A Survey of Local Search Methods for Graph Coloring

Philippe Galinier* Alain Hertz†

September 1, 2004

Abstract

Tabucol is a tabu search algorithm that tries to determine whether the vertices of a given graph can be colored with a fixed number k of colors such that no edge has both endpoints with the same color. This algorithm was proposed in 1987, one year after Fred Glover's article that launched tabu search. While more performing local search algorithms have now been proposed, *Tabucol* remains very popular and is often chosen as a subroutine in hybrid algorithms that combine a local search with a population based method. In order to explain this unfailing success, we make a thorough survey of local search techniques for graph coloring problems, and we point out the main differences between all these techniques.

1 Introduction

The graph coloring problem is a famous difficult combinatorial optimization problem. Examples of applications include time tabling and scheduling [30, 35], radio frequency assignment [19], computer register allocation [5, 8], and printed circuit board testing [20]. While exact algorithms can solve instances with up to 100 vertices, heuristic methods are needed for larger instances. Most recent graph coloring heuristics are either local search methods or *hybrid algorithms* that combine a local search with a population based algorithm. The objective of this paper is to present a survey of metaheuristics proposed for graph coloring. In particular we analyze in detail local search metaheuristics.

*CRT-Ecole Polytechnique, 3000 chemin de la Cote Sainte-Catherine, Montreal (QC) H3T 2A7, Canada, philippe.galinier@polymtl.ca

†GERAD-Ecole Polytechnique, 3000 chemin de la Cote Sainte-Catherine, Montreal (QC) H3T 2A7, Canada, alain.hertz@gerad.ca

One of the first local search methods that has been proposed for solving graph coloring problems is *Tabucol* [24], a tabu search algorithm that was developed in 1986 and published in 1987, one year after Fred Glover's paper [22] that launched Tabu Search. *Tabucol* was originally developed by Hertz and de Werra, and later improved by several researchers (see for example [14, 17]). While *Tabucol* is almost 20 years old, it is now frequently used as local search operator in hybrid algorithms. This is why *Tabucol* is presented and analyzed in detail in the paper. The strategy of *Tabucol* involves accepting solutions with conflicting edges (edges with both endpoints colored with the same color) while using penalties. However, other totally different alternative local search approaches have also been proposed. In the following, we identify four main strategies, making it possible to classify local search heuristics into four classes. Finally, it was observed that some graphs, especially large random graphs, can not be colored efficiently by using pure local search algorithms, and several approaches have therefore been proposed to deal with these difficult instances. The most recent and also most efficient approach is based upon hybrid algorithms that use a particular kind of recombination operator. Another efficient strategy for coloring large graphs is to first extract several stable sets and then color the residual graph.

In the next section we give some definitions and notations while Section 3 is devoted to a historical review of local search methods for graph coloring problems. Section 4 contains a precise description of *Tabucol*. In Section 5, we propose a classification of the existing local search graph coloring algorithms. Section 6 contains a short review of efficient approaches for coloring large graphs. Some computational experiments are reported in Section 7 to compare the performance of each algorithm.

2 Definitions and Notations

Given a graph $G = (V, E)$ with vertex set V and edge set E, and given an integer k, a k-coloring of G is a function $c : V \longrightarrow \{1, \ldots, k\}$. The value $c(x)$ of a vertex x is called the color of x. The vertices with color i ($1 \leq i \leq k$) define a *color class*, denoted V_i. If two adjacent vertices x and y have the same color i, vertices x and y, the edge $[x, y]$ and color i are said *conflicting*. A k-coloring without conflicting edges is said *legal*. A *stable set* is a subset of vertices no two of which are adjacent. Hence, a k-coloring is legal if and only if its color classes are stable sets. The Graph Coloring Problem (GCP for short) is to determine the smallest integer k, called *chromatic number* of G and denoted $\chi(G)$, such that there exists a legal k-coloring of G.

Given a fixed integer k, the problem of determining whether there exists a legal k-coloring of G is called k-GCP. A local search algorithm for the GCP can be used to solve the k-GCP by simply stopping the search as soon as a legal k-coloring is

met. Also, an algorithm that solves the k-GCP can be used to solve the GCP. One can for example use the following scheme.

1. Use a greedy algorithm for finding a legal coloring c.
2. Set k^* equal to the number of colors used in c and $k := k^* - 1$.
3. Solve the k-GCP; if a legal k-coloring is found then go to step 2, else return k^*.

