379*	3 416.2	**3 365**	3 375.2
le450_25c	4 515	4 515	4 525.4	–	–	*4 700*	4 773.3	*4 648*	4 700.7	*4 553*	4 583.8
le450_25d	4 544	4 544	4 550.0	–	–	*4 722*	4 805.7	*4 696*	4 740.3	*4 569*	4 607.6
latin_sqr_10	41 444	*42 223*	42 392.7	–	–	41 444	41 481.5	–	–	*41 492*	41 672.8
C2000.5	132 515	132 515	132 682.0	–	–	–	–	–	–	**132 483**	132 513.9
C4000.5	473 234	473 234	473 211.0	–	–	–	–	–	–	*513 457*	514 639.0
$Suc\#/Total\#$		29/52		18/27		69/77		61/81		85/94	

Like for the lower bounds (Table 4.3), columns 1–2 in Table 4.4 present the best known upper bounds f^b_{UB}, the following 10 columns give the best results f^*_{UB} and the average results $Avg.$ of the four reference algorithms and our HSA algorithm respectively. The "–" marks for the reference algorithms in the table mean that the algorithms did not report results on the tested graphs. The italic entries in the table mean that the reference algorithms fail to attain the best known results on the tested graphs. The last row in Table 4.4 presents the number of cases where the algorithm can achieve the best known result ($Suc\#$) over the total number of the tested graphs ($Total\#$). Once again, we only focus on solution quality and we mention that our timeout limit (2h) is the same as MA and BLS and similar to MASC and EXSCOL on the small instances but shorter than MASC and EXSCOL on the large graphs. Besides, EXSCOL, BLS, MASC, MA, and our HSA algorithm are tested on 52, 27, 77, 81 and 94 graphs respectively.

From Table 4.4, we observe that EXSCOL, BLS, MASC, MA and our HSA algorithm can match the best known results for 29, 18, 69, 61 and 85 graphs, but fail to reach the best results for 23, 9, 8, 20 and 9 instances respectively. In particular, our HSA algorithm can improve the best known results for 24 graphs (bold entries).

Like for the lower bounds, we compare our HSA algorithm with each of the four reference algorithms and summarize the comparisons in Figure 4.5 with the same information as in Figure 4.4. From Figure 4.5, we can observe that HSA obtains better results for 26, 10, 21 and 28 graphs, equal results for 19, 17, 52 and 53 graphs and worse results for 7, 0, 4 and 0 results compared to EXSCOL, BLS, MASC, and MA respectively. This comparison study shows clearly that the proposed HSA algorithm competes very favorably with the reference algorithms in terms of upper bounds of the MSCP.

4.5 Analysis of HSA

In this section, we study first the impact of the joint use of two crossover operators on the performance of the proposed HSA algorithm. Moreover, we perform a fitness distance analysis (FDA) [Jones and Forrest 1995] in order to obtain some insight on the hardness of some benchmark instances, which may help understand the behavior of our HSA algorithm.

4.5.1 Analysis of the double-crossover operator

As indicated in Section 4.2.3, HSA employs two crossover operators (GGX and DGX) to generate offspring solutions. In order to investigate the positive role of this mechanism, we compare HSA with its

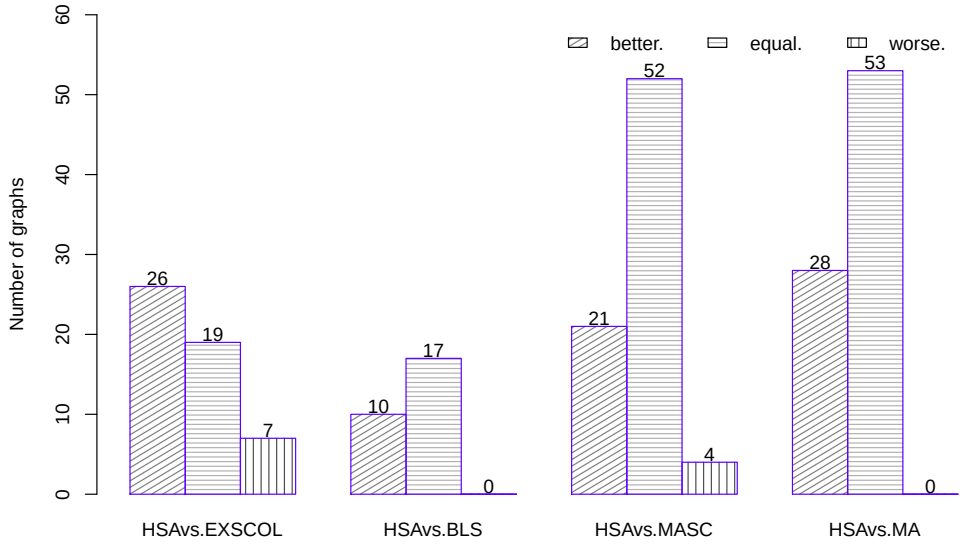


Figure 4.5: Comparisons of HSA and four reference algorithms for the upper bounds.

two variants: HSA$_{GGX}$ uses only the GGX crossover while HSA$_{DGX}$ uses only the DGX crossover. We carry out additional experiments on 20 selected graphs and run HSA, HSA$_{GGX}$ and HSA$_{DGX}$ for 30 times on each graph. These three algorithms use the same parameter settings given in Table 4.1 and the same timeout limits (2 hours).

Table 4.5: Comparisons on 20 selected graphs for the upper and lower bounds of the MSCP

Graph			HSA				HSA$_{GGX}$				HSA$_{DGX}$			
Name	f^b_{UB}	f^b_{LB}	f^*_{UB}	avg_{UB}	f^*_{LB}	avg_{LB}	f^*_{UB}	avg_{UB}	f^*_{LB}	avg_{LB}	f^*_{UB}	avg_{UB}	f^*_{LB}	avg_{LB}
homer	1155	1129	**1150**	1151.8	1129	1129.0	**1150**	1151.9	1129	1129.0	**1151**	1152.2	1129	1129.0
queen11.11	733	726	733	734.4	726	726.0	733	736.2	726	726.0	733	735.2	726	726.0
queen12.12	944	936	**943**	947.0	936	936.0	945	948.5	936	936.0	**942**	947.5	936	936.0
queen13.13	1192	1183	**1191**	1195.4	1183	1183.0	1194	1197.3	1183	1183.0	1192	1197.3	1183	1183.0
miles250	325	318	325	325.0	318	318.0	325	325.0	318	318.0	325	325.0	318	318.0
miles500	705	686	705	705.8	686	686.0	705	706.2	686	686.0	705	705.9	686	686.0
DSJC250.1	973	569	**970**	980.4	**570**	569.2	977	981.0	**570**	569.5	**972**	980.9	**570**	568.8
DSJC250.5	3214	1280	**3210**	3235.6	**1287**	1271.6	3222	3243.7	**1285**	1275.1	**3210**	3235.7	1270	1261.6
DSJC250.9	8277	4279	8277	8277.2	**4311**	4279.4	8277	8279.4	**4294**	4269.4	8277	8277.0	**4312**	4273.1
DSJC500.1	2841	1250	2848	2867.1	1250	1243.4	2850	2870.4	1248	1244.4	2848	2870.4	1245	1241.2
DSJC500.5	10897	2921	10992	11063.2	**2923**	2896.0	10969	11066.2	2918	2898.6	11012	11094.0	2894	2876.1
DSJC500.9	29896	10881	**29886**	29910.4	**11053**	10950.1	**29869**	29900.0	**11049**	10952.0	29900	29927.0	**10982**	10853.3
DSJR500.1	2173	2061	**2156**	2170.7	**2069**	2069.0	**2154**	2167.1	**2069**	2069.0	**2159**	2172.6	**2069**	2069.0
DSJR500.1c	16311	15025	**16286**	16286.0	**15398**	15212.4	**16286**	16286.0	**15313**	15185.6	**16286**	16286.0	**15118**	15006.4
DSJR500.5	25630	22728	**25440**	25684.1	**22974**	22656.7	**25439**	25565.9	22641	22634.7	25935	26029.2	**22999**	22643.0
flat300_28_0	4238	1541	4260	4290.0	**1547**	1536.5	4270	4296.5	**1544**	1535.9	4261	4289.4	1533	1519.3
le450_15a	2632	2329	2634	2648.4	**2331**	2331.0	2637	2649.5	**2331**	2330.9	2642	2658.5	**2331**	2331.0
le450_15b	2642	2348	**2632**	2656.5	2348	2348.0	2644	2656.3	2348	2348.0	**2641**	2659.9	2348	2348.0
le450_15c	3491	2593	**3487**	3792.4	**2610**	2606.6	**3490**	3853.5	**2610**	2607.4	3491	3814.7	**2610**	2606.6
le450_15d	3506	2622	**3505**	3883.1	**2628**	2627.1	3829	3913.8	**2628**	2626.7	**3504**	3774.9	**2628**	2627.2
suc#			16		20		10		17		14		16	

Table 4.5 summarizes the computational results of HSA, HSA$_{GGX}$ and HSA$_{DGX}$. Column 1–3 recall the best known lower and upper bounds of the 20 graphs. Columns 4–15 present the upper bounds, the average upper bounds, the lower bounds and the average lower bounds achieved by HSA, HSA$_{GGX}$ and HSA$_{DGX}$ respectively. The last row gives the number of times an algorithm finds a better or equal result compared to the best known result.

Table 4.6: FDC analysis on 20 selected graphs for the lower and upper bounds of the MSCP

Graph	Upper bounds of the MSCP				Lower bounds of the MSCP			
	$\#lo$	$avg\ d_{lo}$	$avg\ d_{go}$	ρ	$\#lo$	$avg\ d_{lo}$	$avg\ d_{go}$	ρ
homer	1188	166.388	96.875	0.772	1011	540.925	0.542	-0.999
queen11_11	1188	105.468	45.633	0.666	730	108.392	0.006	-0.999
queen12_12	1191	119.169	31.407	0.874	784	131.224	0.084	-0.975
queen13_13	1191	147.310	67.307	0.688	829	154.965	0.000	-0.999
miles250	1137	36.563	3.991	0.865	1200	123.578	0.000	-0.999
miles500	1191	69.946	28.412	0.729	1200	119.666	0.123	-0.704
DSJC250.1	706	87.201	94.048	0.462	1200	246.501	243.506	-0.126
DSJC250.5	1183	205.100	213.692	0.122	1198	236.674	222.969	-0.292
DSJC250.9	1199	238.464	86.344	0.637	1198	214.859	168.488	-0.472
DSJC500.1	942	354.208	184.689	0.675	1200	494.570	491.143	-0.100
DSJC500.5	1200	484.804	482.820	0.150	1200	489.638	488.290	-0.094
DSJC500.9	1200	491.331	490.818	0.056	1200	461.389	460.501	-0.133
DSJR500.1	1185	279.052	226.710	0.646	1200	481.245	286.637	-0.272
DSJR500.1c	1198	421.765	212.482	0.304	1200	440.222	450.543	-0.127
DSJR500.5	1200	491.142	489.561	0.105	769	289.422	385.740	-0.020
flat300_28_0	1199	274.961	262.538	0.341	1200	287.138	286.731	-0.139
le450_15a	1159	273.573	299.082	0.161	1200	442.403	389.603	-0.151
le450_15b	1115	234.293	202.203	0.527	1200	443.224	329.997	-0.130
le450_15c	1108	352.946	230.713	0.921	1200	441.681	426.002	-0.217
le450_15d	1086	350.593	418.864	0.122	1200	441.665	436.336	-0.056

From Table 4.5, we can make the following observations. First, HSA with its two crossover operators can reach the best known lower and upper bounds for 36 out of the 40 cases. HSA$_{GGX}$ and HSA$_{DGX}$ can only reach the best known results for 27 and 30 instances respectively. Second, HSA is able to improve the best known results for 24 instances (bold) while HSA$_{GGX}$ and HSA$_{DGX}$ improve the best results for 16 and 17 instances respectively. Third, HSA obtains better results compared to HSA$_{GGX}$ and HSA$_{DGX}$ for 12 instances and worse results for 4 instances respectively (underlined). Besides, HSA$_{GGX}$ and HSA$_{DGX}$ can complement each other on some graphs, for instances, the DSJC500.9 and DSJR500.9 instances. In summary, using jointly the DGX and GGX crossovers allows HSA to reach a better performance than when these crossovers are used separately. This is particularly useful to handle different graphs with multiple topologies.

4.5.2 Landscape analyses

The fitness-distance correlation (FDC) coefficient ρ measures the correlation between the quality (fitness or objective function value) of local optima and their distances to the optimum of a given problem instance [Jones and Forrest 1995]. If the solution quality increases with the diminution of distance to the optimum, then there is a path to the optimum via solutions with increasing (better) fitness. Even if FDC alone cannot fully characterize the hardness of a problem, it can provide useful information about the landscape of the problem. For a minimization problem, a ρ value close to 1 (the largest possible value) indicates a strong fitness-distance correlation while a ρ value close to -1 (the smallest possible value) means the absence of any correlation. The reverse is true for a maximization problem. In this section, we present for the first time a FDC analysis of the MSCP both for the problems of computing upper bounds (minimization) and lower bounds (maximization).

Table 4.6 presents the results of the FDC analysis on the 20 selected graphs for the problems of calculating lower and upper bounds of the MSCP. For each graph, we collect 1200 local optima and identify the number of distinct local optima among these 1200 collected solutions ($\#lo$), the average distance between local optima ($avg\ d_{lo}$), the average distance between a local optimum and the closest best known local optimum ($avg\ d_{go}$) and the FDC coefficient (ρ).

From Table 4.6, one notices that for the minimization problem of upper bounds, the ρ values of COLOR 2002-2004 instances (close to 1) are larger than the ρ values of most DIMACS instances (close to 0). For the

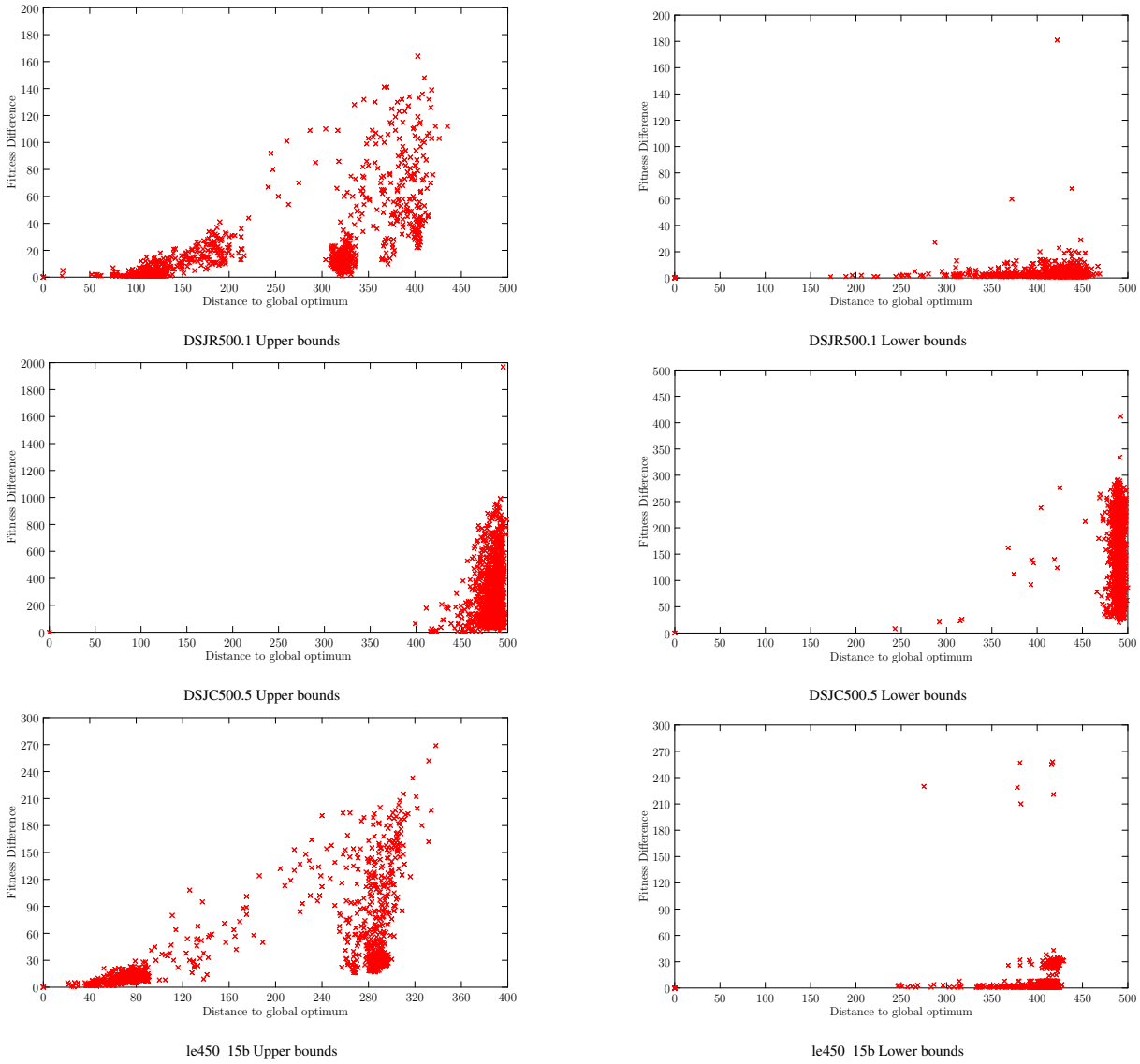


Figure 4.6: FDC plots on 4 graphs for the lower and upper bounds

maximization problem of lower bounds, the ρ values of COLOR 2002-2004 instances are close to -1 while the ρ values of most DIMACS instances are close to 0. These observations indicate that most DIMACS instances would be more difficult to solve compared to the COLOR 2002-2004 instances. This is indeed coherent with the experimental results of Section 4.4.2. In order to investigate the landscape in a visual way, we provide the FDC plots in Figure 4.6 with respect to the fitness difference and the distance between a local optimum and the nearest global optimum on three difficult DIMACS graphs (for the problems of lower and upper bounds). We can clearly see that there is no correlation for DSJC500.5 (for both problems of lower and upper bounds), for DSJR500.1 and le450_15b (lower bounds).

4.6 Conclusion

In this chapter, we presented an efficient hybrid search algorithm (HSA) for the lower and upper bounds of the minimum sum coloring problem (MSCP). HSA combines a double-crossover recombination method, a dedicated iterated double-phase tabu search (IDTS) procedure and a quality and diversity based population updating method. The recombination method jointly applies a diversification-guided crossover (DGX) and a grouping-guided crossover (GGX) to generate promising offspring solutions. The IDTS applies specific

strategies to make transitions between feasible and infeasible solutions and a perturbation mechanism to escape from local optima traps.

Experimental evaluations on 94 benchmark instances showed that the proposed HSA algorithm is highly competitive in comparison with the state-of-the-art algorithms for the MSCP. HSA can match most of the current best known lower and upper bounds. In particular, it is able to improve the best known upper bounds for 24 graphs and the best known lower bounds for 27 graphs.

Additionally, we carried out experiments to verify the merit of the double-crossover recombination method. Moreover, we showed for the first time a landscape analysis on a number of selected instances for the lower and upper bounds of the MSCP, which allows us to understand why some instances are more difficult than others.

5

LHS: Learning-based Hybrid Search for bandwidth (multi)coloring

In this chapter, we consider the bandwidth coloring problem (BCP) and the bandwidth multicoloring problem (BMCP) which are two other important generalizations of the classic vertex coloring problem. This chapter introduces an effective learning-based hybrid search (LHS) for the BCP and BMCP. LHS is based on the cooperation framework of an informed construction procedure and a local search repair procedure. The proposed algorithm is evaluated on two sets of 66 commonly used BCP and BMCP benchmark instances in the literature. The computational results show that our LHS algorithm can achieve the best-known result for most of these benchmark instances established by several existing algorithms. Moreover, LHS finds an improved best solution for 14 instances (2 BCP instances and 12 BMCP instances). The content of this chapter is published in [Jin and Hao 2015a].

Contents

5.1 Introduction

This chapter is dedicated to the bandwidth coloring problem and the bandwidth multicoloring problem which are formally presented in Chapter 2. Recall that given an undirected graph, the BCP consists in

finding a k-coloring with the smallest value of k such that the absolute value of the difference between the colors of adjacent vertices is not less than the weight of the associated edge. The BCP can be generalized as the BMCP where each vertex receives a number of different colors. A legal bandwidth multicoloring must satisfy two constraints: (1) the absolute value of the difference between the colors of adjacent vertices is not less than the weight of the associated edge; (2) the absolute value of the difference between any two distinct colors of a vertex is at least the weight of the loop edge of this vertex. The BMCP is to find a legal bandwidth multicoloring with k minimum.

In this chapter, we propose an effective heuristic approach, called Learning-based Hybrid Search (LHS) for the BCP and BMCP. LHS is based on the cooperation of an informed construction procedure and a local search repair procedure and integrates several distinguishing features. The main contributions of the work can be summarized as follows.

– From the algorithm perspective, the proposed LHS approach establishes an original cooperative framework between an informed construction approach and a local search approach. The construction procedure progressively builds feasible (partial) solutions while relying on 1) a dynamic learning-based guiding function to determine the order of vertices to be colored and 2) a forward checking technique to reduce the available colors of the considered vertex. In particular, the guiding function takes into account both static information of the instance under consideration and dynamic information learned during the construction and the repair processes. When the construction procedure runs into a *dead-end* (i.e., the partial solution under construction can not be extended any more without violating some constraints), the search switches to the repair procedure to try to unlock the dead-end situation in order to switch back to the coloring construction process. The local search repair procedure is based on the tabu search metaheuristic reinforced by a simple perturbation strategy. To our knowledge, this is the first hybrid algorithm of this kind proposed for the BCP and BMCP. Moreover, the underlying cooperative framework could be useful and adapted to other problems.

– From the computational perspective, we evaluate the LHS approach on two sets of 66 commonly used BCP and BMCP benchmark instances in the literature. The computational results show that our LHS algorithm can achieve the best-known result for most of these benchmark instances established by several existing algorithms. Moreover, LHS finds an improved best solution for 14 instances (2 BCP instances and 12 BMCP instances).

The rest of this chapter is organized as follows. Next section presents the learning-based hybrid search for the BCP and BMCP. Section 5.3 shows computational results on the benchmark instances and comparisons with some best performing algorithms. Before concluding, Section 5.4 shows an analysis of the proposed LHS approach.

5.2 Components of the LHS approach

The Learning-based Hybrid Search (LHS) approach is designed for the BCP (more precisely, for the k-BCP problem), since one can easily convert the BMCP into the BCP by defining a new graph as follows (see [Dorne and Hao 1999] for an example): For each vertex v_i of the BMCP, we define a clique $\{v_{i_1}, v_{i_2}, ..., v_{p_i}\}$ of cardinality $p(i)$ with a distance $d(i,i)$ for each edge of the clique. For each edge $\{v_i, v_j\} \in E$, the distance $d(i,j)$ is duplicated for each pair of vertices $\{v_{i_x}, v_{j_y}\}$ of the two corresponding cliques. Then the new graph has $\sum_{v_i \in V} p(i)$ vertices and $\sum_{\{v_i,v_j\} \in E} p(i) \times p(j)$ edges.

To approximate the BCP, we solve a series of k-BCP as performed in [Dorne and Hao 1999, Lai and Lü 2013, Malaguti and Toth 2008], i.e., to seek a legal bandwidth coloring with k colors (k can be initially

fixed to be slightly a value larger than or equal to the best-known k in the literature). As soon as a legal k-coloring is found, we decrease the current value k to $k-1$ and solve the new k-BCP. We repeat this process until no legal k-coloring can be found and return the last k for which a legal k-coloring is reached. We describe below the LHS algorithm which basically solves the decision k-BCP problem.

