Available online at www.sciencedirect.com

ScienceDirect

Computers & Operations Research 35 (2008) 960–975

computers & operations research

www.elsevier.com/locate/cor

A graph coloring heuristic using partial solutions and a reactive tabu scheme

Ivo Blöchliger[a,*,1], Nicolas Zufferey[b]

[a]École Polytechnique Fédérale de Lausanne (EPFL), Chaire de Recherche Opérationnelle Sud-Est (ROSE), CH-1015 Lausanne, Switzerland
[b]Département d'opérations et systèmes de décision, Université Laval, Québec (QC), Canada G1K 7P4

Available online 14 September 2006

Abstract

Most of the recent heuristics for the graph coloring problem start from an infeasible k-coloring (adjacent vertices may have the same color) and try to make the solution feasible through a sequence of color exchanges. In contrast, our approach (called FOO-PARTIALCOL), which is based on tabu search, considers feasible but partial solutions and tries to increase the size of the current partial solution. A solution consists of k disjoint stable sets (and, therefore, is a feasible, partial k-coloring) and a set of uncolored vertices. We introduce a reactive tabu tenure which substantially enhances the performance of both our heuristic as well as the classical tabu algorithm (called TABUCOL) proposed by Hertz and de Werra [Using tabu search techniques for graph coloring, Computing 1987;39:345–51]. We will report numerical results on different benchmark graphs and we will observe that FOO-PARTIALCOL, though very simple, outperforms TABUCOL on some instances, provides very competitive results on a set of benchmark graphs which are known to be difficult, and outperforms the best-known methods on the graph flat300_28_0. For this graph, FOO-PARTIALCOL finds an optimal coloring with 28 colors. The best coloring achieved to date uses 31 colors. Algorithms very close to TABUCOL are still used as intensification procedures in the best coloring methods, which are evolutionary heuristics. FOO-PARTIALCOL could then be a powerful alternative. In conclusion FOO-PARTIALCOL is one of the most efficient simple local search coloring methods yet available.
© 2006 Elsevier Ltd. All rights reserved.

Keywords: Combinatorial optimization; Graph coloring; Tabu search

1. Introduction

The *graph coloring problem*, also called the *vertex coloring problem*, is a very well-studied NP-hard problem [1]. Given a graph $G = (V, E)$, where V is the vertex set and E the edge set, the problem consists of assigning a color (an integer) to each vertex $v \in V$ in such a manner that no two adjacent vertices ever share a color. The goal is to minimize the number of colors used. The minimum number of colors necessary to color a graph G is called the *chromatic number* of G and is denoted by $\chi(G)$ or χ. We define a *stable set* as a set of mutually non-adjacent vertices. A *k-coloring* can be viewed as a partition of the vertex set V into k stable sets $S_1, \ldots, S_k$, called *color classes*.

The graph coloring problem has many practical applications such as the creation of timetables, frequency assignments and stock management (e.g. [2–5]). As a result of its importance and simplicity, many methods have been developed to solve or to approach the graph coloring problem. Elaborated exact methods are able to find optimal colorings for graphs

[*] Corresponding author. Tel.: +41 21 693 2548; fax: +41 21 693 5840.
E-mail address: Ivo.Bloechliger@a3.epfl.ch (I. Blöchliger).
[1] Supported by Grant no. 2100-068172.02 of the Swiss National Science Foundation.

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.05.014

with up to about 100 vertices [6–8]. This limit is, of course, a rule of thumb. There are intractable instances that have fewer than 100 vertices and graphs that can be solved optimally that have more than 150 vertices. For larger graphs, we must use heuristics. Strategies range from very simple greedy approaches to elaborate hybrid evolutive heuristics. A set of well-known benchmark graphs can be found on the web [9]. We have tested our approach on a sample of these benchmark graphs (known to be difficult to color), and we have compared our results to similar methods, as well as to the two genetic hybrid algorithms GH [10] and MMT [11], which rank among the most efficient graph coloring algorithms.

The paper is organized as follows: in Section 2, we present the basic tabu search method and some possible extensions, as well as an application to the graph coloring problem. We then review in Section 3 some other famous heuristic methods for graph coloring. In Section 4, we present various ways (existing and new ones) of managing the tabu tenure. Our proposed adaptation of the reactive tabu search method (called PARTIALCOL) to the graph coloring problem is described in Section 5, and computational experiments are reported in Section 6. Finally, some possible further developments will be given in the conclusion.

2. Tabu search and TABUCOL

Local search heuristics operate in a *search space* S, also called a *solution space*. The elements of this space are called *solutions* even if not all elements are solutions to the initial problem. For every solution $s \in S$, a neighborhood $N(s) \subset S$ is defined. A local search method starts at an initial solution, and then moves repeatedly from the current solution to a neighbor solution in order to try to find better solutions, measured by an appropriate objective function. The passage from one solution to the next solution is called a *move*.

Tabu search is a local search algorithm which was originally proposed by Glover [12,13] and Hansen [14]. Let f be an objective function which must be minimized over the solution space S. Its basic version can be described as follows. First, it needs an initial solution $s_0 \in S$ as input. Then, the algorithm generates a sequence of solutions $s_1, s_2, \ldots$ in the search space S such that $s_{i+1} \in N(s_i)$. When a move is performed from s_i to s_{i+1}, the inverse of that move is stored in a *tabu list* L. For the following t iterations, where t is the *tabu tenure* (for historical reasons also called *tabu list length*), a move stays *tabu* and cannot be used (with some exceptions) to generate a neighbor solution. The solution s_{i+1} is computed as $s_{i+1} = \arg\min_{s \in N'(s_i)} f(s)$, where $N'(s)$ is a subset of $N(s)$ containing all solutions s' which can be obtained from s either by performing a move that is not in L (i.e. not tabu) or such that $f(s') < f(s^\star)$, where $s^\star$ is the best solution encountered along the search so far. The process is stopped when a fixed number of iterations without improving $s^\star$ have been performed or when a given amount of CPU time has elapsed.

In practice, one rarely implements a real list of tabu moves. One rather stores a vector which for each possible move contains the iteration number up to which a move stays tabu. If a move is executed, the corresponding entry in the vector is set to the current number of iterations plus the tabu tenure t. A move is tabu if the current number of iterations is smaller than the corresponding entry in the vector. This is not only easier to implement but is also faster in execution, because to test whether or not a move is tabu takes constant time. Testing the presence of an element in the tabu list takes $O(t)$ time, where t is the length of the list. Moreover, there will be some additional time required for adding and deleting elements from the list.

Many variants and extensions of the basic tabu search can be found in [15]. Most of these extensions deal with the organization of the neighborhood of the search and how to choose the next current solution in the neighborhood.

Today, one of the most efficient adaptations of tabu search to the graph coloring problem is called TABUCOL. It was first proposed by Hertz and de Werra in [16], and an improved version can be briefly described as follows [10]: the search space is the set of k-partitions of the vertex set, and the objective function f to minimize is the total number of edges where both vertices have the same color. A neighbor solution is obtained by modifying the color of a vertex which is adjacent to a vertex of the same color. When the color of a vertex x is modified from i to j, color i is declared tabu for vertex x for t of iterations, and all solutions where x has color i are called *tabu solutions*. At each iteration, TABUCOL determines the best neighbor s' of the current solution s (ties between solutions of equal quality are broken randomly) such that either s' is a non-tabu solution or $f(s') < f(s^\star)$, where $s^\star$ is the best solution found so far. The tabu tenure t is set equal to $\text{UNIFORM}(a, b) + \alpha \cdot n_c$, where $\text{UNIFORM}(a, b)$ returns an integer randomly chosen in $\{a, \ldots, b\}$ and n_c represents the number of conflicting vertices in the current solution s. Based on an experimentation phase, Galinier and Hao proposed in [10] to set $\alpha = 0.6$, $a = 0$ and $b = 9$.

3. Various approaches to vertex coloring

There are two widely explored approaches to vertex coloring. The first consists of producing k-colorings and attempting to decrease k while maintaining a (valid) coloring. Most constructive and greedy heuristics make use of this approach. These heuristics are generally very fast but yield solutions which are far from optimal.