The number of times the algorithm goes through Step 3 depends on the quality of the coloring found in Step 1. At first sight, one could therefore think that the above technique may waste a lot of time in finding legal k-colorings with non minimal values of k. However, finding a legal coloring with $k+1$ colors is generally much easier than one with k colors. Hence, the first iterations take typically much less time than the subsequent ones, and the majority of time is used in finding a legal k^*-coloring and in trying, without success, to find a (k^*-1)-coloring.

It is also possible to use a different scheme where the k-GCP is solved with increasing values of k. In this case, one should start with a lower bound on $\chi(G)$. Notice however that if $k \geq \chi(G)$, then a heuristic used to solve the k-GCP can be stopped as soon as a legal k-coloring is encountered, while there is no such simple stopping criterion when $k < \chi(G)$. Hence, we do not recommend the approach where k increases since most of the time will be spent in solving k-GCPs with $k < \chi(G)$.

3 A historical review

The first local search (LS for short) algorithm for coloring graphs was proposed by Chams et al. [4] in 1987. The problem addressed in the paper is the k-GCP. The proposed algorithm explores the set of (not necessary legal) k-colorings, the objective being to minimize the number of conflicting edges. A move consists in changing the color of a single vertex in the current solution and the algorithm uses the *Simulated Annealing* (SA for short) metaheuristic. Another major contribution of the paper was to propose a second algorithm that performs two successive phases. During the first phase (preprocessing), several color classes are built by extracting stable sets in the graph using a greedy algorithm. Then the SA algorithm is applied during the second phase. The paper shows that the stable sets extraction preprocessing makes it possible to get better results when applied to large graphs. Two among the three authors of the paper, Hertz and de Werra, experimented a few months later a new LS metaheuristic for coloring graphs: the tabu search metaheuristic, just proposed by Fred Glover in [22]. While their LS strategy is the same as the one proposed in [4] (they use the same solution space, neighborhood and objective function), the results are better than those obtained with SA. The authors have also tested the stable sets extraction preprocessing. But instead of using a greedy algorithm for the

extraction of stable sets, they propose to use a simplified version of *Stabulus*, a tabu search algorithm for the maximum stable set problem [15].

In 1991, Johnson et al. compare SA algorithms that use three different strategies [27]. In the first strategy, solutions are (not necessarily legal) colorings, with a non-fixed number of color classes. The objective is to minimize both the number of colors and the number of conflicting edges. The second tested strategy was originally proposed by Morgenstern and Shapiro [32]: it considers as solutions all legal colorings, and the number of colors is not fixed. The objective is now to minimize the number of color classes in a legal coloring. As explained in section 5.1, this strategy requires using much more complicated types of moves, namely Kempe chain interchanges. The third strategy is the penalty strategy used by Chams et al. in [4]. Intensive tests on random graphs did not allow Johnson et al. to identify a best strategy. In addition to the three SA algorithms, Johnson et al. have designed a procedure, called XRLF, for extracting stable sets. Their procedure combines an exhaustive search with a variant of Leighton's RLF algorithm [30]. Moreover, they use the extraction process until the residual graph has less than 70 vertices, making it possible to color optimally the residual graph with an exact coloring algorithm. XRLF is shown to be competitive with the three SA algorithms, and it performs even better on some large graphs.

In the early 90s, Davis implemented a genetic algorithm (GA for short) for the GCP. The GA algorithm uses an order-based encoding [10]. More precisely, a solution is encoded as a permutation of the vertices, and in order to evaluate a solution, a greedy algorithm is applied which sequentially colors the vertices, giving each vertex the first color still available. This algorithm delivers results of poor quality. Experiments performed with this GA are also presented in [14].

Costa et al [9] (in 1995) and Fleurent and Ferland [14] (in 1996) were the first to experiment a genetic local search (GLS for short) algorithm for coloring graphs. The GLS uses a population of solutions and a crossover operator, very much like a GA algorithm; but the random mutation operator of the GA is replaced here by an LS operator. In [9], the LS operator is a simple descent method. Costa et al. report results that are better than those obtained with *Tabucol*. In [14], the LS operator is *Tabucol* and the crossover operator resembles the uniform crossover proposed in standard GAs. The authors report that the GLS slightly improves the results of *Tabucol* on some graphs, a fact they attribute to the diversification induced by the use of crossovers. They also report results obtained on large random graphs by first using a stable set extraction preprocessing, and then running *Tabucol* for coloring the residual graph. Although their algorithm is not very different from the one used by Hertz and de Werra in [24], the results are quite better, mainly for two reasons. The first reason is that they used an improved version of *Tabucol*. The second reason is that the tabu search algorithm *Stabulus* used to extract stable sets does not have the same objective as in [24]: instead of maximizing the number of vertices in the

extracted stable sets, Fleurent and Ferland propose to minimize the number of edges in the residual graph.