5.2.1 General procedure

Our Learning-based Hybrid Search (LHS) approach repeats the following two phases: a *coloring construction phase* (Section 5.2.3) to extend in a step-by-step way a partial legal solution by coloring a new vertex at each step and a *repair phase* (Section 5.2.4) using tabu search [Glover and Laguna 1999] to solve constraint violations when the partial legal solution cannot be further extended. The main procedure of the LHS approach is summarized in Algorithm 6.

The coloring construction phase operates with partial (legal) solutions and tries to expand a partial solution to a complete solution without violating the problem constraints. Starting from an empty solution, the construction procedure selects at each step, according to a learning-based guiding function (see Section 5.2.2), an uncolored vertex and tries to assign to it an available color. For the selected vertex, if a color can be assigned to it without violating any distance constraints, the vertex receives the color and the construction phase continues. If no feasible coloring is possible for the selected vertex, a dead-end is encountered (in this case, the last selected vertex is called a *dead-end vertex*) and LHS switches to the tabu search repair phase to escape from the dead-end.

Suppose the partial legal solution is composed of $L-1$ colored vertices when the dead-end is encountered. Then the tabu search repair phase takes as its input the partial solution and extends it by assigning a randomly selected color from the given k colors to the dead-end vertex. Obviously, this extension leads inevitably to an illegal coloring (with L vertices) which violates some distance constraints. The purpose of the repair phase is then to try to find a legal coloring for the set of L vertices by re-coloring these vertices. At the end of the repair process, there are two possibilities. If the dead-end is resolved, i.e., a legal partial coloring is found for the set of L vertices, LHS switches back to the construction phase to continue its coloring construction. On the other hand, if the repair procedure fails to find a legal partial coloring for the set of L vertices, LHS drops the on-going process and prepares to restart a new round of construction-repair process. In order to learn from this failure, LHS updates the guiding function of some critical variables (see Section 5.2.2) with the help of an adaptive reinforcement learning strategy. As such, the next round of the construction phase will benefit from some learned information to re-order the vertices such that the critical vertices which are difficult to color will be considered with a high priority. LHS repeats the above process until a pre-fixed number of maximum tries is reached or a complete legal k-coloring is obtained.

5.2.2 Learning-based guiding function

As we explain above and in Section 5.2.3, the construction procedure employs a guiding function $\mathcal{F}$ to dynamically determine the order of vertices for color assignment. This guiding function constitutes thus one of the most critical components of the LHS algorithm and needs to be designed with care.

In our case, the guiding function $\mathcal{F}$ dynamically ranks each vertex v by taking into account both static and learning-based dynamic information and is called at each step of the construction process to select a vertex for color assignment (a vertex with the highest rank is selected, ties are broken at random). This function takes the following form:

Algorithm 6 Learning-based Hybrid Search for the Bandwidth Coloring Problem

Require: A graph $G = (V, E)$, an integer k
Ensure: A feasible k-coloring C_* found or null
$T \leftarrow 0$ /* T counts the failed 'construction-repair' rounds */
$C \leftarrow \emptyset$ /* C is the current feasible coloring under construction */
while $T \leq maxTries$ **do**
  repeat
    /* Construction phase */
    $(C, v_i) \leftarrow$ Construct_partial_solution(C) /* v_i is the dead-end vertex encountered, Sect. 5.2.3 */
    /* Tabu search based conflict repair phase */
    $C_* \leftarrow$ Tabu_search_repair(C, v_i) /* Apply tabu search to solve conflicts, Sect. 5.2.4 */
    if C_* is still a conflicting coloring **then**
      /* The current round of construction-local search fails */
      Update the learning-based guiding function $\mathcal{F}$ /* Sect. 5.2.2 */
      $C \leftarrow \emptyset$; $C_* \leftarrow \emptyset$
      break
    else
      $C \leftarrow C_*$
    end if
  until $|C_*| = n$
  if $|C_*| = n$ **then**
    return C_* /* C_* is a complete and legal k-coloring, return C_* */
  end if
  $T \leftarrow T + 1$
end while
return C_*

$$\mathcal{F}(v) = \begin{cases} deg(v) + fr\_deg(v), & T = 0 \\ fb\_val(v) + fr\_deg(v), & T > 0, \forall v \in V \end{cases} \tag{5.1}$$

where T is the number of the failed 'construction-repair' rounds.

The $deg(v)$ part of $\mathcal{F}$ represents the connection degree of vertex v, i.e., $deg(v) = |\Gamma(v)|$ where $\Gamma(v) = \{u \in V : \{v, u\} \in E\}$. For a given graph, this part remains static and captures a basic and main characteristic of the graph. The use of this information within $\mathcal{F}$ is based on the consideration that a vertex with a large degree exhibits a stronger influence to its adjacent vertices than a vertex with a small degree. So a vertex with a high degree is selected with a higher priority for color assignment. This static part of $\mathcal{F}$ is only considered for the first round of the construction-repair process ($T = 0$) when there is no learning-based information available yet.

The freedom degree $fr\_deg(v)$ is the number of adjacent vertices of vertex v which received a color. Let $K(v) = \{u \in \Gamma(v) : u \text{ is colored}\}$, then $fr\_deg(v) = |K(v)|$. Clearly, $fr\_deg(v)$ takes values in $\{0, \ldots, deg(v)\}$. For each vertex v, $fr\_deg(v)$ is initially set to 0. Then each time an adjacent vertex of v receives a color, $fr\_deg(v)$ is increased by 1. As such, $\mathcal{F}$ evolves dynamically with fr_deg to favor the coloring of those vertices which become more constrained by the coloring of its adjacent vertices. The fr_deg component of $\mathcal{F}$ is based on the following consideration. When $fr\_deg(v)$ is small relative to $deg(v)$ (say close to 0), few of its adjacent vertices are colored. As a consequence, vertex v has a large freedom in the sense that it is easy to color. In this case, vertex v will be given a low $\mathcal{F}$ value, thus a low rank. Reversely, if $fr\_deg(v)$ is close to its maximum value $deg(v)$, almost all of its neighboring vertices have already received a color. In this case, coloring vertex v is more difficult since this is strongly constrained by the colors of its adjacent vertices. Consequently, we give a high $\mathcal{F}$ value (thus a high rank) to such a vertex whose coloring becomes critical.

The feedback value $fb\_val(v)$ is used to learn from each failed construction-repair process in order to influence the rank of vertices for the next round of coloring construction process. This part is initially set to 0 for each vertex at the beginning of the whole search process. Then two types of updates are dynamically operated after each failed construction-repair process, i.e., when a dead-end is encountered, which cannot be unblocked by the subsequent tabu search repair process (see Sections 5.2.3 and 5.2.4).

– *Update of the dead-end vertex*: Suppose that v is the last vertex under consideration when the construction phase encounters a dead-end (i.e., no color can be assigned to v without violating some distance constraints). Since the dead-end involving vertex v cannot be resolved by the subsequent tabu search repair phase, we consider the underlying vertex v to be difficult or critical to color. In order to favor the selection of this vertex for color assignment for the next round of the coloring construction phase, we increase its feedback value $fb\_val(v)$ (thus its $\mathcal{F}$ rank) by a quantity $\Delta = \max_{u \in 1,\ldots,n}\{deg(u) : u \in V\}$.

– *Update of the conflicting vertices*: Suppose that c is the conflicting partial coloring after an improving or sideways (the number of conflicts is not changed) move of the tabu search repair process. We consider that the vertices that are still involved in constraint violations are difficult or critical to color. In order to favor the selection of these vertices for the next round of the construction phase, we raise their rank. Precisely, let X be the set of colored vertices in c. We first identify the set CV of conflicting vertices in c: $CV = \{v \in X : \exists u \in K(v), |c(v) - c(u)| < d(u, v)\}$ where $K(v)$ is the set of colored vertices adjacent to v. Then for each vertex v of CV, its feedback value $fb\_val(v)$ is increased by 1.

As such, if a vertex is repeatedly involved in constraint violations which are difficult to repair, its rank will progressively be augmented and the vertex will be selected for coloring with a high priority during the next round of the construction-repair process.

5.2.3 Construction phase with forward checking

The construction phase is the main component of the LHS approach responsible for generating legal k-colorings. The whole procedure of the construction phase is illustrated in Algorithm 7. During the construction phase, we maintain two sets of vertices $\mathcal{S} \subset V$ and $\mathcal{U} = V \setminus \mathcal{S}$. $\mathcal{S}$ is the current partial legal solution representing the set of vertices with their respective assigned colors while $\mathcal{U}$ contains the set of remaining vertices waiting for color assignment.

The construction phase initially starts with $\mathcal{S} = \emptyset$ an $\mathcal{U} = V$ and then iteratively extends $\mathcal{S}$ by including a new vertex v from $\mathcal{U}$ with a legal color $c(v)$ (i.e., $c(v)$ satisfies the distance constraints expressed in Eq. (2.1)). The construction phase relies on two key elements: the learning-based guiding function $\mathcal{F}$ (Section 5.2.2) and a constraint satisfaction technique called forward-checking [Haralick and Elliott 1980].

The learning-based guiding function $\mathcal{F}$ provides a dynamic order for the uncolored vertices of $\mathcal{U}$. Using $\mathcal{F}$, the construction procedure selects always the vertex with the largest $\mathcal{F}$ value (ties are broken at random). For each selected vertex, the forward-checking technique is applied to remove incompatible colors for the selected vertex v with respect to its adjacent vertices (i.e., the distance constraints). Forward-checking is an important component of the construction procedure. We explain its functioning in the rest of this section.

Let $v \in \mathcal{U}$ designate the selected vertex for color assignment, let $D(v)$ be the set of the currently available colors of v initially set to $\{1, 2, \ldots, k\}$. Two forward-checking operations are triggered to reduce $D(v)$ with the computational complexity $\Theta(k \times |\Gamma(v)|)$.

Algorithm 7 Pseudo-code of the construction phase with forward checking

Require: $\mathcal{S}$ a partial feasible coloring
Ensure: Either $\mathcal{S}$ a complete legal coloring or $(\mathcal{S}, v)$ an extended partial coloring of $\mathcal{S}$ with a dead-end vertex v

```
1: U ← V\S                    /* U is the set of uncolored vertices of graph G */
2: while U ≠ ∅ do
3:    Select a vertex v ∈ U with the largest F value (break ties randomly)
4:    D(v) ← {1, 2, ..., k} /* Initial color set of vertex v */
5:    for each κ ∈ D(v) do
6:       /* Let Γ(v) be the set of vertices adjacent to vertex v */
7:       for each μ ∈ Γ(v) do
8:          if μ is a colored adjacent vertex of v then
9:             if |κ − c(μ)| < d(v, μ) then
10:               /* Distance constraint violation, delete κ from D(v) */
11:               D(v) ← D(v) − {κ}
12:            end if
13:         else
14:            /* μ is an uncolored adjacent vertex of v */
15:            if κ + d(v, μ) > k and κ − d(v, μ) < 1 then
16:               /* Color κ for v is incompatible with an uncolored vertex μ, delete κ from D(v) */
17:               D(v) ← D(v) − {κ}
18:            end if
19:         end if
20:      end for
21:   end for
22:   if D(v) ≠ ∅ then
23:      Choose the smallest color κ ∈ D(v) and assign κ to v
24:      Extend S with v and its color κ
25:      For each uncolored vertex μ ∈ Γ(v), update its guiding function value F(μ)
26:   else
27:      /* D(v) becomes empty, a dead-end is detected, return S and the dead-end vertex v for the repair phase */
28:      return (S, v)
29:   end if
30: end while
31: return S
```

– *First forward checking operation*: This operation aims to eliminate any color from $D(v)$ which is incompatible with the *colored* vertices in $\Gamma(v)$. More precisely, for a color $\kappa \in D(v)$, if there exists an adjacent colored vertex $\mu \in \Gamma(v)$ such that $|\kappa - c(\mu)| < d(v, \mu)$, color κ can not be assigned to vertex v (due to distance constraint violation) and can be removed from $D(v)$ (See Algorithm 7, lines 8-12).

– *Second forward checking operation*: This operation aims to eliminate any color from $D(v)$ which is incompatible with the *uncolored* vertices in $\Gamma(v)$ if the color is assigned to v. More precisely, for a color $\kappa \in D(v)$, if there exists an adjacent uncolored vertex $\mu \in \Gamma(v)$ such that $\kappa + d(v, \mu) > k$ and $\kappa - d(v, \mu) < 1$, color κ cannot be assigned to vertex v (since the uncolored vertex μ has no available colors if κ is assigned to v) and can be removed from $D(v)$ (see Algorithm 7, lines 14-18).

After these forward checking operations, if the color domain $D(v)$ is not empty, the current partial solution is extended by vertex v with the smallest color of $D(v)$. Before moving to the next iteration of the construction phase, the algorithm updates the guiding function value $\mathcal{F}(\mu)$ (i.e., $fr\_deg(\mu)$) for each uncolored vertex $\mu \in \Gamma(v)$ (Algorithm 7, lines 22-25).

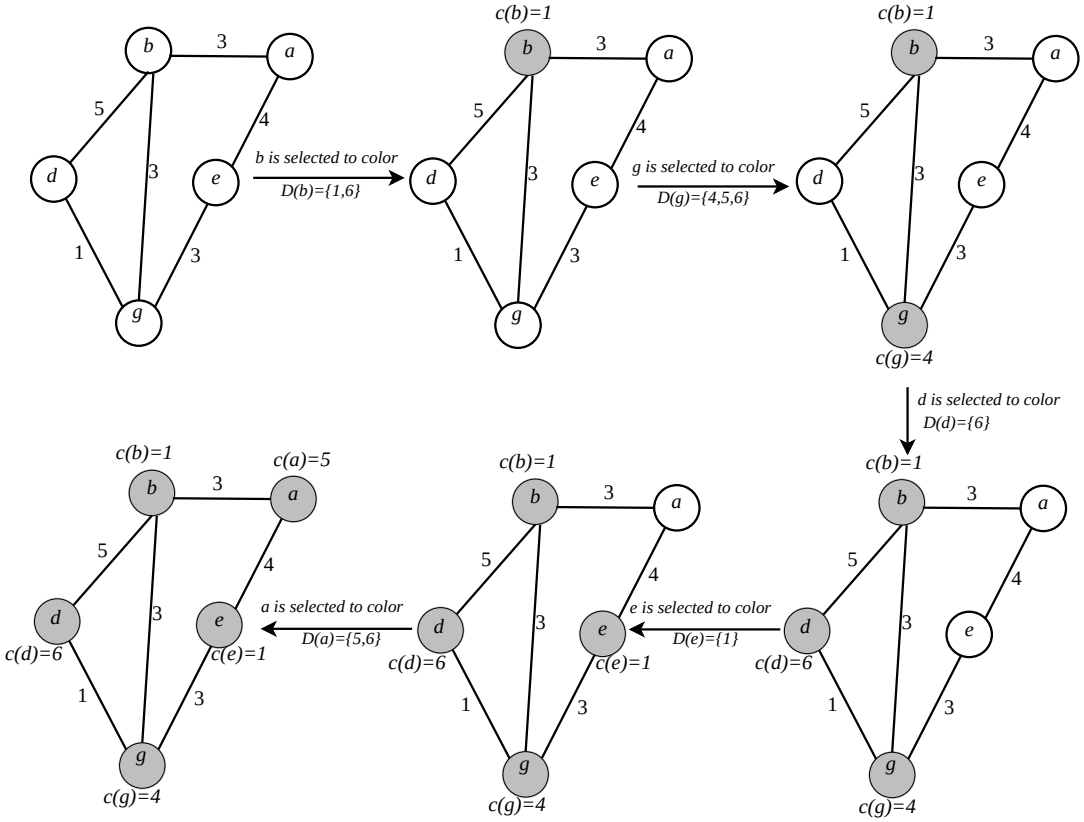


Figure 5.1: An illustrative example of the construction phase with forward checking.

On the other hand, if the color domain $D(v)$ becomes empty (i.e., no color can be assigned to v without violating some distance constraints), a dead-end is detected. At this point, the construction procedure is stopped and the search process switches to the tabu search repair procedure (Algorithm 7, lines 26-28).

Figure 5.1 illustrates the construction phase with forward checking. In the example, we use six colors (k = 6) to color a graph G with five vertices a, b, d, e and g and six edge weights. At the first construction step, the guiding function is $\mathcal{F}(a, b, d, e, g)$ = {2, 3, 2, 2, 3}, b and g are thus the vertices with the largest $\mathcal{F}$ value. Suppose a random selection between b and g gives vertex b and $D(b) = \{1, 2, 3, 4, 5, 6\}$. According to the first forward checking operation, no color can be removed from $D(b)$. Then, according to the second forward checking operation, {2, 3, 4, 5} can be removed from $D(b)$. Hence, $D(b) = \{1, 6\}$ and the smallest color 1 is chosen to color the vertex b. Then the function $\mathcal{F}\{a, b, d, e, g\}$ is updated to {3, -, 3, 2, 4}. At this point, one construction step is successfully accomplished. The next steps of the construction phase will handle the remaining vertices g, d, e and a in this order and assign them colors 4, 6, 1, and 5 respectively.

In this example, no dead-end is encountered during the construction phase. Generally, the application of forward-checking can eliminate all the colors of the vertex currently under consideration. In this case, the search switches to the repair phase for conflict resolution that we explain in the next section.

5.2.4 Tabu search repair phase

When the construction phase encounters a dead-end where the selected vertex v has no available colors (all its colors are removed by the forward-checking operations, see Section 5.2.3), LHS assigns a random color from $\{1, ..., k\}$ to vertex v and updates the guiding function value $\mathcal{F}(\mu)$ (i.e., $fr\_deg(\mu)$) for each uncolored vertex $\mu \in \Gamma(v)$. By doing this, some distance constraints are inevitably violated causing the current partial solution to be illegal. The purpose of the repair phase is then to try to transform this conflicting solution into a legal partial bandwidth coloring in order to switch back to the construction phase.

To achieve this, we develop a tabu search [Glover and Laguna 1999] repair procedure (TSRP) for the k-BCP which combines a basic tabu search procedure (TS) and a simple perturbation mechanism. The TS procedure is an adaptation of the Tabucol algorithm first introduced in [Hertz and de Werra 1987] for the conventional graph coloring problem and later improved in [Dorne and Hao 1999, Galinier and Hao 1999].

Tabu search procedure

The general scheme of the tabu search repair procedure is shown in Algorithm 8. As shown in the algorithm, TSRP alternates between the tabu search procedure (lines 4-18, Algorithm 8) and the perturbation mechanism.

Suppose that the partial illegal solution $\mathcal{S}$ is composed of L vertices $\{v^1, ..., v^L\} \subset V$. Then the TS procedure explores a subset of the space $\Omega = \{1, ..., k\}^L$ to seek a legal bandwidth coloring by using an evaluation function f, a neighborhood N and a tabu list (see Section 5.2.4). Notice that Ω contains both legal and illegal bandwidth k-colorings. The purpose of TS is to find a legal solution by making successive improvements.

From the partial illegal coloring $\mathcal{S}$, TS improves its solutions by iteratively moving from the current solution to one neighboring solution guided by the evaluation function. The best solution (in terms of the evaluation function f) is recorded in $\mathcal{S}_*$. At each iteration, TS moves from the current solution $\mathcal{S}$ to a best authorized neighboring solution, records the transition in the tabu list to prevent the search from revisiting solution $\mathcal{S}$ and possibly updates the best solution $\mathcal{S}_*$ (lines 7-15). If the best solution cannot be improved for a given number $maxIters$ of consecutive iterations, the search is considered to be trapped in a local optimum. To escape from the local optimum, TSRP triggers a perturbation mechanism (see Section 5.2.4) to modify the current solution which becomes the starting solution of the next round of the tabu search procedure (lines 19-20). The repair phase using TSRP stops either when a legal coloring is found by the tabu search procedure (lines 10-11) or after reaching a prefixed number $maxTSruns$ of the tabu search runs (or perturbations) (lines 3, 22-23).

Evaluation function, constrained neighborhood and tabu list

We next describe the key elements of the tabu search procedure, i.e., the evaluation function to measure the quality of a candidate solution (bandwidth) coloring, the neighborhood to identify the neighboring solutions that can be attained at each iteration, and the tabu list to avoid short-term cycling.

Evaluation function f: Recall that a distance $d(i, j)$ is defined for each edge $\{v_i, v_j\} \in E$ and the distance constraint states that the absolute value of the difference between the colors assigned to adjacent vertices v_i and v_j must be at least the distance $d(i, j)$. Given a partial (illegal) solution $\mathcal{S}$, we use the evaluation function defined in Eq. (5.2) [Lai and Lü 2013] to quantify the quality of $\mathcal{S}$.

$$f(\mathcal{S}) = \sum_{\{i,j\} \in E} max\{0, d(i, j) - |c(v_i) - c(v_j)|\} \tag{5.2}$$

This function basically measures the degree of constraint violations induced by a solution. Given two solutions $\mathcal{S}_1$ and $\mathcal{S}_2$, if $f(\mathcal{S}_1) < f(\mathcal{S}_2)$ (i.e., $\mathcal{S}_1$ has a smaller degree of constraint violations than $\mathcal{S}_2$), $\mathcal{S}_1$ is better than $\mathcal{S}_2$. If $f(\mathcal{S})$ equals to 0, $\mathcal{S}$ is a feasible solution.

Constrained neighborhood N: TS uses a constrained neighborhood N which can be described by the move operator $OneMove(v, i, j)$. Let $\mathcal{S}$ be a solution composed of L vertices $X = \{v^1, ..., v^L\}$. Let CV be the set of conflicting vertices such that $CV = \{v \in X$: the color of v is conflicting with the color of at least one vertex$\}$. The $OneMove$ operator changes the current color i of a conflicting vertex $v \in CV$ to

Algorithm 8 Tabu search repair procedure

Require: Graph G, color number k, partial illegal solution to be repaired $\mathcal{S}$, maximum number of launching TS $maxTSruns$
Ensure: a legal bandwidth k-coloring if found or the best solution found

```
1: S* ← S            /* S* records the best solution found so far */
2: α ← 0             /* Counts the number of launching TS */
3: while α < maxTSruns do
4:    /* Tabu search, see Section 5.2.4 */
5:    β ← 0          /* Counts consecutive iterations failing to improve S* */
6:    repeat
7:       Select one best eligible move OneMove(v, i, j)
8:       S ← S ⊕ OneMove(v, i, j)
9:       Update the tabu list
10:      if f(S) = 0, i.e., S is legal coloring then
11:         return S
12:      end if
13:      if f(S) < f(S*), i.e., S is better than S* then
14:         S* ← S; β ← 0
15:      else
16:         β ← β + 1
17:      end if
18:   until β = maxIters
19:   /* The perturbation mechanism, see Section 5.2.4 */
20:   S ← Perturbation(S*) /* The perturbed solution becomes the starting point of the next TS run */
21:   α ← α + 1
22: end while
23: return S*
```

another color j. Let $OneMove(v, i, j)$ designate such a move and $\mathcal{S} \oplus OneMove(v, i, j)$ be the resulting neighboring solution from $\mathcal{S}$. Then the neighborhood N of $\mathcal{S}$ is composed of all possible solutions that can be obtained by applying the $OneMove$ operator to $\mathcal{S}$, i.e.,

$$N(\mathcal{S}) = \{\mathcal{S} \oplus OneMove(v, i, j) \ : v \in CV\}$$

With this neighborhood, TS explores a much restricted space of $k^{|CV|}$ (instead of k^L) where CV is usually a very small subset of vertices of the current coloring $\mathcal{S}$. In order to render the neighborhood exploration as fast as possible, we adopt the incremental technique based on special data structures as explained in [Dorne and Hao 1999] to streamline the calculations.