The second approach consists of fixing k and starting with an *improper* k-coloring. An *improper* coloring may contain *conflicts*. If two adjacent vertices x and y have the same color, we say that there is a *conflict* between vertices x and y, respectively, vertices x and y are *conflicting vertices*, and the edge $\{x, y\}$ is a *conflicting edge*. One may try then to reduce the number of conflicts until, in the best case, a k-coloring is found. If a k-coloring is found, the method is restarted in order to search for a $(k-1)$-coloring, and so on. If no k-coloring is found, we restart the method to search for a $(k+1)$-coloring, and so on. The process is stopped as soon as a fixed k is considered twice by the method. The output solution is a k-coloring with the smallest k. Most state-of-the-art heuristics are derived from this approach.

A third and barely explored approach uses *partial k-colorings* with a fixed k and a strategy for adapting k derived from the second approach described above. A *partial k-coloring* consists of k mutually disjoint stable sets $S_1, \ldots, S_k$ and a set $\mathcal{O}$ of non-colored vertices such that $\mathcal{O} = V \backslash (S_1 \cup \cdots \cup S_k)$. Note that a partial k-coloring has no conflict. The goal of this third approach is to increase the size of the partial solution, which is equivalent to reducing the size of $\mathcal{O}$. Our method, called PARTIALCOL, is based on this third approach. Morgenstern proposed a complex algorithm based on partial solutions in [17]. More details on his algorithm, henceforth called MOR, will be given in Section 5.3.

Some practical frequency assignment problems [18], for which we know the set of all the interference constraints, could be formulated as graph coloring problems. In this case, a frequency (instead of a color) must be assigned to each antenna (instead of a vertex) while respecting a subset of the set of the interference constraints. For example, there could be an interference between antennae x and y if they transmit on the same frequency. In the *minimum interference frequency assignment problem*, the goal is to minimize the sum of the interferences. In this case, all the antennae must receive a frequency, and interferences are penalized. To solve this, the approach using improper colorings could be appropriate. However, if the goal is to assign a frequency to as many antennae as possible while respecting all the interference constraints (and not only a subset), then the partial solution approach is useful. This problem is known as the *maximum service frequency assignment problem*.

Constructive heuristics build a solution in a step-by-step fashion. In each step, a vertex is chosen and a color is assigned to it such that no conflict is generated. There exist numerous strategies for determining the order in which the vertices are chosen. The most simple approach is the *greedy algorithm*, which chooses vertices randomly and assigns the smallest possible color (supposing the colors are numbered) to the examined vertex. A more successful method called DSATUR [19] consists of always choosing the vertex with the highest number of differently colored adjacent vertices; the vertex that has the highest number of incident edges is used in the event of a tie. Other published methods of this type are RLF [4], KP [5], and the ITERATED GREEDY coloring algorithm [20]. In addition, Laguna and Marti proposed in [21] a *Greedy Randomized Adaptive Search Procedure* (GRASP) for coloring sparse graphs. These methods usually find solutions quickly, but these solutions are often far from optimal. Note that the ITERATED GREEDY algorithm is often combined with local search procedures in order to obtain better results.

Most *local search heuristics* for the graph coloring problem use one of the two following search spaces: (1) a space of all colorings, where the heuristic simply tries to reduce the number of colors used, or (2) a space with improper colorings but a fixed number of colors, where the heuristic attempts to reduce the number of conflicts to zero. The most-studied search strategies are simulated annealing [22,23], tabu search (called TABUCOL and described in Section 2) [10,16], and variable neighborhood search [5,24]. More details and results on these methods can be found in [5]. In [25], Dorne and Hao proposed a tabu search coloring heuristic close to the one described in [10]. In their method, Dorne and Hao use a candidate list strategy, an aspiration criterion, incremental evaluation, and dynamic tabu tenure as for TABUCOL, but with $\alpha \in \{2, 4\}$. Such a method does not outperform the one proposed by Galinier and Hao in [10] and, consequently, we will not compare our algorithms to this method.

Evolutive hybrid heuristics work with a population P of several solutions (or pieces of solutions) at a time and attempt to produce better solutions by recombining solutions (or pieces of solutions) of P. When new solutions are generated from P, the algorithm applies a local search procedure in an effort to improve these solutions. The so obtained solutions are finally used to update P. In such methods, almost all CPU time is consumed by the local search procedure. The most successful coloring methods are of this type and usually use a variant of TABUCOL as a local search procedure. The choice of TABUCOL is often preferred to other local search methods because it is the most simple, rapid, and efficient algorithm among the best-known local search coloring procedures. We would like to draw attention to several genetic hybrid algorithms [10,11,25,26], a scatter search method [27], and an adaptive memory heuristic [5,28]. The last cited method is the most simple method among the methods of this type. It is close to the one proposed in [10]. In both of these cases, an appropriate recombination operator and an efficient and quick local search procedure (TABUCOL) are responsible for the excellent performance exhibited by those methods. The very recent MMT algorithm [11] shows an excellent performance and has improved the best-known coloring on several graphs. It is based on the same neighborhood structure as the PARTIALCOL algorithm to be presented in this paper. It is worth noting that the recombination operator and the variant of local search method used internally must be adapted specifically to the given problem for optimum results.

4. Managing the tabu tenure

We will focus on methods to dynamically adjust the tabu tenure (length of the tabu list) along the search, according to different criteria. We will first define a classification of such methods and give some known examples. We will define three groups of methods to adjust the tabu tenure: *static*, *dynamic*, and *reactive* schemes. Each method may be used in either a *deterministic* or a *randomized* way. We will finally propose a new scheme of automatic adjustment of the tabu tenure called the FOO-*scheme*.

4.1. Static, dynamic, and reactive tabu tenures

We define a *static tabu tenure* as a tenure which does not change during the search process. In the original version of tabu search, the tabu tenure is chosen once and for all and is kept fixed for all instances. This leads to results that compare poorly to other methods of choosing the tabu tenure. Letting the user supply the tabu tenure is not desirable, because the user should not be part of a heuristic. An improvement can be made by choosing the tabu tenure as a function of the instance size or, even better, as a function of the mean size of the neighborhood of a solution. The idea behind this is that the larger the neighborhood of a solution s is, the more possibilities there are to reach this solution. Therefore, the tabu tenure should be larger in order to avoid revisiting s.

We define a *dynamic tabu tenure* as a tenure which depends on the current solution and on the move which has been executed to obtain the current solution. No information about previous visited solutions is required. For the graph coloring heuristic TABUCOL which uses improper k-colorings, the dynamic tenure $\alpha \cdot n_c$, where α is a constant (typically 0.6) and n_c is the number of vertices involved in a conflict in the current improper coloring, has proven to be very efficient over a wide range of instances [10,26]. The above method can be generalized to other problems, provided that the value of the optimal solution is known. This is the case for feasibility problems where the tenure will be determined to be proportional to the number of constraint violations. If the value of the optimal solution is not known, one could compare the quality of the current solution with the quality of the best solution found so far, but this does not fit our definition of a *dynamic tabu tenure*, since it involves information about previously visited solutions.

We define a *reactive tabu tenure* as a tenure determined on the basis of (possibly) the entire search history. We use this more general definition for our purposes. The term *reactive tabu search* was first introduced by Battiti and Tecchiolli in [29]. They propose modifying the size of the tabu tenure when a *cycle* occurs, i.e. when a solution is visited twice by the search process. In their algorithm, an explicit check for the repetition of solutions is added to the basic scheme of tabu search. The appropriate tabu tenure is learned in an automated fashion by reacting to the occurrence of cycles. Each time a cycle is detected, the tabu tenure is increased because we must assume that it was too small to prevent the detected cycle. The tabu tenure is decreased if no cycle is detected during a given number of iterations. In addition, if the search appears to be repeating an excessive number of solutions too often, then the search is diversified by making a number of random moves, proportional to the average of the cycle length, in order to escape a local attractor.