The Second DIMACS Challenge took place in 1993. The purpose of the Challenge was to encourage high quality research on empirical issues in combinatorial optimization. Three problem classes were chosen: finding cliques in graphs, coloring graphs, and solving the satisfiability problem (SAT). The results of the challenge are published in [28]. In order to solve the k-coloring problem, Morgenstern explores a new LS strategy in which solutions are the legal partial k-colorings and the goal is to maximize the number of colored vertices; more precisely, the objective is to minimize the sum of the degrees of the uncolored vertices. A move, called i-swap, consists in assigning a color (say i) to an uncolored vertex (say v), and to remove the color on each neighbor of vertex v that has color i. This type of move is combined with another neighborhood, called s-chain, which is a generalization of Kempe chain interchanges. As local search method, Morgenstern uses a simplified version of SA in which the temperature remains constant. Several variants of the algorithm are tested and the best one uses a population which is initialized by using the XRLF algorithm. This quite complicated technique made it possible to improve the best known results on some large graphs.

A new type of GLS algorithms was proposed by Dorne and Hao in [12], and Galinier and Hao in [17]. Both algorithms use a simple (and quite rustic) way to manage the population while the LS operator is *Tabucol*. They obtained remarkable results on large graphs, better than those produced by the algorithms mentioned above. Unlike all other efficient coloring algorithms for large graphs [9, 14, 33], the GLS algorithms in [12, 17] do not use a stable sets extraction preprocessing. The reason of the efficiency or these new hybrid algorithms is due to the power of the crossover operators. Galinier and Hao explain the major difference between their new specialized crossover and the ones proposed so far: their crossover operator recombines color classes instead of color assingments (see more details in section 6.2). Glass and Pruegel-Bennett [21] have recently analyzed the effect of replacing *Tabucol* by a steepest descent method in Galinier and Hao's algorithm while using a much larger population. Their experiments show that solutions with the same quality can still be obtained, but the price to pay is a large increase in the computational effort.

In 1997, Mladenovic and Hansen [31] have proposed a new optimization technique called *variable neighborhood search* (VNS for short) in which a combination of several neighborhoods is used during the search. Avanthay et al. [1] have designed twelve different large neighborhoods. They use *Tabucol* in combination with jumps in the search space performed by means of these neighborhoods. Experiments reported in [1] clearly show that the method is more efficient than simply using *Tabucol*.

The *adaptive memory algorithm* is another recent metaheuristic proposed by Rochat and Taillard [34]. Unlike the GA (and GLS) algorithm that uses a population of solutions and performs crossovers, the adaptive memory algorithm stores

portions of solutions in a central memory, and recombines them in order to build new solutions. In addition, the algorithm uses an LS operator, as it is the case in the GLS algorithm. Galinier et al. [18] have recently proposed an adaptive memory algorithm, called *Amacol* for coloring graphs. The recombination operator builds a new solution by combining color classes stored in the central memory, and the LS operator used in the algorithm is *Tabucol*. The results obtained by *Amacol* are comparable to those obtained by the GLS proposed in [17].

We are aware of the following recent working papers. Bloechliger and Zufferey [2] have designed two *reactive* tabu search algorithm for the k-GCP [2]. The first one is a variation on *Tabucol* where the tabu list size reacts to the oscillations of the objective function. The second one uses the same solution space, neighborhood and objective function as in [33], but a reactive tabu search in used instead of the simplified SA algorithm used by Morgenstern. Chiarandini and Stuetzle [6] are currently working on an algorithm called *iterated local search* which is a simplified version of the above VNS algorithm. With Dumitrescu [7], they are studying a neighborhood called *cyclic exchange neighborhood* which can be seen as a variation on the *chain neighborhood* studied in [1]. They are currently experimenting the use of this new neighborhood within a tabu search or an iterated local search.

4 The *Tabucol* algorithm

As mentioned in Section 3, *Tabucol* was introduced in 1987 by Hertz and de Werra [24]. Since that time, many authors used *Tabucol* as a subroutine (see section 3). Although these authors preserved the main characteristics of the initial *Tabucol*, they also introduced some modifications that made the algorithm more efficient. We present here below an improved version of *Tabucol* [17]. We then describe and discuss the main features of *Tabucol*, including the differences between the different versions.