Given this neighborhood, each iteration of TS selects the best $OneMove(v, i, j)$ operator among all the eligible candidate moves to be applied to the current solution. Ties are broken at random.

Tabu list management: The tabu list is introduced to record forbidden moves that have been performed in the recent past. Each time TS makes a move $OneMove(v, i, j)$, the pair (v, j) is added to the tabu list, meaning that it is forbidden to remove color j from vertex v for the next TT (tabu tenure) iterations. In our case, the tabu tenure TT is adaptively determined by $TT = f(\mathcal{S}) + random(10)$ [Dorne and Hao 1999, Galinier and Hao 1999] where $f(\mathcal{S})$ is the value of the evaluation function defined in Eq. (5.2) and $random(10)$ is a random integer from 1 to 10. Moreover, a forbidden move is always accepted if it leads to a neighboring solution better than the best solution found so far (*aspiration criterion*).

The perturbation mechanism

The tabu search procedure described above can be trapped in a local optimum, leading to search stagnation. When this happens, we employ a simple perturbation strategy to change (or perturb) the current solution $\mathcal{S}$. Recall that $\mathcal{S}_*$ records the best solution found so far. The perturbation procedure always replaces $\mathcal{S}$ by $\mathcal{S}_*$ and thus uses $\mathcal{S}_*$ as the new starting point of the next round of TS.

The perturbation strategy is based on the consideration that $\mathcal{S}_*$ is a high quality configuration (i.e., having few conflicts) which could be close to a proper (legal) coloring. Using $\mathcal{S}_*$ as its starting point, the tabu search procedure will explore a new search trajectory and hopefully encounters a proper coloring.

5.2.5 Discussions

In this section, we discuss the relation of our LHS algorithm with the Forward Checking Coloration Neighborhood Search (FCNS) approach [Prestwich 2008] which is probably the closest study for the BCP. Indeed, both approaches employ forward checking to build a partial color assignment. However there are three notable differences between LHS and FCNS. First, unlike LHS, the heuristic used for vertex selection in FCNS does not integrate any learning technique. Second, when a dead-end is encountered (i.e., no color can be assigned to the vertex under consideration), LHS employs a tabu search procedure to repair the conflicting coloring while FCNS just uncolors one or more vertices in a heuristic way to resolve the conflict. Third, the process of dropping an on-going construction phase provides LHS with an opportunity of learning from previous failures while FCNS does not use such similar technique.

From a more general perspective, LHS is also related to the general GRASP metaheuristic [Feo and Resende 1995] in the sense that both LHS and GRASP are composed of a construction phase and a local optimization phase. Yet, unlike GRASP which applies a local search procedure to improve a complete solution (complete color assignment in our case), the tabu search routine of LHS repairs partial (and conflicting) solutions.

As shown in the next section, the proposed LHS algorithm, equipped with its particular features, is a very competitive method for the bandwidth coloring problem.

5.3 Experimental results

5.3.1 Benchmark instances and experimental protocol

Our LHS approach was tested on two sets of 66 well-known benchmark instances (33 graphs for the BCP and 33 for the BMCP [Johnson et al. 2002]). These instances belong to three types: GEOMn, GEOMna and GEOMnb (where n denotes the number of vertices of the graph). The first type refers to sparse graphs, while the two other types correspond to dense graphs.

The LHS algorithm is coded in C++ and compiled using g++ with the '-O2' option on an Intel Xeon E5440 processor (2.83GHz and 4GB RAM). The run time required for solving the DIMACS machine benchmarks (available at: `ftp://dimacs.rutgers.edu/pub/dsj/clique/`) on our machine is 0.44, 2.63 and 9.85 seconds for graphs r300.5, r400.5 and r500.5 respectively. The computational results reported in this section were obtained with the parameter values shown in Table 5.1. Given the stochastic nature of our LHS algorithm, each problem instance is independently solved 20 times. As explained in Section 5.2, LHS solves the k-BCP problem by decreasing the k values. For the experiments reported in

Table 5.1: Settings of parameters

Parameter	Sect.	Description	Value
$maxTries$	5.2.1	Maximum number of tries	10^3
$maxIters$	5.2.4	Maximum number of non-improving moves for TS	10^4
$maxTSruns$	5.2.4	Maximum iterations of launching TS	50

Table 5.2: LHS: Detailed computational results on BCP instances

Characteristics of the graphs				k_*	LHS		
Name	n	m	d		k	SR	$t(s)$
GEOM20	20	20	0.1053	20	*21*	20/20	0.0
GEOM20a	20	37	0.1947	20	20	20/20	0.0
GEOM20b	20	32	0.1684	13	13	20/20	0.0
GEOM30	30	50	0.1149	27	*28*	20/20	0.0
GEOM30a	30	81	0.1862	27	27	20/20	0.0
GEOM30b	30	81	0.1862	26	26	20/20	0.0
GEOM40	40	78	0.1000	27	*28*	20/20	0.0
GEOM40a	40	146	0.1872	37	37	20/20	0.0
GEOM40b	40	157	0.2013	33	33	20/20	0.0
GEOM50	50	127	0.1037	28	28	20/20	0.0
GEOM50a	50	238	0.1943	50	50	20/20	0.1
GEOM50b	50	249	0.2033	35	35	20/20	1.2
GEOM60	60	185	0.1045	33	33	20/20	0.0
GEOM60a	60	339	0.1915	50	50	20/20	0.1
GEOM60b	60	366	0.2068	41	41	20/20	214.7
GEOM70	70	267	0.1106	38	38	20/20	0.0
GEOM70a	70	459	0.1901	61	61	20/20	23.7
GEOM70b	70	488	0.2020	47	47	20/20	665.4
GEOM80	80	349	0.1104	41	41	20/20	0.1
GEOM80a	80	612	0.1936	63	63	20/20	6.6
GEOM80b	80	663	0.2098	60	60	20/20	19.9
GEOM90	90	441	0.1101	46	46	20/20	0.0
GEOM90a	90	789	0.1970	63	63	20/20	23.8
GEOM90b	90	860	0.2147	69	69	20/20	779.2
GEOM100	100	547	0.1105	50	50	20/20	1.0
GEOM100a	100	992	0.2000	67	67	8/20	1557.4
					68	20/20	189.8
GEOM100b	100	1050	0.2121	72	**71**	12/20	2038.6
					72	20/20	759.6
GEOM110	110	638	0.1064	50	50	20/20	1.3
GEOM110a	110	1207	0.2013	71(72)	71	19/20	2218.7
					72	20/20	324.2
GEOM110b	110	1256	0.2095	78	**77**	10/20	2598.7
					78	20/20	94.3
GEOM120	120	773	0.1083	59	59	20/20	0.5
GEOM120a	120	1434	0.2000	82	82	20/20	171.1
GEOM120b	120	1491	0.2088	84	84	2/20	3568.1
					85	20/20	1829.7

this chapter, we set the initial value of k to be the best-known (i.e., the smallest) value k_* from the literature for all the graphs.

5.3.2 Bandwith coloring: Computational results

This section is dedicated to an evaluation of the LHS's performance for the bandwidth coloring problem using the 33 BCP benchmark graphs. Columns 1–4 in Table 5.2 present the characteristics of the tested graphs and column k_* gives the current best-known results reported in the published literature and the unpublished paper [Lai et al. 2014]. The current best-known k_* reported in the published literature is also given in parentheses. Columns 6–8 present detailed computational results of our LHS algorithm: The best result in terms of the number of colors (k), the success rate (SR, number of runs over 20 to attain the best result k) and the average running time to reach k (t, in seconds).

Table 5.3: Comparisons with four state-of-the-art algorithms on BCP instances

Graph		FCNS		EA		MITS		PR		LHS	
Name	k_*	k	$t(s)$	k	$t(s)$	k	$t(s)$	k	$t(s)$	k	$t(s)$
GEOM20	20	21	0.0	21	0.0	-	-	21	0.0	21	0.0
GEOM20a	20	*20*	0.0	*20*	0.0	*20*	0.0	*20*	0.0	*20*	0.0
GEOM20b	13	*13*	0.0	*13*	0.0	*13*	0.0	*13*	0.0	*13*	0.0
GEOM30	27	28	0.0	28	0.0	-	-	28	0.0	28	0.0
GEOM30a	27	*27*	0.0	*27*	0.0	*27*	0.0	*27*	0.0	*27*	0.0
GEOM30b	26	*26*	0.0	*26*	0.0	*26*	0.0	*26*	0.0	*26*	0.0
GEOM40	27	28	0.0	28	0.0	-	-	28	0.0	28	0.0
GEOM40a	37	*37*	0.0	*37*	0.0	*37*	0.0	*37*	0.0	*37*	0.0
GEOM40b	33	*33*	0.0	*33*	0.0	*33*	0.0	*33*	0.0	*33*	0.0
GEOM50	28	*28*	0.0	*28*	0.0	*28*	0.0	*28*	0.0	*28*	0.0
GEOM50a	50	*50*	2.0	*50*	0.0	*50*	0.0	*50*	0.0	*50*	0.1
GEOM50b	35	*35*	0.0	*35*	0.0	*35*	3.0	*35*	1.0	*35*	1.2
GEOM60	33	*33*	0.0	*33*	0.0	*33*	0.0	*33*	0.0	*33*	0.0
GEOM60a	50	*50*	1.0	*50*	0.0	*50*	1.0	*50*	0.0	*50*	0.1
GEOM60b	41	43	0.0	*41*	29.0	*41*	277.0	*41*	105.0	*41*	214.7
GEOM70	38	*38*	0.0	*38*	0.0	*38*	0.0	*38*	0.0	*38*	0.0
GEOM70a	61	62	2.0	*61*	12.0	*61*	45.0	*61*	47.0	*61*	23.7
GEOM70b	47	48	1.0	48	52.0	*47*	8685.0	*47*	6678.0	*47*	665.4
GEOM80	41	*41*	0.0	*41*	0.0	*41*	0.0	*41*	0.0	*41*	0.1
GEOM80a	63	*63*	12.0	*63*	150.0	*63*	21.0	*63*	12.0	*63*	6.6
GEOM80b	60	61	0.0	*60*	145.0	*60*	322.0	*60*	191.0	*60*	19.9
GEOM90	46	*46*	3.0	*46*	0.0	*46*	0.0	*46*	0.0	*46*	0.0
GEOM90a	63	64	2.0	*63*	150.0	*63*	230.0	*63*	191.0	*63*	23.8
GEOM90b	69	72	2.0	70	1031.0	*69*	20144.0	*69*	23560.0	*69*	779.2
GEOM100	50	*50*	0.0	*50*	2.0	*50*	2.0	*50*	2.0	*50*	1.0
GEOM100a	67	68	9.0	68	273.0	*67*	11407.0	*67*	5556.0	*67*	1557.4
										68	189.8
GEOM100b	72	73	15.0	73	597.0	*72*	24561.0	*72*	41832.0	**71**	2038.6
										72	759.6
GEOM110	50	*50*	4.0	*50*	3.0	*50*	2.0	*50*	5.0	*50*	1.3
GEOM110a	71(72)	73	7.0	72	171.0	72	1529.0	*71*	5140.0	*71*	2218.7
										72	324.2
GEOM110b	78	79	2.0	*78*	676.0	*78*	24416.0	*78*	18136.0	**77**	2598.7
										78	94.3
GEOM120	59	60	4.0	*59*	0.0	*59*	1.0	*59*	2.0	*59*	0.5
GEOM120a	82	84	4.0	84	614.0	*82*	34176.0	*82*	62876.0	*82*	171.1
GEOM120b	84	86	9.0	*84*	857.0	-	-	85	66301.0	*84*	3568.1
										85	1829.7

From Table 5.2, one observes that except for three small instances (indicated in italics), LHS can match the best-known results of the other 30 instances. Remarkably, LHS is able to improve the current best-known result for two hard instances (GEOM100b and GEOM110b indicated in bold). Furthermore, LHS achieves robust results with a success rate SR = 20/20 except for five cases (GEOM100a, GEOM100b, GEOM110a, GEOM110b and GEOM120b). Besides, Table 5.2 also lists the k value of these five graphs when SR = 20/20. The average running time of LHS ranges from 0 second to 1 hour. Each computing time corresponds to the average time for LHS to reach a legal coloring with the k value indicated in the table.

In order to further evaluate our LHS method, we compare our results with those obtained by four best performing algorithms in the literature: Forward checking colouration neighbourhood search (FCNS) [Prestwich 2008], evolutionary algorithm (EA) [Malaguti and Toth 2008], multistart iterated tabu search (MITS) [Lai and Lü 2013] and path relinking (PR) [Lai et al. 2014]. For this purpose, we restrict our attention to solution quality, i.e., the smallest k used to obtain a legal k-coloring. Computing times are included only for indicative purposes since there is no sense to compare the computing times of two methods if they achieve colorings with different k values. As one can observe in Table 5.3, there are many such cases. Indeed, it is generally more difficult to find a legal k-coloring than a legal $(k+1)$-coloring. This is particularly true when k is close to the best-known value k_* (see for instance the cases GEOM100a, GEOM100b, GEOM110a and GEOM110b in Table 5.3). Finally, the experimental platforms used by the reference algorithms are as follows: A 733MHz Pentium III PC for FCNS, a PIV 2.4MHz computer with 512 MB RAM for EA and a 2.8GHz computer with 4GB RAM for MITS and PR.

Table 5.3 presents the comparative results of LHS and the four reference methods (FCNS, EA, MITS and PR). The "-" marks for the reference MITS algorithm in Table 5.3 mean that MITS fails to reach the best-known result for the tested graph and the best obtained k is not reported for these graphs. From Table 5.3, one observes that the reference algorithms can achieve the best reported k_* for 17, 24, 28 and 29 instances respectively, while LHS achieves the best-known results for 30 instances. Table 5.3 also discloses that LHS obtains no worse results than FCNS, EA, MITS and PR. More importantly, LHS can improve the best-known results in the literature for two instances (entries in bold). Finally, to find a legal coloring with the same k value, LHS requires comparable computing times with respect to FCNS and EA, and shorter times than MITS and PR for many cases. These outcomes provide evidence of the efficacy of our LHS approach for the BCP.

5.3.3 Bandwith multicoloring: Computational results

We turn now our attention to an evaluation of the LHS algorithm on the bandwidth multicoloring problem using the set of 33 "GEOM" benchmark instances for the BMCP. LHS is designed for solving the BCP, we do not adjust the LHS algorithm but transform each instance of the BMCP to the corresponding instance of the BCP. We split each vertex v_i into a clique of cardinality $p(i)$ (each vertex v_i receives a subset $S(i) \subset \{1, 2, ..., k\}$ of $p(i)$ different colors), with each edge of the clique having distance $d(i, i)$, corresponding to the distance of the loop edge of vertex v_i in the original graph. As a result, the new graph has $\sum_{i=1,...,n} p(i)$ vertices [Lim et al. 2005, Malaguti and Toth 2008].

In this section, we first present our detailed computational results, and then show a comparison between LHS and the five state-of-the-art algorithms OF-SW [Chiarandini and Stützle 2007], FCNS [Prestwich 2008], EA [Malaguti and Toth 2008], MITS [Lai and Lü 2013] and PR[Lai et al. 2014].

Table 5.4 shows the detailed characteristics of each graph, the best-known result k_* (the best-known k_* in the published literature is also given in parentheses) and the result of our LHS approach. In addition to the best colors obtained (k) with the average running time to reach k (t in seconds), we also provide the success rate (SR) of LHS for attaining the best result k. Furthermore, we list the k value for each graph when LHS could achieve the robust results with a SR = 20/20. Table 5.5 reports the comparative results between LHS and the five reference algorithms. The experimental platform used by OF-SW algorithm is a 2GHz AMD Athlon MP 2400+ processor with 256 KB cache and 1 GB RAM, and the platforms of the other four reference algorithms are the same as given in Section 5.3.2.

From the results in Table 5.4, we observe that our LHS approach can match the best-known k_* for all 33 instances. More importantly, LHS finds an improved best result for 12 out of 33 instances (entries in bold). This is remarkable given that 14 of these 33 best-known results were reported very recently in the unpublished work [Lai et al. 2014]. In particular, LHS can consistently achieve the best-known k_* in the literature with a perfect success rate for all graphs except GEOM80a, GEOM90a and GEOM110b. Besides, the indicated computing time corresponds to the average time for LHS to reach a legal coloring with the k value indicated in the table. From Table 5.4, one also observes that the running time increases when k decreases since this makes the problem more difficult to solve.

Table 5.5 lists the comparative results of LHS and the five reference algorithms (OF-SW, FCNS, EA, MITS and PR). The "-" marks for the OF-SW algorithm mean that no result is available for the concerned graphs. From Table 5.5, one observes that OF-SW, FCNS, EA and MITS reach the best-known results for 6, 8, 13 and 18 cases respectively. While both the unpublished PR algorithm and our LHS algorithm attain the best-known results for all 33 instances (in italics), LHS requires a much shorter computing time for most instances. More importantly, our LHS algorithm can find an improved best solution for 12 instances (in bold).

Table 5.4: Detailed computational results of LHS on the set of 33 BMCP instances

Characteristics of the graphs				k_*	LHS		
Name	n	m	d		k	SR	$t(s)$
GEOM20	118	1048	0.1518	149	149	20/20	1.8
GEOM20a	100	1327	0.2681	169	169	20/20	0.5
GEOM20b	40	132	0.1692	44	44	20/20	0.0
GEOM30	143	1419	0.1398	160	160	20/20	0.1
GEOM30a	171	3288	0.2262	209	209	20/20	16.2
GEOM30b	69	447	0.1905	77	77	20/20	0.0
GEOM40	220	3074	0.1276	167	167	20/20	0.2
GEOM40a	203	4473	0.2182	213	213	20/20	9.0
GEOM40b	84	743	0.2131	74	74	20/20	1.5
GEOM50	285	4935	0.1219	224	224	20/20	0.3
GEOM50a	302	9649	0.2123	312(314)	**311**	20/20	1452.6
					312	20/20	307.5
GEOM50b	104	1140	0.2128	83	83	20/20	72.1
GEOM60	315	6174	0.1248	258	258	20/20	1.3
GEOM60a	362	13105	0.2005	354(356)	**353**	2/20	9007.1
					354	20/20	4996.0
GEOM60b	127	1785	0.2231	113	113	20/20	910.7
GEOM70	384	8584	0.1167	266(270)	266	20/20	2534.0
GEOM70a	379	14821	0.2069	466(467)	**465**	6/20	36604.9
					466	20/20	12622.1
GEOM70b	148	2212	0.2033	116	**115**	3/20	3640.7
					116	20/20	1844.7
GEOM80	465	12927	0.1198	380(381)	**379**	20/20	357.8
					380	20/20	164.0
GEOM80a	389	15545	0.2060	358(361)	**357**	2/20	43403.0
					358	7/20	25852.0
					360	20/20	13302.9
GEOM80b	169	3028	0.2133	138(139)	138	20/20	46.5
GEOM90	530	16180	0.1154	328(330)	328	20/20	162.2
GEOM90a	454	20455	0.1989	372(375)	372	3/20	16782.1
					373	20/20	3164.5
GEOM90b	184	3602	0.2139	144	**142**	7/20	7680.8
					143	20/20	4858.5
					144	20/20	1331.8
GEOM100	581	19829	0.1177	404	404	20/20	64.9
GEOM100a	528	28496	0.2048	436(442)	**429**	1/20	78363.1
					434	20/20	10767.1
					436	20/20	2310.7
GEOM100b	200	4429	0.2226	156	**153**	1/20	10840.1
					155	20/20	3147.6
					156	20/20	726.3
GEOM110	643	24799	0.1201	375(381)	375	20/20	1598.8
GEOM110a	602	38783	0.2144	482(488)	**478**	1/20	49457.1
					480	20/20	2921.5
					482	20/20	375.3
GEOM110b	220	5163	0.2143	201(204)	201	3/20	5388.4
					203	20/20	303.1
GEOM120	680	27759	0.1202	396	396	20/20	626.1
GEOM120a	664	46429	0.2109	539(554)	**536**	2/20	69518.6
					539	20/20	20286.1
GEOM120b	235	5779	0.2102	189	**187**	8/20	8025.8
					188	20/20	1642.0
					189	20/20	315.2

Table 5.5: Comparisons of LHS with five state-of-the-art algorithms on the set of 33 BMCP instances

Graph		OF-SW		FCNS		EA		MITS		PR		LHS	
Name	k_*	k	$t(s)$	k	$t(s)$	k	$t(s)$	k	$t(s)$	k	$t(s)$	k	$t(s)$
GEOM20	149	-	-	*149*	4.0	*149*	18.0	*149*	2.0	*149*	1.0	*149*	1.8
GEOM20a	169	-	-	170	2.0	*169*	9.0	*169*	15.0	*169*	7.0	*169*	0.5
GEOM20b	44	*44*	30.0	*44*	0.0	*44*	5.0	*44*	0.0	*44*	0.0	*44*	0.0
GEOM30	160	-	-	*160*	0.0	*160*	1.0	*160*	0.0	*160*	0.0	*160*	0.1
GEOM30a	209	*209*	380.0	214	11.0	210	954.0	*209*	10.0	*209*	26.0	*209*	16.2
GEOM30b	77	*77*	80.0	*77*	0.0	*77*	0.0	*77*	0.0	*77*	0.0	*77*	0.0
GEOM40	167	-	-	*167*	1.0	*167*	20.0	*167*	0.0	*167*	1.0	*167*	0.2
GEOM40a	213	214	500.0	217	299.0	214	393.0	*213*	328.0	*213*	133.0	*213*	9.0
GEOM40b	74	*74*	140.0	*74*	4.0	*74*	1.0	*74*	2.0	*74*	4.0	*74*	1.5
GEOM50	224	-	-	*224*	1.0	*224*	1197.0	*224*	8.0	*224*	2.0	*224*	0.3
GEOM50a	312(314)	315	1080.0	323	51.0	316	4675.0	314	40373.0	*312*	270860.0	**311**	1452.6
												312	307.5
GEOM50b	83	84	200.0	86	1.0	*83*	197.0	*83*	1200.0	*83*	723.0	*83*	72.1
GEOM60	258	*258*	710.0	*258*	77.0	*258*	139.0	*258*	19.0	*258*	23.0	*258*	1.3
GEOM60a	354(356)	356	1420.0	373	10.0	357	8706.0	356	38570.0	*354*	34580.0	**353**	9007.1
												354	4996.0
GEOM60b	113	117	300.0	116	12.0	115	460.0	*113*	104711.0	*113*	63579.0	*113*	910.7
GEOM70	266(270)	267	1060.0	277	641.0	272	1413.0	270	7602.0	*266*	130844.0	*266*	2534.0
GEOM70a	466(467)	478	1470.0	482	315.0	473	988.0	467	38759.0	*466*	6952.0	**465**	36604.9
												466	12622.1
GEOM70b	116	120	380.0	119	55.0	117	897.0	*116*	213545.0	*116*	26110.0	**115**	3640.7
												116	1844.7
GEOM80	380(381)	382	1490.0	398	361.0	388	132.0	381	212213.0	*380*	34493.0	**379**	357.8
												380	164.0
GEOM80a	358(361)	360	1510.0	380	109.0	363	8583.0	361	41235.0	*358*	41772.0	**357**	43403.0
												360	13302.9
GEOM80b	138(139)	139	490.0	141	37.0	141	1856.0	139	255.0	*138*	705.0	*138*	46.5
GEOM90	328(330)	334	1810.0	339	44.0	332	4160.0	330	4022.0	*328*	134941.0	*328*	162.2
GEOM90a	372(375)	377	1910.0	382	13.0	382	5334.0	375	10427.0	*372*	282456.0	*372*	16782.1
												373	3164.5
GEOM90b	144	147	590.0	147	303.0	*144*	1750.0	*144*	211366.0	*144*	14648.0	**142**	7680.8
												144	1331.8
GEOM100	404	*404*	2170.0	424	7.0	410	3283.0	*404*	40121.0	*404*	16355.0	*404*	64.9
GEOM100a	436(442)	437	2500.0	461	26.0	444	12526.0	442	381.0	*436*	9108.0	**429**	78363.1
												436	2310.7
GEOM100b	156	159	690.0	159	367.0	*156*	3699.0	*156*	213949.0	*156*	86308.0	**153**	10840.1
												156	726.3
GEOM110	375(381)	378	2510.0	392	43.0	383	2344.0	381	183.0	*375*	25401.0	*375*	1598.8
GEOM110a	482(488)	490	3120.0	500	29.0	490	2318.0	488	926.0	*482*	9819.0	**478**	49457.1
												482	375.3
GEOM110b	201(204)	208	790.0	208	5.0	206	480.0	204	944.0	*201*	47653.0	*201*	5388.4
												203	303.1
GEOM120	396	397	2730.0	417	9.0	*396*	2867.0	-	-	*396*	15341.0	*396*	626.1
GEOM120a	539(554)	549	3690.0	565	41.0	559	3873.0	554	1018.0	*539*	45147.0	**536**	69518.6
												539	20286.1
GEOM120b	189	191	910.0	196	3.0	191	3292.0	*189*	213989.0	*189*	14371.0	**187**	8025.8
												189	315.2

