Recent Advances in Graph Vertex Coloring

Philippe Galinier, Jean-Philippe Hamiez, Jin-Kao Hao, and Daniel Porumbel

Abstract Graph vertex coloring is one of the most studied NP-hard combinatorial optimization problems. Given the hardness of the problem, various heuristic algorithms have been proposed for practical graph coloring, based on local search, population-based approaches and hybrid methods. The research in graph coloring heuristics is very active and improved results have been obtained recently, notably for coloring large and very large graphs. This chapter surveys and analyzes graph coloring heuristics with a focus on the most recent advances.

1 Introduction

The graph coloring problem is one of the most important and the most studied problems in combinatorial optimization. The problem has applications in many domains, such as timetabling and scheduling, frequency assignment, register allocation, routing and wavelenth assigment, and many others.

In the following, we consider a non-oriented graph $G = (V,E)$, with a set V of n vertices and a set E of m edges ($d = \frac{2m}{n(n-1)}$ defines the *density* of G). Given an integer k, a k-coloring c is a function that assigns to each vertex v of the graph an integer $c(v)$ chosen in set $\{1,2,...,k\}$ (the set of colors), all vertices colored the same defining a "color class". A k-coloring c is a *proper* coloring if the endpoints of

Philippe Galinier
CRT – École Polytechnique, 3 000 chemin de la Côte Sainte-Catherine, Montréal, Québec, Canada, e-mail: philippe.galinier@polymtl.ca

Jean-Philippe Hamiez · Jin-Kao Hao
LERIA – Université d'Angers, 2 boulevard Lavoisier, 49 045 Angers CEDEX 01, France, e-mail: {hamiez,hao}@info.univ-angers.fr

Daniel Porumbel
LGI2A – Université d'Artois, Technoparc Futura, 62 400 Béthune CEDEX, France, e-mail: daniel.porumbel@univ-artois.fr

any edge are assigned different colors. A graph is *k*-colorable if it admits a proper *k*-coloring. The chromatic number $\chi(G)$ of a given graph *G* is the smallest integer *k* for which *G* is *k*-colorable. A proper *k*-coloring such that $k = \chi(G)$ is named an *optimal* coloring. The *graph coloring* problem is the problem to find an optimal coloring of a given graph. For a given graph *G* and a given integer *k*, the *k-coloring* problem is the problem to determine if *G* is *k*-colorable and, if it is the case, to find a proper *k*-coloring of *G*.

Graph coloring is NP-hard and *k*-coloring is NP-complete for any integer $k \geq 3$ (but 2-coloring is polynomial) [23, 40]. Therefore, no algorithm can solve graph coloring in polynomial time in the general case (assuming that N $\neq$ NP). In addition, note that the problem to find a *k*-coloring with no more than twice the optimal number of colors is still NP-hard. Also, finding an optimal coloring turns out to be particularly difficult in practice. As a matter of fact, there are graphs with as few as 125 vertices that can not be solved optimally even by using the best performing exact algorithms. For larger graphs, it is therefore necessary to resort to *heuristics*, i.e., algorithmic techniques that provide sub-optimal solutions within an acceptable amount of time.

A simple and classical way to generate a sub-optimal proper coloring is to use a greedy heuristic. A greedy coloring heuristic builds a solution step by step by fixing on each step the color of a given vertex. In the *greedy sequential heuristic*, the order in which the vertices are colored is determined beforehand (i.e., statically), randomly or according to a specific criterion. On each step, the considered vertex is assigned the smallest possible color number such that no conflict is created with the already colored vertices. More efficient greedy heuristics like DSATUR [5] and Recursive Largest First (RLF) [42] employ refined rules to dynamically determine the next vertex to color.

Greedy heuristics are generally fast, but they tend to need much more colors than the chromatic number to color a graph. Better results can be obtained by using much more powerful heuristics, such as notably *local search heuristics* and *evolutionary algorithms* – a survey of heuristics and metaheuristics is proposed in [4]. In fact, almost all of the most popular metaheuristics have also been applied and implemented on the graph coloring problem. From a historical point of view, simulated annealing and tabu search are among the first local search approaches that were successfully applied to graph coloring. In local search, a candidate solution is progressively improved by local transformations (named "moves"). Population-based approaches (e.g., memetic algorithms, quantum annealing) represent another family of heuristics that established themself as one of the most effective coloring methods. Thanks to the introduction of a pool of solutions, powerful search operators are made available like various solution recombinations and hybridization with local search. Finally, a latest approach based on independent set extraction and progressive coloring proves to be quite useful to color very large graphs.

In the following sections, we present the different coloring heuristics grouped in four categories: Local search heuristics in Sect. 3, evolutionary algorithms in Sect. 4, independent set extraction approaches in Sect. 5, and other heuristics (such as quantum annealing, neural networks, ant colonies, etc.) in Sect. 6. Then, we review ex-

tensions of graph coloring problems and applications related to graph coloring in Sect. 7. Finally, Sect. 8 reports on standard benchmark graphs along with the results obtained by the best performing coloring heuristics on these graphs.

2 General Solution Strategies Applicable to Coloring Problems

It is important to note that the search space and the evaluation function used by a heuristic are not necessarily the same as those of the original problem. In the following, we name "strategy" a particular way to define the search space and the evaluation function. Thus, a strategy can be seen as a reformulation of the original problem. Solution strategies designed for k-coloring are presented below.

- In the *k-fixed penalty strategy* [33, 39], a configuration is any (non necessarily proper) k-coloring and its cost is the number of conflicts – i.e., edges whose endpoints are assigned the same color. This strategy makes it possible to use a very simple type of move, named "1-moves". A 1-move consists in changing the color of a single vertex.
- In the *k-fixed partial proper strategy* (named *Impasse neighborhood* in [50]), a configuration is any partial (proper) k-coloring – a partial proper k-coloring is similar to a proper k-coloring except that some vertices may remain uncolored. The cost of a given configuration is the number of unassigned vertices – it can also be the sum of the degrees of unassigned vertices. A move (named "i-swap") consists in assigning a color i to an uncolored vertex v and, at the same time, to deassign all neighbors of v whose color is i in the current configuration.

A k-coloring heuristic can also be used to solve the graph coloring problem. This can be done as follows: First, find a proper k-coloring by using a greedy heuristic. Then apply the k-coloring heuristic and reduce the value of k each time a proper k-coloring is found. Strategies designed to tackle directly the graph coloring problem are briefly presented below.

- In the *proper strategy* [39], a configuration is any proper k-coloring. The evaluation function is designed so as to promote the variance of the size of the different color classes and, as a side-effect, to reduce their number. A move (named a "Kempe chain") consists in making an exchange of vertices between two color classes, while preserving the legality of the solution. See [39] for more details.
- In the *penalty strategy*, a configuration is any k-coloring. The evaluation function is designed so as to decrease both the number of conflicts and the number of colors, see [39]. The moves used in this strategy are 1-moves (defined above).
- In the *order-based strategy*, a configuration is any ordering of the vertices, which is subsequently "decoded" (i.e., transformed into a proper coloring) by using the greedy sequential heuristic [13].
- In the *edge orienting strategy*, a configuration is any orientation of the edges of the input graph, and its cost is the length of a longest path in the resulting digraph. Types of move mechanisms applicable to this strategy are presented in [24].

3 Local Search Heuristics

Local search is a simple yet very powerful approach. The most basic local search heuristic is iterative improvement. An *iterative improvement* procedure (or *descent*) consists in choosing on each iteration a move that produces a decrease in the cost function until a *local optimum* is reached. Unfortunately, iterative improvement may get trapped rapidly in a poor local optimum. Advanced local search heuristics resort to different kinds of mechanisms in order to circumvent the problem posed by local optima. For example, *simulated annealing* relies on randomness in order to allow "uphill" moves (i.e., moves that increase the cost function), thanks to the *temperature* parameter. *Tabu search* uses a short term diversification mechanism named a tabu list. *Iterated local search* and *variable neighborhood search* combine descents and perturbations (equivalent to jumps performed in the search space). Prominent local search heuristics proposed for coloring graphs will be presented and analyzed in the remaining of this section.

3.1 Local Search Heuristics Proposed for Coloring Graphs

Two heuristics based on the *k*-fixed penalty strategy were proposed in 1987: A simulated annealing algorithm in [9] and a tabu algorithm named TABUCOL in [33]. The tabu mechanism used in this latter algorithm is as follows: After performing a move (i.e., changing the color of a vertex), it is forbidden to reassign to this vertex its former color for a few iterations. In spite of its simplicity, this technique turned out to be remarkably efficient. In particular TABUCOL outperformed the simulated annealing heuristic proposed in [9].