Tabucol is a tabu search algorithm for the k-GCP. It first generates an initial random k-coloring, which contains typically a large number of conflicting edges. Then, the heuristic iteratively modifies the color of a single vertex, the objective being to decrease progressively the number of conflicting edges until a legal k-coloring is obtained. A tabu list is used in order to escape from local optima and to avoid short term cycling. A more precise description is now presented.

The search space S explored by *Tabucol* is the set of k-colorings of a given graph G. A solution $c \in S$ is therefore a partition of the vertex set into k subsets $V_1, \ldots, V_k$. The evaluation function f measures the number of conflicting edges. Hence, for a solution $c = (V_1, \ldots, V_k)$ in S, $f(c)$ is equal to $\sum_{i=1}^{k} |E_i|$, where E_i denotes the set of edges with both endpoints in V_i. The goal of *Tabucol* is to determine a k-coloring c such that $f(c) = 0$. An elementary transformation, called *1-move*, consists in

changing the color of a single vertex. For a vertex v and a color $i \neq c(v)$, we denote (v, i) the 1-move that assigns color i to v, and the solution resulting from this 1-move is denoted $c+(v, i)$. Hence, the k-coloring $c' = c+(v, i)$ can be described as follows:

- $c'(v) = i$
- $c'(w) = c(w)$ for all $w \in V - \{v\}$.

The neighborhood $N(c)$ of a solution $c \in S$ is defined as the set of k-colorings that can be reached from c by applying a single 1-move. Hence, $N(c)$ contains $|V|(k-1)$ solutions. The *performance* of a 1-move (v, i) on a solution c can be measured by $\delta(v, i) = f(c+(v, i)) - f(c)$. A strictly negative (respectively positive) value of $\delta(v, i)$ indicates that the 1-move (v, i) leads to a decrease (an increase) of the cost function.

Another important feature of *Tabucol* involves the notion of *conflicting vertices* (i.e. vertices involved in a conflicting edge). We denote $F(c)$ the number of conflicting vertices in c. A 1-move (v, i) that involves a conflicting vertex v is said *critical*. For efficiency reasons, *Tabucol* performs only critical 1-moves (see Section 4.2).

The tabu list in *Tabucol* stores moves. When a 1-move (v, i) is performed on a solution c, the tabu list stores the pair $(v, c(v))$, which means that it is forbidden to reassign color $c(v)$ to v for some number of iterations. The duration of the tabu status of $(v, c(v))$ depends on the number of conflicting vertices in c and on two parameters L and λ. More precisely, when the 1-move (v, i) is applied on c, then $(v, c(v))$ becomes a tabu 1-move for $L + \lambda F(c)$ iterations. A 1-move (v, i) is defined as *candidate* if it is both critical and not tabu, or if $f(c+(v, i)) = 0$ (i.e. if $\delta(v, i) = -f(c)$). The last condition is a very elementary aspiration criterion which avoids missing a legal k-coloring.

At each iteration, *Tabucol* performs the best candidate 1-move (ties are broken randomly). The algorithm stops as soon as $f(c) = 0$ and it returns c, which is a legal k-coloring. In the pseudo-code given below, there is a second stopping criterion which is based on a total number of iterations. Other stopping conditions can naturally also be used, such as a limit in computing time, or a maximum number of iterations without improving the best found solution, etc.

Algorithm Tabucol
Input: A graph $G = (V, E)$, an integer $k > 0$.
Parameters: $MaxIter$, L and λ.
Output: Solution c^*.

Build a random solution c;
Set $c^* := c$ and $iter = 0$;
Set the tabu list to the empty set; /* No move is tabu */

Repeat until $f(c) = 0$ **or** $iter = MaxIter$ /* stopping criterion */

Set $iter := iter + 1$;
Choose a candidate 1-move (v, i) with minimum value $\delta(v, i)$;
Introduce move $(v, c(v))$ into the tabu list for $L + \lambda F(c)$ iterations;
Set $c := c + (v, i)$; if $f(c) < f(c^*)$ then set $c^* := c$;

In the next subsections, we discuss thoroughly the following different features of *Tabucol*. We give the rationale of each of these features, and comment on their importance.