Table 5.6: Assessment of the learning-based guiding function

Graph		LHS$_{random}$			LHS		
Name	k_*	k	SR	$t(s)$	k	SR	$t(s)$
GEOM50	224	*224*	20/20	0.3	*224*	20/20	0.3
GEOM50a	312(314)	313	2/20	28288.6	**311**	20/20	1452.6
GEOM50b	83	*83*	16/20	3492.4	*83*	20/20	72.1
GEOM60	258	*258*	20/20	1.6	*258*	20/20	1.3
GEOM60a	354(356)	*354*	7/20	12349.2	**353**	2/20	9007.1
GEOM60b	113	*113*	1/20	3549.1	*113*	20/20	910.7
GEOM70	266(270)	*266*	7/20	14235.4	*266*	20/20	2534.0
GEOM70a	466(467)	*466*	9/20	30681.1	**465**	6/20	36604.9
GEOM70b	116	117	3/20	7880.7	**115**	3/20	3640.7
GEOM80	380(381)	**379**	6/20	18108.5	**379**	20/20	357.8
GEOM80a	358(361)	360	1/20	64551.1	**357**	2/20	43403.0
GEOM80b	138(139)	*138*	20/20	529.6	*138*	20/20	46.5
GEOM90	328(330)	*328*	3/20	15547.1	*328*	20/20	162.2
GEOM90a	372(375)	373	3/20	26154.4	*372*	3/20	16782.1
GEOM90b	144	145	3/20	11794.4	**142**	7/20	7680.8
GEOM100	404	*404*	20/20	302.8	*404*	20/20	64.9
GEOM100a	436(442)	437	3/20	45299.8	**429**	1/20	78363.1
GEOM100b	156	159	4/20	7565.8	**153**	1/20	10840.1
GEOM110	375(381)	*375*	3/20	32435.4	*375*	20/20	1598.8
GEOM110a	482(488)	**481**	6/20	24425.6	**478**	1/20	49457.1
GEOM110b	201(204)	202	1/20	15397.1	*201*	3/20	5388.4
GEOM120	396	*396*	20/20	14012.6	*396*	20/20	626.1
GEOM120a	539(554)	542	1/20	76969.1	**536**	2/20	69518.6
GEOM120b	189	190	2/20	12950.6	**187**	8/20	8025.8

Once again, we do not emphasize on computing time since the compared approaches provide results with different k values. The computing times of FCNS [Prestwich 2008] are much shorter, but its results are much worse in terms of solution quality. To find solutions of the same quality (with the same k), LHS does not consume more time than OF-SW [Chiarandini and Stützle 2007] and EA [Malaguti and Toth 2008]. Compared to the most recent and the two best performing algorithms MITS [Lai and Lü 2013] and PR [Lai et al. 2014], LHS requires less computing times to find equal or better solutions.

In summary, LHS competes very favorably with the five high performing reference algorithms in the literature for the BMCP.

5.4 Analysis of LHS

In this section, we perform an additional experiment to assess the impact of the learning-based guiding function $\mathcal{F}$ defined in Eq. (5.1) (Section 5.2.2) which is a key element of the LHS approach.

Indeed, as explained in Section 5.2.1, the construction phase uses the guiding function $\mathcal{F}$ to decide the next vertex for color assignment. This experiment aims to show its influence to the performance of LHS. For this purpose, we compare LHS (with $\mathcal{F}$ of Eq. (5.1)) and a LHS variant called LHS$_{random}$. LHS$_{random}$ discards the guiding function $\mathcal{F}$ and selects randomly vertices for color assignment.

For this experiment, we focus on 24 most difficult and challenging benchmark instances for the BMCP. With the same experimental protocol, we run 20 times each of these two LHS procedures (LHS and LHS$_{random}$) to solve the 24 BMCP benchmark instances. The computational outcomes are reported in Table 5.6.

From Table 5.6, one observes that LHS achieves always a better or equal result compared to LHS$_{random}$. In particular, the result of LHS is better for 15 out of the 24 instances, i.e., LHS requires a smaller number of colors to find a legal coloring, with a color reduction ranging from 1 up to 8. Given that finding a legal k-coloring with k close to the best-known value k^* is already a difficult task, the improvement of LHS over

LHS_{random} is significant. Moreover, LHS requires less average time to reach its best solutions compared to LHS_{random}. In summary, discarding the guiding function $\mathcal{F}$ makes LHS less effective and weakens its performance.

Finally, one notices that even the weakened LHS_{random} procedure is competitive compared with the best existing methods since LHS_{random} is able to match the best-known results in most cases and even finds two improved best results (indicated in bold). Thus this experiment indirectly shows the interest of the general cooperative approach of LHS which combines a construction procedure and a repair procedure.

5.5 Conclusion

In this chapter, we presented the learning-based hybrid search approach for the bandwidth coloring problem (BCP) and the bandwidth multicoloring problem (BMCP), which are two important generalizations of the vertex coloring problem. LHS alternates between an informed construction phase and a repair procedure until attaining a feasible solution. The construction phase is guided by a learning-based function to choose the next vertex for color assignment and applies a forward checking technique to eliminate incompatible colors for unassigned vertices. The tabu search based repair procedure is used to resolve dead-end situations when the construction phase cannot further extend the current partial solution.

Experimental evaluations on two sets of 66 benchmark instances showed that the proposed LHS approach is highly competitive in comparison with the current most effective algorithms for the BCP and BMCP. LHS can reach the best-known results for most benchmarks of both BCP and BMCP. In particular, LHS improves the best-known results for two BCP instances and 12 BMCP instances.

Finally, the general LHS algorithm follows a new framework which is different from the existing approaches. In the future, we hope to investigate its usefulness for solving other constrained combinatorial problems.

IV

General conclusion

6

Conclusions and perspectives

6.1 Conclusions

This thesis focuses on designing effective hybrid metaheuristic algorithms for solving three NP-hard generalizations of the vertex coloring problem, i.e., the minimum sum coloring problem (MSCP), the bandwidth coloring problem (BCP) and the bandwidth multicoloring problem (BMCP). Considering the theoretical intractability and the widespread real world applications of these problems, we studied several solution approaches to find high quality suboptimal solutions in acceptable computing time. The resulting algorithms are evaluated on a number of well-known benchmark instances and shown to be highly competitive in comparison with the best performing algorithms in the literature.

After providing an overview of the most representative algorithms proposed in the literature as well as the well-known benchmark instances for the studied coloring problems, we presented two hybrid algorithms for solving the minimum sum coloring problem in Chapter 3 and 4. The hybrid algorithms follow the general framework which combines the population-based evolutionary search and local optimization procedure. These two algorithms are effective in solving the MSCP and highly competitive with other state-of-the-art algorithms in the literature. Moreover, we can make the following conclusions. First, the quality of the initial population influences the performance of the hybrid algorithms especially on hard and large instances. Second, crossover operator is an important ingredient for the hybrid metaheuristics and the idea of the partition crossover is still useful for the MSCP. However, the crossover operator for the MSCP needs to be designed differently from the well-known GPX crossover since the objective function of the MSCP is different from that of the vertex coloring problem. Third, the landscape analysis on a number of instances for the MSCP demonstrates why some instances are more difficult than others.

In Chapter 5, we proposed a learning-based hybrid search approach (LHS) for the bandwidth coloring problem and the bandwidth multicoloring problem. LHS alternates between an informed construction phase and a repair procedure until attaining a feasible solution. The construction phase is guided by a learning-based function to choose the next vertex for color assignment and applies a forward checking technique to eliminate incompatible colors for unassigned vertices. The tabu search based repair procedure is used to resolve dead-end situations when the construction phase cannot further extend the current partial solution. Experimental evaluations on two sets of 66 benchmark instances showed that the proposed LHS approach is highly competitive in comparison with the current most effective algorithms for the BCP and BMCP. Moreover, we can make the following observations. First, this framework is general, simple and intuitive.

Second, this framework follows the constructive approach while incorporating local optimization to eliminate conflicts. Third, the learning technique plays an important role in this hybrid algorithm which needs to be carefully designed.

Additionally, we presented a general and unified swap-based tabu search algorithm (SBTS) for solving the maximum independent set problem, which is used to generate initial solutions in the hybrid search algorithm for the MSCP. SBTS explores the search space by a dynamic alternation between intensification and diversification steps. The search process is driven by the $(k, 1)$-swap operator combined with specific rules to examine four different neighborhoods. For the purpose of intensification, SBTS uses (0,1)-swap to improve the solution and (1,1)-swap to make side-walks with specific selection rules. To overcome local optima, SBTS adopts an adaptive perturbation strategy which applies either a (2,1)-swap for a weak perturbation or a $(k, 1)$-swap (k > 2) for a strong perturbation. A tabu mechanism is also employed to prevent the search from short-term cycles. We tested the proposed algorithm on two sets of 120 well-known instances (DIMACS and BHOSLIB) with multiple topologies and densities. Computational results show that SBTS competes favorably with 5 state-of-the-art algorithms in the literature.

6.2 Perspectives

The hybrid approaches seem highly promising given their excellent performance on the studied generalizations of the vertex coloring problem. However, several interesting ideas can be considered in future studies, which may reinforce the performance of the hybrid approaches.

First, concerning the hybrid algorithms for the MSCP, we found that the independent set extraction is effective for large and hard problem instances. One immediate extension is to design a combination of the EXSCOL algorithm [Wu and Hao 2012] which applies the independent set extraction and our hybrid memetic algorithm to benefit from their respective advantages. Another alternative is to design a local search to dynamically exploit the extraction and expansion of independent sets to construct a set of good initial solutions, and then use our hybrid search algorithm to improve these solutions.

Second, the hybrid search algorithm LHS for solving the BCP and BMCP establishes an original cooperative framework between an informed construction approach and a local search approach. We hope to adapt this LHS algorithm to other constrained combinatorial problems in order to investigate its usefulness. However, LHS drops all the colored vertices when it fails to find a legal coloring and starts from scratch although some information is learned but still some computing time is wasted. One possibility is to design a strategy to uncolor a subset of colored vertices instead of dropping all the colored vertices. Another possibility is to propose a memetic algorithm which combines the population-based evolutionary search and the LHS optimization procedure.

Finally, the proposed SBTS algorithm for the maximum independent set problem achieves highly competitive results on the well-known benchmarks, but the best results of some hard graphs are attained occasionally. More studies are needed to improve the stability and search capacity of the approach. One possibility is to collect some information during the search process and then use the learned information to guild the search trajectory. Another possibility is to develop a meaningful solution recombination mechanism that can be used within a population-based hybrid algorithm.

Appendix

7

SBTS: Swap-Based Tabu Search for maximum independent set

This Appendix presents a general swap-based tabu search (SBTS) heuristic for the maximum independent set problem (MIS), which is used to generate initial solutions in the hybrid search algorithm for the minimum sum coloring problem (see Chapter 4). Given a graph G, the MIS aims to determine a subset $S \subseteq V$ of maximum cardinality such that no two vertices of S are adjacent. SBTS integrates distinguished features including a general and unified $(k, 1)$-swap operator, four constrained neighborhoods and specific rules for neighborhood exploration. Extensive evaluations on two popular benchmarks (DIMACS and BHOSLIB) of 120 instances show that SBTS attains the best-known results for all the instances with very different structures and topologies. The best-known results are also attained on an additional set of 11 real instances from code theory. The content of this Appendix is published in [Jin and Hao 2015b].

Contents

7.1 Introduction

Given a simple undirected graph $G = (V, E)$ with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E \subset V \times V$, an independent set S is a subset of V such that no two vertices are adjacent, i.e., $\forall v_i, v_j \in S, \{v_i, v_j\} \notin E$. An independent set is said maximum if it has the largest cardinality among all the independent sets of G. The maximum independent set problem (MIS) is to determine a maximum independent set of an arbitrary graph. As one of Karp's 21 NP-complete problems [Karp 1972], MIS is among the most popular problems in combinatorial optimization [Johnson and Garey 1979, Johnson and Trick 1996].

In graph theory, there are two tightly related problems: The maximum clique problem (MC) and minimum vertex cover problem (MVC). A clique C of G is a subset of V such that all vertices in C are pairwise adjacent, i.e., $\forall v_i, v_j \in C, \{v_i, v_j\} \in E$. MC is to find a clique C of maximum cardinality. A vertex cover VC of G is a subset of V such that each edge of E is incident to at least one vertex of VC, i.e., $\forall \{v_i, v_j\} \in E, v_i \in VC \vee v_j \in VC$. MVC is to determine a vertex cover of minimum cardinality.

Let $\bar{G} = (V, \bar{E})$ be the complementary graph of $G = (V, E)$ such that $\bar{E} \subset V \times V$ and $\forall v_i, v_j \in V, \{v_i, v_j\} \in \bar{E}$ if and only if $\{v_i, v_j\} \notin E$. Then given a subset S of V, the following three statements are equivalent: S is an independent set in G, $V \backslash S$ is a vertex cover in G and S is a clique in $\bar{G}$. As a consequence, MIS, MC and MVC are three equivalent problems such that any algorithm designed for one of these problems can be directly applied to solve the other two problems. These problems are relevant to a wide variety of applications such as code theory, information retrieval, signal transmission, classification theory, experimental design and many more others [Bomze et al. 1999, Johnson and Trick 1996, Wu and Hao 2015]. In this work, we focus on studying the MIS problem.

During the past decades, a large number of solution procedures for solving MIS, MC and MVC have been reported in the literature. Among them are several exact algorithms based on the general branch-and-bound framework [Carraghan and Pardalos 1990, Li and Quan 2010, Östergård 2002, San Segundo et al. 2011, Tomita and Kameda 2007]. These exact methods are applicable to problem instances of limited sizes. For larger cases, various heuristics have been proposed to obtain near-optimal solutions. The most representative heuristics include tabu search [Battiti and Protasi 2001, Friden et al. 1989, Wu et al. 2012, Wu and Hao 2013a], stochastic local search [Andrade et al. 2012, Katayama et al. 2005, Pullan 2006; 2008], parallel hyper-heuristics mixing several low-level heuristics [Pullan et al. 2011]), simulated annealing [Geng et al. 2007], variable neighborhood search [Hansen et al. 2004], breakout local search [Benlic and Hao 2013], local search with edge weighting [Cai et al. 2014, Richter et al. 2007] and evolutionary algorithms [Brunato and Battiti 2011, Zhang et al. 2005]. According to the reported results on benchmark instances, in particular those of the well-known Second DIMACS Implementation Challenge on Cliques, Coloring, and Satisfiability [Johnson and Trick 1996], it seems that ILS and GLP [Andrade et al. 2012], BLS [Benlic and Hao 2013], NuMVC [Cai et al. 2014], PLS [Pullan 2006; 2008], CLS [Pullan et al. 2011], COVER [Richter et al. 2007], MN/TS [Wu et al. 2012] and AMTS [Wu and Hao 2013a] are among the top performing heuristics in the literature. Nevertheless, due to the large variety of structures of these instances (they are random graphs or transformed from different real problems), no single approach can attain the best-known results for all the DIMACS instances. For more information, the reader can refer to [Wu and Hao 2015] which provides an updated and comprehensive review on both exact and heuristic MC algorithms, with a special focus on recent developments.

In this work, we introduce a general Swap-Based Tabu Search (SBTS) heuristic for the maximum independent set problem. SBTS inspects the search space by a dynamic alternation between intensification

and diversification steps [Glover and Laguna 1999, Lourenço et al. 2001, Schrimpf et al. 2000]. The search process is driven by a general and unified $(k, 1)$-swap ($k \geq 0$) operator combined with specific rules to explore four constrained neighborhoods. Given an independent set S, $(k, 1)$-swap exchanges one vertex (which is strategically selected) in $V \backslash S$ against its k adjacent vertices in S. For the purpose of intensification, SBTS uses (0,1)-swap to improve the solution and (1,1)-swap to make side-walks with the help of specific selection rules. To overcome local optima, SBTS adopts an adaptive perturbation strategy which applies either a (2,1)-swap for a weak perturbation or a (k,1)-swap ($k > 2$) for a strong perturbation. A tabu mechanism is also employed to prevent the search from short-term cycles. Compared with existing local search algorithms, SBTS distinguishes itself by its unified $(k, 1)$-swap operator, its specific neighborhoods and its dedicated rules for neighborhood exploration.

The proposed SBTS algorithm attains the best-known results for all 120 instances of the well-known DIMACS and BHOSLIB benchmarks with very different structures and topologies. This is the first time a single heuristic reaches such a performance. The best-known results are also attained on an additional set of 11 real instances from code theory.

7.2 Components of the SBTS approach

Our Swap-Based Tabu Search (SBTS) algorithm for MIS follows the iterated local search framework [Lourenço et al. 2001] and shares similarities with other methods like variable neighborhood search [Hansen et al. 2004] and the ruin-and-recreate search [Schrimpf et al. 2000]. However, as we explain in this section, SBTS possesses some particular features like four constrained neighborhoods and the specific rules for an effective exploration of these neighborhoods.

7.2.1 General procedure

The general SBTS procedure is summarized in Algorithm 9. SBTS uses a fast randomized construction procedure (Section 7.2.3) to obtain a first feasible independent set S (i.e., no two vertices of S are adjacent, S is also called *a feasible solution* or simply *a solution* in this Appendix). From this initial solution, SBTS tries to find improved solutions (i.e., larger independent sets) by a series of intensification and diversification steps (Sections 7.2.7 and 7.2.8). Both intensification and diversification steps are based on the general $(k, 1)$-swap operator (Section 7.2.5).

Specifically, each intensification step makes a $(k, 1)$-swap move ($k = 0, 1$) to increase the cardinality of the independent set or search new solutions while keeping the cardinality unchanged. Inversely, a diversification step applies a $(k, 1)$-swap move ($k \geq 2$) to decrease temporarily the quality of the current solution (the current solution loses $k - 1$ vertices). Whenever there exist intensification moves, they are always preferred over diversification moves. Diversification moves are only applied to escape from a local optimum (i.e., when no eligible $(k, 1)$-swap move ($k = 0, 1$) is available). As we explain in Sections 7.2.7 and 7.2.8, both intensification and diversification are subject to dedicated rules which govern the way $(k, 1)$-swap moves are executed.

SBTS uses a global variable S_* to record the best solution ever discovered during the search and a tabu list to prevent short-term cycles (see Section 7.2.6). The algorithm stops when a fixed number of iterations are realized.

7.2.2 Search space and evaluation function

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

Algorithm 9 General procedure of the SBTS algorithm for MIS

Input: A graph G, $Iters_{max}$ (maximum allowed iterations per run)
Output: The largest independent set S_* found.
$S \leftarrow Initialization()$ /* Generate a feasible independent set S, Sect. 7.2.3 */
$S_* \leftarrow S$ /* S_* records the largest independent set found so far */
$f_* \leftarrow f(S)$ /* f_* records the cardinality of S_* */
Initialize tabu_list /* Initialize the tabu list, Sect. 7.2.6 */
for $iters \leftarrow 1$ **to** $Iters_{max}$ **do**
  if there exists an intensification move **then**
    $S \leftarrow$ IntensificationStep(S) /* Apply $(k, 1)$-swap ($k \leq 1$) to improve solution S, Sect. 7.2.7 */
    if $f(S) > f_*$ **then**
      $S_* \leftarrow S$, $f_* \leftarrow f(S)$
    end if
  else
    $S \leftarrow$ DiversificationStep(S) /* Apply $(k, 1)$-swap ($k > 1$) to perturb solution S, Sect. 7.2.8 */
  end if
  Update tabu_list /* Sect. 7.2.6 */
end for
return S_*

For a given graph $G = (V, E)$, the search space Ω explored by SBTS is the set of all the independent sets of G, i.e., $\Omega = \{S \subseteq V : v_i, v_j \in S, \{v_i, v_j\} \notin E\}$. For any feasible solution $S \in \Omega$, its quality is directly assessed by the cardinality of S, i.e., $f(S) = |S|$. Given two independent sets S and S', S is better than S' if and only if $f(S) > f(S')$.

7.2.3 Initial solution

The initial solution used by the SBTS algorithm is generated by the following sequential randomized heuristic (V is the vertex set of graph G).

1. Set S to empty
2. Select randomly a vertex $u \in V$ and add u into S
3. Remove from V vertex u and all its adjacent vertices $v \in V$ ($\{u, v\} \in E$)
4. Repeat steps 2–3 until V becomes empty and return S

It is easy to observe that the resulting solution S is a feasible (and maximal) independent set. Due to the random choices at step 2, each run of this construction procedure may lead to a different solution. Given the stochastic nature of SBTS, this feature is useful for multiple runs of SBTS.

7.2.4 Preliminary definitions

To explain the intensification and diversification mechanisms of SBTS, we introduce some key concepts (measures) which are particularly useful to define the different neighborhoods and the application rules of the general $(k, 1)$-swap operator.

Definition 1. (Mapping Degree K^M) Given a graph $G = (V, E)$ and an independent set S, the Mapping Degree of a vertex v_i in $V \backslash S$ is the number of its adjacent vertices v_j in S, i.e., $\forall v_i \in V \backslash S$, $K^M(v_i) = |\{v_j \in S : \{v_i, v_j\} \in E\}|$. A similar definition of Mapping Degree can be found in [Andrade et al. 2012].