Very often, one can improve the performance of a tabu search by randomizing the tabu tenure. Suppose that we use some arbitrary scheme to determine the tabu tenure t. Then, instead of using t as the tabu tenure, we determine a randomized tenure t_r according to some distribution which normally depends on t. A typical distribution will be a uniform distribution over an interval $[a(t), b(t)]$, i.e. $t_r := \text{UNIFORM}(a(t), b(t))$. Typically a and b are chosen more or less proportional to t. For the TABUCOL heuristic, a very good performance has been reported for $a(t) = t$ and $b(t) = t + 10$, where $t = \alpha \cdot n_c$ [10,26].

4.2. The FOO-scheme: a new scheme

The FOO-*scheme* provides a reactive tabu tenure based on the *fluctuation of the objective* function. The idea behind this scheme is to observe how the objective function evolves along the search process. If the objective function does not change its value (or changes its value very little) during a long period, we have a reason to believe that the search process is trapped in an uninteresting region of the search space. As a reaction, we increase the tabu tenure in an attempt to escape the region where the search process has been trapped. As a counterweight for increasing the tenure, we make the tenure evaporate slowly along the search process. This makes the search process alternate between intensification (when the tenure is low) and diversification (when the tenure is high).

We define three parameters for the FOO-scheme: φ, η, and b. Every φ iterations, we determine Δ, which is the difference between the maximum and minimum values that the objective function has taken during the last φ iterations. We refer to φ as a *frequency*, even if this is technically incorrect (a frequency is the number of events per time, and not the time per event). If Δ is less than or equal to the *threshold* b, we conclude that the search process is probably trapped and increase the tabu tenure t by the *increment* η. Otherwise, if Δ is larger than the threshold b, we decrease t by one (but we forbid negative values of t).

The parameters φ, η, and b of the FOO-scheme must be tuned for a specific problem. We refer to the process of determining good parameter combinations as the *tuning phase*. There are two extreme scenarios to avoid. In the first scenario, the tabu tenure grows beyond all limits. This can happen because the threshold b or the increment η are too large or because φ is too small. In the second scenario, the tabu tenure always stays close to zero and the search process is quickly blocked in an uninteresting region of the search space. This can happen because the threshold b or the increment η are too small or because φ is too large. It is important to mention that, instead of fixing the values of φ, η, and b, they can be chosen randomly every φ iterations from a uniform distribution over some intervals to be determined. The intervals should be chosen such that they include parameter combinations determined in the tuning phase to be good. Randomizing the parameters can avoid over-training in the tuning phase and can make the scheme more robust.

5. The PARTIALCOL algorithm

In this section, we describe a new tabu search method for the graph coloring problem. We first describe the solution space and its associated neighborhood structure. We then present a method of choosing a neighbor solution. Finally, we propose two ways of managing the tabu tenure.

PARTIALCOL is a local search method which takes as input a graph $G = (V, E)$ with vertex set V and edge set E, along with the desired number k of colors to find a k-coloring. A pseudocode is given in Algorithm 1.

5.1. Definition of the search space

The search space consists of *partial k-colorings*. A *partial k-coloring* consists of k disjoint stable sets $S_1, \ldots, S_k$ and a set $\mathcal{O} = V \setminus \bigcup_{i=1}^{k} S_i$. We note $s = (S_1, \ldots, S_k; \mathcal{O})$. Each vertex $v \in V$ is in exactly one of the sets $S_1, \ldots, S_k, \mathcal{O}$. If $v \in S_i$, then v has color i. If $v \in \mathcal{O}$, then v is not colored.

We can see that there is a major difference between the structure of a solution in TABUCOL and the one used for PARTIALCOL. In the approach chosen for PARTIALCOL, all the conflicts are in the set $\mathcal{O}$; however, in TABUCOL, the conflicts could be in any of the k color classes. Consequently, the solution spaces induced by these two solution structures are very different.

Note that the idea to deal with partial assignments is not new. For example, Morgenstern proposed a complex method based on partial k-colorings in [17] for the graph coloring problem (see Section 5.3 for more details). In addition, Vasquez proposed in [30] a tabu search for the frequency assignment problem with polarizations. He worked with partial assignments too. Because these two algorithms both performed well, one of our motivations is to propose a method based on partial assignments, but which is much simpler than the one proposed by Morgenstern.

5.2. Strategy and generation of an initial solution

We first have to determine the first tested k for which we will try to find a k-coloring. In our algorithms, we chose $k=\chi(G)$ if $\chi(G)$ is known, and $k=k^\star$ otherwise, where $k^\star$ is the smallest k for which a k-coloring has been found by a known heuristic. Two possible cases arise: (1) a k-coloring is found or (2) no k-coloring is found. If a k-coloring has been found, we decrease k by one, start over, and continue until no k-coloring can be found. Otherwise, if no k-coloring could be found, we increase k by one and start over, continuing until a k-coloring has been found.

In order to generate an initial partial k-coloring $(S_1, \dots, S_k; \mathcal{O})$, we use the following greedy algorithm. At each step we randomly choose a non-considered vertex v and insert it to S_i with the smallest possible i, such that S_i is still a stable set. If no such set S_i exists, we add v to $\mathcal{O}$.

In order to generate an initial k-partition (or improper k-coloring) $(\mathcal{C}_1, \dots, \mathcal{C}_k)$, we also use a greedy algorithm which is very close to the previous one. At each step, we randomly choose a non-considered vertex v and put it into $\mathcal{C}_i$ with the smallest possible i such that we avoid creating any conflict. If it is impossible to do so without creating a conflict, we put v in a randomly selected class $\mathcal{C}_j$.

Note that the two above greedy algorithms do not consume more than a fraction of a second to generate an initial solution.

5.3. Neighborhood of a solution and selection of a neighbor solution

Let $A(v)$ be the subset of V containing all the vertices adjacent to v. A neighbor solution $s'=(S'_1, \dots, S'_k; \mathcal{O}')$ can be obtained from a solution $s=(S_1, \dots, S_k; \mathcal{O})$ by coloring an uncolored vertex $u \in \mathcal{O}$ with a color c. We call this a *move* $(u \hookrightarrow S_{\mathrm{c}})$. To execute this move, we first set $\mathcal{O}'=\mathcal{O}\backslash\{u\}$ and $S'_{\mathrm{c}}=S_c \cup \{u\}$. It is now possible that S'_c is no longer stable, so we remove vertices adjacent to u in order to make S'_c a stable set. The sets for the new solution s' will be: $\mathcal{O}'=(\mathcal{O}\backslash\{u\}) \cup (A(u) \cap S_c)$, $S'_c=(S_c \cup \{u\})\backslash(A(u) \cap S_c)$; and $S'_j=S_j$ for $j \in \{1, \dots, k\}-\{c\}$.

This kind of neighborhood was first proposed by Morgenstern in [17], but his method is much more complicated than the one proposed here. In Morgenstern's method, a neighbor solution is chosen as it is in a simulated annealing search, i.e. a temperature parameter T is required, but this parameter is difficult to tune. Further, Morgenstern's method uses a second neighborhood structure when the first one cannot improve the best solution found after a fixed number of iterations. This other neighborhood structure is complex and deals with *s-chain interchanges* (refer to [17] for a definition). Moreover, Morgenstern's method is based on a pool of good partial k-colorings and not only on a single partial k-coloring. Finally, Morgenstern uses a specific method called XRLF (presented in [23]) to generate the initial pool of partial solutions.

Note that, with such a neighborhood and the number of uncolored vertices as an evaluation function, many neighbor solutions have the same value. This property makes the search more random and less guided. It might be interesting to cast a new objective function which discriminates more efficiently between different neighbor solutions. However, the danger with such an objective function would be that it might constrict the search and remove too much randomness. As a consequence, the algorithm with this new objective function may perform very well on some graphs with a special structure but poorly on others. An idea to more discriminate between the neighbor solutions is to use another objective function, as proposed in [17], which is defined as follows. Let $deg(x)$ be the degree of vertex x and let the objective function be $\sum_{x\in\mathcal{O}} deg(x)$ instead of $\sum_{x\in\mathcal{O}} 1=|\mathcal{O}|$. Preliminary tests with this objective function have not been conclusive, except for the DIMACS benchmark graphs le450_25c and le450_25d [9], where colorings with 26 colors were found within seconds. Apparently, the objective function is very well suited for the structure of these two graphs but not for others. Interestingly, with this objective function, the algorithm is no longer capable of finding colorings with 15 colors for the graphs le450_15c and le450_15d but only colorings with 16 colors. With the original objective function $(f(s)=|\mathcal{O}|)$, a coloring with 15 colors is found within seconds.