1) The initial solution is built randomly.
2) Only critical moves are considered.
3) All legal moves are considered - as opposed to using only a random sample.
4) An elementary aspiration criterion is used.
5) The size of the tabu list increases with the number of conflicting vertices and depends on two parameters L and λ.
6) Efficient data-structures can reduce dramatically the computational effort.
7) *Tabucol* is a basic tabu search, without long term strategies such as intensification or diversification.

4.1 Initial solution

In the above *Tabucol* algorithm, the initial solution is built randomly. This is the simplest possible option. Another alternative is to build a k-coloring by using a greedy heuristic. However, there is no tangible advantage to use this option for most of the graphs.

4.2 Critical 1-moves

Tabucol only considers 1-moves (v, i) where v is a conflicting vertex. This makes it easier to guide the search towards good regions of the search space. Indeed, we have observed that by using 1-moves involving non conflicting vertices, it is much more difficult to escape from a local minimum. The reason is that there are typically some non conflicting vertices v than can receive a new color i without creating any conflicting edge, in which case $\delta(v, i) = 0$. By imposing that v must be conflicting in a 1-move (v, i), one forces the algorithm to deal with those vertices that are responsible for the existence of conflicts.

Notice also that the objective function decreases dramatically in the early stages of the search and the heuristic generally uses most of its time trying to eliminate the last conflicting edges. Therefore, except at the very beginning of the search, there is a limited number of conflicting vertices. Hence, a possible advantage of this

policy is to make each iteration faster. How much faster depends on the implementation. We will notice in Section 4.6 that if appropriate data-structures are used, the computational time of an iteration is not proportional to the number of critical 1-moves, and the benefit in this case may not be really significant.

4.3 Complete neighborhood

In the above *Tabucol* algorithm, the 1-move performed at each iteration is the best one among all critical 1-moves (ties are broken randomly). This was not the case in the initial version of *Tabucol* [24], where the best move was chosen among a random sample of critical 1-moves. The size of the sample was typically a small percentage of the total number of possible critical 1-moves. Experiments have shown that the use of a too small sample may compromise the efficiency of the heuristic [24].

4.4 Aspiration criterion

The aspiration criterion used in the above *Tabucol* algorithm is very elementary: the tabu status of a 1-move (v, i) is cancelled if it leads to a legal coloring, in which case the search can be stopped. A more complex aspiration criterion was proposed in [24]. However, it is very rarely used and its removal does not seem to have a negative impact on the algorithm.

4.5 Tabu tenure

The number of critical 1-moves is proportional to the number of conflicting vertices and may therefore considerably vary during the search process. The tabu tenure therefore varies dynamically. In our experiments, we observed clearly that a constant tabu tenure was generally inappropriate - only small portions of the search space are explored with a too small tabu tenure while too large values appear to be too restrictive. Notice however that a constant tabu tenure equal to 7 was used in [24] (as proposed in some early work of Fred Glover), in combination with a small random sample of critical 1-moves (to be opposed to a complete neighborhood). A constant tabu tenure may be a correct option in this case.

Another important issue is how to determine the value of parameters L and λ. Galinier and Hao suggest to choose L randomly in [0,9] and to set λ=0.6. These values seem to be reasonably good, at least for the graphs they tested in their experiments [17]. However, these values may be inappropriate for other graphs and these two parameters should ideally be set instance by instance.

4.6 Data structures

Most of the computing time in *Tabucol* is spent at each iteration in choosing the best possible critical 1-move. A naive technique for doing this would consist in evaluating the performance $\delta(v,i)$ of each critical 1-move and choosing the best one. Notice that for a k-coloring $c = (V_1, \ldots, V_k)$, $\delta(v,i)$ is equal to the number of vertices adjacent to v in V_i minus the number of vertices adjacent to v in $V_{c(v)}$. This value can be computed in $O(|\Gamma(v)|)$, where $\Gamma(v)$ denotes the set of vertices adjacent to v. Indeed, one can initially set $\delta(v,i) = 0$ and then consider each neighbor $w \in \Gamma(v)$: $\delta(v,i)$ is increased by one unit if $c(w) = i$, and decreased by one unit if $c(w) = c(v)$. Since there are $O(kF(c))$ critical 1-moves, the total time needed to perform an iteration is in $O(kF(c)|V|)$. But there is another more efficient implementation that makes it possible to reduce the complexity of this computation. Let $\gamma(v,i)$ denote the number of vertices that are adjacent to v and have color i in the current solution c, and assume that these values are stored in a $|V|$ x k matrix which we call γ-matrix. The γ-matrix can be initialized at the beginning of the search process, and then updated at each iteration when a 1-move is performed on c to modify the current solution. The performance of a 1-move (v,i) can now be be computed in constant time since $\delta(v,i) = \gamma(v,i) - \gamma(v,c(v))$. Notice also that one can determine in constant time if a vertex v is conflicting since this is the case if and only if $\gamma(v,c(v)) > 0$. Finding the best critical 1-move can be performed in $O(kF(c))$ by scanning the γ-matrix. When moving from a solution c to $c + (v,i)$, one can update the γ matrix in $O(|\Gamma(v)|)$. Indeed, for each vertex $w \in \Gamma(v)$, it is sufficient to increase $\gamma(w,i)$ by one unit and to decrease $\gamma(w,c(v))$ by one unit. Hence, one iteration takes $O(min\{kF(c), |\Gamma(v)|\})$. Notice that the complexity of finding the best critical 1-move can further be improved by storing the γ-matrix in a heap structure.