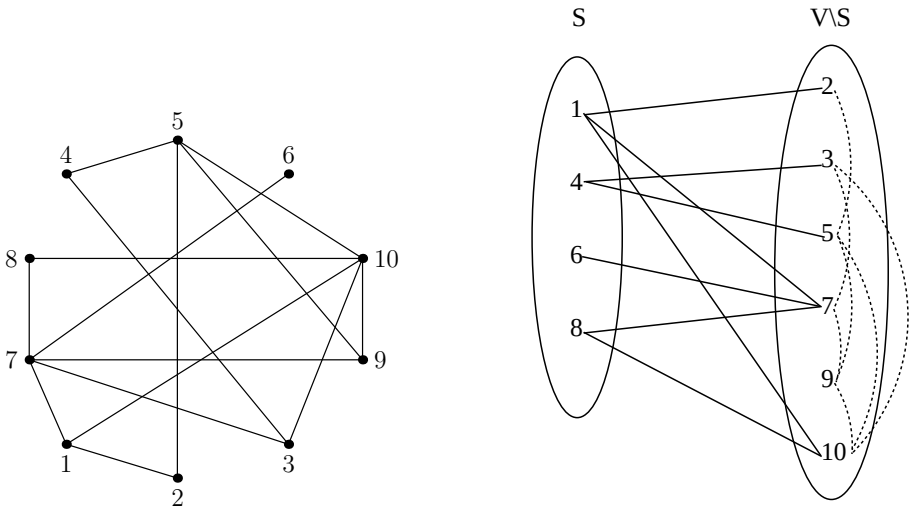


Figure 7.1: An illustrative example of graph G

Fig. 7.1 shows a graph G with 10 vertices and an independent set $S = \{1, 4, 6, 8\}$ and $V\backslash S = \{2, 3, 5, 7, 9, 10\}$. According to the definition, vertex 2 in $V\backslash S$ has one adjacent vertex (1) in S, hence the Mapping Degree $K^M(2) = 1$. Similarly, the Mapping Degrees of the other vertices in $V\backslash S$ are shown in Table 7.1. The Mapping Degree is used to partition the vertices of $V\backslash S$ into four subsets which define the neighborhoods used by SBTS (see Section 7.2.5).

Definition 2. (Expanding Degree K^E) The Expanding Degree of a vertex v_i in S is the number of its adjacent vertices v_j in $V\backslash S$ whose Mapping Degree K^M equals to 1, i.e., $\forall v_i \in S, K^E = |\{v_j \in V\backslash S : \{v_i, v_j\} \in E, K^M(v_j) = 1\}|$.

In Fig. 7.1, among the 5 vertices of S, only vertices 1 and 4 have adjacent neighbors in $V\backslash S$ with a Mapping Degree of 1, thus their Expanding Degree is $K^E(1) = 1$ and $K^E(4) = 2$ while vertices 6 and 8 have zero Expanding Degree (see Table 7.1). The Expanding Degree is used to define the rule to exploit the neighborhood induced by (1,1)-swap (see Section 7.2.7).

Definition 3. (Diversifying Degree K^D) Given a graph $G = (V, E)$ and an independent set S, the Diversifying Degree of a vertex v_i in $V\backslash S$ is the number of adjacent vertices v_j in $V\backslash S$, i.e. $\forall v_i \in V\backslash S, K^D(v_i) = |\{v_j \in V\backslash S : \{v_i, v_j\} \in E\}|$.

The Diversifying Degree is used to differentiate vertices with the same Expanding Degrees when the neighborhood induced by (1,1)-swap is exploited (see Section 7.2.7). It is also employed to define the rule to select degrading, i.e., $(k, 1)$-swap $(k > 1)$ moves to escape from local optima (see Section 7.2.8).

For the details of our example shown in Fig. 7.1, see Table 7.1. As shown in the Section 7.2.9, these measures will be dynamically updated after each iteration of the algorithm and this can be achieved efficiently in an incremental way.

7.2.5 $(k, 1)$-swap, neighborhoods and exploration of neighborhoods

The search process of the SBTS algorithm is basically driven by the general $(k, 1)$-swap $(k = 0, 1, 2, \ldots)$ operator. In this section, we provide a detailed presentation of this operator, the different neighborhoods induced by the operator and the way these neighborhoods are explored.

Table 7.1: Mapping Degree, Expanding Degree and Diversifying Degree on the illustrative graph.

Vertices in $V\backslash S$	Neighbors in S	Mapping Degree K^M	Neighbors in $V\backslash S$	Diversifying Degree K^D
2	1	1	5	1
3	4	1	7, 10	2
5	4	1	2, 9, 10	3
7	1, 6, 8	3	3, 9	2
9		0	5, 7, 10	3
10	1, 8	2	3, 5, 9	3

Vertices in S	Neighbors in $V\backslash S$ whose Mapping Degree K^M = 1	Expanding Degree K^E
1	2	1
4	3, 5	2

Let S be an independent set and $V\backslash S$ its complementary set. Let "$S \oplus (k,1)$-swap" denote the application of $(k,1)$-swap ($k \geq 0$) to S. Then the resulting solution S' is given by $S' \leftarrow S \oplus (k,1)$-swap.

According to the value of k, $(k,1)$-swap changes differently the cardinality of the current solution S. The resulting solution has a larger or equal cardinality when $k \leq 1$ (i.e., $k = 0, 1$). Otherwise (i.e., $k \geq 2$), the resulting solution is deteriorated by $k-1$ units. Moreover, whatever the value k takes, applying a $(k,1)$-swap move to an independent set always leads to a feasible solution.

To define the rules to apply this general $(k,1)$-swap operator, we introduce four different neighborhoods. Precisely, given an independent set S (the current solution), we partition its complementary set $V\backslash S$ into four subsets according to the Mapping Degree of each vertex (see Section 7.2.4).

1. NS_k: the set of vertices v_i in $V\backslash S$ whose Mapping Degree K^M equals to k, i.e., $NS_k = \{v_i \in V\backslash S : K^M(v_i) = k\}, (k = 0, 1, 2)$.
2. $NS_{>2}$: the set of vertices v_i in $V\backslash S$ whose Mapping Degree K^M is larger than 2, i.e., $NS_{>2} = \{v_i \in V\backslash S : K^M(v_i) > 2\}$.

For the example of Fig. 7.1 where $S = \{1, 4, 6, 8\}$ and $V\backslash S = \{2, 3, 5, 7, 9, 10\}$, we have $NS_0 = \{9\}, NS_1 = \{2, 3, 5\}, NS_2 = \{10\}$ and $NS_{>2} = \{7\}$.

Clearly, each NS_k ($k = 0, 1, 2, > 2$) set defines unambiguously a different (and constrained) neighborhood when it is employed by the $(k,1)$-swap operator. Precisely, for a given NS_k, the associated neighborhood is composed of all the solutions obtained by swapping a vertex of NS_k with its k adjacent vertices in S. For this reason, we will also use NS_k ($k = 0, 1, 2, > 2$) to denote the associated neighborhoods interchangeably.

To explore the search space, the SBTS algorithm selects at each iteration a particular vertex from one NS_k ($k = 0, 1, 2, > 2$) as follows. SBTS first examines NS_0 to see whether an improving (0,1)-swap move is applicable. If NS_0 is not empty, a (0,1)-swap move is applied with a vertex randomly chosen from NS_0. Otherwise, if NS_1 offers eligible vertices, a side-walk (1,1)-swap move is applied to a vertex of NS_1 which is selected according to the specific rule presented in Section 7.2.7. If NS_1 is empty, SBTS makes a degrading $(k,1)$-swap move with a vertex from NS_2 or $NS_{>2}$ following the rule defined in Section 7.2.8. Hence, at each iteration, SBTS only checks a smaller number (i.e., $|NS_k|$ instead of $|V\backslash S|$) of neighboring solutions to explore the search space. According to the value of k, each iteration corresponds to either an intensification step ($k = 0, 1$) or a diversification step ($k \geq 2$). After each iteration, K^M, K^E, K^D, along with the neighborhoods NS_k and their sizes are updated accordingly (see the Section 7.2.9).

One understands intuitively that during the search process, NS_0 will be exhausted very rapidly. Among the remaining NS_k ($k = 1, 2, > 2$) sets, NS_k with $k \geq 2$ can only deteriorate the solution and should not be used frequently. The only set that could lead to a hopeful improvement of the solution is NS_1. In Section 7.2.7 and 7.2.8, we introduce dedicated rules for the exploration of the neighborhoods NS_k with $k \geq 1$.

7.2.6 Tabu list and aspiration rule

The SBTS algorithm uses a tabu list to avoid short-term cycles [Glover and Laguna 1999]. Precisely, each time a $(k, 1)$-swap move is executed, the k vertices which are swapped out from the independent set S are classified tabu in order to prevent these vertices from moving back to S for the next tt iterations (tt is called tabu tenure). On the other hand, the vertex which joins S is not subject to tabu prohibition. Thus the tabu list is updated only after a $(k, 1)$-swap with $k \geq 1$.

Suppose $(k, 1)$-swap exchanges vertex $v_i \in NS_k$ ($k \geq 1$) and its k adjacent vertices $(v_{j_1}, v_{j_2}...v_{j_k})$ in S. For each vertex v_{j_p} ($p = 1...k$), its tabu tenure tt is adaptively set as follows.

- $k = 1$: If $|NS_1| < |NS_2| + |NS_{>2}|$, $tt = 10 + Random(|NS1|)$ where $Random(A)$ returns a random value from the domain $\{0...A-1\}$; Otherwise, $tt = |NS_1|$.
- $k > 1$: tt is set to 7.

These tabu tenure rules are purely empirical. However, for the case of $k = 1$, the first part corresponds to situations which occur usually and the adopted tabu tenure ($tt = 10 + Random(|NS1|)$) is inspired from the literature [Dorne and Hao 1999, Galinier and Hao 1999, Wu and Hao 2013a]. The second part (which occurs occasionally) is based on the consideration that when there are many side-walk moves (i.e., $|NS_1|$ is very high relative to $|NS_2|$ and $|NS_{>2}|$), the vertex that just left the solution will not be considered before having tried a number of side-walk moves as high as $|NS_1|$. For the case of $k > 1$, since there are several vertices (at least two) leaving the independent set and these k vertices are not chosen according to specific objectives, there is no reason to prevent them from joining the solutions for a long period of time. For this reason, the tabu tenure for them can be set to a relatively small value. In fact, we observe that as long as the tabu tenure remains in the range of 4 to 10, it does not really impact the performance of the algorithm. So we set the tabu tenure to the middle value 7 which proves to be robust in our experiments. In Section 7.4.2, we provide more information about the tabu tenure.

Notice that vertices in NS_0 are never forbidden by the tabu list given that any vertex in NS_0 can increase the current solution by one unit. This can be considered as an aspiration condition [Glover and Laguna 1999] that revokes the tabu status of any vertex if it belongs to NS_0.

Finally, given an independent set S and the associated sets NS_k ($k = 1, 2, > 2$), a vertex from any NS_k is said *eligible* if it is not forbidden by the tabu list.

7.2.7 Intensification

Intensification of the SBTS algorithm aims to find better solutions or to reach new solutions without deteriorating the current solution. For this purpose, whenever NS_0 is not empty, SBTS applies $(0, 1)$-swap to improve the solution. For instance, in Fig. 7.1, given the current solution $S = \{1, 4, 6, 8\}$, $NS_0 = \{9\}$ and $NS_1 = \{2, 3, 5\}$, SBTS will select vertex 9 to apply $(0, 1)$-swap to generate a better solution $S = \{1, 4, 6, 8, 9\}$. When NS_0 becomes empty, SBTS checks then the NS_1 neighborhood for a possible (1,1)-swap (side-walk) move.

If NS_1 offers multiple choices for a (1,1)-swap, one must decide which vertex of NS_1 is selected for the (1,1)-swap. One trivial strategy is to make this decision at random. However, as we illustrate below, the order of examining the vertices in NS_1 for (1,1)-swap may impact the solution quality. To make this decision as fruitful as possible, we devise a selection rule which takes into account problem specific information relative to the Expanding Degree and Diversifying Degree (see Section 7.2.4). The proposed selection rule favors the (1,1)-swap moves that tend to create new promising (e.g., improving) moves for future iterations.

Selection Rule for the NS_1 neighborhood examination:

1. Collect in set NS_1^- any vertex $v_i \in NS_1$ such that its adjacent neighbor v_j in S has the largest Expanding Degree;
2. If NS_1^- is composed of a single vertex, select this vertex; otherwise, select the vertex $v_i \in NS_1^-$ with the largest Diversifying Degree (ties are broken at random).

The first part of this *Selection Rule* is based on the following consideration. When swapping $v_i \in NS_1$ with $v_j \in S$ such that v_j has the largest Expanding Degree, we encourage the emergence of improving (0,1)-swap moves. For instance, in Fig. 7.1, given $NS_1 = \{2, 3, 5\}$ and suppose that all the vertices in NS_1 are eligible for a (1,1)-swap move (the notion of eligibility under the tabu rule is explained in Section 7.2.6). Since vertices 3 and 5 of NS_1 have the adjacent vertex 4 in S with an Expanding Degree of 2 while vertex 2 of NS_1 has the adjacent vertex 1 in S with an Expanding Degree of 1, we have $NS_1^- = \{3, 5\}$ (i.e., vertices 3 and 5 are preferred than vertex 2). At this point, one notices a (1,1)-swap using any vertex 3 or 5 of NS_1^- (say 3) will change the Mapping Degree of the other vertex (vertex 5) to 0. This makes the other vertex to become a member of the updated NS_0 and could be added to the independent set at the next iteration. In comparison, since vertex $2 \in NS_1$ has an Expanding Degree of 1, swapping 2 into S cannot create any improving moves.

The second part of this *Selection Rule* is based on the consideration that the vertices of NS_1 with a larger Diversifying Degree could make the search more diversified after a (1,1)-swap move. Indeed, after swapping $v_i \in NS_1$ and $v_j \in S$, we need to update the Mapping Degree, the Expanding Degree and Diversifying Degree concerned by v_i and v_j (see Section 7.2.9), leading to modifications of the neighborhoods NS_k $(k = 0, 1, 2, > 2)$. By definition, a vertex $v_i \in NS_1$ with a larger Diversifying Degree has more adjacent vertices in $V \backslash S$. Selecting such a vertex for a (1,1)-swap move leads to more changes in $V \backslash S$, thus more changes in the neighborhoods NS_k $(k = 0, 1, 2, > 2)$. In this sense, this helps to diversify the choices of the next iteration of the search procedure. For our example in Fig. 7.1, vertices 3 and 5 have respectively a Diversifying Degree of 2 and 3. According to the *Selection Rule*, vertex 5 (instead of vertex 3) is selected to take part in the swap move with vertex 4 in S. After this move, three vertices (2, 9 and 10 which are adjacent to 5 in $V \backslash S$) take part in neighborhood updating. In comparison, vertex 3 in NS_1 (with a small Diversifying Degree) will induce fewer changes in the neighborhoods.

One notices that this heuristic selection rule has no theoretical guarantee of being able to always lead to the best choice. However, the rule is designed to favor a good choice when such a choice is available. The computational results shown in this work confirm its usefulness in practice.

Further reducing NS_1 neighborhood examination:

As explained above, among the vertices of NS_1, those v_i whose adjacent neighbor v_j in S has an Expanding Degree of 1 are less promising than the other vertices since using these v_i in (1,1)-swap can only lead to new side-walk (or degrading) moves and can never create new improving moves for the next iteration. In order to prevent the search from making uselessly too many side-walk moves, we define an additional rule to reduce NS_1 as follows. If there are more (1,1)-swap moves than $(k, 1)$-swap $(k > 1)$ moves (i.e., $|NS_1| > |NS_2| + |NS_{>2}|$), we exclude from NS_1 any v_i such that its adjacent neighbor v_j

has an Expanding Degree of 1. For instance, in Fig. 7.1, $NS_1 = \{2, 3, 5\}$. If we apply this reduction rule, vertex 2 will be excluded from NS_1 since the Expanding Degree of its neighbor in S (vertex 1) equals to 1. Experiments show that this reduction rule could improve the search efficiency for a number of situations where a large number of side-walk moves frequently appear during the search process.

Algorithm 10 The Intensification Step for MIS

```
1: Input: A feasible independent set S
2: Output: The independent set S'.
3: /* Explore neighborhood NS_0 with an improving (0,1)-swap move */
4: if NS_0 is not empty then
5:     Choose randomly a vertex v_i from NS_0;
6:     S' ← S ⊕ (0,1)-swap;
7: else
8:     /* Explore neighborhood NS_1 with a side-walk (1,1)-swap move */
9:     if |NS_1| > |NS_2| + |NS_>2| then
10:        Exclude vertices v_i from NS_1 whose neighbor v_j in S satisfies K^E(v_j) = 1;
11:    end if
12:    if A vertex v_i (v_i ∈ NS_1) is chosen according to Selection Rule then
13:        S' ← S ⊕ (1,1)-swap;
14:    else
15:        S' ← S;
16:    end if
17: end if
18: Perform the updating procedure; /* see Section 7.2.9 */
19: Return S';
```

The pseudo-code of one intensification iteration is given in Algorithm 10 where S is the current independent set and NS_k $(k = 0, 1, 2, > 2)$ are the associated neighborhoods.

Notice that after each $(1, 1)$-swap, the neighborhoods NS_k are updated accordingly (see Section 7.2.9). Additionally, the vertex that is swapped out from S is added to the tabu list to prevent it from being moved back to S for a number of next iterations (see Section 7.2.6).

If NS_0 is empty and NS_1 does not offer any eligible (1,1)-swap move (i.e., NS_1 is empty or all the vertices of NS_1 are forbidden by the tabu list), the search continues with a diversification step which is explained in the next section.

7.2.8 Diversification

When the current solution cannot be further improved by a (0,1)-swap or changed by a (1,1)-swap, the search procedure is trapped in a local optimum. To escape from this local optimum, the SBTS algorithm resorts to $(k, 1)$-swap $(k \geq 2)$ moves to perturb the current solution in order to displace the search to a new search zone. These swap moves are carried out according to some dedicated rules which once again depend on problem specific information.

One observes first that a $(k, 1)$-swap $(k \geq 2)$ move applied to a solution S deteriorates the cardinality of S by exactly $k-1$ units. Consequently, a smaller k (e.g., $k = 2$) perturbs more weakly a solution while a larger k (e.g., $k > 2$) changes more strongly the solution. To control the perturbation strength, SBTS adopts an adaptive strategy relying on the number of possible $(1, 1)$-swap moves (i.e., $|NS_1|$) and the number of of $(k, 1)$-swap $(k > 1)$ moves (i.e., $|NS_2| + |NS_{>2}|$):

1. If $|NS_1| > |NS_2| + |NS_{>2}|$, SBTS uses $NS_{>2}$ to perform a strong perturbation by a $(k, 1)$-swap $(k > 2)$ move as follows: Select an eligible vertex v_i of $NS_{>2}$ with the largest Diversifying Degree (ties are broken at random) and swap the chosen vertex v_i with its k neighbors in S.
2. Otherwise, SBTS applies with equal probability either NS_2 or $NS_{>2}$ to perform either a weak or strong perturbation.
   - $k = 2$: Select an eligible vertex v_i of NS_2 with the largest Diversifying Degree (ties are broken at random) and then swap v_i with its two neighbors in S.
   - $k > 2$: Determine an eligible v_i of $NS_{>2}$ at random without considering the tabu list and then swap the chosen vertex v_i with its k neighbors in S.

The underlying rationale for point (1) is that when a local optimum is reached, all the vertices of NS_1 are prohibited by the tabu list (i.e., they have been removed recently from the independent set S, see Section 7.2.6). A large NS_1 indicates thus that in the recent past, the search has gone through a high number of side-walk moves. This situation corresponds to a kind of deep local optimum which is diffcult to escape. To displace the search into a new search zone, we need to apply a strong perturbation which is achieved by employing a $(k, 1)$-swap move with $k > 2$.

The second case corresponds to the situation where the search has made a relative small number of side-walk moves. In this case, we alternate probabilistically the perturbation strength to try to find better solutions in a zone around the current local optimum (with a weak perturbation) or far from current local optimum (with a strong perturbation).

Like for an intensification step, after a $(k, 1)$-swap $(k \geq 2)$, the neighborhood sets NS_k $(k = 0, 1, 2, > 2)$ are updated accordingly (see Section 7.2.9). The k vertices that are swapped out from S are added to the tabu list (Section 7.2.6).

7.2.9 Information updating procedure

After each $(k, 1)$-swap, SBTS updates the Mapping degree, Expanding degrees and Diversifying degree of some vertices as well as the associated neighborhoods NS_k $(k = 0, 1, 2, > 2)$. We explain below the updating procedure.

Suppose that $v_i \in NS_k$ $(k = 0, 1, 2, > 2)$ is swapped with its k adjacent vertices $v_j \in S$. Let $v_{ia} \in V \backslash S$ be any adjacent vertex of v_i (i.e., $v_{ia} \in V \backslash S$, $\{v_{ia}, v_i\} \in E$). For each v_j, let $v_{ja} \in V \backslash S$ be any adjacent vertex of v_j (i.e., $v_{ja} \in V \backslash S$, $\{v_{ja}, v_j\} \in E$). Then the updating procedure realizes the following operations:

1. First, for v_i and each v_j: Since v_i moves from NS_k to S, its Expanding Degree is initially set to 0. Since v_j is removed from S, its Mapping Degree and Diversifying Degree is initially set to 0.
2. Then, for each vertex v_{ia}: its Diversifying Degree decreases by 1 and its Mapping Degree increases by 1. Meanwhile, the Mapping Degree of v_j increases by 1.

   The Expanding Degree of v_i increases by 1 if the Mapping Degree of v_{ia} increases from 0 to 1 (including v_j) while the Expanding Degree of v_i decreases by 1 if the Mapping Degree of v_{ia} increases from 1 to 2.

   When the Mapping Degree of v_{ia} changes from k to $k + 1$ $(k = 0, 1, 2)$, v_{ia} moves from NS_k to NS_{k+1} for $k = 0, 1$ and to $NS_{>2}$ for $k = 2$. If $k > 2$, v_{ia} stays in $NS_{>2}$. Notice that v_j belongs now to NS_1.
3. Finally, for each v_j and its adjacent vertices v_{ja} in $V \backslash S$: The Diversifying Degree of v_{ja} increases by 1 while the Mapping Degree of v_{ja} decreases by 1. Meanwhile, the Diversifying Degree of v_j increases by 1 for each v_{ja}.

For any $v'_j \in S$ adjacent to v_{ja} ($\{v'_j, v_{ja}\} \in E, v'_j \neq v_j$), its Expanding Degree increases by 1 if the Mapping Degree of v_{ja} decreases from 2 to 1.

According to the decrease of the Mapping Degree of v_{ja} from $k+1$ to k ($k = 0, 1, 2$), v_{ja} displaces from NS_{k+1} ($NS_{>2}$ if $k > 2$) to NS_k. If $k > 2$, v_{ja} stays in $NS_{>2}$.

Notice that no v_j is swapped out from S if a (0,1)-swap is applied. In this case, only v_i and its adjacent vertices in $V \backslash S$ need to be updated.

This procedure can be efficiently performed in $O(k + |\{v_{ia}\}| + k|\{v_{ja}\}|)$. For Fig. 7.1, if vertex 9 is added into the solution S ={1, 4, 6, 8} after a (0,1)-swap, the Mapping Degree of its neighbors {5, 7, 10} becomes $K^M(5) = 2, K^M(7) = 4$ and $K^M(10) = 3$. The Expanding Degree of vertex 4 becomes $K^E(4) = 1$. The new neighborhoods become: $NS_0 = \emptyset$, $NS_1 = \{2, 3\}$, $NS_2 = \{5\}$ and $NS_{>2} = \{7, 10\}$.