PARTIALCOL examines every possible neighbor solution. For each combination of $v \in \mathcal{O}$ and $c \in \{1, \dots, k\}$, our method evaluates the cardinality $|\mathcal{O}'|$ and chooses the move $(v \hookrightarrow S_c)$ that leads to the smallest $|\mathcal{O}'|$, provided either that the move is not tabu or, if it is tabu, that it leads to a value $|\mathcal{O}'|$ which is smaller than the best value already encountered (denoted by $|\mathcal{O}^\star|$). If several moves lead to the same value of $|\mathcal{O}'|$, then one of them is randomly chosen.

Due to the very large number of times we have to evaluate solutions in local search methods, it is imperative that these evaluations can be performed with maximum efficiency, even at the expense of efficiency in other parts of the method. In our case, we must evaluate, for every vertex $v \in \mathcal{O}$ and every color $c \in \{1, \ldots, k\}$, the number $\gamma_{v,c}$ of vertices adjacent to v with color c. In order to do this efficiently, we store all values $\gamma_{v,c}$ in a matrix Γ of size $n \times k$ (n is the number of vertices and k is the number of colors). We do this for every vertex and not only vertices in $\mathcal{O}$. Once Γ is computed, the evaluation of a move takes constant time. The evaluation of the complete neighborhood takes a time proportional to $k \cdot |\mathcal{O}|$. After a move $(v \hookrightarrow S_c)$ has been selected and executed, Γ is updated as follows.

- For every vertex u adjacent to v, we increase $\gamma_{u,c}$ by one. This update can be done in a time proportional to the number of adjacent vertices to v by storing a list of adjacent vertices for each vertex.
- For every vertex w which is moved back from S_c to $\mathcal{O}$, we decrease the value $\gamma_{u_w,c}$ by one for every neighbor u_w of w. The complexity of this update is small in practice and comparable to the complexity of the first update, which is due to the fact that only very few vertices w will be removed in general. This characteristic is a result of the fact that our algorithm precisely chooses moves minimizing the number of such vertices w.

5.4. Tabu tenure schemes

For our experiments, we have considered two schemes and used them to govern adjustments to the tabu tenure. The first scheme is a dynamic scheme called the DYN-scheme. The other is a reactive scheme and is named the FOO-scheme. We discuss the details of how these schemes were adapted to PARTIALCOL and TABUCOL.

First, in the DYN-scheme, we are taking advantage of the fact that finding a k-coloring is really a feasibility problem. This fact allows us to know the optimal value of the objective function, provided that a k-coloring exists. The tabu tenure will be set to a value proportional to the objective function. For the TABUCOL algorithm, we have used the following tabu tenure: $t = 0.6 \cdot n_c + \text{UNIFORM}(0, 9)$. Recall that this dynamic tenure has been successfully applied to graph coloring in [10,26]. The algorithm using this scheme will be denoted by DYN-TABUCOL. For the PARTIALCOL algorithm, preliminary tests have shown that the same dynamic tabu tenure as for TABUCOL leads to good results. We have replaced n_c (the number of vertices involved in a conflict in the current solution) with the number of uncolored nodes $|\mathcal{O}|$, i.e. $t = 0.6 \cdot |\mathcal{O}| + \text{UNIFORM}(0, 9)$. The algorithm using this scheme will be denoted by DYN-PARTIALCOL.

Second, for the FOO-scheme (see Section 4.2 for a detailed description of this reactive scheme), we have used the same randomized parameter set (obtained by the tuning phase) for both the PARTIALCOL and TABUCOL algorithms, which are: $\varphi \in [500; 5000]$, $\eta \in [5; 30]$, and $b \in [1; 2]$. Every φ iterations, the three parameters are drawn uniformly from each range. The algorithms using the FOO-schemes will be denoted by FOO-PARTIALCOL and FOO-TABUCOL. Note that the above values of parameters were obtained after performing several numerical experiments which will not be described here.

Algorithm 1. PARTIALCOL: search for a k-coloring for graph G (source code is available at http://www.bloechligair.ch/science/)

```
Input: Graph $G$, initial solution $s = \{S_1, \ldots, S_k; \mathcal{O}\}$, integers $k \geqslant 1$, $i_{\max} \geqslant 0$, a scheme to adjust the tabu tenure.
Output: A $k$-coloring of $G$ or a message that no $k$-coloring has been found.
1.  $i \leftarrow 0$ /* iteration counter */
2.  initialize the tabu tenure $t$ according to the chosen scheme
3.  $s^\star \leftarrow s$ /* best solution encountered */
4.  $|\mathcal{O}^\star| \leftarrow |\mathcal{O}|$
5.  while $i \leqslant i_{\max}$ and $\mathcal{O} \neq \emptyset$ do
6.      compute all moves $M \leftarrow \{(v \hookrightarrow S_q) | v \in \mathcal{O},\ 1 \leqslant q \leqslant k\}$
7.      remove tabu moves from $M$ which do not lead to a new best solution
8.      choose a move $(u \hookrightarrow S_q) \in M$ minimizing $|\mathcal{O}'|$, break ties randomly
9.      apply move $(u \hookrightarrow S_q)$ to $s$
10.     determine the tenure $t$ according to the chosen scheme
```

```
11.         set the move $(w \hookrightarrow S_q)$ tabu for the next $t$ iterations where $w$ runs over all vertices removed from $S_q$
12.         $i \leftarrow i + 1$
13.         **if** $|\mathcal{C}| < |\mathcal{C}^\star|$ **then**
14.             $s^\star \leftarrow s,\ |\mathcal{C}^\star| \leftarrow |\mathcal{C}|,\ i \leftarrow 0$
15.         **end if**
16.     **end while**
17.     **if** $\mathcal{C} = \emptyset$ **then**
18.         **return** $k$-coloring $s$
19.     **end if**
20.     **return** no $k$-coloring is found
```

6. Numerical results

In this section, we present numerical results in order to compare the four following algorithms: DYN-PARTIALCOL, FOO-PARTIALCOL, DYN-TABUCOL, and FOO-TABUCOL. In addition, we will compare their performance with other methods which are considered among the best ones. Two series of tests were made with a limit of, respectively, 60 and 600 min of CPU-time per run (on a 2 GHz Pentium 4 with 512 MB of RAM).

6.1. Instances

We have chosen a sample of 20 graphs (see below), which are well known as they were given at the COLORING02 workshop [9]. These graphs encompass all large and difficult graphs used for the famous DIMACS benchmarks [31]. Other graphs in this collection of benchmark instances are very easy to solve, in the sense that a sophisticated greedy algorithm or a short application of a very basic tabu search finds an optimal solution in a very short time (see [28] for details). In addition, most of the non-considered benchmark graphs are very easy to color in the sense that all our heuristics could color them very quickly with either the optimal number of colors where known, or with the least known number of colors. The following graphs could be colored 20 out of 20 times by all four algorithms within less than 1 min CPU time: anna, fpsol2.i.1, games120, homer, huck, inithx.i.1, jean, le450_25a, le450_25b, le450_5a, le450_5b, le450_5c, le450_5d, miles1500, miles250, mulsol.i.1, myciel3, myciel4, myciel5, myciel6, myciel7, queen10_10, queen11_11, queen12_12, queen13_13, queen5_5, queen6_6, queen7_7, queen8_12, queen8_8, queen9_9, school1, school1_nsh, and zeroin.i.1. Note that in almost all cases, the coloring was found within seconds. For the following graphs, a coloring was found by all four algorithms in at least 10 out of 20 runs within a minute: flat300_20_0, flat300_26_0, david, inithx.i.2, inithx.i.3, miles1000, miles500, miles750, mulsol.i.2, mulsol.i.3, mulsol.i.4, mulsol.i.5, zeroin.i.2, and zeroin.i.3. The following graphs were colored by at least one of the four algorithms one out of 20 runs within less than 1 min: fpsol2.i.2, fpsol2.i.3, le450_15a, le450_15b, queen14_14, queen15_15, and queen16_16. Consequently, all these experiments confirm the ones in [28].