An exhaustive experimental study is conducted in [39] in order to evaluate and compare several graph coloring heuristics, notably three simulated annealing heuristics based on three different strategies. Note that none of those strategies (the *k*-fixed penalty strategy, the proper strategy, and the penalty strategy) dominated clearly the others in these experiments.

One of the first and most efficient local search coloring heuristic is the simulated annealing algorithm proposed in [50] – as its temperature parameter remains constant, this algorithm is named a Metropolis algorithm. This Metropolis algorithm is based on the *k*-fixed partial proper strategy and it combines two types of moves: *i*-swaps and *s*-chains – a *s*-chain is a generalization of Kempe chains, see [50]. More recently, a tabu heuristic (named PARTIALCOL) also based on the *k*-fixed partial proper strategy has been developed in [3]. Experiments conducted with TABUCOL and PARTIALCOL indicate that, although they are based on different strategies, the two tabu algorithms obtain similar results on most graphs.

3.2 Recent Advances

Several authors have tried to improve the efficiency of a tabu coloring heuristic, notably by a more careful management of the tabu list. The original TABUCOL uses a static tabu list (i.e., the tabu tenure is constant) [33]. However, it can be noticed that the size of the neighborhood changes throughout the search. For this reason, it is suggested in [20] to set the tabu tenure accordingly. A still more sophisticated and robust technique proposed in [3] allows to tune automatically the tabu tenure throughout the search.

A different idea explored in [55] makes it possible to improve the efficiency of TABUCOL by modifying its evaluation function. While any edge violation entails the same cost in the original algorithm, a specific penalty is assigned to each edge. Two new evaluation functions are proposed in [55]. In the first one, the penalty assigned to a given edge depends on the degree of its endpoints – edges whose endpoints have a larger degree (assumed to be more difficult to satisfy) are assigned a larger penalty. In the second one, the penalty of each edge is dynamically adjusted during the search.

In [56], it is proposed to guide a local search operator throughout the search space by measuring distances between solutions. Two algorithms that exploit this idea are presented. The first one uses a learning process to guide a basic tabu operator (TABUCOL) toward yet unvisited spheres. The second algorithm makes deep investigations within a bounded region by organizing it as a tree-like structure of connected spheres. Experiments show that these algorithms outperform the underlying tabu algorithm and obtain remarkable results.

The iterated local search and variable neighborhood search metaheuristics have also been investigated for coloring graphs. The variable neighborhood search algorithm proposed in [2] uses TABUCOL as a local search operator and applies different types of perturbation operators. Also, an iterated local search algorithm is presented in [10].

A new local search coloring heuristic, named "Variable Space Search" (VSS-Col), is proposed in [34]. The VSS-Col heuristic alternates the use of three different local search heuristics that adopt different strategies: TABUCOL, PARTIALCOL, and a third tabu algorithm (detailed in [24]), which is not competitive by itself but complements adequately the two others. This algorithm obtains better results than each of its components and produces some of the best results obtained with local search heuristics.

Several other graph coloring local search heuristics have been proposed recently. In particular, large-scale neighborhoods are explored in [69].

3.3 Discussion

To finish with local search, we underline three important points:

- *The speed of a graph coloring local search heuristic depends highly on its implementation.* A local search heuristic must determine on each iteration the performance of a large number of moves – i.e., its impact on the increase or decrease of the cost of the configuration. It is therefore important to compute efficiently the performance of each move. This can be done by using an incremental technique such as the one described in [19, 21];
- *Local search is a key ingredient for coloring graphs.* Most efficient coloring heuristics developed so far use local search. They are either "pure" local search heuristics, or they use a local search operator as one of their components. In particular, greedy algorithms, "pure" genetic algorithms (see Sect. 4), or neural networks (see Sect. 6) are generally unable to compete with heuristics that use local search;
- *Local search is not sufficient to efficiently color large graphs.* "Pure" local search heuristics are well-suited for the coloring of small and medium-sized graphs, but their results are often far from the optimum when it comes to coloring large graphs (graphs having 500 vertices or more). For large graphs, better solutions can be obtained by using the independent set extraction approach (see Sect. 5) or evolutionary algorithms using specialized recombination operators (see Sect. 4.1).

4 The Evolutionary Approach

Evolutionary algorithms have long been a popular and successful optimization approach for graph coloring. The first algorithms based on genetic and evolutionary principles were developed in the early 1990s [13]. Since then, important progress has been made and a number of specific coloring techniques have been established. For instance, it is now generally accepted that traditional uniform crossovers are outperformed by *class-oriented* crossovers – i.e., crossovers that construct offspring by "blending" *color classes* from parents (see Sect. 4.1). Other recent developments concern population diversity issues, the exploitation-exploration balance, population dynamics (mating and survival selection), hybridization with exact algorithms (see Sect. 4.2), etc. However, let us first present a number of general principles that are used by most evolutionary coloring algorithms.

As opposed to local search (Sect. 3), the evolutionary approach is based on a population (pool) of several different individuals (colorings) that are improved via an "evolutionary" process of optimization. There exists a large number of evolutionary paradigms (evolution strategies, genetic algorithms, evolutionary programming, scatter search, etc) and they are generally inspired by Darwinian principles, as for example: "Survival of the fittest" (e.g., remove the less fit low-quality individuals), preferential mating (higher quality individuals are more easily allowed to become parents), recombination or crossover (the selected parents are recombined to generate offspring), adaptation and evolution (e.g., by local search and optimization).

The most popular evolutionary algorithms for graph coloring are the classical *steady-state* genetic algorithms. These coloring algorithms often use local search, and so, the approach can also be regarded as an instance of *memetic* computing [51]. The population (referred to as **Pop**) is defined as a set of "individuals" $\mathbf{Indiv}_1, \mathbf{Indiv}_2, \ldots \mathbf{Indiv}_{|\mathbf{Pop}|}$. Each individual represents a k-coloring (with k fixed or not) that is evaluated according to a fitness function – in the evolutionary terminology, the objective function from optimization is referred to as the fitness function. At each "generation", two or more parent individuals are selected and recombined to generate offspring solutions. This offspring can be improved via local search and then replace some individuals selected for elimination. The generic schema of this genetic process is described below.

1: Initialize parent population $\mathbf{Pop} \leftarrow \{\mathbf{Indiv}_1, \mathbf{Indiv}_2, \ldots, \mathbf{Indiv}_{|\mathbf{Pop}|}\}$
2: **while** a stopping condition is not met **do**
3: $\quad$ *parents* $\leftarrow$ matingSelection(**Pop**)
4: $\quad$ **Offspr** $\leftarrow$ recombination(*parents*)
5: $\quad$ **Offspr** $\leftarrow$ localSearch(**Offspr**, *maxIter*)
6: $\quad$ $R \leftarrow$ eliminationSelection(**Pop**)
7: $\quad$ $\mathbf{Pop} \leftarrow \mathbf{Pop} \cup \{\mathbf{Offspr}\} - R$
8: **end while**
9: Return the fittest individual ever visited

The last decade has seen a surge of interest in integrating local search in genetic coloring algorithms (see step 5 in the above schema), making the memetic approach [51] more and more popular. Besides classical genetic algorithms, even more particular evolutionary paradigms (crossover-free distributed search [8, 49], scatter search [29] or adaptive memory algorithms [22]) do incorporate a local search coloring routine (often based on TABUCOL [33] or on the k-fixed partial proper strategy [49]).

In this section, we discuss some key issues of evolutionary coloring algorithms like crossover design, population dynamics and diversity, hybridizations with other search methods.

4.1 Specialized Recombination Operators

The recombination is a crucial concept in many areas of evolutionary computing and defines the way information is transmitted from parents to offspring. A challenging issue is to make this "inheritance" process as meaningful for the problem as possible, i.e., so as to preserve the good features of the parents and disrupts the bad ones [60]. Recombination ideas in earlier work [13, 19] were based on the standard *uniform* crossover or on the *order-based* uniform crossover.

The first of the above crossovers is based on a straightforward array encoding that associate vertices to integer colors. In principle, the offspring solution is constructed by assigning to each vertex $v \in V$ either the color of v in the first parent or the color of v in the second parent. Regarding the second crossover (order-based),

it does not encode individuals directly as colorings but (indirectly) as permutations of V. A permutation indicates an order of vertices that can be decoded into an actual coloring by greedily assigning colors one by one in this order. Since indirect encodings lie outside the scope of this chapter, we refer the reader to [52] for interesting recombination issues in this context. However, according to [16, 19], both early uniform crossovers failed to make the evolutionary search reach significantly better results that local search.