7.3 Experimental results

7.3.1 Benchmark instances

To evaluate the efficiency of our proposed SBTS algorithm, we carry out experiments on three different data sets: DIMACS, BHOSLIB and CODE:

- **DIMACS benchmark:** This set was established for the Second DIMACS Implementation Challenge [Johnson and Trick 1996]. It contains 12 varieties of instances with multiple topologies and densities. It is composed of 80 graphs with size ranging from less than 50 vertices and 1 000 edges up to more than 4 000 vertices and 5 000 000 edges. These instances are the most popular and frequently used for evaluating algorithms for MIS, MC and MVC. Among these 80 DIMACS instances, the maximum clique is now known for 74 of them except 6 graphs: 3 (large) random graphs with at least 500 vertices (C500.9, C1000.9, C2000.9) and 3 structured graphs (hamming10-4, johnson32_2_4, keller6) [McCreesh and Prosser 2013, Wu and Hao 2015]. These instances are available from `http://www.cs.hbg.psu.edu/txn131/clique.html`.
- **BHOSLIB benchmark:** This set arose from the SAT'04 Competition. The BHOSLIB instances were translated from hard random SAT instances [Xu et al. 2005]. Each of the 40 instances has a known, but hidden optimal solution. These instances have a size ranging from less than 500 vertices and 100 000 edges up to more than 1 500 vertices and 10 000 000 edges. The set is more and more used in the literature for performance evaluation. These instances are available from `http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm`.
- **CODE benchmark:** This set is composed of 11 large graphs arising from code theory with size ranging from 1 024 vertices and 7 936 edges up to 4 096 vertices and more than 184 320 edges. These instances have unknown optima and are the least frequently used in the literature. They are available from `http://neilsloane.com/doc/graphs.html`.

Since the original DIMACS graphs are proposed for MC, we use their complement graphs to test our SBTS algorithm. For BHOSLIB and CODE benchmarks, the original graphs are used.

7.3.2 Experimental protocol

Our SBTS algorithm is coded in C++ and compiled using g++ with the '-O2' option on a Cluster running Linux with 2.83GHz and 8GB. When we run the DIMACS machine benchmark program [1] on our machine, we obtain the following results: 0.20 CPU seconds for graph r300.5, 1.23 CPU seconds for r400.5 and 4.68 CPU seconds for r500.5.

1. `ftp://dimacs.rutgers.edu/pub/dsj/clique/`

Given its stochastic nature, we run SBTS independently 100 times to solve each instance with initial solutions generated by the procedure of Section 7.2.3. The stop condition of each run is a maximum of 10^8 iterations which are divided into 10^4 restarts, each restart being limited to 10^4 iterations (i.e., $Iters_{max} = 10^4$, Algorithm 9). This experimental protocol is typically used in the literature (see next section). SBTS runs with the self-tuned tabu tenure tt given in Section 7.2.6. Though fine-tuning tabu tenure would lead to improved results for some graphs, for our experiments, we used the above tabu tenure except as otherwise stated. No other parameter is required by SBTS.

7.3.3 Computational results of SBTS on DIMACS, BHOSLIB and CODE instances

Tables 7.2, 7.3 and 7.4 show respectively the computational statistics of the SBTS algorithm on the three sets of benchmark instances with respect to f_{bk} which designates the optimal value or the best lower bound (i.e., the largest independent set ever reported in the literature). Notice that for the popular DIMACS and BHOSLIB benchmarks, recent heuristics can attain the f_{bk} value for many cases, as it is shown in Table 7.5 of Section 7.3.4.

Table 7.2 shows the computational statistics of the SBTS algorithm on the set of 80 DIMACS instances. Columns 1-4 give the characteristics of each graph: Name, number of vertices and edges, and optimal or best-known result f_{bk} (optimal values are marked with '*'). The columns under heading "SBTS" list our best result f_*, the average result f_{avg}, the successful runs $Success$ for reaching f_* over the 100 independent runs, the average iterations $AvgIters$ and the average CPU time $t(s)$ (seconds) over the successful runs.

Table 7.2: Detailed computational results of SBTS on the set of 80 DIMACS instances. Each instance is solved 100 times and each run is limited to a maximum of 10^8 iterations.

Characteristics of the graphs				SBTS				
Name	n	m	f_{bk}	f_*	$f_{avg}(Std.)$	$Success$	$AvgIters$	$t(s)$
brock200_1	200	14834	21*	21	21.00	100/100	329	0.0005
brock200_2	200	9876	12*	12	12.00	100/100	12859	0.0450
brock200_3	200	12048	15*	15	15.00	100/100	11606	0.0296
brock200_4	200	13089	17*	17	17.00	100/100	33277	0.0768
brock400_1	400	59723	27*	27	27.00	100/100	15572682	66.3977
brock400_2	400	59786	29*	29	29.00	100/100	4159016	20.2432
brock400_3	400	59681	31*	31	31.00	100/100	340265	1.7676
brock400_4	400	59765	33*	33	33.00	100/100	160679	0.8298
brock800_1	800	207505	23*	23	21.52(0.88)	26/100	56901393	963.6288
brock800_2	800	208166	24*	24	22.29(1.49)	43/100	46449467	784.3647
brock800_3	800	207333	25*	25	24.16(1.35)	72/100	40752010	651.0726
brock800_4	800	207643	26*	26	25.90(0.70)	98/100	25784326	421.6047
C125.9	125	6963	34*	34	34.00	100/100	85	0.0001
C250.9	250	27984	44*	44	44.00	100/100	492	0.0005
C500.9	500	112332	57	57	57.00	100/100	23184	0.0540
C1000.9	1000	450079	68	68	68.00	100/100	1438740	8.3696
C2000.5	2000	999836	16*	16	16.00	100/100	7628	0.9211
C2000.9	2000	1799532	80	80	77.29(0.64)	2/100	79591805	1515.5700
C4000.5	4000	4000268	18*	18	18.00	100/100	4289342	1553.2406
DSJC500.5	500	125248	13*	13	13.00	100/100	524	0.0074
DSJC1000.5	1000	499652	15*	15	15.00	100/100	26455	2.2891
keller4	171	9435	11*	11	11.00	100/100	27	0.0001
keller5	776	225990	27*	27	27.00	100/100	2101	0.0194
keller6	3361	4619898	59	59	59.00	100/100	12311511	754.8754
MANN_a9	45	918	16*	16	16.00	100/100	3	0.0000
MANN_a27	378	70551	126*	126	126.00	100/100	635	0.0008
MANN_a45	1035	533115	345*	345	345.00	100/100	4763131	27.5632
MANN_a81	3321	5506380	1100*	1100	1100.00	100/100	716340	22.6991
hamming6-2	64	1824	32*	32	32.00	100/100	18	0.0000
hamming6-4	64	704	4*	4	4.00	100/100	2	0.0000
hamming8-2	256	31616	128*	128	128.00	100/100	217	0.0001
hamming8-4	256	20864	16*	16	16.00	100/100	10	0.0000
hamming10-2	1024	518656	512*	512	512.00	100/100	365	0.0004
hamming10-4	1024	434176	40	40	40.00	100/100	255	0.0037
gen200-p0.9-44	200	17910	44*	44	44.00	100/100	947	0.0005
gen200-p0.9-55	200	17910	55*	55	55.00	100/100	576	0.0004

Continued on next page

Continued from previous page

Characteristics of the graphs				SBTS				
Name	n	m	f_{bk}	f_*	$f_{avg}(Std.)$	$Success$	$AvgIters$	$t(s)$
gen400-p0.9-55	400	71820	55*	55	55.00	100/100	1504	0.0027
gen400-p0.9-65	400	71820	65*	65	65.00	100/100	186	0.0006
gen400-p0.9-75	400	71820	75*	75	75.00	100/100	432	0.0010
c-fat200-1	200	1534	12*	12	12.00	100/100	34	0.0001
c-fat200-2	200	3235	24*	24	24.00	100/100	119	0.0009
c-fat200-5	200	8473	58*	58	58.00	100/100	89	0.0005
c-fat500-1	500	4459	14*	14	14.00	100/100	63	0.0013
c-fat500-2	500	9139	26*	26	26.00	100/100	76	0.0020
c-fat500-5	500	23191	64*	64	64.00	100/100	102	0.0031
c-fat500-10	500	46627	126*	126	126.00	100/100	157	0.0045
johnson8-2-4	28	210	4*	4	4.00	100/100	1	0.0000
johnson8-4-4	70	1855	14*	14	14.00	100/100	3	0.0000
johnson16-2-4	120	5460	8*	8	8.00	100/100	1	0.0000
johnson32-2-4	496	107880	16	16	16.00	100/100	1	0.0000
p_hat300-1	300	10933	8*	8	8.00	100/100	35	0.0002
p_hat300-2	300	21928	25*	25	25.00	100/100	22	0.0001
p_hat300-3	300	33390	36*	36	36.00	100/100	32	0.0001
p_hat500-1	500	31569	9*	9	9.00	100/100	105	0.0028
p_hat500-2	500	62946	36*	36	36.00	100/100	157	0.0011
p_hat500-3	500	93800	50*	50	50.00	100/100	607	0.0028
p_hat700-1	700	60999	11*	11	11.00	100/100	1620	0.0476
p_hat700-2	700	121728	44*	44	44.00	100/100	57	0.0014
p_hat700-3	700	183010	62*	62	62.00	100/100	153	0.0021
p_hat1000-1	1000	122253	10*	10	10.00	100/100	48	0.0069
p_hat1000-2	1000	244799	46*	46	46.00	100/100	343	0.0150
p_hat1000-3	1000	371746	68*	68	68.00	100/100	700	0.0127
p_hat1500-1	1500	284923	12*	12	12.00	100/100	85232	13.0180
p_hat1500-2	1500	568960	65*	65	65.00	100/100	365	0.0202
p_hat1500-3	1500	847244	94*	94	94.00	100/100	683	0.0169
san200_0.7_1	200	13930	30*	30	30.00	100/100	6682	0.0139
san200_0.7_2	200	13930	18*	18	18.00	100/100	765	0.0015
san200_0.9_1	200	17910	70*	70	70.00	100/100	890	0.0008
san200_0.9_2	200	17910	60*	60	60.00	100/100	37	0.0001
san200_0.9_3	200	17910	44*	44	44.00	100/100	3192	0.0025
san400_0.5_1	400	39900	13*	13	13.00	100/100	2586	0.0248
san400_0.7_1	400	55860	40*	40	40.00	100/100	137032	0.8490
san400_0.7_2	400	55860	30*	30	30.00	100/100	11758	0.0691
san400_0.7_3	400	55860	22*	22	22.00	100/100	14543	0.0818
san400_0.9_1	400	71820	100*	100	100.00	100/100	3823	0.0075
san1000	1000	250500	15*	15	15.00	100/100	1410471	46.4781
sanr200_0.7	200	13868	18*	18	18.00	100/100	435	0.0008
sanr200_0.9	200	17863	42*	42	42.00	100/100	665	0.0004
sanr400_0.5	400	39984	13*	13	13.00	100/100	552	0.0041
sanr400_0.7	400	55869	21*	21	21.00	100/100	559	0.0026

Table 7.2 demonstrates that SBTS obtains quite competitive results on the set of DIMACS instances. Specifically, SBTS can consistently reach the previous best-known solutions for 75 out of the 80 instances with a perfect success rate. Furthermore, SBTS can reach the best-known results for *all* the instances with various topologies and densities, including the most difficult graphs (brock $800\_x(x = 1, 2, 3, 4)$, C2000.9, MANN_$a45$ and MANN_$a81$). To the best of our knowledge, the top-performing heuristics in the literature miss at least one best-known result on these difficult graphs. On the other hand, one observes that SBTS has a low success rate (less than 50%) for 3 graphs. Notice that for the 6 open instances (C500.9, C1000.9, C2000.9, hamming10-4, johnson32_2_4, keller6), SBTS hits the best lower bounds for each run except for C2000.9. One can speculate that these lower bounds (except for C2000.9) would be close to or would be optimal solutions and thus are difficult to improve, even though this observation does not constitute a proof. As for the computing time, SBTS requires on average less than 1000 seconds except for C2000.9 and C4000.5.

Table 7.3 reports the computational results of SBTS on the set of 40 BHOSLIB instances. The column 1-4 gives the characteristic of the graphs and column 5-9 presents the detailed results of the proposed SBTS algorithm. From this table, one finds that SBTS also performs well for this benchmark set. Specifically, SBTS reaches the optimal results with a perfect success rate for the instances with up to 1000 vertices. The BHOSLIB set is known to be more difficult compared to most of the DIMACS benchmark. Yet, SBTS can still attain the optimal results for all the 40 instances. On the other hand, SBTS has a low or very low success rate (less than 50%) for 11 graphs and requires a large computing time for the largest instances.

Table 7.3: Detailed computational results of SBTS on the set of 40 BHOSLIB instances. Each instance is solved 100 times and each run is limited to a maximum of 10^8 iterations.

Characteristics of the graphs				SBTS				
Name	n	m	f_{bk}	f_*	$f_{avg}(Std.)$	$Success$	$AvgIters$	$t(s)$
frb30-15-1	450	83198	30*	30	30.00	100/100	8182	0.0264
frb30-15-2	450	83151	30*	30	30.00	100/100	15420	0.0562
frb30-15-3	450	83216	30*	30	30.00	100/100	14226	0.0498
frb30-15-4	450	83194	30*	30	30.00	100/100	15057	0.0564
frb30-15-5	450	83231	30*	30	30.00	100/100	44397	0.1611
frb35-17-1	595	148859	35*	35	35.00	100/100	171245	0.6895
frb35-17-2	595	148868	35*	35	35.00	100/100	96263	0.4014
frb35-17-3	595	148784	35*	35	35.00	100/100	26607	0.1027
frb35-17-4	595	148873	35*	35	35.00	100/100	232484	0.8445
frb35-17-5	595	148572	35*	35	35.00	100/100	61897	0.2411
frb40-19-1	760	247106	40*	40	40.00	100/100	46853	0.2384
frb40-19-2	760	247157	40*	40	40.00	100/100	4316200	20.6966
frb40-19-3	760	247325	40*	40	40.00	100/100	550507	2.7152
frb40-19-4	760	246815	40*	40	40.00	100/100	2187043	11.6519
frb40-19-5	760	246801	40*	40	40.00	100/100	9738592	55.9365
frb45-21-1	945	386854	45*	45	45.00	100/100	2800152	23.0019
frb45-21-2	945	387416	45*	45	45.00	100/100	6617043	53.1387
frb45-21-3	945	387795	45*	45	45.00	100/100	13044028	98.7293
frb45-21-4	945	387491	45*	45	45.00	100/100	5276955	42.9933
frb45-21-5	945	387461	45*	45	45.00	100/100	10366230	80.3013
frb50-23-1	1150	580603	50*	50	49.75(0.43)	75/100	37020418	368.6880
frb50-23-2	1150	579824	50*	50	49.49(0.50)	49/100	40728061	302.2116
frb50-23-3	1150	579607	50*	50	49.13(0.34)	13/100	62285352	517.4469
frb50-23-4	1150	580417	50*	50	50.00	100/100	10398673	94.6139
frb50-23-5	1150	580640	50*	50	50.00	100/100	17773506	119.3523
frb53-24-1	1272	714129	53*	53	52.03(0.17)	3/100	55616513	498.3600
frb53-24-2	1272	714067	53*	53	52.30(0.46)	30/100	34698236	321.4827
frb53-24-3	1272	714229	53*	53	52.66(0.47)	66/100	43238016	359.6429
frb53-24-4	1272	714048	53*	53	52.22(0.41)	22/100	42301012	340.4545
frb53-24-5	1272	714130	53*	53	52.91(0.29)	91/100	27705528	273.8870
frb56-25-1	1400	869624	56*	56	55.07(0.26)	7/100	54577332	551.6357
frb56-25-2	1400	869899	56*	56	55.06(0.31)	8/100	51319979	470.2750
frb56-25-3	1400	869921	56*	56	55.30(0.46)	30/100	41946192	383.5283
frb56-25-4	1400	869262	56*	56	55.86(0.35)	86/100	34350025	335.2178
frb56-25-5	1400	869699	56*	56	55.79(0.41)	79/100	38964414	568.9961
frb59-26-1	1534	1049256	59*	59	58.02(0.20)	2/100	25836232	261.1367
frb59-26-2	1534	1049648	59*	59	57.96(0.24)	1/100	78884128	762.2700
frb59-26-3	1534	1049729	59*	59	57.94(0.37)	4/100	42283734	388.9075
frb59-26-4	1534	1048800	59*	59	58.00(0.32)	5/100	74758342	969.9380
frb59-26-5	1534	1049829	59*	59	58.81(0.46)	84/100	40168986	394.9430

Table 7.4 reports the computational statistics of our SBTS algorithm on the set of 11 CODE instances where the best-known results f_{bk} are from [Andrade et al. 2012, Sloane 2000]. The results of SBTS are obtained with the default tabu tenure tt except those of 1et.2048, 1tc2048 and 1zc.4096 for which $tt = 40 + Random(|NS1|)$. From the table, one finds that SBTS attains the best-known results for all the CODE instances in a short time. SBTS reaches the best-known results with a perfect success rate for 9 out of 11 instances. However, for one case (1zc.4096), its success rate is very low (1%).

From the results on DIMACS, BHOSLIB and CODE benchmarks, one observes that there is no clear correlation between the problem size and the necessary time to solve it since the difficulty of an instance also depends on its structure.

7.3.4 Comparisons with seven state-of-the-art algorithms

To assess the performance of the proposed SBTS algorithm relative to the state-of-the-art methods, we compare in this section SBTS with some best-performing algorithms for MIS, MC and MVC in the literature. We present two comparisons which concern the DIMACS and BHOSLIB sets on the one hand and the CODE set on the other hand.

Table 7.4: Detailed computational results of SBTS on the set of 11 CODE instances. Each instance is solved 100 times and each run is limited to a maximum of 10^8 iterations.

Characteristics of the graphs				SBTS				
Name	n	m	f_{bk}	f_*	$f_{avg}(Std.)$	$Success$	$AvgIters$	$t(s)$
1dc.1024	1024	24063	94	94	94.00	100/100	10764	0.0289
1dc.2048	2048	58367	172	172	172.00	100/100	33895	0.1678
1et.1024	1024	9600	171	171	171.00	100/100	38018	0.0669
1et.2048	2048	22528	316	316	316.00	100/100	2016754	6.5911
1tc.1024	1024	7936	196	196	196.00	100/100	10798	0.0157
1tc.2048	2048	18944	352	352	352.00	100/100	1622215	6.5195
1zc.1024	1024	33280	112	112	111.99(0.01)	99/100	9984898	29.7898
1zc.2048	2048	78848	198	198	198.00	100/100	18931972	81.6927
1zc.4096	4096	184320	379	379	373.00(3.57)	1/100	96720177	1187.4100
2dc.1024	1024	169162	16	16	16.00	100/100	1371	0.0327
2dc.2048	2048	504451	24	24	24.00	100/100	154941	6.9823

Comparisons with five references algorithms on DIMACS and BHOSLIB benchmarks

For this comparison, we focus on 45 most difficult instances from DIMACS and BHOSLIB sets and ignore the other instances since they can be easily solved with a 100% success rate by all the compared algorithms. First we summarize below the experimental conditions used by 5 reference algorithms which are implemented on sequential architectures and report state-of-the-art computational results on both DIMACS and BHOSLIB benchmarks.

- MN/TS [Wu et al. 2012]: This is a multi-neighborhood tabu search algorithm which is designed for the equivalent maximum clique (and its weighted generalization). It is run on a PC with 2.83 GHz CPU and 8 GB RAM, and the stop condition is a maximum of 10^8 iterations.
- BLS [Benlic and Hao 2013]: This is an iterated local search algorithm which combines a descent procedure with a dedicated and adaptive perturbation strategy. BLS is run on a Xeon E5440 with 2.83 GHz and 2 GB, and the stop condition is a maximum of $1.6 * 10^8$ iterations.
- PLS [Pullan 2006; 2008]: This is a highly effective phased local search algorithm which relies on three sub-algorithms using different vertex selection rules. It is run on a Pentium IV machine with 512KB L2 cache and 512 MB RAM, and the stop condition is a maximum of 10^8 iterations for all instances except for MANN_a45 and MANN_a81 where 10^9 iterations are allowed.
- COVER [Richter et al. 2007]: This is a local search algorithm designed for the equivalent minimum vertex cover problem which uses edge weighting techniques. It is run on a machine with 2.13 GHz and 2 GB RAM, and the stop condition is a maximum of 10^8 iterations.
- NuMVC [Cai et al. 2014]: This is a very recent local search algorithm for MVC using edge weighting techniques. It is run on a machine with 3 GHz CPU and 4 GB RAM, and the stop condition is a cutoff time which is set to 2,000 seconds.

Table 7.5 summarizes the results of the competing algorithms. All the results are based on 100 independent runs for each graph. The reported results of the reference algorithms are extracted from the corresponding papers while the results of SBTS are from Tables 7.2 and 7.3.

For each compared algorithm, we show its best result f_*, followed by the average result f_{avg} given in parenthesis over 100 runs if the success rate is lower than 100%, and the average time in seconds $t(s)$ over the successful runs. For COVER, as stated in [Richter et al. 2007], $t(s)$ is the median run time which is indicative of a typical run of the algorithm.

"Best #" in the last row of Table 7.5 shows the number of instances for which an algorithm cannot reach the best-known results in the literature and "Avg." indicates the average value of f_{avg} for the 45 instances for each algorithm. Note that, "-" in Table 7.5 means that the result is unavailable.

Table 7.5: Comparisons of SBTS with five reference algorithms on 45 most difficult DIMACS and BHOSLIB instances.