The instances which will be considered in our experiments are of different types, are known to be difficult to color, and most of them have been studied by other researchers (e.g. [5,10,17,25–28]).

- There are *random graphs* [23], named "DSJC$n.d$", where n is the number of vertices and d is 10 times the density. They are generated in such a way that the probability of an edge being present between two given vertices equals the density.
- There are *geometric random graphs* [23], named "DSJR$n.r$" and "R$n.r$". They are generated by choosing randomly n points in the unit square, which will be the vertices of the graph, and by joining two vertices by an edge, if the two corresponding points are at a distance less than r from each other. Instances with a letter "c" appended to their name are simply the complement of an instance without the "c".
- There are *flat graphs* [20] named "flatn_χ_δ", where n is the number of vertices, χ is the chromatic number, and δ is a *flatness parameter* giving the maximal allowed difference between the degrees of two vertices.
- The *Leighton graphs* are generated to have a given chromatic number using a sophisticated randomized algorithm [4]. The instances we use have been generated by Morgenstern and they are named "le$n_\chi x$" where n is the number

Table 1
Results for DYN-PARTIALCOL with a CPU-time limit of 60 min

Graph	$\|V\|$	χ, $k^\star$	k	Success	CPU sec.	iter. (in thousands)
DSJC1000.1	1000	?,20	20	3 of 50	2301	292 947.5
			21	50 of 50	2	277.7
DSJC1000.5	1000	?,83	89	6 of 50	2786	45 502.6
DSJC1000.9	1000	?,224	228	30 of 50	2275	14 826.7
DSJC500.1	500	?,12	12	50 of 50	193	38 819.9
DSJC500.5	500	?,48	49	1 of 50	811	55 679.3
			50	50 of 50	291	22 684.8
DSJC500.9	500	?,126	127	1 of 50	1680	43 409.5
			128	50 of 50	347	9885.4
DSJR500.1c	500	?,85	85	3 of 50	989	56 980.8
DSJR500.5	500	?,122	126	28 of 50	1544	79 620.2
			127	44 of 50	631	34 271.7
			128	47 of 50	147	7867.2
R1000.1c	1000	?,98	99	20 of 50	1895	28 967.6
R1000.5	1000	?,234	249	10 of 50	2098	25 024.1
			250	30 of 50	2020	24 472.9
			251	47 of 50	1515	18 487.7
			252	50 of 50	845	10 178.7
R250.1c	250	?,64	64	4 of 50	635	89 068.3
			65	13 of 50	488	64 974.2
			66	16 of 50	225	29 462
R250.5	250	?,65	66	6 of 50	2400	296 613
			67	45 of 50	241	31 809.9
flat1000_50_0	1000	50,50	50	50 of 50	26	107.9
flat1000_60_0	1000	60,60	60	50 of 50	91	390.4
flat1000_76_0	1000	76,83	88	9 of 50	2376	40 543.7
flat300_28_0	300	28,31	28	13 of 50	1878	154 261.2
			29	35 of 50	1398	133 092.5
			30	46 of 50	1221	131 767.5
			31	49 of 50	652	79 871.8
le450_15c	450	15,15	15	50 of 50	3	615.7
			16	50 of 50	7	1617.4
le450_15d	450	15,15	15	50 of 50	22	4682.1
			16	50 of 50	7	1713.6
le450_25c	450	25,25	27	50 of 50	10	1583.3
le450_25d	450	25,25	27	50 of 50	7	1151.4

of vertices, χ the chromatic number, and x is a lower case letter to distinguish different graphs with similar parameter settings.

6.2. *Analysis of the results in the first series of tests (1 h)*

When we allow a limit of 60 min, a comparison of the results for all four algorithms, namely DYN-PARTIALCOL, FOO-PARTIALCOL, DYN-TABUCOL, and FOO-TABUCOL, is given in Tables 1–4. For each graph, each algorithm and each tested k, we executed 50 runs with a different random seed. For each tested instance, we report the number $|V|$ of vertices, the chromatic number χ when known, and the value $k^\star$ which is the smallest k for which a k-coloring has been found by an algorithm (at time of publication). In addition, we report the number of colors k for which a k-coloring was found at least one time out of 50. If more than one k was tested, we report (at most) the four smallest values of k for which a k-coloring could be found. For each algorithm, we report the number of colors k for which a k-coloring was found, the number of successful runs out of the 50, the average number of iterations (in thousands) needed to obtain such k-colorings, and the average CPU time in seconds needed to find the k-colorings. A summary for these results is reported in Table 5, where we only present the best number of colors found by each algorithm. We indicate the best results (among these four methods) in bold.

Table 2
Results for FOO-PARTIALCOL with a CPU-time limit of 60 min

Graph	$\|V\|$	$\chi, k^\star$	k	Success	CPU sec.	iter. (in thousands)
DSJC1000.1	1000	?,20	21	50 of 50	3	441.9
DSJC1000.5	1000	?,83	89	5 of 50	1893	32 399.6
DSJC1000.9	1000	?,224	228	20 of 50	2296	13 373.1
DSJC500.1	500	?,12	12	23 of 50	1811	324 770.8
DSJC500.5	500	?,48	50	50 of 50	337	26 470.5
DSJC500.9	500	?,126	128	48 of 50	1260	32 862
DSJR500.1c	500	?,85	85	50 of 50	438	21 243.1
DSJR500.5	500	?,122	128	24 of 50	1808	56 146.6
R1000.1c	1000	?,98	98	2 of 50	604	8070.7
			99	30 of 50	1769	26 593.4
R1000.5	1000	?,234	252	2 of 50	1967	15 678.4
			253	35 of 50	1920	16 310.1
R250.1c	250	?,64	64	50 of 50	4	453.9
			65	50 of 50	0	20.4
			66	50 of 50	0	8
R250.5	250	?,65	67	39 of 50	1331	128 496.8
flat1000_50_0	1000	50,50	50	50 of 50	111	581.4
flat1000_60_0	1000	60,60	60	50 of 50	527	2929.9
flat1000_76_0	1000	76,83	88	10 of 50	2332	40 546.5
flat300_28_0	300	28,31	28	35 of 50	1905	179 148.1
			29	24 of 50	1587	162 908.3
			30	26 of 50	1525	168 488.1
			31	34 of 50	1464	184 134
le450_15c	450	15,15	15	50 of 50	1	230
			16	50 of 50	1	137.5
le450_15d	450	15,15	15	50 of 50	3	592.9
			16	50 of 50	1	179.2
le450_25c	450	25,25	27	50 of 50	4	707.5
le450_25d	450	25,25	27	50 of 50	3	583.6

If we compare FOO-PARTIALCOL and FOO-TABUCOL, FOO-PARTIALCOL is better on four graphs but worse on six graphs. Both algorithms obtain the same results on 10 graphs. In such a situation, we give a score denoted by 6-4-(10) in favor of FOO-TABUCOL. Based on this score, the methods are comparable. Thus, we need to go further in the comparison and count their success rates over the 50 runs. In this situation, the score is 6-3-(1) in favor of FOO-PARTIALCOL. If we compare DYN-PARTIALCOL and DYN-TABUCOL, the score is 4-4-(12). When comparing the success rates, the score is 6-4-(2) in favor of DYN-TABUCOL. Consequently, when allowing 1 h of computation, TABUCOL and PARTIALCOL obtain comparable results.