An important breakthrough in evolutionary graph coloring is represented by the first crossover models that manipulate *color classes* instead of *color values*. As such, the Greedy Partition Crossover (GPX) [20] established genetic algorithms as one of the most competitive methods for graph coloring. In this approach, an individual is seen as a partition of V (into k classes). The parents transmit classes to the offspring in an alternate manner until the offspring receives k classes. Any remaining uncolored vertex is assigned a random color (some studies suggest selecting this color using a greedy criterion). This class-based approach requires a very low computing time compared to the number of local search iterations. It seems to be very useful for many other combinatorial optimization problems called "grouping problems" [18].

During the last few years, a number of new developments and improvements of GPX have been presented. For instance, the "GHZ" recombination operator [22] combines conflict-free classes (independent sets) from all individuals of the population. An interesting issue of the class-oriented approach is the risk of inheriting *too many* classes from only one parent. To overcome such risks, GPX considers the two parents *alternately* when deciding which parent to transmit classes. Other studies propose using several parents and forbidding each transmitting parent to re-transmit for a number of generations [43].

In order to minimize the number of vertices that remain uncolored in the end (recall that only k classes are transmitted to the offspring), the above crossovers always transmit classes of maximum size (this is why they are called "Greedy"). However, this criterion (maximum size) can be refined to more meaningful measures [57]. For instance, one can consider that a large class with conflicts could be less useful than a slightly smaller class with no conflicts. The number of parents has also been a topic of interest but several studies seem to indicate that using many parents does not always improve results. For instance, `MACOL` [43] uses a random number between 2 and 6. In [57], certain epistasis arguments (the analysis of the impact of the interactions between classes) argue that more parents are only useful for instances with large class size (e.g., 50).

One should be aware that the large graph coloring literature also contains references to other types of crossover operators, based on unifying pairs of conflict-free sub-classes [14], on sexual reproduction using a splitting of the graph into two parts [48], distance-preserving crossovers [63], etc.

4.2 Other Evolutionary Advances and Algorithms

Besides recombination operators, research in graph coloring has also led to several other advances in understanding genetic algorithms. A general challenge in genetic algorithms is to make the algorithm reach a proper balance between two conflicting aspects of heuristic search: (i) Exploration – typically assured via crossover or other population-based operators and (ii) exploitation – often represented by local search. The local search exploitation potential explains its success in improving genetic algorithms: A local search routine is able to find higher-quality individuals in the proximity of lower-quality offspring generated by recombination. The balance question is naturally formulated: What is the best way to allocate a given amount of computing time between exploration (crossover) and exploitation (local search).

This balance can be controlled by the maximum number of iterations that are allowed after each crossover (*maxIter* at step 5 in the algorithmic schema at page 7). At a first glance, it might seem that more local search iterations leads to more exploitation and less exploration. However, certain papers [20, 22] indicate that a longer local search routine can actually improve the exploration capacity of the search process and preserve diversity – more local search iterations can scatter the offspring to more distant and diverse areas. As such, most implementations spend (considerably) less computing time on crossover or selection routines (usually linear in n) than on local search iterations (*maxIter* can be an order of magnitude higher than n). Finally, notice that this local search operator does not need to be very complex: Even very rudimentary local search methods (steepest descent) could lead genetic algorithms to good performances under certain conditions [12, 26].

A related central topic in genetic algorithms is maintaining population diversity – a recurrent issue in genetic algorithms *in general* that can be approached from numerous perspectives. Very often, the general diversity objective is to control the population dynamics using so-called "population management" schemes: By creating diversity-oriented subpopulations, by proposing better diversity-guided elimination rules, new offspring rejection mechanisms, by adapting the crossover, etc. In some cases, even the classical genetic algorithm template can be modified so as to adapt to certain diversity objectives. For instance, the scatter-search evolutionary paradigm is used in [29]: A "diversity-oriented" subpopulation is specifically generated so as to keep reference to very diverse individuals. The algorithm proposed in [22] uses an adaptive memory that contains a number of independent sets instead of a population of individuals. When a set-based diversity measure falls below a certain threshold, the algorithm triggers a "diversifying" action by allowing more local search.

If one wants to stay in the classical genetic algorithm template, diversity can still be achieved by modifying the population dynamics at certain steps, e.g., by rejecting certain offspring that do not satisfy distance and similarity criteria. For instance, in [45], if the offspring is too similar to one of the existing solutions, then it is not inserted in the population. The similarity in this case is based on conflict numbers and number of uncolored vertices. However, the partition distance [27, 59] conveys a more direct and semantic sense of distance and it is often used in coloring

papers for various purposes [20, 22, 26, 29, 43, 57]. A simple offspring rejection rule based on this metric is presented in [57] and it seems that the approach can be generalized to other problems as well [58]. Another point where one can intervene is the elimination step. Indeed, an approach that uses a scoring function (based on the distance metric) to decide which individual should be replaced by the offspring is presented in [43]. Notice that the parent selection step, while very influent in other evolutionary computing areas, seems to have a secondary impact in graph coloring.

Finally, let us comment on the fact that genetic algorithms are better suited than local search for hybridization with exact algorithms. At least in the column generation approach described in [46], the `MMT` genetic algorithm [45] can generate more diverse columns than local search. `MMT` works with conflict-free classes (independent sets) that constitute partial colorings improved via the k-fixed partial proper strategy [49]. The resulting independent sets fit very well in a set covering coloring formulation: Construct a linear program that minimizes the number of independent sets necessary to cover all vertices of the graph. Let us denote by $\mathscr{I}$ the set of all existing independent sets of G and let $x_i = 1$ if and only if $i \in \mathscr{I}$ is utilized by the current coloring. The objective of the program is to minimize $\sum_{i \in \mathscr{I}} x_i$. The constraints impose to cover each vertex by at least one selected independent set. Since $\mathscr{I}$ can not be usually completely generated, this set is first sampled from the (diverse) classes provided by the population of `MMT`. In a second stage, more independent sets are constructed via column generation: The slave problem requires finding columns $i \in \mathscr{I}$ that could improve the objective function (because they violate the dual constraints).

4.3 Discussion and Concluding Remarks

To conclude the evolutionary part, a number of points can be underlined:

1. Genetic algorithms represent a popular and flexible approach to optimization problems that seems very effective for many classes of difficult graphs;
2. Much attention has been paid to the recombination operator and it is generally accepted that:
   - Recombination operators must be specialized to the problem. Blindly using unspecialized operators (e.g., standard uniform crossover) can render quite impractical a genetic algorithm otherwise well-designed;
   - The most effective coloring crossovers are class-oriented. The parent classes are chosen for transmission using greedy selections;
   - Refining more detailed aspects can be very useful in certain cases, e.g., instances with large color classes seem to benefit from using more than two parents.
3. A chiseled population dynamics (using partition distances to modify certain steps, by applying scatter search or adaptive memory frameworks) seems a

promising choice for future research (diversity preservation is an important topic in general genetic algorithm research).

5 Approaches Based on Independent Set Extraction

5.1 Basic Approach

As observed in many studies, it is difficult, if not impossible, to find a proper k-coloring of a large graph G (e.g., with 1 000 vertices or more) with k close to $\chi(G)$ by applying directly a given coloring algorithm on G. A basic approach to deal with large graphs is to apply the general principle of "reduce-and-solve". This approach is composed of a *preprocessing phase* followed by a *coloring phase*.

The preprocessing phase typically identifies and removes some (large) independent sets from the original graph to obtain a reduced subgraph (called "residual" graph). The subsequent coloring phase determines a proper coloring for the residual graph. Given the residual graph is of reduced size, it is expected to be easier to color than the initial graph. Now it suffices to consider each extracted independent set as a new color class (i.e., by assigning a new color to all the vertices of each of these sets). The coloring of the residual graph and all the extracted independent sets give a proper coloring of the initial graph. This approach were explored with success in early studies like [9, 19, 33, 39].

Algorithms based on this approach can use different methods to find a large independent set in the graph. In [9], this was achieved with a simple greedy heuristic while in [19, 33], large independent sets were identified by a dedicated tabu search algorithm. In [39], the authors introduced the XRLF heuristic which operates in two steps. First, a number of independent sets are collected using Leighton's Recursive Largest First (RLF) heuristic [42]. Then, an independent set is iteratively selected and extracted from the graph such that its removal minimizes the density of the reduced graph. This process continues until the residual graph reaches a given threshold.