Graph		MN/TS		BLS		PLS		COVER		NuMVC		SBTS	
Name	f_{bk}	f_*	$t(s)$	f_*	$t(s)$	f_*	$t(s)$	f_*	$t(s)$	f_*	$t(s)$	f_*	$t(s)$
brock200_1	21*	21	0.01	21	0.01	21	0.00	21	0.01	-	-	21	0.00
brock200_2	12*	12	0.06	12	0.18	12	0.03	12	0.43	12	0.13	12	0.05
brock200_3	15*	15	0.07	15	0.57	15	0.03	15	7.62	-	-	15	0.03
brock200_4	17*	17	0.09	17	0.43	17	0.08	17	7.90	17	1.26	17	0.08
brock400_1	27*	27	10.27	27	121.40	27	1.08	*25(25.00)*	-	-	-	27	66.40
brock400_2	29*	29	1.34	29	17.40	29	0.38	*28(27.01)*	-	29(28.84)	572.39	29	20.24
brock400_3	31*	31	0.63	31	5.08	31	0.18	31(30.50)	135.26	-	-	31	1.77
brock400_4	33*	33	0.28	33	3.17	33	0.10	33(32.70)	112.98	33	4.98	33	0.83
brock800_1	23*	23(22.72)	188.14	23(22.40)	1568.24	23	30.09	*21(21.00)*	-	-	-	23(21.52)	963.63
brock800_2	24*	24(23.88)	156.47	24(23.04)	1078.13	24	24.41	*22(22.00)*	-	*21(21.00)*	-	24(22.29)	784.36
brock800_3	25*	25	118.57	25(24.52)	1020.11	25	15.08	*23(23.00)*	-	-	-	25(24.16)	651.07
brock800_4	26*	26	62.38	26	601.74	26	6.54	*24(24.00)*	-	*21(21.00)*	-	26(25.90)	421.60
C125.9	34*	34	0.01	34	0.00	34	0.00	34	0.01	34	0.00	34	0.00
C250.9	44*	44	0.01	44	0.00	44	0.00	44	0.01	44	0.00	44	0.00
C500.9	57	57	0.06	57	0.00	57	0.19	57	0.31	57	0.13	57	0.05
C1000.9	68	68	0.63	68	35.70	68	1.88	68	5.82	68	2.02	68	8.37
C2000.5	16*	16	0.07	16	2.90	16	0.73	16	3.78	16	2.93	16	0.92
C2000.9	80	80(78.37)	563.70	80(78.60)	4811.17	*78(78.00)*	-	*78(77.84)*	-	80(78.71)	1393.30	80(77.29)	1515.57
C4000.5	18*	18	144.37	18	654.60	18	149.65	18	689.74	18	252.81	18	1553.24
keller4	11*	11	0.01	11	0.00	11	0.00	11	0.01	11	0.00	11	0.00
keller5	27*	27	0.05	27	0.09	27	0.05	27	0.07	27	0.04	27	0.02
keller6	59	59	97.87	59	24.80	59(57.75)	550.95	59	15.63	59	2.51	59	754.88
MANN_a27	126*	126	3.42	126	35.20	126	0.03	126	0.01	126	0.00	126	0.00
MANN_a45	345*	*340(340.00)*	90.58	*342(340.82)*	-	*344(344.00)*	28.76	345(344.41)	-	345	86.36	345	27.56
MANN_a81	1100*	*1090 (1090.00)*	632.24	*1094 (1092.17)*	-	*1098 (1098.00)*	269.66	1100 (1098.11)	-	1100 (1099.06)	732.90	1100	22.70
frb50-23-1	50*	50(49.84)	116.92	50(49.96)	882.22	50(49.72)	1045.59	50(49.89)	171.92	50	38.14	50(49.75)	368.69
frb50-23-2	50*	50(49.47)	161.77	50(49.56)	1074.17	50(49.45)	1171.15	50(49.30)	1.72	50	176.59	50(49.49)	302.21
frb50-23-3	50*	50(49.15)	214.58	50(49.08)	1037.68	50(49.16)	1041.40	50(49.24)	2.61	50(49.95)	532.81	50(49.13)	517.45
frb50-23-4	50*	50	11.91	50	55.51	50	126.44	50	16.94	50	7.89	50	94.61
frb50-23-5	50*	50	50.90	50	142.93	50(49.99)	436.29	50(49.98)	88.94	50	19.53	50	119.35
frb53-24-1	53*	53(52.03)	240.36	53(52.04)	2306.74	53(52.06)	1707.39	53(52.09)	11.31	53(52.86)	715.12	53(52.03)	498.36
frb53-24-2	53*	53(52.30)	209.89	53(52.16)	2015.13	53(52.23)	1548.89	53(52.34)	4.24	53	205.35	53(52.30)	321.48
frb53-24-3	53*	53(52.91)	253.96	53(52.88)	1199.95	53(52.66)	1185.68	53(52.91)	157.80	53	51.23	53(52.66)	359.64
frb53-24-4	53*	53(52.45)	178.01	53(52.54)	1361.23	53(52.46)	1423.27	53(52.24)	10.74	53	266.87	53(52.22)	340.45
frb53-24-5	53*	53(52.90)	278.31	53(52.90)	1100.00	53(52.85)	979.85	53(52.84)	253.05	53	39.89	53(52.91)	273.89
frb56-25-1	56*	56(55.22)	174.02	56(55.20)	2304.83	56(55.10)	1240.19	56(55.15)	20.73	56	470.68	56(55.07)	551.64
frb56-25-2	56*	56(55.12)	127.16	56(55.06)	1500.44	56(54.93)	1702.39	56(55.12)	30.33	56(55.97)	617.49	56(55.06)	470.28
frb56-25-3	56*	56(55.25)	209.48	56(55.20)	1409.09	56(55.08)	1476.61	56(55.76)	435.30	56	121.30	56(55.30)	383.53
frb56-25-4	56*	56(55.85)	158.14	56(55.86)	999.51	56(55.66)	1304.03	56(55.84)	291.11	56	49.45	56(55.86)	335.22
frb56-25-5	56*	56	85.57	56	591.49	56(55.81)	1089.08	56(55.98)	89.58	56	26.76	56(55.79)	569.00
frb59-26-1	59*	59(58.05)	242.75	59(57.96)	3298.21	*58(57.85)*	-	59(58.11)	30.76	59(58.88)	687.85	59(58.02)	261.14
frb59-26-2	59*	59(58.01)	396.38	59(58.00)	2399.92	*58(57.63)*	-	59(58.06)	40.86	59(58.38)	1160.02	59(57.96)	762.27
frb59-26-3	59*	59(58.23)	197.36	59(58.31)	2338.59	59(57.77)	1929.05	59(58.12)	65.04	59(58.96)	580.03	59(57.94)	388.91
frb59-26-4	59*	59(58.10)	192.45	59(58.20)	1823.63	59(57.71)	2044.91	59(58.01)	73.92	59(58.79)	741.21	59(58.00)	969.94
frb59-26-5	59*	59(58.99)	96.09	59	403.30	59(58.77)	1193.22	59(58.89)	292.60	59	61.91	59(58.81)	394.94
Best $\#(Avg.)$		-2(74.02)		-2(74.05)		-5(74.18)		-7(74.01)		-($\geq$ 2)($\leq$ 74.36)		0(74.21)	

One observes from Table 7.5 that except SBTS, each reference algorithm fails to find the best-known results for at least two instances (entries in italic). Indeed, given that these instances have very different characteristics and structures, it is known that it is very difficult for a single heuristic to perform well on all the instances [Pullan et al. 2011]. Besides, SBTS has a slightly better average result of 74.21 against 74.18 of PLS which is the best among the reference algorithms (except NuMVC whose average is an optimistic upper bound since its results are missing for six instances).

For the DIMACS instances, MN/TS, BLS and PLS (which are maximum clique or maximum independent set algorithms) reach the best reported results for the groups "brock", "C", and "keller" with a high success rate except for C2000.9 which is among the most difficult instance. For this instance, MN/TS and BLS achieve the best-known result (80) with an average of 78.37 and 78.60 respectively while PLS fails to find solutions larger than 78. For the group "MANN", MN/TS, BLS and PLS cannot reach the best-known results for MANN_a45 (345) and MANN_a81 (1100). The largest solutions they find have a size of 340, 342, and 344 for MANN_a45, and a size of 1090, 1094, and 1098 for MANN_a81 respectively. Generally,

it seems that the typical MC or MIS algorithms (e.g., MN/TS, BLS, PLS) have serious difficulties to solve these two "MANN" instances.

By contrast, the typical MVC algorithms COVER and NuMVC perform well on the group "MANN" with a high success rate while they clearly encounter difficulties for the group "brock". Indeed, COVER fails to reach the best-known result for 6 out of the 12 brock instances. For the 6 brock instances tested by NuMVC, two results do not match the best-known values. Besides, for C2000.9, NuMVC can achieve the best-known result of 80 while COVER can only achieve a solution of size 78.

Our SBTS algorithm achieves the best-known results for all 25 DIMACS instances including the two "problematic" groups "brock" and "MANN". In particular, SBTS can attain the best results for MANN_a45 and MANN_a81 with a perfect success rate, which is better than the typical MC or MIS algorithms.

The average results given in parenthesis show that MN/TS, BLS, PLS, COVER and NuMVC can attain the reported best results in every single run for 20, 19, 21, 14, and 20 cases out of the 25 DIMACS instances respectively, while SBTS has a perfect success rate for 20 cases, which is more than BLS and COVER, equal to MN/TS and NuMVC and one less than PLS. However, for C2000.9, the average result of SBTS is slightly worse than the reference algorithms.

For the BHOSLIB instances, the reference algorithms can attain the best-known results except PLS which fails to reach the optimal solutions for frb59-26-1 and frb59-26-2. One observes from the average results that MN/TS, BLS, PLS, COVER and NuMVC can attain the optimal solutions in every single run for 3, 4, 1, 1, and 13 cases respectively. Our SBTS algorithm is able to reach the best-known results with a perfect success rate for 2 cases, which is more than PLS and COVER but less than MN/TS, BLS, and NuMVC.

Finally, it is more delicate to make a fully fair comparison of the computing time given that the compared algorithms are coded in different languages with different data structures, run on different platforms and more importantly lead to results of different quality for a number of graphs. As an indicative, we observe that to reach a result of equal quality, SBTS is more time consuming than MN/TS, COVER and NuMVC, but remains competitive with BLS and PLS.

Comparisons with two reference algorithms on CODE benchmark

The CODE benchmark is less popular than the DIMACS and BHOSLIB sets and few papers report results on the 11 CODE instances including [Andrade et al. 2012, Butenko et al. 2009, Etzion and Ostergard 1998]. However, we think the CODE instances are of interest since they come from real problems (code theory) and known to be relatively difficult. For this study, we adopt as our reference two most recent algorithms that use the CODE benchmark: ILS and GLP [Andrade et al. 2012]. Both ILS and GLP are run on a computer equipped with a 3.16 GHz Intel Core 2 Duo CPU and 4 GB of RAM. Unlike the reference studies of the last section which make 100 independent runs, the results of ILS and GLP reported in [Andrade et al. 2012] are based on 15 runs. The stop condition for each run is the average arc (edge) scans limited to 2^{17} [Andrade et al. 2012].

In addition to the 11 CODE instances, the authors of [Andrade et al. 2012] also report results on a subset of 33 DIMACS and 9 BHOSLIB instances (8 instances as they are introduced in Section 7.3.1 plus one additional challenging instance frb100-40). To make a fair comparison, we re-run SBTS 15 times (like ILS and GLP) on these 11 CODE instances and the 42 DIMACS/BHOSLIB instances. Since there is no evident way to relate the number of average arc (edge) scans used by ILS and GLP to the number of iterations used by SBTS, SBTS is run under the stop condition given in Section 7.3.4. To report the results, we only retain

Table 7.6: Comparisons of SBTS with two typical MIS algorithms ILS and GLP [Andrade et al. 2012] on 32 representative instances.

Graph		ILS		GLP		SBTS	
Name	f_{bk}	f_*	$t(s)$	f_*	$t(s)$	f_*	$t(s)$
brock400_1	27*	*25(25.0)*	11.00	27(25.1)	22.00	27	59.21
brock400_2	29*	*25(25.0)*	11.00	29(27.7)	20.00	29	25.39
brock400_3	31*	31(27.0)	11.00	31	19.00	31	1.01
brock400_4	33*	33(30.3)	11.00	33	16.00	33	0.86
brock800_1	23*	*21(21.0)*	60.00	23(21.1)	112.00	23(21.3)	1127.50
brock800_2	24*	*21(21.0)*	60.00	*21(21.0)*	111.00	24(22.0)	787.93
brock800_3	25*	*22(22.0)*	60.00	25(22.2)	111.00	25(24.4)	763.61
brock800_4	26*	26(21.3)	60.00	26(21.7)	112.00	26(25.7)	400.82
C2000.9	80	*77(76.9)*	103.00	*79(77.5)*	182.00	*78(77.1)*	1558.14
C4000.5	18*	18(17.1)	1897.00	18	3708.00	18	1553.24
MANN_a45	345*	345(344.5)	3.00	*344(343.8)*	5.00	345	17.99
MANN_a81	1100*	1100	10.00	*1098(1097.6)*	17.00	1100	22.29
frb30-15-1	30*	30	9.00	30	22.00	30	0.03
frb35-17-1	35*	35(34.9)	13.00	35	32.00	35	1.93
frb40-19-1	40*	40	19.00	40	43.00	40	0.87
frb45-21-1	45*	45(44.7)	27.00	45(44.9)	62.00	45	34.50
frb50-23-1	50*	50(48.9)	36.00	*49(48.6)*	82.00	50(49.7)	250.32
frb53-24-1	53*	53(51.5)	42.00	*52(51.3)*	93.00	*52(52.0)*	52.36
frb56-25-1	56*	*55(54.2)*	49.00	*55(54.1)*	111.00	*55(55.0)*	123.34
frb59-26-1	59*	*58(57.3)*	57.00	*57(57.0)*	126.00	59(58.0)	436.48
frb100-40	100*	*96(95.3)*	249.00	*95(94.1)*	495.00	*96(95.4)*	862.18
1dc.1024	94	94(93.1)	14.00	94(93.1)	31.00	94	0.00
1dc.2048	172	172(171.1)	32.00	172(171.5)	74.00	172	0.06
1et.1024	171	171	8.00	171(170.9)	16.00	171	0.16
1et.2048	316	316	16.00	316	40.00	316	4.13
1tc.1024	196	196	8.00	196	18.00	196	0.03
1tc.2048	352	352	15.00	352	37.00	352	19.10
1zc.1024	112	112(111.1)	10.00	112	28.00	112	43.44
1zc.2048	198	198(197.3)	22.00	198(197.8)	65.00	198	56.94
1zc.4096	379	379(367.7)	51.00	379(374.4)	160.00	379(372.6)	99.83
2dc.1024	16	16	50.00	16	198.00	16	0.01
2dc.2048	24	24(23.8)	165.00	24	527.00	24	3.07

the 12 (out of 33) most difficult graphs for the DIMACS set while keeping the 11 CODE instances and the 9 BHOSLIB instances since for the remaining instances, all three compared algorithms reach the same results.

Table 7.6 shows the comparison of ILS (columns 3-4), GLP (columns 5-6), and SBTS (columns 7-8): best result f_* followed by the average results f_{avg} given in parenthesis over 15 runs and the average time in seconds $t(s)$ over the successful runs.

From Table 7.6, one observes that ILS and GLP cannot reach the best-known results for 9 (entries in italic) out of 21 difficult DIMACS and BHOSLIB instances while SBTS fails to reach the best-known result for 4 instances with its 15 runs (corresponding to the cases where its success rate is lower than 15%, see Tables 7.2 and 7.3). Furthermore, the average results of SBTS on the instances which cannot be solved with a 100% success rate are all better than ILS and GLP except for C2000.9 where the average of SBTS (77.1) is worse than GLP (77.5) but better than ILS (76.9). We do not emphasize the computing time since the compared algorithms give several results of different quality (f_*).

For the CODE set, the three compared algorithms can achieve the best-known result for the 11 instances. Furthermore, ILS and GLP reach the best results with a 100% success rate for 5 and 6 cases respectively against 10 cases for our SBTS algorithm (i.e., except 1zc.4096).

To conclude, the comparative results indicate that the proposed SBTS algorithm is quite competitive with the reference algorithms not only for the best obtained solutions but also for the average solutions. SBTS seems to be the most comprehensive approach to solve the DIMACS, BHOSLIB and CODE instances with multiple topologies and densities.

Table 7.7: Comparisons of SBTS$_{random}$ with SBTS

Graph		SBTS$_{random}$		SBTS	
Name	f_{bk}	$f_*(f_{avg})$	$t(s)$	$f_*(f_{avg})$	$t(s)$
brock400_1	27*	27	118.95	27	66.40
brock400_2	29*	29	18.81	29	20.24
brock400_3	31*	31	4.63	31	1.77
brock400_4	33*	33	0.81	33	0.83
brock800_1	23*	23(21.40)	874.88	23(21.52)	963.63
brock800_2	24*	24(22.29)	823.12	24(22.29)	784.36
brock800_3	25*	25(23.89)	937.43	25(24.16)	651.07
brock800_4	26*	26(25.65)	671.61	26(25.90)	421.60
C2000.9	80	*78(76.30)*	2117.21	80(77.29)	1515.57
C4000.5	18*	18(17.99)	5482.05	18	1553.24
MANN_a45	345*	345(344.06)	302.25	345	27.56
MANN_a81	1100*	*1098(1098.00)*	452.82	1100	22.70
frb30-15-1	30*	30	0.16	30	0.03
frb35-17-1	35*	35	48.40	35	0.69
frb40-19-1	40*	40	36.51	40	0.24
frb45-21-1	45*	45(44.97)	327.35	45	23.00
frb50-23-1	50*	50(48.92)	394.40	50(49.75)	368.69
frb53-24-1	53*	*52(51.25)*	609.09	53(52.03)	498.36
frb56-25-1	56*	*55(54.11)*	594.24	56(55.07)	551.64
frb59-26-1	59*	*58(57.04)*	890.92	59(58.02)	261.14
frb100-40	100*	*95(94.22)*	2094.28	*97(95.48)*	862.18
1dc.1024	94	94	0.09	94	0.03
1dc.2048	172	172	0.20	172	0.17
1et.1024	171	171	0.02	171	0.07
1et.2048	316	316(315.68)	116.37	316	6.59
1tc.1024	196	196	0.02	196	0.02
1tc.2048	352	352(351.96)	110.99	352	6.52
1zc.1024	112	112(111.99)	44.57	112(111.99)	29.79
1zc.2048	198	198(197.40)	172.03	198	81.69
1zc.4096	379	*377(356.02)*	727.18	379(372.66)	99.83
2dc.1024	16	16	0.35	16	0.03
2dc.2048	24	24	188.08	24	6.98

7.4 Analysis of SBTS

Now we turn our attention to an analysis of the important features of the proposed SBTS algorithm: the selection rule for intensification (Section 7.2.7) and the tabu tenure (Section 7.2.6).

7.4.1 Influence of the selection rule for intensification

As described in Section 7.2.7, SBTS uses a Selection Rule for the NS_1 neighborhood for (1,1)-swap moves. In this section, we carry out an experiment to verify the importance of this dedicated Selection Rule compared to a random selection rule. For this purpose, we create a variant of SBTS (denoted by SBTS$_{random}$) by replacing its Selection Rule with a random selection rule. With SBTS$_{random}$, when NS_1 offers multiple eligible vertices, one of them is picked at random and used by the (1,1)-swap move.

For this experiment, we run SBTS and SBTS$_{random}$ 100 times on each of the 32 instances (DIMACS, BHOSLIB, CODE) of Section 7.3.4 under the same condition as before. The results are given in Table 7.7 which shows for each algorithm the best result f_*, the average result f_{avg} (in parenthesis) and the average time in second $t(s)$ to reach the best result f_*. From Table 7.7, one notices that SBTS performs better than SBTS$_{random}$ both in terms of the best result f_* and the average result f_{avg}. Precisely, SBTS achieves the best-known result for all the instances except frb100-40 while SBTS$_{random}$ attains the best-known result for only 25 cases out of 32 instances. Besides, SBTS has a perfect success rate for 20 cases against 13 cases for SBTS$_{random}$. This experiment demonstrates the usefulness of using the proposed Selection Rule to explore the NS_1 neighborhood.

Table 7.8: Comparisons of SBTS$_{unique}$ with SBTS

Graph		SBTS$_{unique}$		SBTS	
Name	f_{bk}	f_*	$t(s)$	f_*	$t(s)$
brock400_1	27*	27(26.98)	59.04	27	66.40
brock400_2	29*	29	6.07	29	20.24
brock400_3	31*	31	0.71	31	1.77
brock400_4	33*	33	0.23	33	0.83
brock800_1	23*	23(21.46)	452.06	23(21.52)	963.63
brock800_2	24*	24(22.44)	653.53	24(22.29)	784.36
brock800_3	25*	25(24.10)	739.92	25(24.16)	651.07
brock800_4	26*	26(25.95)	458.71	26(25.90)	421.60
C2000.9	80	*79(76.85)*	605.79	80(77.29)	1515.57
C4000.5	18*	18	1981.74	18	1553.24
MANN_a45	345*	345(344.25)	318.49	345	27.56
MANN_a81	1100*	1100(1098.41)	1680.60	1100	22.70
frb30-15-1	30*	30	0.03	30	0.03
frb35-17-1	35*	35	0.47	35	0.69
frb40-19-1	40*	40	0.74	40	0.24
frb45-21-1	45*	45	68.51	45	23.00
frb50-23-1	50*	50(49.55)	356.14	50(49.75)	368.69
frb53-24-1	53*	53(52.03)	438.46	53(52.03)	498.36
frb56-25-1	56*	56(55.09)	453.20	56(55.07)	551.64
frb59-26-1	59*	59(57.86)	777.11	59(58.02)	261.14
frb100-40	100*	*97(95.28)*	849.67	*97(95.48)*	862.18
1dc.1024	94	94	0.03	94	0.03
1dc.2048	172	172	0.07	172	0.17
1et.1024	171	171	0.22	171	0.07
1et.2048	316	316	27.23	316	6.59
1tc.1024	196	196	0.01	196	0.02
1tc.2048	352	352	13.73	352	6.52
1zc.1024	112	112	52.26	112(111.99)	29.79
1zc.2048	198	198(197.89)	178.46	198	81.69
1zc.4096	379	*377(365.26)*	377.07	379(372.66)	99.83
2dc.1024	16	16	0.01	16	0.03
2dc.2048	24	24	2.78	24	6.98

7.4.2 Analysis of the tabu tenure tuning technique

Recall that the tabu tenure tt is set differently according to $k = 1$ or $k > 1$ (see Section 7.2.6). In this section, we carry out an experiment to show the usefulness of this tuning technique. For this purpose, we adopt for the case $k > 1$ the same tabu tenure as for the case $k = 1$ and denote the resulting variant by SBTS$_{unique}$. We use SBTS$_{unique}$ to solve 100 times each of the 32 instances under the same condition as before.

Table 7.8 shows the comparative results of SBTS$_{unique}$ (column 3-4) and SBTS (column 5-6). For each algorithm, we show the best result f_* followed by the average result f_{avg} (in parenthesis) and the average time in second $t(s)$. We observe that contrary to SBTS which finds all the best-known results except frb100-40 instance, SBTS$_{unique}$ fails to do so for 3 instances. SBTS has a perfect success rate of reaching the best-known result for 20 cases against 17 cases for SBTS$_{unique}$. This study shows the interest of the adopted tabu tenue technique and confirms the importance of tuning the tabu tenure carefully.

7.5 Conclusion

In this Appendix, we have presented SBTS, a general and unified swap-based tabu search algorithm for solving the maximum independent set problem. The proposed algorithm explores the search space by a dynamic alternation between intensification and diversification steps. The search process is driven by the $(k, 1)$-swap operator combined with specific rules to examine four different neighborhoods. For the purpose of intensification, SBTS uses (0,1)-swap to improve the solution and (1,1)-swap to make side-walks with specific selection rules. To overcome local optima, SBTS adopts an adaptive perturbation strategy which

applies either a (2,1)-swap for a weak perturbation or a (k,1)-swap ($k > 2$) for a strong perturbation. A tabu mechanism is also employed to prevent the search from short-term cycles.

We have tested the proposed SBTS on two sets of 120 well-known instances (DIMACS and BHOSLIB) with multiple topologies and densities. Computational results show that SBTS competes favorably with 5 state-of-the-art algorithms in the literature. In particular, SBTS can achieve the best-known results for all the 120 instances. An additional test of SBTS on a set of 11 instances from code theory has confirmed its competitiveness relative to two other reference methods.

Even though the proposed approach achieves competitive results on the three benchmarks, one observes that some best results can only be reached occasionally. More studies are needed to improve the stability and search capacity of the approach. One possibility would be to introduce multiple search strategies and apply them dynamically and adaptively according to learned guiding information. Another possibility would be to combine SBTS with the memetic search framework where a meaningful solution recombination mechanism must be sought.

List of figures

List of tables

List of publications

International journals

– Yan Jin and Jin-Kao Hao. Effective learning-based hybrid search for bandwidth coloring. Accepted to *IEEE Transactions on Systems, Man, and Cybernetics: Systems* Sept 2014.
DOI: http://dx.doi.org/10.1109/TSMC.2014.2360661

– Yan Jin and Jin-Kao Hao. General swap-based multiple neighborhood tabu search for finding maximum independent set. *Engineering Applications of Artificial Intelligence* 37: 20-33, 2015.

– Yan Jin, Jin-Kao Hao, Jean-Philippe Hamiez. A memetic algorithm for the minimum sum coloring problem. *Computers & Operations Research* 43(3): 318-327, 2014.

Submitted and on-going papers

– Yan Jin and Jin-Kao Hao. Hybrid evolutionary search for the minimum sum coloring problem of graphs. Submitted, Mar. 2015.

– Yan Jin, Jean-Philippe Hamiez and Jin-Kao Hao. A review on algorithms for the minimum sum coloring problem. Apr. 2015.