If we compare FOO-PARTIALCOL and DYN-PARTIALCOL, the score is 6-1-(13) in favor of DYN-PARTIALCOL. If we compare FOO-TABUCOL and DYN-TABUCOL, the score is 5-4-(11), which is slightly in favor of DYN-TABUCOL. When comparing the success rates, the score is 5-4-(1) in favor of FOO-TABUCOL. Consequently, when allowing 1 h of computation, DYN seems to perform slightly better than FOO. As the results are varied, one may suggest to implement both ways of managing the tabu tenures with a mechanism to alternate between them.

No obvious conclusion can be made when comparing TABUCOL and PARTIALCOL according to the density of the graphs. But it could be interesting to compare TABUCOL and PARTIALCOL according to the "nature" of the graphs.

On the *random graphs* (the DSJCs, the DSJRs and the R's, which represent 12 graphs), TABUCOL performs generally better than PARTIALCOL. On the "flat" and "Leighton's" graphs, the results are much less predictable. While one algorithm finds a k-coloring very quickly, the other algorithm might even be challenged to find a $(k+1)$-coloring.

On the "*Leighton's*" *graphs* le450_25c and le450_25d, TABUCOL finds colorings with 26 colors quite easily, while PARTIALCOL is only able to find 27-colorings. Surprisingly, this situation is reversed for the graphs le450_15c and le450_15d, where PARTIALCOL finds optimal colorings very quickly, but TABUCOL finds optimal colorings only rarely or not at all. It seems that this is mainly due to the objective function. Preliminary tests have shown that if we use the sum of the degrees of all vertices in $\mathcal{O}$ as the objective function for PARTIALCOL, it easily finds a 26-coloring for the

Table 3
Results for DYN-TABUCOL with a CPU-time limit of 60 min

Graph	$\|V\|$	$\chi, k^\star$	k	Success	CPU sec.	iter. (in thousands)
DSJC1000.1	1000	?,20	20	14 of 50	1855	224 021.7
			21	50 of 50	1	161.8
DSJC1000.5	1000	?,83	89	48 of 50	1224	17 482.6
DSJC1000.9	1000	?,224	227	48 of 50	1520	7198.4
			228	50 of 50	718	3384.8
DSJC500.1	500	?,12	12	50 of 50	48	8878.8
DSJC500.5	500	?,48	49	11 of 50	1550	69 803.6
			50	50 of 50	25	1567.4
DSJC500.9	500	?,126	127	50 of 50	328	7198.4
			128	50 of 50	32	718.7
DSJR500.1c	500	?,85	85	1 of 50	685	55 458.3
DSJR500.5	500	?,122	126	5 of 50	746	56 818.8
			127	12 of 50	154	10 387
			128	18 of 50	313	19 643.7
R1000.1c	1000	?,98	98	47 of 50	812	28 505.3
			99	44 of 50	308	12 207.7
R1000.5	1000	?,234	249	41 of 50	1245	29 509.4
			250	44 of 50	740	17 605.1
			251	50 of 50	381	8517.2
			252	48 of 50	205	4569.9
R250.1c	250	?,64	66	2 of 50	0	0.1
R250.5	250	?,65	67	11 of 50	528	64 103.3
flat1000_50_0	1000	50,50	50	50 of 50	421	732.8
flat1000_60_0	1000	60,60	60	49 of 50	1415	2099.6
flat1000_76_0	1000	76,83	88	46 of 50	1173	16 532.9
flat300_28_0	300	28,31	31	50 of 50	378	32 521.3
le450_15c	450	15,15	16	50 of 50	4	847.7
le450_15d	450	15,15	15	1 of 50	12	2246.6
			16	49 of 50	14	3572.4
le450_25c	450	25,25	26	49 of 50	9	954.6
			27	50 of 50	0	14.4
le450_25d	450	25,25	26	50 of 50	12	1313.3
			27	50 of 50	0	15.7

le450_25c/d graphs but no 15-coloring for the le450_15c/d graphs. However, such a discriminating objective function does not generally lead to better results on the other graphs. Consequently, it may be an interesting avenue of research to test intermediate objective functions within our approach, such as $\sum_{x\in O} \lceil deg(x)/\beta \rceil$ for some $\beta \geqslant 1$, where $deg(x)$ is the degree of the vertex x. The advantage of this formula is that we can continuously shift β from 1 to the maximal degree $deg(G)$ of the graph G to get the two extremes. For $\beta = deg(G)$, we obtain simply $|\mathcal{O}|$; and, for $\beta = 1$, we obtain the sum of the degrees of the vertices in $\mathcal{O}$.

On the "*flat*" *graphs*, the partial solution neighborhood seems to be more appropriate. For the graph flat300_28_0, we even find an optimal coloring using 28 colors! At the time of publication, the best coloring ever discovered by other algorithms required 31 colors. Note that for the "flat" graphs, we take into account the very important remark appearing on the web page http://www.cs.ualberta.ca/joe/Coloring/, thus, our results are relevant. This remark mentioned that, in older versions of the instances, without first permuting the order of the vertices, a simple greedy approach may optimally color some of the "flat" graphs.

Note that we also performed additional experiments (with a time limit of 1 h too) on several random graphs of size $|V| \in \{100; 250; 500; 1000\}$ and density $d \in \{0.1; 0.5; 0.9\}$ (we generated three graphs for each pair of values $\{|V|; d\}$ and executed each algorithm five times on each graph). These graphs are obtained by linking a pair of vertices by an edge with probability d, independently for each pair. On all graphs with $d = 0.1$, and on the graphs of size $|V| \in \{100; 250\}$, the four above algorithms obtained exactly the same results in terms of the number of used colors. However, on the remaining graphs and as generally observed on the random graphs from the DIMACS instances (i.e. the DSJCs,

Table 4
Results for FOO-TABUCOL with a CPU-time limit of 60 min

Graph	$\lvert V \rvert$	$\chi, k^\star$	k	Success	CPU sec.	iter. (in thousands)
DSJC1000.1	1000	?,20	21	50 of 50	2	188.8
DSJC1000.5	1000	?,83	89	22 of 50	2001	31 773.6
DSJC1000.9	1000	?,224	227	39 of 50	2068	8760
			228	50 of 50	1120	5074.6
DSJC500.1	500	?,12	12	50 of 50	242	40 328
DSJC500.5	500	?,48	49	6 of 50	1776	101 117.2
			50	50 of 50	59	3925.4
DSJC500.9	500	?,126	127	47 of 50	1037	20 129
			128	50 of 50	58	1249.8
DSJR500.1c	500	?,85	85	13 of 50	1347	68 509.1
DSJR500.5	500	?,122	128	17 of 50	1494	31 236.8
R1000.1c	1000	?,98	98	50 of 50	652	15 795.8
			99	50 of 50	140	3801.1
R1000.5	1000	?,234	254	8 of 50	1140	7532.8
			255	31 of 50	645	4909.4
R250.1c	250	?,64	64	49 of 50	144	17 726.6
			65	50 of 50	6	779.1
			66	50 of 50	0	37.1
R250.5	250	?,65	67	13 of 50	1834	108 584.7
flat1000_50_0	1000	50,50	73	2 of 50	2987	8246.6
			74	2 of 50	2570	8120.4
			75	6 of 50	2489	8425.5
			76	12 of 50	2367	8377.5
flat1000_60_0	1000	60,60	79	1 of 50	3313	16 661.2
			80	2 of 50	2912	16 527.6
			81	10 of 50	2943	18 253.6
			82	19 of 50	2870	20 018.5
flat1000_76_0	1000	76,83	87	1 of 50	3060	36 751.4
			88	14 of 50	1962	31 241.9
flat300_28_0	300	28,31	29	1 of 50	1208	77 181
			30	2 of 50	1007	79 914.7
			31	50 of 50	515	50 391.7
le450_15c	450	15,15	15	32 of 50	1072	193 694.3
			16	50 of 50	4	193.4
le450_15d	450	15,15	15	11of 50	995	161 652.8
			16	50 of 50	3	189.7
le450_25c	450	25,25	26	50 of 50	19	2123.1
			27	50 of 50	0	28.2
le450_25d	450	25,25	26	50 of 50	25	2819.7
			27	50 of 50	0	27

the DSJRs and the R's), TABUCOL performed in average better than PARTIALCOL (i.e. TABUCOL found colorings with fewer colors than PARTIALCOL), and the DYN-scheme was in average slightly better than the FOO-scheme.