For the subsequent residual graph coloring, various methods have been used including exhaustive search [39], tabu search [33], simulated annealing [9, 39] and hybrid genetic tabu search [19].

5.2 Enhancing Independent Set Extraction

Very recently, this basic independent set extraction approach was revisited and improved results were reported on several large graphs [71, 72, 31].

As stated previously, a proper k-coloring of a given graph G corresponds to a partition of V into k independent sets. Suppose that we want to color G with k

colors. Assume now that we extract $t < k$ independent sets $I_1, ..., I_t$ from G. It is clear that if we maximize the number of vertices covered by the t independent sets (i.e., $|I_1 \cup ... \cup I_t|$ is as large as possible), we obtain a residual graph with fewer vertices when $I_1 \cup ... \cup I_t$ are removed from G. This would make the residual graph easier to color using the remaining $k-t$ colors.

The basic strategy for independent set extraction described in Sect. 5.1 operates greedily by removing exactly one independent set at a time. This preprocessing phase was improved in [71] by identifying and removing at each extraction iteration a *maximum number of disjoint independent sets* of the same size from the graph. This extraction procedure can be summarized as follows.

1. Identify a maximum independent set I in G;
2. Collect in a set M as many other independent sets of size $|I|$ as possible;
3. Find from M a maximal number of *disjoint* independent sets $I_1, ..., I_t$;
4. Remove the vertices of $I_1 \cup \ldots \cup I_t$ from G;
5. Repeat the above steps until the subgraph becomes small enough.

The number of vertices of the last subgraph (residual graph) depends on the subsequent coloring algorithm applied to the residual graph. In [71, 72, 31], values from 500 to 800 are suggested.

In step 1 of the above extraction process, one needs to identify a maximum independent set. Notice that the problem of finding a maximum independent set in a graph is NP-hard [23, 40] in the general case. For this purpose, the authors of [71] use the so-called adaptive tabu search heuristic designed for the equivalent maximum clique problem to find large independent sets [70]. The same heuristic is also employed to build the pool M composed of independent sets of a given size (step 2).

In step 3, it is required to find among the candidates of M a maximal number of *disjoint* independent sets. This latter task corresponds in fact to the maximum set packing problem, which is equivalent to the maximum clique (thus the maximum independent set) problem [23, 40]. To see this, it suffices to construct an instance of the independent set problem from $M = \{I_1, ..., I_t\}$ as follows. Define a new graph $G' = (V', E')$ where $V' = \{1, ..., t\}$ and $\{i, j\} \in E'$ $(i, j \in V')$ if I_i and I_j share at least one element, i.e., $I_i \cap I_j \neq \emptyset$. Now it is clear that there is a strict equivalence between an independent set in G' and a set of disjoint independent sets in M. Consequently, one can apply again any maximum clique or independent set algorithm to approximate the problem.

As shown in [71], this extraction strategy packs more vertices than with the conventional one-by-one extraction strategy with the same number of color classes. This generates smaller residual graphs that tend to be easier to color. The EXTRACOL algorithm described in [71] combines this independent set extraction strategy with the memetic coloring algorithm MACOL [43]. Evaluation of EXTRACOL on the set of the largest graphs (with 1 000 to 4 000 vertices) of the DIMACS challenge benchmarks showed remarkable results (see Sect. 8.2).

5.3 An Extraction and Expansion Approach

Preprocessing a graph by extracting independent sets reduces the initial graph and tends to ease the coloring task. Such a preprocessing strategy relies on the hypothesis that each extracted independent set defines a color class of the final coloring. Unfortunately, this may not be the case all the time. For instance, it is shown that EXTRACOL [71] performs poorly on some large geometric graphs although its performance is remarkable on all the other tested graphs. A close check discloses that due to the particular structure of these graphs, many largest (extracted) independent sets are not part of a final coloring. In this case, it is a harmful to remove definitively these independent sets from the graph since this will prevent inevitably the subsequent coloring algorithm from reaching an optimal coloring. To mitigate this difficulty, one solution is to allow the subsequent coloring algorithm to "reconsider" the extracted independent sets and allow some vertices of these extracted sets to change their colors.

In [72, 31], the approach of independent set extraction followed by a coloring phase is expended to an "extraction and expansion" approach. This approach can be summarized by the following procedure composed of three phases:

1. The *extraction* phase simplifies the initial graph G by removing iteratively large independent sets from the original graph. This phase stops when the residual graph becomes sufficiently small fixed by a threshold. The residual graph will be first colored (phase 2) and then extended during the expansion and backward coloring process (phase 3);
2. The *initial coloring* phase applies a graph coloring algorithm to the residual graph to determine a $(k-t)$-coloring where t is the number of extracted independent sets. If a proper $(k-t)$-coloring c for the residual graph is found, then c plus the t independent sets extracted during the phase 1 constitutes a proper k-coloring of the initial graph G, return this k-coloring and stop. Otherwise, continue to phase 3 to trigger the expansion and backward coloring phase;
3. The *expansion and backward coloring* phase extends the current subgraph G' by adding back one or more extracted independent sets to obtain an extended subgraph G''. Then the coloring algorithm is run on G'' by starting from the current coloring of G' extended with the added independent sets as new color classes. Once again, if a proper coloring is found for the subgraph G'', this coloring plus the remaining independent sets forms a proper k-coloring of the initial graph G and the whole procedure stops. Otherwise, one repeats this expansion and backward coloring phase until no more independent set is left or a proper coloring is found for the current subgraph under consideration.

Notice that the approaches described in Sect. 5.1 and 5.2 correspond to phases 1 and 2 and consequently can be considered as a special case of the extraction and expansion approach. The expansion and backward coloring phase (phase 3) is critical since the extracted independent sets are re-examined by the coloring process. If some vertices of an extracted independent set should not receive the same color, they have a chance to be assigned the right color by the applied coloring algorithm.

The key issues to be considered concern the way to select the independent sets to add back to the current coloring (and to rebuild the corresponding subgraph). Several strategies are possible for this selection [31]:

- First, we can consider how many independent sets are to be selected for expansion. Basically, this decision can be made according to one of two rules: One independent set or several independent sets at a time. This choice may have influences on the subsequent coloring process. Indeed, adding back one independent set at a time implies limited extensions to the current coloring (only one new color class is added). This leads thus to a more gradual coloring optimization. On the other hand, using several independents sets to extend the current subgraph and colorings offers more freedom and diversification for coloring optimization;
- Second, we can also consider which independent sets are to be selected. This decision can be achieved following the reverse of extraction order, extraction order, or random order. This decision can also be based on the size of the independent sets. Notice that given the way independent sets are extracted during the extraction phase, applying the reverse of extraction order handles the independent sets from the smallest to the largest [72] while applying extraction order does the opposite.

It is clear that any combination of the above two decisions leads to a strategy that can be used to determine the independent sets for backward coloring optimization.

Finally, it is easy to see that the underlaying coloring algorithm employed for the initial coloring (phase 2) and subsequent colorings (phase 3) also impacts the performance of this extraction and expansion coloring approach. In [72], the task of coloring is ensured by a perturbation-based tabu search (a variant of the TABUCOL procedure [33], see Sect. 3.1) while in [31], the more powerful memetic algorithm MACOL [43] is preferred.

Experimental evaluations of this extraction and expansion approach reported in [72, 31] demonstrated a remarkable performance on some very hard and large graphs with more than 2 000 vertices by improving very recent upper bounds of several large benchmark graphs.

6 Other Approaches

In this section, we review a few other strategies based on concepts or metaphors that are (almost) different from those presented in the precedent sections.

Quantum Annealing: In [66], an effective quantum annealing algorithm is described for k-coloring. This strategy can be seen as a population-based heuristic (an individual is called here a "replica") and inherits ideas from *quantum* mechanics. Compared with simulated annealing, it includes an additional parameter Γ (with the classical temperature parameter). While simulated annealing evolves in a neighborhood of constant radius, Γ is used here to modify the radius of the neighborhood (to control diversity) and to reinforce the evaluation function with a "kinetic" energy

relying on interactions between replicas (roughly speaking, it quantifies the similarity of replicas). This kinetic energy aims to help escaping local optima. Experiments reported in [66] shown remarkable results on some tested DIMACS graphs.

Neural Networks: In [36], an artificial neural network algorithm is described for the graph coloring problem. This algorithm, called RBA-AIS, follows the partial proper strategy defined in Sect. 2. In RBA-AIS, the value of k (initially, $k = \Delta + 1$, where Δ is the maximum connection degree) and the network size ($k \times n$ cells) can be modified by an *adaptive multiple restart local search* approach (the RBA component). Roughly speaking, RBA runs several times a randomized dynamical algorithm (called "Stochastic Steep Descent", SSD) where the initial state of each SSD call (except the first one which is constant) is determined by the results of previous SSD executions (AIS component). Other coloring algorithms using neural networks are reported in [41, 64]. Except for some structured graphs, existing neural network coloring algorithms are not competitive.