References

Aardal, K. I., Van Hoesel, S. P., Koster, A. M., Mannino, C., and Sassano, A. (2007). Models and solution techniques for frequency assignment problems. *Annals of Operations Research*, 153(1):79–129. (Cited on pages 11 and 25.)

Aicha, M., Malika, B., and Habiba, D. (2010). Two hybrid ant algorithms for the general T-colouring problem. *International Journal of Bio-Inspired Computation*, 2(5):353–362. (Cited on page 25.)

Allen, S. M., Smith, D. H., and Hurley, S. (2002). Generation of lower bounds for minimum span frequency assignment. *Discrete Applied Mathematics*, 119(1):59–78. (Cited on page 25.)

Andrade, D. V., Resende, M. G., and Werneck, R. F. (2012). Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18(4):525–547. (Cited on pages 96, 98, 108, 111, 112, and 120.)

Bar-Noy, A. and Kortsarz, G. (1998). Minimum color sum of bipartite graphs. *Journal of Algorithms*, 28(2):339–365. (Cited on page 19.)

Battiti, R. and Protasi, M. (2001). Reactive local search for the maximum clique problem. *Algorithmica*, 29(4):610–637. (Cited on page 96.)

Benlic, U. and Hao, J.-K. (2012). A study of breakout local search for the minimum sum coloring problem. *Simulated Evolution and Learning*, pages 128–137. (Cited on pages 17, 20, 21, 42, 46, 55, and 63.)

Benlic, U. and Hao, J.-K. (2013). Breakout local search for maximum clique problems. *Computers & Operations Research*, 40(1):192–206. (Cited on pages 96 and 109.)

Bernardo, F., Agustí, R., Pérez-Romero, J., and Sallent, O. (2010). An application of reinforcement learning for efficient spectrum usage in next-generation mobile cellular networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(4):477–484. (Cited on page 25.)

Bomze, I. M., Budinich, M., Pardalos, P. M., and Pelillo, M. (1999). The maximum clique problem. *Handbook of Combinatorial Optimization*, pages 1–74. (Cited on page 96.)

Bonomo, F., Durán, G., Marenco, J., and Valencia-Pabon, M. (2011). Minimum sum set coloring of trees and line graphs of trees. *Discrete Applied Mathematics*, 159(5):288–294. (Cited on page 19.)

Bouziri, H. and Jouini, M. (2010). A tabu search approach for the sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36:915–922. (Cited on pages 17, 20, and 21.)

Brélaz, D. (1979). New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256. (Cited on pages 19 and 30.)

Brunato, M. and Battiti, R. (2011). R-evo: a reactive evolutionary algorithm for the maximum clique problem. *IEEE Transactions on Evolutionary Computation*, 15(6):770–782. (Cited on page 96.)

Bui, T. N. and Nguyen, T. H. (2006). An agent-based algorithm for generalized graph colorings. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 19–26. ACM. (Cited on pages 25, 27, and 29.)

Butenko, S., Pardalos, P., Sergienko, I., Shylo, V., and Stetsyuk, P. (2009). Estimating the size of correcting codes using extremal graph problems. *Optimization*, pages 227–243. (Cited on page 111.)

Cai, S., Su, K., Luo, C., and Sattar, A. (2014). Numvc: An efficient local search algorithm for minimum vertex cover. *arXiv preprint arXiv:1402.0584*. (Cited on pages 96 and 109.)

Carraghan, R. and Pardalos, P. M. (1990). An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9(6):375–382. (Cited on page 96.)

Castelino, D., Hurley, S., and Stephens, N. (1996). A tabu search algorithm for frequency assignment. *Annals of Operations Research*, 63(2):301–319. (Cited on page 25.)

Chiarandini, M. and Stützle, T. (2007). Stochastic local search algorithms for graph set T-colouring and frequency assignment. *Constraints*, 12(3):371–403. (Cited on pages 83 and 86.)

Costa, D. (1993). On the use of some known methods for T-colorings of graphs. *Annals of Operations Research*, 41(4):343–358. (Cited on page 25.)

Di Gaspero, L. and Schaerf, A. (2006). Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modelling and Algorithms*, 5(1):65–89. (Cited on pages 39 and 40.)

Dorigo, M., Caro, G., and Gambardella, L. (1999). Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172. (Cited on page 28.)

Dorne, R. and Hao, J.-K. (1995). An evolutionary approach for frequency assignment in cellular radio networks. In *IEEE International Conference on Evolutionary Computation*, volume 2, pages 539–544. (Cited on page 25.)

Dorne, R. and Hao, J.-K. (1999). Tabu search for graph coloring, T-colorings and set T-colorings. In *Meta-Heuristics*, pages 77–92. Springer. (Cited on pages 25, 72, 78, 79, and 101.)

Douiri, S. M. and Elbernoussi, S. (2011). New algorithm for the sum coloring problem. *International Journal of Contemporary Mathematical Sciences*, 6(10):453–463. (Cited on page 17.)

Douiri, S. M. and Elbernoussi, S. (2012). A new ant colony optimization algorithm for the lower bound of sum coloring problem. *Journal of Mathematical Modelling and Algorithms*, 11(2):181–192. (Cited on page 17.)

Etzion, T. and Ostergard, P. R. (1998). Greedy and heuristic algorithms for codes and colorings. *IEEE Transactions on Information Theory*, 44(1):382–388. (Cited on page 111.)

Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133. (Cited on page 80.)

Fleurent, C. and Ferland, J. A. (1996). Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63(3):437–461. (Cited on page 54.)

Friden, C., Hertz, A., and de Werra, D. (1989). Stabulus: A technique for finding stable sets in large graphs with tabu search. *Computing*, 42(1):35–44. (Cited on page 96.)

Galinier, P. and Hao, J.-K. (1999). Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397. (Cited on pages 22, 36, 37, 54, 78, 79, and 101.)

Gamst, A. (1986). Some lower bounds for a class of frequency assignment problems. *IEEE Transactions on Vehicular Technology*, 35(1):8–14. (Cited on page 25.)

Garey, M. R. and Johnson, D. S. (2002). *Computers and intractability*, volume 29. WH Freeman. (Cited on page 23.)

Geng, X., Xu, J., Xiao, J., and Pan, L. (2007). A simple simulated annealing algorithm for the maximum clique problem. *Information Sciences*, 177(22):5064–5071. (Cited on page 96.)

Glover, F. and Laguna, M. (1999). Tabu search. *Handbook of Combinatorial Optimization*, pages 2093–2229. (Cited on pages 39, 41, 54, 73, 78, 97, and 101.)

Hajiabolhassan, H., Mehrabadi, M. L., and Tusserkani, R. (2000). Minimal coloring and strength of graphs. *Discrete Mathematics*, 215(1):265–270. (Cited on page 19.)

Hale, W. K. (1980). Frequency assignment: Theory and applications. *Proceedings of the IEEE*, 68(12):1497–1514. (Cited on page 25.)

Hamiez, J.-P. and Hao, J.-K. (2002). Scatter search for graph coloring. In *Artificial Evolution*, pages 168–179. Springer Berlin Heidelberg. (Cited on page 37.)

Hansen, P., Mladenović, N., and Urošević, D. (2004). Variable neighborhood search for the maximum clique. *Discrete Applied Mathematics*, 145(1):117–125. (Cited on pages 96 and 97.)

Hao, J.-K. (2012). Memetic algorithms in discrete optimization. In *Handbook of Memetic Algorithms*, pages 73–94. Springer. (Cited on page 36.)

Hao, J.-K. and Dorne, R. (1996). Study of genetic search for the frequency assignment problem. In *Artificial Evolution*, pages 333–344. Springer. (Cited on page 25.)

Hao, J.-K., Dorne, R., and Galinier, P. (1998). Tabu search for frequency assignment in mobile radio networks. *Journal of Heuristics*, 4(1):47–62. (Cited on page 25.)

Haralick, R. M. and Elliott, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313. (Cited on page 75.)

Helmar, A. and Chiarandini, M. (2011). A local search heuristic for chromatic sum. In *Proceedings of the 9th Metaheuristics International Conference*, pages 161–70. (Cited on pages 17, 20, 21, 23, 44, 46, 57, and 60.)

Hertz, A. and de Werra, D. (1987). Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351. (Cited on pages 36, 56, and 78.)

Jansen, K. (2000). Approximation results for the optimum cost chromatic partition problem. *Journal of Algorithms*, 34(1):54–89. (Cited on page 19.)

Jiang, T. and West, D. B. (1999). Coloring of trees with minimum sum of colors. *arXiv preprint math/9904140*. (Cited on page 19.)

Jin, Y. and Hao, J.-K. (2015a). Effective learning-based hybrid search for bandwidth coloring. Accepted to IEEE Transactions on Systems, Man, and Cybernetics: Systems Sept 2014, DOI: http://dx.doi.org/10.1109/TSMC.2014.2360661. (Cited on pages 13, 25, 28, 29, and 71.)

Jin, Y. and Hao, J.-K. (2015b). General swap-based multiple neighborhood tabu search for the maximum independent set problem. *Engineering Applications of Artificial Intelligence*, 37:20–33. (Cited on pages 13, 51, and 95.)

Jin, Y. and Hao, J.-K. (2015c). Hybrid search for upper and lower bounds of the minimum sum coloring problem. Submitted for publication. (Cited on pages 13, 17, 21, 22, 23, and 49.)

Jin, Y., Hao, J.-K., and Hamiez, J.-P. (2014). A memetic algorithm for the minimum sum coloring problem. *Computers & Operations Research*, 43:318–327. (Cited on pages 12, 17, 21, 22, 35, 56, and 63.)

Johnson, D. S. and Garey, M. R. (1979). Computers and intractability: A guide to the theory of NP-completeness. *Freeman, San Francisco, New York*, page 32. (Cited on page 96.)

Johnson, D. S., Mehrotra, A., and Trick, M. A. (2002). Color02/03/04: Graph coloring and its generalizations. (Cited on pages 12, 30, and 80.)

Johnson, D. S., Mehrotra, A., and Trick, M. A. (2008). Special issue on computational methods for graph coloring and its generalizations. *Discrete Applied Mathematics*, 156(2):145–146. (Cited on page 25.)

Johnson, D. S. and Trick, M. A. (1996). *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*. American Mathematical Soc. (Cited on pages 12, 96, and 105.)

Jones, T. and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *ICGA*, volume 95, pages 184–192. Citeseer. (Cited on pages 65 and 67.)

Karp, R. M. (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103. (Cited on pages 11 and 96.)

Katayama, K., Hamamoto, A., and Narihisa, H. (2005). An effective local search for the maximum clique problem. *Information Processing Letters*, 95(5):503–511. (Cited on page 96.)

Kokosiński, Z. and Kwarciany, K. (2007). On sum coloring of graphs with parallel genetic algorithms. *Adaptive and Natural Computing Algorithms*, pages 211–219. (Cited on pages 17 and 22.)

Kroon, L. G., Sen, A., Deng, H., and Roy, A. (1997). The optimal cost chromatic partition problem for trees and interval graphs. In *Graph-Theoretic Concepts in Computer Science*, pages 279–292. Springer. (Cited on pages 17, 19, and 23.)

Kubicka, E. (1989). *The chromatic sum of a graph*. PhD thesis, Western Michigan University. (Cited on pages 11 and 17.)

Kubicka, E. (2004). The chromatic sum of a graph: History and recent developments. *International Journal of Mathematics and Mathematical Sciences*, 2004(30):1563–1573. (Cited on page 18.)

Kubicka, E. (2005). Polynomial algorithm for finding chromatic sum for unicyclic and outerplanar graphs. *Ars Combinatoria 76*. (Cited on page 17.)

Kubicka, E., Kubicki, G., and Kountanis, D. (1991). Approximation algorithms for the chromatic sum. In *Computing in the 90's*, pages 15–21. Springer. (Cited on page 19.)

Kubicka, E. and Schwenk, A. J. (1989). An introduction to chromatic sums. In *Proceedings of the 17th Conference on ACM Annual Computer Science Conference*, pages 39–45. ACM. (Cited on page 17.)

Lai, X. and Lü, Z. (2013). Multistart iterated tabu search for bandwidth coloring problem. *Computers & Operations Research*, 40(5):1401–1409. (Cited on pages 25, 28, 29, 30, 72, 78, 82, 83, and 86.)

Lai, X., Lü, Z., Hao, J.-K., Glover, F., and Xu, L. (2014). Path relinking for bandwidth coloring problem. CoRR abs/1409.0973. (Cited on pages 25, 29, 30, 81, 82, 83, and 86.)

Leighton, F. T. (1979). A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506. (Cited on page 19.)

Li, C.-M. and Quan, Z. (2010). Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 2010 22nd*, volume 1, pages 344–351. IEEE. (Cited on page 96.)

Li, Y., Lucet, C., Moukrim, A., and Sghiouer, K. (2009). Greedy algorithms for the minimum sum coloring problem. *Logistique et Transports*. (Cited on pages 17, 19, 21, 44, 46, and 56.)

Lim, A., Zhu, Y., Lou, Q., and Rodrigues, B. (2005). Heuristic methods for graph coloring problems. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, pages 933–939. ACM. (Cited on pages 25, 27, 29, and 83.)

Lourenço, H. R., Martin, O. C., and Stutzle, T. (2001). Iterated local search. *arXiv preprint math/0102188*. (Cited on page 97.)

Lü, Z. and Hao, J.-K. (2010). A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241–250. (Cited on pages 36, 38, and 55.)

Lü, Z., Hao, J.-K., and Glover, F. (2011). Neighborhood analysis: a case study on curriculum-based course timetabling. *Journal of Heuristics*, 17(2):97–118. (Cited on pages 39 and 40.)

Malafiejski, M. (2004). Sum coloring of graphs. *Graph Colorings*, 352:55–65. (Cited on pages 17 and 19.)

Malaguti, E. (2009). The vertex coloring problem and its generalizations. *4OR*, 7(1):101–104. (Cited on page 26.)

Malaguti, E., Monaci, M., and Toth, P. (2008). A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2):302–316. (Cited on page 36.)

Malaguti, E. and Toth, P. (2008). An evolutionary approach for bandwidth multicoloring problems. *European Journal of Operational Research*, 189(3):638–651. (Cited on pages 25, 29, 30, 72, 82, 83, and 86.)

Malaguti, E. and Toth, P. (2010). A survey on vertex coloring problems. *International Transactions in Operational Research*, 17(1):1–34. (Cited on page 27.)

Marti, R., Gortazar, F., and Duarte, A. (2010). Heuristics for the bandwidth colouring problem. *International Journal of Metaheuristics*, 1(1):11–29. (Cited on pages 28 and 29.)

McCreesh, C. and Prosser, P. (2013). Multi-threading a state-of-the-art maximum clique algorithm. *Algorithms*, 6(4):618–635. (Cited on page 105.)

Moscato, P. and Cotta, C. (2003). A gentle introduction to memetic algorithms. In *Handbook of Metaheuristics*, pages 105–144. Springer. (Cited on pages 36 and 50.)

Moukrim, A., Sghiouer, K., Lucet, C., and Li, Y. (2010). Lower bounds for the minimal sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36:663–670. (Cited on pages 17, 23, 57, and 60.)

Moukrim, A., Sghiouera, K., Lucetb, C., and Li, Y. (2013). Upper and lower bounds for the minimum sum coloring problem. https://www.hds.utc.fr/~moukrim/dokuwiki/doku.php?id=en:mscp. (Cited on pages 17, 21, 22, 23, 42, 44, 46, 56, 57, 60, and 63.)

Neri, F., Cotta, C., and Moscato, P. (2012). *Handbook of memetic algorithms*, volume 379. Springer. (Cited on pages 36 and 50.)

Östergård, P. R. (2002). A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1):197–207. (Cited on page 96.)

Porumbel, D. C., Hao, J.-K., and Kuntz, P. (2010). An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers & Operations Research*, 37(10):1822–1832. (Cited on pages 36, 37, 41, and 55.)

Prestwich, S. (2002). Coloration neighbourhood search with forward checking. *Annals of Mathematics and Artificial Intelligence*, 34(4):327–340. (Cited on pages 28 and 30.)

Prestwich, S. (2008). Generalised graph colouring by a hybrid of local search and constraint programming. *Discrete Applied Mathematics*, 156(2):148–158. (Cited on pages 25, 28, 29, 80, 82, 83, and 86.)

Pullan, W. (2006). Phased local search for the maximum clique problem. *Journal of Combinatorial Optimization*, 12(3):303–323. (Cited on pages 96 and 109.)

Pullan, W. (2008). Approximating the maximum vertex/edge weighted clique using local search. *Journal of Heuristics*, 14(2):117–134. (Cited on pages 96 and 109.)

Pullan, W., Mascia, F., and Brunato, M. (2011). Cooperating local search for the maximum clique problem. *Journal of Heuristics*, 17(2):181–199. (Cited on pages 96 and 110.)

Richter, S., Helmert, M., and Gretton, C. (2007). A stochastic local search approach to vertex cover. *KI 2007: Advances in Artificial Intelligence*, pages 412–426. (Cited on pages 96 and 109.)

Roberts, F. S. (1991). T-colorings of graphs: recent results and open problems. *Discrete Mathematics*, 93(2):229–245. (Cited on page 25.)

Salavatipour, M. R. (2003). On sum coloring of graphs. *Discrete Applied Mathematics*, 127(3):477–488. (Cited on pages 17 and 19.)

Salcedo-Sanz, S., Santiago-Mozos, R., and Bousoño-Calzón, C. (2004). A hybrid hopfield network-simulated annealing approach for frequency assignment in satellite communications systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(2):1108–1116. (Cited on page 25.)

San Segundo, P., Rodríguez-Losada, D., and Jiménez, A. (2011). An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research*, 38(2):571–581. (Cited on page 96.)

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., and Dueck, G. (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171. (Cited on page 97.)

Sen, A., Deng, H., and Guha, S. (1992). On a graph partition problem with application to vlsi layout. *Information Processing Letters*, 43(2):87–94. (Cited on pages 17 and 19.)

Sloane, N. (2000). Challenge problems: Independent sets in graphs. (Cited on page 108.)

Sörensen, K. and Sevaux, M. (2006). MA|PM: Memetic algorithms with population management. *Computers & Operations Research*, 33(5):1214–1225. (Cited on page 55.)

Supowit, K. J. (1987). Finding a maximum planar subset of a set of nets in a channel. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, pages 93–94. (Cited on pages 11 and 17.)

Tesman, B. A. (1989). *T-colorings, list T-colorings, and set T-colorings of graphs*. UMI. (Cited on page 25.)

Thomassen, C., Erdös, P., Alavi, Y., Malde, P. J., and Schwenk, A. J. (1989). Tight bounds on the chromatic sum of a connected graph. *Journal of Graph Theory*, 13(3):353–357. (Cited on page 22.)

Tomita, E. and Kameda, T. (2007). An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization*, 37(1):95–111. (Cited on page 96.)

Walser, J. P. (1996). Feasible cellular frequency assignment using constraint programming abstractions. In *Proceedings of the Workshop on Constraint Programming Applications, in conjunction with the Second International Conference on Principles and Practice of Constraint Programming (CP96)*. (Cited on page 25.)

Wang, L., Liu, W., and Shi, H. (2008). Noisy chaotic neural networks with variable thresholds for the frequency assignment problem in satellite communications. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):209–217. (Cited on page 25.)

Wang, Y., Hao, J.-K., Glover, F., and Lü, Z. (2013). Solving the minimum sum coloring problem via binary quadratic programming. *arXiv preprint arXiv:1304.5876*. (Cited on pages 17, 19, and 22.)

Wu, Q. and Hao, J.-K. (2012). An effective heuristic algorithm for sum coloring of graphs. *Computers & Operations Research*, 39(7):1593–1600. (Cited on pages 17, 19, 21, 42, 46, 60, 63, and 92.)

Wu, Q. and Hao, J.-K. (2013a). An adaptive multistart tabu search approach to solve the maximum clique problem. *Journal of Combinatorial Optimization*, 26(1):86–108. (Cited on pages 96 and 101.)

Wu, Q. and Hao, J.-K. (2013b). Improved lower bounds for sum coloring via clique decomposition. CoRR abs/1303.6761. (Cited on pages 17, 23, 57, and 60.)

Wu, Q. and Hao, J.-K. (2015). A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709. (Cited on pages 96 and 105.)

Wu, Q., Hao, J.-K., and Glover, F. (2012). Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, 196(1):611–634. (Cited on pages 96 and 109.)

Xu, K., Boussemart, F., Hemery, F., and Lecoutre, C. (2005). A simple model to generate hard satisfiable instances. *arXiv preprint cs/0509032*. (Cited on page 105.)

Zhang, Q., Sun, J., and Tsang, E. (2005). An evolutionary algorithm with guided mutation for the maximum clique problem. *IEEE Transactions on Evolutionary Computation*, 9(2):192–200. (Cited on page 96.)

Zoellner, J. A. and Beall, C. L. (1977). A breakthrough in spectrum conserving frequency assignment technology. *IEEE Transactions on Electromagnetic Compatibility*, 19(3):313–319. (Cited on page 25.)

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

université angers

Thèse de Doctorat

Yan JIN

Hybrid metaheuristic algorithms for sum coloring and bandwidth coloring

Métaheuristiques hybrides pour la somme coloration et la coloration de bande passante

Résumé

Le problème de somme coloration minimum (MSCP) et le problème de coloration de bande passante (BCP) sont deux généralisations importantes du problème de coloration des sommets classique avec de nombreuses applications dans divers domaines, y compris la conception de circuits imprimés, la planication, l'allocation de ressource, l'affectation de fréquence dans les réseaux mobiles, etc. Les problèmes MSCP et BCP étant NP-difficiles, les heuristiques et métaheuristiques sont souvent utilisées en pratique pour obtenir des solutions de bonne qualité en un temps de calcul acceptable. Cette thèse est consacrée à des métaheuristiques hybrides pour la résolution efcace des problèmes MSCP et BCP. Pour le problème MSCP, nous présentons deux algorithmes mémétiques qui combinent l'évolution d'une population d'individus avec de la recherche locale. Pour le problème BCP, nous proposons un algorithme hybride à base d'apprentissage faisant coopérer une méthode de construction "informée" avec une procédure de recherche locale. Les algorithmes développés sont évalués sur des instances biens connues et se révèlent très compétitifs par rapport à l'état de l'art. Les principaux composants des algorithmes que nous proposons sont également analysés.

Mots clés

Somme coloration de graphe, Coloration de bande passante, Stable maximum, Optimisation combinatoire, Métaheuristiques, Recherche tabou, Algorithme hybride.

Abstract

The minimum sum coloring problem (MSCP) and the bandwidth coloring problem (BCP) are two important generalizations of the classical vertex coloring problem with numerous applications in diverse domains, including VLSI design, scheduling, resource allocation and frequency assignment in mobile networks, etc. Since the MSCP and BCP are NP-hard problems, heuristics and metaheuristics are practical solution methods to obtain high quality solutions in an acceptable computing time. This thesis is dedicated to developing effective hybrid metaheuristic algorithms for the MSCP and BCP. For the MSCP, we present two memetic algorithms which combine population-based evolutionary search and local search. An effective algorithm for maximum independent set is devised for generating initial solutions. For the BCP, we propose a learning-based hybrid search algorithm which follows a cooperative framework between an informed construction procedure and a local search heuristic. The proposed algorithms are evaluated on well-known benchmark instances and show highly competitive performances compared to the current state-of-the-art algorithms from the literature. Furthermore, the key issues of these algorithms are investigated and analyzed.

Key Words

Graph sum coloring, Bandwidth coloring, Maximum independent set, Combinatorial optimization, Metaheuristics, Tabu search, Hybrid algorithm.