6.3. Comparison with other algorithms

We will informally compare the four considered algorithms with the following methods: MOR [17], GH [10], and MMT [11], where the last two are considered as the most efficient graph coloring heuristics in terms of the number of used colors. However, because we do not report the CPU times required by GH, MOR, and MMT, it is not possible to ensure that conditions were equivalent, so any comparison between these algorithms presented here must be made with care. We give results of tests made with GH, MOR, and MMT so that the reader may have a baseline by which he may evaluate DYN-PARTIALCOL, FOO-PARTIALCOL, DYN-TABUCOL, and FOO-TABUCOL. In Table 5, we have included the best colorings found by the GH, the MOR and the MMT algorithm, if available, for comparison. Note first that Table 5 shows that the only graph on which PARTIALCOL outperforms every other method is the graph flat300_28_0.

Table 5
Comparison of the four algorithms with a CPU-time limit of 60 min

Graph	$\|V\|$	χ, $k^\star$	PARTIALCOL		TABUCOL		GH	MOR	MMT
			FOO	DYN	FOO	DYN			
DSJC1000.1	1000	?,20	21	**20**	21	**20**	20	21	20
DSJC1000.5	1000	?,83	**89**	**89**	**89**	**89**	83	88	83
DSJC1000.9	1000	?,224	228	228	**227**	**227**	224	226	226
DSJC500.1	500	?,12	**12**	**12**	**12**	**12**	12	12	12
DSJC500.5	500	?,48	50	**49**	**49**	**49**	48	49	48
DSJC500.9	500	?,126	128	**127**	**127**	**127**	126	128	127
DSJR500.1c	500	?,85	**85**	**85**	**85**	**85**	–	85	85
DSJR500.5	500	?,122	128	**126**	128	**126**	–	123	122
R1000.1c	1000	?,98	**98**	99	**98**	**98**	–	98	98
R1000.5	1000	?,234	252	**249**	254	**249**	–	241	237
R250.1c	250	?,64	**64**	**64**	**64**	66	–	64	64
R250.5	250	?,65	67	**66**	67	67	–	65	65
flat1000_50_0	1000	50,50	**50**	**50**	73	**50**	50	50	50
flat1000_60_0	1000	60,60	**60**	**60**	79	**60**	60	60	60
flat1000_76_0	1000	76,83	88	88	**87**	88	83	89	82
flat300_28_0	300	28,31	**28**	**28**	29	31	31	31	31
le450_15c	450	15,15	**15**	**15**	**15**	16	15	15	15
le450_15d	450	15,15	**15**	**15**	**15**	**15**	15	15	15
le450_25c	450	25,25	27	27	**26**	**26**	26	25	25
le450_25d	450	25,25	27	27	**26**	**26**	26	25	25

Only the smallest value of k for which a k-coloring has been found is reported for each algorithm. The lowest values of k for each graph among the four algorithms FOO-PARTIALCOL, DYN-PARTIALCOL, FOO-TABUCOL, and DYN-TABUCOL are in bold face. The CPU times for the three algorithms GH, MOR, and MMT are not reported and are sensibly larger than 60 min for some large and difficult graphs. However, these results are given so that the reader may qualitatively compare our methods with state-of-the-art methods.

It is interesting to compare PARTIALCOL with MOR [17], because both methods use the same solution structure, i.e. partial k-colorings. Recall that the MOR-algorithm is a simulated annealing algorithm with sophisticated management of the temperature and alternative neighborhoods to escape local minima. Moreover, MOR uses a pool of solutions instead of only one. Nevertheless, we can observe that the results are similar. While the two methods find the same results on nine graphs (namely DSJC500.1, DSJC500.5, DSJR500.1c, R1000.1c, R250.1c, flat1000_50_0, flat1000_60_0, le450_15c, and le450_15d), MOR is better on seven graphs (namely DSJC1000.5, DSJC1000.9, DSJR500.5, R1000.5, R250.5, le450_25c, and le450_25d) and the PARTIALCOL algorithms find better colorings for four graphs (namely DSJC1000.1, DSJC500.9, flat1000_76_0, and flat300_28_0).

On the 13 graphs for which we have results from GH (excluding flat300_28_0 on which our algorithms are better), PARTIALCOL finds colorings with the same number of colors in six cases and a larger number of colors in seven cases.

PARTIALCOL's performance equals that of MMT in 10 cases, but MMT outperforms PARTIALCOL in nine cases.

However, one should consider that PARTIALCOL and TABUCOL are only simple local search methods, and that GH, MOR, and MMT are much more complex. Further, for large and difficult graphs, GH, MOR, and MMT required more CPU time than 60 min to which PARTIALCOL and TABUCOL were limited for our experiments, so the comparison has to be done with care. In addition, recall that GH (resp. MMT) is an hybrid evolutive method using DYN-TABUCOL (resp. a local search procedure based on the same neighborhood structure presented in PARTIALCOL) as intensification procedure. Thus, these two methods have lots of other sophisticated ingredients which are not used in PARTIALCOL and TABUCOL.

6.4. Analysis of the results in the second series of tests (10 h)

Based on the results of the first series of tests, we have selected 10 graphs where the four considered algorithms are not as good as the best-known methods. The results are given in Table 6. Our goal is to measure the improvement of the results if we allow much more time to each method.

Table 6
Test results for a CPU-time limit of 10 h and a limit of 2×10^9 iterations without improvement

Graph	$\|V\|$	$\chi, k^\star$	k	PARTIALCOL		TABUCOL	
				FOO	DYN	FOO	DYN
DSJC1000.5	1000	?,83	88	0 of 5	0 of 5	1 of 5	2 of 5
DSJC1000.9	1000	?,224	225	0 of 5	0 of 5	0 of 5	2 of 5
			226	0 of 5	2 of 5	5 of 5	5 of 5
			227	3 of 5	5 of 5	5 of 5	5 of 5
DSJC500.5	500	?,48	48	2 of 5	0 of 5	0 of 5	0 of 5
			49	3 of 5	3 of 5	4 of 5	5 of 5
DSJC500.9	500	?,126	126	0 of 5	0 of 5	0 of 5	1 of 5
			127	0 of 5	1 of 5	5 of 5	5 of 5
DSJR500.5	500	?,122	125	0 of 5	1 of 5	0 of 5	0 of 5
			126	0 of 5	5 of 5	0 of 5	1 of 5
			127	0 of 5	5 of 5	1 of 5	2 of 5
R1000.5	1000	?,234	247	0 of 5	0 of 5	0 of 5	1 of 5
			248	0 of 5	5 of 5	0 of 5	3 of 5
			249	0 of 5	5 of 5	0 of 5	4 of 5
			250	0 of 5	5 of 5	0 of 5	4 of 5
			251	0 of 5	5 of 5	0 of 5	5 of 5
R250.5	250	?,65	66	0 of 5	2 of 5	0 of 5	0 of 5
flat1000_76_0	1000	76,83	87	0 of 5	1 of 5	1 of 5	1 of 5
le450_25c	450	25,25	25	2 of 5	4 of 5	2 of 5	0 of 5
			26	1 of 5	4 of 5	5 of 5	5 of 5
le450_25d	450	25,25	25	3 of 5	3 of 5	2 of 5	0 of 5
			26	2 of 5	3 of 5	5 of 5	5 of 5

For each graph and each reported value of k, five runs with different random seeds were executed. We report the number of successful runs.

FOO-PARTIALCOL is now able to color the graph DSJC1000.9 with 227 colors instead of 228, which is again not as good as the best methods. But FOO-PARTIALCOL can color DSJC500.5 with 48 colors instead of 50, and the graphs le450_25c and le450_25d with 25 colors instead of 27! For the three latter graphs, the obtained results are now as good as the best known. In addition, it is important to mention that no other local search method was able to color DSJC500.5 with 48 colors, and that GH only colored le450_25c and le450_25d with 26 colors.