Ant Colony: ALS-COL [54] seems to be the most recent ant colony optimization algorithm for k-coloring. It follows the partial proper strategy defined in Sect. 2, each ant running a tabu search inspired from PARTIALCOL [3] (let us call it ALS-TS). While decisions in classical ant colony optimization algorithms rely on a parametric probability that is time consuming (since it is based on *all* possible decisions), ALS-TS uses less parameters and just considers *non-tabu* moves (to save time). Results reported in [54] indicate that ALS-COL outperforms PARTIALCOL and some previous ant colony optimization heuristics proposed for graph coloring (see e.g., [35, 62]). Given sufficient time, the algorithm obtains very competitive results on several graphs.

Bounding Strategies: A column generation approach was recently described in [32] (let us call it CG) for computing lower bounds of χ. CG tries to determine the *fractional* chromatic number χ_f [44, 73] since $\chi(G) \geq \lceil \chi_f(G) \rceil$. CG computes a lower bound $\underline{\chi_f}$ of χ_f using a recursive depth-first search or a local search with restarts. A *restricted* linear programming model is also solved to compute an upper bound $\overline{\chi_f}$ of χ_f. CG was tested on 136 instances (including DIMACS graphs), $\underline{\chi_f}$ and $\overline{\chi_f}$ being determined for 119 of them. In *all* the 119 cases, it turned out that $\left\lceil \underline{\chi_f} \right\rceil = \lceil \overline{\chi_f} \rceil$ (this avoids the floating-point inaccuracy of linear programming solvers). Improved lower bounds of χ were proposed, reducing (sometimes considerably) the gap to upper bounds. For some large graphs where CG cannot compute $\lceil \chi_f \rceil$, CG was run on subgraphs. Here again, this leads to improved lower bounds.

7 Extensions and Applications of Graph Coloring

7.1 Extensions and Generalizations of Graph Coloring

In *T-coloring* [28, 37], one associates to each edge $\{v_i, v_j\} \in E$ a set $t_{i,j}$ of positive integer values (0 included) and imposes $|c(v_i) - c(v_j)| \notin t_{i,j}$. The *span* of a

T-coloring c is the difference between the smallest and the highest color in c. Given a graph G, the T-coloring problem (in its optimization version) is to determine the minimum span for all possible T-colorings of G. Graph (k-)coloring corresponds to a special case of T-coloring where each $t_{i,j} = \{0\}$.

Set T-coloring [65] is a generalization of T-coloring (with the same objective) where an integer "demand" $\delta_i > 0$ is further associated to each $v_i \in V$, corresponding to the number of different colors required by v_i: $\delta_i = l \Rightarrow c(v_i) = \{c_{i,1}, \ldots, c_{i,l}\}$. A "co-node" constraint must then be verified by all v_i: $\{c_{i,\alpha}, c_{i,\beta\neq\alpha}\} \in c(v_i) \times c(v_i) \Rightarrow |c_{i,\alpha} - c_{i,\beta}| \notin t_{i,i}$. Finally, the constraint previously introduced for T-coloring must also hold here: $\{v_i, v_j\} \in E \Rightarrow |c_{i,\alpha} - c_{j,\beta}| \notin t_{i,j}\ \forall \{c_{i,\alpha}, c_{j,\beta}\} \in c(v_i) \times c(v_j)$. The graph ($k$-)coloring problem is a special case of set T-coloring where each $\delta_i = 1$ (and $t_{i,j} = \{0\}$).

Bandwidth coloring [25] is a restriction of T-coloring where the constraint on adjacent vertices is replaced by $|c(v_i) - c(v_j)| \geq t_{i,j}$ ($t_{i,j}$ is not a set here, but a numerical value). *Multi-coloring* [47] is a special case of set T-coloring where all $t_{i,j} = \{0\}$. The *bandwidth multi-coloring* problem [47] is a combination of bandwidth coloring and a variant of set T-coloring where the co-node constraint is simplified to $\{c_{i,\alpha}, c_{i,\beta\neq\alpha}\} \in c(v_i) \times c(v_i) \Rightarrow |c_{i,\alpha} - c_{i,\beta}| \geq t_{i,i}$ ($t_{i,i}$ is a numerical value). In the case of *list-coloring* [53], the color of each vertex must be chosen in a predefined set of authorized colors.

More than 200 *open* problems related to (k-)coloring are presented in [38], see also the programs from the meetings of the "DIMACS / DIMATIA / Rényi Working Group on Graph Colorings and their Generalizations" (`http://dimacs.rutgers.edu/Workshops/GraphColor/main.html`) and the "Graph Coloring and its Generalizations" symposium series (COLOR, 2002 – ..., `http://mat.gsia.cmu.edu/COLOR03`) for instance.

7.2 Applications of Graph Coloring

There are few real-world problems that can be directly modeled by a graph coloring problem. On the other hand, many applications are similar to graph coloring or are somehow related to graph coloring problems or their extensions [1, 7, 15, 61]. We give several examples below for illustration.

A **timetabling** problem is the task to assign a time slot to events (exams, lectures) subject to pairwise constraints [61]. The constraints involve pairs of events that can not be assigned to the same time slot – for example, two lectures given by the same professor. Graph coloring has been used for a long time in order to model timetabling problems. In a standard representation, the events are represented by vertices, constraints by edges and time slots by colors. In addition to (mandatory) constraints expressed by coloring constraints, other additional hard constraints may be present and soft constraints can be used in order to discriminate among legal solutions.

Register allocation is an important problem in compiler optimization. Given a target source code, the problem is to choose among a large number of variables those that will be assigned to registers – the other variables will be kept in RAM and require much slower access times. Each of these variables must be assigned a register subject to the constraint that two variables that are in use at the same time cannot be assigned to the same register. Register allocation can be modeled as a weighted partial coloring problem (the problem to find a partial proper coloring such that the weighted sum of colored vertices is maximized) [7]. Variables are represented by vertices, constraints by edges and registers by colors. The weight of a vertex expresses the savings of keeping the corresponding variable in a register rather than in RAM.

Frequency assignment problems have been playing an important role in telecommunication networks for more than twenty years. In this kind of problem, frequencies chosen in a specified set (the frequency domain) must be assigned to physical communication equipments (antennas or links, depending on the context). A common feature of these problems is the presence of distance constraints imposed on pairs of frequencies in order to avoid (or at least reduce) interference between geographically close communication equipments [1]. Therefore, frequency assignment problems can be seen as an extension of the bandwidth coloring problem where vertices represent equipments, and colors frequencies. In a typical case, the number of frequencies is fixed (and very limited) and the problem is to find a non necessarily legal solution with a minimum number of conflicts (interference level).

Routing and wavelength assignment (RWA) problems play an important role in all-optical networks. A network is represented by a non-oriented graph whose nodes figure stations and edges represent links. One is also given a set of pairs (origin-destination) of stations, named "demands". In order to satisfy a demand, a lightpath must be constructed, i.e., a route (path) in the graph, along with a frequency. A legal solution of the problem is a set of lightpaths such that two lightpaths that share a common link never use the same frequency [15]. In the max-RWA problem, the number of available frequencies is fixed and the goal is to satisfy a maximum number of demands. The max-RWA problem can be modeled as a non-standard variant of the partial coloring problem. The paths in the graph are represented by vertices, the constaints by edges and the frequencies by colors. In practice, solution techniques exploit a graph that contains only a limited number of paths.

8 Benchmarks and Computational Results

8.1 Benchmarks

The first federative library of graph coloring instances was created in the 1990' for the second DIMACS challenge on graph coloring and maximum clique. Originally [68], it was composed of the 32 graphs available from `ftp://dimacs.`

rutgers.edu/pub/challenge/graph/benchmarks/volume/Color (the ".b" suffix indicates that files are compressed, see ftp://dimacs.rutgers.edu/pub/challenge/graph/translators/binformat for unpacking). Additional well-known instances are available from http://mat.gsia.cmu.edu/COLOR/instances.html. Various classes of graphs are represented: Random, real-world inspired, translation from other problems, etc.

A *recent* more expanded library, including most of the previous mentioned benchmarks, has been proposed along with the "Graph Coloring and its Generalizations" symposium series (COLOR, 2002 – ...). It is available from http://mat.gsia.cmu.edu/COLOR02. New graph domains include matrix partitioning or optical network design for instance.