DYN-PARTIALCOL is now able to save one color on the graphs DSJC500.5 and R1000.5, and to save two colors on the graphs DSJC1000.9, flat1000_76_0, le450_25c, and le450_25d. For the two latter graphs, the obtained results are now as good as the best known.

FOO-TABUCOL is now able to save one color on the graphs DSJC500.5, DSJC500.9, and DSJC500.5, and to optimally color the graphs le450_25c and le450_25d.

DYN-TABUCOL is now able to save one color on the graphs DSJC1000.5, DSJC500.9, and flat1000_76_0, and to save two colors on the graphs R1000.5 and DSJC1000.9.

Consequently, it was worth to perform the four algorithms for a longer time because most of the results were improved.

It could be interesting to analyze, using Tables 5 and 6, the performance of DYN-TABUCOL and FOO-PARTIALCOL, because the former algorithm is an existing method introduced in [10], and the latter method is the newest among the four considered algorithms in this paper (the way of managing the tabu tenure is new and the neighborhood structure was never used in a tabu search before). We can observe that DYN-TABUCOL is better on seven graphs but worse on six graphs. If we compare DYN-TABUCOL with the best-known methods, it got similar results on nine graphs and worse results on 11 graphs. If we compare FOO-PARTIALCOL with the best-known methods, it obtained similar results on 11 graphs, worse results on eight graphs, but a better result on one graph. Consequently, FOO-PARTIALCOL could be considered at least as good as DYN-TABUCOL.

When considering all the efficient local search coloring heuristic (e.g. [16,17,22–24]), FOO-PARTIALCOL is obviously better than the methods proposed in [16,22–24], and it is comparable but much more simpler than the algorithm proposed in [17]. Consequently, we put forth that FOO-PARTIALCOL should be considered as one of the best local search coloring heuristic developed to date.

7. Conclusion and further work

In this paper, we have presented FOO-PARTIALCOL, a local search approach to the graph coloring problem. We have shown that this heuristic obtains competitive results on a large sample of benchmark graphs which are generally agreed to be difficult to color. In contrast to several coloring algorithms, PARTIALCOL is very simple and efficient.

Contemporary state-of-the-art coloring methods are hybrid evolutive algorithms using TABUCOL as the internal local search procedure. PARTIALCOL now offers a powerful alternative to TABUCOL and its variations. Recent advances [11] show the potential of this approach.

The FOO-scheme can improve the performances of both TABUCOL and PARTIALCOL. The reactive tabu tenure adjusts itself depending not only on the graph but also on the state of the search. Moreover, the proposed reactive tabu tenure is very easy to implement and, in contrast to other methods, it does not need to perform an explicit check for the repetition of configurations. Determination of the tabu tenure requires only the variation of the objective function. Hence, in contrast to the dynamic tabu tenure strategies proposed by Galinier and Hao in [10], and Dorne and Hao in [25], FOO-scheme's nature does not rely on the specificity of the problem (recall that we need the number of conflicting vertices for the DYN-scheme). Consequently, the use of this new and simple way of adjusting the tabu tenure could easily be tested on other problems where a tabu method has been applied. It should improve the results of the method (especially if the tabu status is static) and simplify its use, once the parameters have been tuned.

Finally, we would like to mention that the paradigm of partial solutions is applicable to other combinatorial problems such as timetabling and frequency assignments.

References

[1] Garey M, Johnson DS. Computer and intractability. San Francisco: Freeman; 1979.
[2] Chaitin GJ. Register allocation and spilling via graph coloring. In: Proceedings of ACM SIGPLAN 82 symposium on compiler construction. 1982. p. 98–105.
[3] Gamst A, Rave W. On the frequency assignment in mobile automatic telephone systems. In: Proceedings of GLOBECOM'92. 1992. p. 309–15.
[4] Leighton FT. A graph coloring algorithm for large scheduling problems. Journal of Research of the National Bureau Standard 1979;84: 489–505.
[5] Zufferey N. Heuristiques pour les Problèmes de la Coloration des Sommets d'un Graphe et d'Affectation de Fréquences avec Polarités. PhD Thesis, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne; 2002.
[6] Hertz A. Application des Métaheuristiques à la Coloration des Sommets d'un Graphe. Traités IC2 Hermès, 2003. Chapitre 3 dans Métaheuristiques et outils nouveaux en recherche opérationnelle.
[7] Mehrotra A, Trick M. A column generation approach for graph coloring. INFORMS Journal On Computing 1996;8(4):344–54.
[8] Peeöller J. A correction to Brélaz's modification of Brown's coloring algorithm. Communications of the ACM 1983;26:593–7.
[9] Dimacs graph coloring benchmarks, ⟨http://mat.gsia.cmu.edu/COLORING02/⟩. Webpage.
[10] Galinier P, Hao J-K. Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization 1999;3:379–97.
[11] Malaguti E, Monaci M, Toth P. A metaheuristic approach for the vertex coloring problem. Technical report OR/05/3, DEIS—University of Bologna, Italy; 2005.
[12] Glover F. Tabu search—part I. ORSA Journal on Computing 1989;1:190–205.
[13] Glover F. Tabu search—part II. ORSA Journal on Computing 1990;2:4–32.
[14] Hansen P. The steepest ascent mildest descent heuristic for combinatorial programming. In: Congress on numerical methods in combinatorial optimization. Capri, Italy, 1986.
[15] Glover F, Laguna M. Tabu search. Boston: Kluwer Academic Publishers; 1997.
[16] Hertz A, de Werra D. Using tabu search techniques for graph coloring. Computing 1987;39:345–51.
[17] Morgenstern C. Distributed coloration neighborhood search. Discrete Mathematics and Theoretical Computer Science 1996;26:335–58.
[18] Aardal KI, van Hoesel SPM, Koster AMCA, Mannino C, Sassano A. Models and solution techniques for frequency assignment problems. 4OR 2003;1(4):261–317.
[19] Brélaz D. New methods to color vertices of a graph. Communications of the Association for Computing Machinery 1979;22:251–6.
[20] Culberson JC. Iterated greedy graph coloring and the difficulty landscape. Technical report, Department of Computer Sciences, University of Alberta; 1992.
[21] Laguna M, Marti R. A GRASP for coloring sparse graphs. Computational Optimization and Applications 2001;19:165–78.
[22] Chams M, Hertz A, de Werra D. Some experiments with simulated annealing for coloring graphs. European Journal of Operational Research 1987;32:260–6.

[23] Johnson DS, Aragon CR, McGeoch LA, Schevon C. Optimization by simulated annealing: an experimental evaluation, part II graph coloring and number partitioning. Operations Research 1991;39:378–406.

[24] Avanthay C, Hertz A, Zufferey N. A variable neighborhood search for graph coloring. European Journal of Operational Research 2003;151: 379–88.

[25] Dorne R, Hao J-K. A new genetic local search algorithm for graph coloring. Lecture Notes in Computer Sciences 1998;1498:745–54.

[26] Fleurent C, Ferland JA. Genetic and hybrid algorithms for graph coloring. Annals of Operations Research 1996;63:437–61.

[27] Hamiez J-P, Hao J-K. Scatter search for graph coloring. Lecture Notes in Computer Sciences 2002;2310:168–79.

[28] Hertz A, Galinier P, Zufferey N. An adaptive memory algorithm for the graph coloring problem. Discrete Applied Mathematics 2005, accepted for publication.

[29] Battiti R, Tecchiolli G. The reactive tabu search. ORSA Journal on Computing 1994;6:126–40.

[30] Vasquez M. Arc-consistency and tabu search for the frequency assignment problem with polarization. In: Proceedings of CPAIOR'02. Le Croisic, France, March 2002. p. 359–72.

[31] Johnson DS, Trick MA. Proceedings of the second DIMACS implementation challenge. DIMACS series in discrete mathematics and theoretical computer sciences, vol. 26. Providence, RI: American Mathematical Society; 1996.