Many classes of random graphs exist, the most studied of them in the graph coloring research community being those following the "uniform" $\mathscr{G}_{n,p}$ model [17]. These graphs constitute a real challenge and their optimums are usually unknown. To our knowledge, just one federative library has been proposed [33] (see also [16, 30] for 3-coloring). The 30 smallest instances used in [33] ($n \in \{100, 300\}$) are available from http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files (gcol1.txt – gcol 30.txt). The 8 largest instances ($n \in \{500, 1\,000\}$) are available from the second author upon request.

Some generators have been developed to build (specific) classes of graphs. See http://www.cs.ualberta.ca/~joe/Coloring for register-interference, Brockington, evacuation, quasi-random k-partite, and timetabling-inspired graphs or ftp://dimacs.rutgers.edu/pub/challenge/graph/contributed/morgenstern for Leighton, random, k-partite, geometrical, and planar graphs. The following are the most popular hard instances:

1. Five large and very large random graphs (dsjc500.5, dsjc500.9, dsjc1000.1, dsjc1000.5, and dsjc1000.9) following the uniform $\mathscr{G}_{n,p}$ model and proposed in [39]. The first and second number in the name of each graph represent respectively the number n of vertices and the probability p that an edge exists between any pair of vertices. The chromatic numbers of these graphs are unknown;
2. Two random "flat" graphs (flat300_28_0 and flat1000_76_0). They are structured graphs with known chromatic number (respectively 28 and 76);
3. Two random "leighton" graphs (le450_25c and le450_25d) with $n = 450$ and $\chi = 25$ containing numerous cliques of various sizes;
4. Four large random geometric graphs (dsjr500.1c, dsjr500.5, r1000.1c, and r1000.5). These graphs are generated by picking random points (vertices) in a plane and by linking two points situated within a certain geometrical distance [39]. The chromatic number is unknown for r1000.1c and is equal to 84 for dsjr500.1c, 122 for dsjr500.5 and 234 for r1000.5;
5. Three very large random graphs (c2000.5, c2000.9, and c4000.5). The chromatic numbers of these graphs are unknown. Due to the size and difficulty of these graphs, they are not always used in computational experiments in the literature;
6. One "latin square" graph (latin_square_10) with unknown chromatic number that models a problem related to latin squares;

7. Eight "'wap" graphs (wap01 to wap08). These graphs stem from real-life optical network design problems. Each vertex corresponds to a lightpath in the network, edges correspond to intersecting paths. These structured graphs have unknown chromatic number except wap05 (50).

The graphs of families 1 to 6 were initially collected for the 2nd DIMACS challenge while the wap graphs were made available for the COLOR competitions. One notices that contrary to most DIMACS graphs, the wap graphs are much less studied in the literature [6, 11, 22, 31].

8.2 Results of Some Heuristics

Most graph coloring research papers provide both a theoretical algorithmic description and a number of empirical results that we summarize in this section. First, one should keep in mind that the evaluation and comparison of practical results is always complicated by the fact that the computational environments can be very different from algorithm to algorithm (e.g., different compilers, stopping criteria, implementation styles, etc). Besides this, certain papers introduce metaheuristic ideas of general interest, while others algorithms propose very specific graph coloring techniques. As such, the summary tables from this section do not aim at providing an absolute ranking of algorithms, but rather to convey the difficulty of coloring certain graphs.

Let us first introduce in Table 1 the so-called "easy instances" of the DIMACS benchmark. Numerous algorithms are able to find proper colorings with k^* colors for all these graphs, but there is no mention of a proper coloring with less colors. By using $k^* - 1$ colors, the instance probably moves into an "unSAT" side or it becomes overly difficult. In what follows, we concentrate only on the rest of the instances (*hard* instances), as most coloring research papers do.

Regarding these hard instances, we provide in Table 2 the best upper bounds reported by a dozen of *coloring approaches* from the literature. In this context, a *coloring approach* might simply refer to a unique algorithm or to *a class of algorithms* – certain results are reported by combining the performances of several algorithm versions. The instances are grouped in three classes, depending on the graph density d: Sparse graphs ($d < \frac{1}{4}$), medium density instances ($\frac{1}{4} \leq d \leq \frac{3}{4}$) and dense graphs ($d > \frac{3}{4}$). Notice that certain large wap instances can also be considered to be hard. However, since these instances are much less tested in the literature, we do not include them in the table.

One can see that the correlation between graph difficulty and density is rather limited, or inexistent. Other criteria that might influence graph difficulty include the graph order (n) and the graph family (random, geometrical, etc). We observe that the largest performance variations of the most recent algorithms are reported on: flat300.28_0 (a small graph), r1000.5, latin_square_10, c2000.5, and c4000.5.

Furthermore, for the huge graphs c2000.5 and c4000.5, very large time limits are often used (days or weeks).

G	k^*	G	k^*	G	k^*	G	k^*
dsjc125.1	5	r125.1	5	le450_5a	5	flat300.20_0	20
dsjc125.5	17	r125.5	36	le450_5b	5	flat300.26_0	26
dsjc125.9	44	r125.1c	46	le450_5c	5	flat1000.50_0	50
dsjc250.1	8	r250.1	8	le450_5d	5	flat1000.60_0	60
dsjc250.5	28	r250.1c	64	le450_15a	15	school1	14
dsjc250.9	72	r250.5	65	le450_15b	15	school1_nsh	14
dsjc500.1	12	r1000.1	20	le450_25a	25		
		dsjr500.1	12	le450_25b	25		

Table 1 Easy DIMACS coloring instances. Numerous papers report exactly the same values of the best upper bound k^* for these graphs.

9 Conclusion

In this chapter, we reviewed some recent advances for solving the well-known graph coloring problem. Given the hardness of the problem, a large number of heuristics have been devised in order to find satisfactory solutions. These heuristics are typically based on greedy method, local search, population-based search, and various hybrid approaches. Among all these approaches, local search algorithms play an important role, especially because they constitute an indispensable component of many successful and more sophisticated hybrid techniques. Population-based algorithms are very powerful thanks to their specific solution recombination operator (memetic algorithms) or solution interaction mechanism (quantum annealing). To handle very large graphs, approaches that combine independent set extractions and progressive coloring expansions prove to be quite promising. Finally, these different coloring approaches should be considered to provide a set of complementary solution tools for approximating this difficult problem. After all, no single coloring algorithm can dominate all the other approaches on all the graphs.

Finally, we remark that graph coloring is a quite generic problem – no particularly skewed constraints and a very simple objective function. Consequently, ideas of the coloring algorithms would have a general interest and could be applied to similar and related problems.

Acknowledgements The work is partially supported by the “Pays de la Loire” Region (France) within the RaDaPop (2009 – 2013) and LigeRO (2010 – 2013) projects.

Graph name	n, m, d	χ/k^*	Local Search Algorithms					Hybrid algorithms (population-based, independent set extraction, quantum annealing, etc.)									
			2002 ILS [10]	2003 VNS [2]	2008 PARTIALCOL [3]	2008 VSS-Col [34]	2010 TS-Div/Int [56]	1996 DCNS [49]	1996 HGA [19]	1999 HEA [20]	2008 AmaCol [22]	2008 MMT [45]	2010 EvoDiv [57]	2010 MACOL [43]	2011 QA-col [66]	2011 DHQA [67]	2012 ECol [71]
dsjc1000.1	1 000, 49 629, 0.1	$\geq 10/20$	—	—	**20**	**20**	**20**	—	—	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
le450.25c	450, 17 343, 0.17	25/25	26	—	**25**	26	**25**	**25**	**25**	26	26	**25**	**25**	**25**	**25**	**25**	—
le450.25d	450, 17 425, 0.17	25/25	26	—	**25**	26	**25**	**25**	**25**	—	26	**25**	**25**	**25**	**25**	**25**	—
dsjc500.5	500, 62 624, 0.5	$\geq 43/48$	49	49	**48**	**48**	**48**	49	49	**48**	**48**	**48**	**48**	**48**	**48**	**48**	
dsjc1000.5	1 000, 249 826, 0.5	$\geq 73/83$	89	90	88	87	85	89	84	**83**	84	**83**	**83**	**83**	**83**	**83**	**83**
dsjr500.5	500, 58 862, 0.47	122/122	124	—	126	125	—	123	130	—	125	**122**	**122**	**122**	**122**	**122**	—
r1000.5	1 000, 238 267. 0.48	234/234	—	248	—	241	—	268	—	—	—	**234**	237	245	238	**234**	249
flat300.28_0	300, 21 695, 0.48	28/28	31	31	**28**	**28**	**28**	31	33	31	31	31	29	29	31	**28**	—
flat1000.76_0	1 000, 246 708, 0.49	76/82	—	89	88	86	85	89	84	83	84	**82**	**82**	**82**	**82**	**82**	**82**
c2000.5	2 000, 999 836, 0.5	$\geq 99/146$	—	—	—	—	—	150	153	—	—	—	148	148	—	147	**146**
c2000.9	2 000, 1 799 532, 0.9	$\geq 80/409$	—	—	—	—	—	—	—	—	—	—	—	413	—	—	**409**
c4000.5	4 000, 4 000 268, 0.5	$\geq 107/260$	—	—	—	—	—	—	280	—	—	—	271	272	—	—	**260**
dsjc500.9	500, 112 437, 0.9	$\geq 123/126$	**126**	—	**126**	**126**	**126**	—	—	—	**126**	127	**126**	**126**	**126**	**126**	—
dsjc1000.9	1 000, 449 449, 0.9	$\geq 216/222$	—	—	225	224	223	226	—	224	224	225	223	223	**222**	**222**	**222**
dsjr500.1c	500, 121 275, 0.97	84/85	—	—	**85**	**85**	—	**85**	**85**	—	86	**85**	**85**	**85**	**85**	—	—
r1000.1c	1 000, 485 090, 0.97	98/98	—	—	**98**	—	**98**	**98**	99	—	—	**98**	**98**	**98**	**98**	—	101
latin_square_10	900, 307 350, 0.76	$\geq 90/97$	99	—	—	—	—	98	106	—	104	101	98	99	98	**97**	—

Table 2 The upper bounds reported by 15 coloring approaches from the literature (bold entries signal results reaching the best-known coloring k^*). For each graph, we provide: (i) The name (Column 1); (ii) The number of vertices (n), edges (m) and the density d (Column 2); (iii) The chromatic number (or a lower bound if unknown) and the best known upper bound even found by a coloring algorithm (Column 3); And (iv) the upper bounds reported by 15 papers in the last 15 columns. Notice that new upper bounds were reported very recently for flat1000.76_0 ($k^* = 81$), c2000.5 ($k^* = 145$), c2000.9 ($k^* = 408$) and c4000.5 ($k^* = 259$) [31].

References

1. Aardal, K., van Hoesel, S.P.M., Koster, A.M.C.A., Mannino, C., Sassano, A.: Models and solution techniques for frequency assignment problems. Ann. of Oper. Res. **153**(1), 79–129 (2007)
2. Avanthay, C., Hertz, A., Zufferey, N.: A variable neighborhood search for graph coloring. Eur. J. of Oper. Res. **151**(2), 379–388 (2003)
3. Blöechliger, I., Zufferey, N.: A graph coloring heuristic using partial solutions and a reactive tabu scheme. Comput. & Oper. Res. **35**(3), 960–975 (2008)
4. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput. Surveys, **35**(3), 268–308 (2003)
5. Brélaz, D.: New methods to color the vertices of a graph. Commun. of the ACM **22**(4), 251–256 (1979)
6. Bui, T.N., Nguyen, T.V.H., Patel C.M., Phan K.-A.T.: An ant-based algorithm for coloring graphs. Discrete Appl. Math. **156**(2), 190–200 (2008)
7. Chaitin, G.J.: Register allocation and spilling via graph coloring. ACM SIGPLAN Notices **17**(6), 98–105 (1982)
8. Chalupa, D.: Population-based and learning-based metaheuristic algorithms for the graph coloring problem. In: N. Krasnogor, P. Lanzi (eds.) Proc. of the 13th annual Genet. and Evol. Comput. Conf. *(GECCO, Dublin, Ireland, July 12–16, 2011)*, pp. 465–472. ACM Press, N. Y., USA (2011)
9. Chams, M., Hertz, A., de Werra, D.: Some experiments with simulated annealing for coloring graphs. Eur. J. of Oper. Res. **32**(2), 260–266 (1987)
10. Chiarandini, M., Stützle, T.: An application of iterated local search to graph coloring. In: D. Johnson, A. Mehrotra, M. Trick (eds.) Proc. of the Comput. Symp. on Graph Color. and its Gen. *(COLOR, Ithaca, N. Y., USA, Sept. 7–8, 2002)*, pp. 112–125 (2002)
11. Chiarandini, M., Stützle, T.: An analysis of heuristics for vertex colouring. In: P. Festa (ed.) Experimental Algorithms, *Lect. Notes in Comput. Sci.*, vol. 6049, pp. 326–337. Springer, Heidelberg, Ger. (2010)
12. Costa, D., Hertz, A., Dubuis, O.: Embedding of a sequential procedure within an evolutionary algorithm for coloring problems in graphs. J. of Heuristics **1**(1), 105–128 (1995)
13. Davis, L.: Order-based genetic algorithms and the graph coloring problem. In: L. Davis (ed.) Handbook of Genetic Algorithms, pp. 72–90. Van Nostrand Reinhold, N. Y., USA (1991)
14. Dorne, R., Hao, J.K.: A new genetic local search algorithm for graph coloring. In: A. Eiben, T. Bäck, M. Schoenauer, H. Schwefel (eds.) Parallel Problem Solving from Nature - PPSN V, *Lect. Notes in Comput. Sci.*, vol. 1498, pp. 745–754. Springer, Heidelberg, Ger. (1998)
15. Dzongang, C., Galinier, P., Pierre, S.: A Tabu search heuristic for the routing and wavelength assignment problem in optical networks. IEEE Commun. Lett. **9**(5), 426–428 (1998)
16. Eiben, A., van der Hauw, J., van Hemert, J.: Graph coloring with adaptive evolutionary algorithms. J. of Heuristics **4**(1), 24–46 (1998)
17. Erdős, P., Rényi, A.: On random graphs I. Publ. Math. Debr. **6**, 290–297 (1959)
18. Falkenauer, E.: Genetic Algorithms and Grouping Problems. John Wiley & Sons, Inc., N. Y., USA (1997)
19. Fleurent, C., Ferland, J.: Genetic and hybrid algorithms for graph coloring. Ann. of Oper. Res. **63**(3), 437–461 (1996)
20. Galinier, P., Hao, J.K.: Hybrid evolutionary algorithms for graph coloring. J. of Comb. Optim. **3**(4), 379–397 (1999)
21. Galinier, P., Hertz, A.: A survey of local search methods for graph coloring. Comput. & Oper. Res. **33**(9), 2547–2562 (2006)
22. Galinier, P., Hertz, A., Zufferey, N.: An adaptive memory algorithm for the *k*-coloring problem. Discrete Appl. Math. **156**(2), 267–279 (2008)
23. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completness. W.H. Freeman and Co., San Franc., USA (1979)

24. Gendron, B., Hertz, A., St-Louis, P.: On edge orienting methods for graph coloring. J. of Comb. Optim. **13**(2), 163–178 (2007)
25. Gendron, B., Hertz, A., St-Louis, P.: On a generalization of the Gallai-Roy-Vitaver theorem to the bandwidth coloring problem. Oper. Res. Lett. **36**(1), 345–350 (2008)
26. Glass, C., Prügel-Bennett, A.: Genetic algorithm for graph coloring: Exploration of Galinier and Hao's algorithm. J. of Comb. Optim. **7**(3), 229–236 (2003)
27. Gusfield, D.: Partition-distance: A problem and class of perfect graphs arising in clustering. Inf. Process. Lett. **82**(3), 159–164 (2002)
28. Hale, W.K.: Frequency assignment: Theory and applications. IEEE Trans. on Veh. Technol. **68**(12), 1497–1514 (1980)
29. Hamiez, J.P., Hao, J.K.: Scatter search for graph coloring, *Lect. Notes in Comput. Sci.*, vol. 2310, pp. 168–179. Springer, Heidelberg, Ger. (2002)
30. Hamiez, J.P., Hao, J.K., Glover, F.: A study of tabu search for coloring random 3-colorable graphs around the phase transition. Int. J. of Appl. Metaheuristic Comput. **1**(4), 1–24 (2010)
31. Hao, J.K., Wu, Q.: Improving the extraction and expansion approach for large graph coloring. Submitted manuscr. (Sept. 2011)
32. Held, S., Cook, W., Sewell, E.: Safe lower bounds for graph coloring. In: O. Günlük, G. Woeginger (eds.) Proc. of the *15th Int. Conf. on Integer Program. and Comb. Optim.* (IPCO, N. Y., USA, June 15-17, 2011), *Lect. Notes in Comput. Sci.*, vol. 6655, pp. 261–273. Springer, Heidelberg, Ger. (2011)
33. Hertz, A., de Werra, D.: Using tabu search techniques for graph coloring. Comput. **39**(4), 345–351 (1987)
34. Hertz, A., Plumettaz, M., Zufferey, N.: Variable space search for graph coloring. Discrete Appl. Math. **156**(13), 2551–2560 (2008)
35. Hertz, A., Zufferey, N.: A new ant algorithm for graph coloring. In: Proc. of the *Workshop on Nat. Inspired Coop. Strateg. for Optim.* (NISCO, Granada, Spain, June 29–30, 2006), pp. 51–60 (2006)
36. Jagota, A.: An adaptive, multiple restarts neural network algorithm for graph coloring. Eur. J. of Oper. Res. **93**(2), 257–270 (1996)
37. Janczewski, R.: T-coloring of graphs, *Contemp. Math.*, vol. 352, chap. 5, pp. 67–77. Am. Math. Soc., New Providence, USA (2004)
38. Jensen, T., Toft, B.: Graph Coloring Problems. Wiley-Interscience Ser. in Discrete Math. and Optim. John Wiley & Sons, Inc., N. Y., USA (1994)
39. Johnson, D., Aragon, C., McGeoch, L., Schevon, C.: Optimization by simulated annealing: An experimental evaluation; Part II, Graph coloring and number partitioning. Oper. Res. **39**(3), 378–406 (1991)
40. Karp, R.: Reducibility among combinatorial problems, pp. 85–103. Plenum Press, N. Y., USA (1972)
41. Korst, J., Aarts, E.: Combinatorial optimization on a boltzmann machine. J. of Parallel and Distrib. Comput. **6**(2), 331–357 (1989)
42. Leighton, F.: A graph coloring algorithm for large scheduling problems. J. of Res. of the Natl. Bureau of Stand. **84**(6), 489–506 (1979)
43. Lü, Z., Hao, J.K.: A memetic algorithm for graph coloring. Eur. J. of Oper. Res. **203**(2), 241–250 (2010)
44. Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. J. of the ACM **41**(5), 960–981 (1994)
45. Malaguti, E., Monaci, M., Toth, P.: A metaheuristic approach for the vertex coloring problem. INFORMS J. on Comput. **20**(2), 302–316 (2008)
46. Malaguti, E., Monaci, M., Toth, P.: An exact approach for the vertex coloring problem. Discrete Optim. **8**(2), 174–190 (2011)
47. Malaguti, E., Toth, P.: An evolutionary approach for bandwidth multicoloring problems. Eur. J. of Oper. Res. **189**(3), 638–651 (2008)
48. Marino, A., Prügel-Bennett, A., Glass, C.: Improving graph colouring with linear programming and genetic algorithms. In: Proc. of the Short Course on Evol. Algorithms in Eng. and Comput. Sci. *(EUROGEN, Jyväskylä, Finland, May 30 – June 3, 1999)*, pp. 113–118 (1999)

49. Morgenstern, C.: Distributed coloration neighborhood search. In: D. Johnson, M. Trick (eds.) Cliques, Coloring, and Satisfiability, *DIMACS Ser. in Discrete Math. and Theor. Comput. Sci.*, vol. 26, pp. 335–357. Am. Math. Soc., New Providence, USA (1996)
50. Morgenstern, C., Shapiro, H.: Coloration neighborhood structures for general graph coloring. In: D. Johnson (ed.) Proc. of the 1st Ann. ACM-SIAM Symp. on Discrete Algorithms *(SODA, San Franc., USA, January 22–24, 1990)*, pp. 226–235. Soc. for Ind. and Appl. Math., Phila., USA (1990)
51. Moscato, P.: Memetic algorithms: A short introduction. In: D. Corne, F. Glover, M. Dorigo (eds.) New Ideas in Optimization, pp. 219–234. McGraw-Hill Educ., N. Y., USA (1999)
52. Mumford, C.: New order-based crossovers for the graph coloring problem. In: T. Runarsson, H.G. Beyer, E. Burke, J. Merelo-Guervós, L. Whitley, X. Yao (eds.) Parallel Problem Solving from Nature - PPSN IX, *Lect. Notes in Comput. Sci.*, vol. 4193, pp. 880–889. Springer, Heidelberg, Ger. (2006)
53. Piwakowski, K.: List coloring of graphs, *Contemp. Math.*, vol. 352, chap. 11, pp. 153–162. Am. Math. Soc., New Providence, USA (2004)
54. Plumettaz, M., Schindl, D., Zufferey, N.: Ant Local Search and its efficient adaptation to graph colouring. J. of the Oper. Res. Soc. **61**(5), 819–826 (2010)
55. Porumbel, D., Hao, J.K., Kuntz, P.: A study of evaluation functions for the graph K-coloring problem, *Lect. Notes in Comput. Sci.*, vol. 4926, pp. 124–135. Springer, Heidelberg, Ger. (2008)
56. Porumbel, C., Hao, J., Kuntz, P.: A search space "cartography" for guiding graph coloring heuristics. Comput. & Oper. Res. **37**, 769–778 (2010)
57. Porumbel, D., Hao, J.K., Kuntz, P.: An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. Comput. & Oper. Res. **37**(10), 1822–1832 (2010)
58. Porumbel, D., Hao, J.K., Kuntz, P.: Spacing memetic algorithms. In: N. Krasnogor, P. Lanzi (eds.) Proc. of the 13th annual Genet. and Evol. Comput. Conf. *(GECCO, Dublin, Ireland, July 12–16, 2011)*, pp. 1061–1068. ACM Press, N. Y., USA (2011)
59. Porumbel, D., Hao, J.K., Kuntz, P.: An efficient algorithm for computing the distance between close partitions. Discrete Appl. Math. **159**(1), 53–59 (2011)
60. Radcliffe, N.: The algebra of genetic algorithms. Ann. of Math. and Artif. Intell. **10**(4), 339–384 (1994)
61. Schaerf, A.: A survey of automated timetabling. Artif. Intell. Rev. **13**(2), 87–127 (1999)
62. Shawe-Taylor, J., Žerovnik, J.: Ants and graph coloring. Tech. Rep. 952, Univ. of Ljubljana / Math. Dep., Slovenia (2004). URL `http://www.imfm.si/preprinti/PDF/00952.pdf`
63. Tagawa, K., Kanesige, K., Inoue, K., Haneda, H.: Distance based hybrid genetic algorithm: An application for the graph coloring problem. In: Proc. of the 1999 Congr. on Evol. Comput., pp. 2325–2332 (1999)
64. Takefuji, Y., Lee, K.: Artificial neural networks for four-coloring map problems and K-colorability problems. IEEE Trans. on Circuits and Syst. **38**(3), 326–333 (1991)
65. Tesman, B.: Set T-colorings of graphs. Congr. Numer. **77**, 229–242 (1990)
66. Titiloye, O., Crispin, A.: Quantum annealing of the graph coloring problem. Discrete Optim. **8**(2), 376–384 (2011)
67. Titiloye, O., Crispin, A.: Graph coloring with a distributed hybrid quantum annealing algorithm. In: J. O'Shea, N. Nguyen, K. Crockett, R. Howlett, L. Jain (eds.) Agent and Multi-Agent Systems: Technologies and Applications, *Lect. Notes in Comput. Sci.*, vol. 6682, pp. 553–562 (2011)
68. Trick, M.: Appendix: Second DIMACS challenge test problems, *DIMACS Ser. in Discrete Math. and Theor. Comput. Sci.*, vol. 26, pp. 653–657. Am. Math. Soc., New Providence, USA (1996)
69. Trick, M., Yildiz, H.: A large neighborhood search heuristic for graph coloring. In: P. Van Hentenryck, L. Wolsey (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, *Lect. Notes in Comput. Sci.*, vol. 4510, pp. 346–360. Springer, Heidelberg, Ger. (2007)

70. Wu, Q., Hao, J.K.: An adaptive multistart tabu search approach to solve the maximum clique problem. DOI: 10.1007/s10878-011-9437-8 (2012)
71. Wu, Q., Hao, J.K.: Coloring large graphs based on independent set extraction. Comput. & Oper. Res. **39**(2), 283–290 (2012)
72. Wu, Q., Hao, J.K.: An extraction and expansion approach for graph coloring. Asia-Pac. J. of Oper. Res. (2012). In press
73. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. Theory of Comput. **3**, 103–128 (2007)