The tabu list can be stored in a $|V|$ x k matrix T, where $T(v,i)$ represents the index of the iteration at which the 1-move (v,i) will cease to be tabu (i.e., (v,i) is tabu if and only if $T(v,i) \geq iter$). Hence, with this data structure, it is possible to know in constant time whether a 1-move is tabu. Moreover, when a 1-move (v,i) is performed on c, the update of T takes a constant time since it is sufficient to set $T(v,i)$ equal to $iter + L + \lambda F(c)$.

4.7 Intensification and diversification

The role of the tabu list in a tabu search algorithm is to forbid some moves that would annihilate the effect of the moves performed recently. A tabu list may however forbid some interesting moves, and this is why an aspiration criterion makes it possible to suspend the tabu status of a move. While the tabu list and the aspiration criterion are basic features of a tabu search, Glover proposed to use some additional techniques

Search space	k not fixed		k fixed	
	legal colorings	colorings	k-colorings	partial legal k-colorings
Objective function	$-\sum_{i=1}^{k} \lvert V_i \rvert^2$	$\sum_{i=1}^{k} \lvert V_i \rvert (2\lvert E_i \rvert - \lvert V_i \rvert)$	$\sum_{i=1}^{k} \lvert E_i \rvert$	$\sum_{v \in V_{k+1}} d(v)$
Neighborhood structure	Kempe chain interchanges	1-moves	critical 1-moves	i-swaps Kempe chain interchanges
Authors	Morgenstern and Shapiro (1988) Johnson et al. (1991)	Johnson et al. (1991)	Chams et al. (1987) Hertz and de Werra (1987) Johnson et al. (1991)	Morgenstern (1996) Bloechliger and Zufferey (2003)
Search strategy	legal	penalty	k-fixed penalty	k-fixed partial legal

Figure 1: LS strategies.

such as intensification and diversification in order to guide the search in the long term [23]. We notice that *Tabucol* does not use these additional techniques.

5 Local Search Strategies for Graph Coloring

When designing an LS algorithm for solving a particular problem, one has to define the search space to be explored, the evaluation function to be minimized, and the neighborhood function. This triplet is what we call a *search strategy*. We propose to classify the search strategies for graph coloring into four categories. Two of them solve the GCP, while the two others consider the number k of colors as fixed, and therefore deal with the k-GCP. The strategies for the k-GCP are used within the global approach described in Section 2. We define each strategy by its search space and the goal of the search. The classification is summarized in Table 1.

- The *legal strategy*: the search space S contains all legal colorings and the goal is to find a solution $c \in S$ that uses as few colors as possible.

- The *k-fixed partial legal strategy*: the number k of colors is fixed, the search space S contains all partial legal k-colorings, and the goal is to determine a solution $c \in S$ in which all vertices are colored.

- The *k-fixed penalty strategy*: the number k of colors is fixed, the search space S contains all (not necessarily legal) k-colorings, and the goal is to determine a legal k-coloring $c \in S$.

- The *penalty strategy*: the search space S contains all (not necessarily legal) colorings and the goal is to determine a legal coloring $c \in S$ that uses as few colors as possible.

5.1 The legal strategy

The search space S in a legal strategy contains all legal colorings. One could therefore think about generating neighbors of a solution c by moving a vertex either to another existing color class or to a new one. Since all neighbor solutions need to belong to S, they must all be legal. There are therefore typically very few color classes where a vertex can move. This makes this kind of moves apparently useless. A more efficient neighborhood structure for this search space, called *Kempe chain interchange*, was proposed in [32]. Let V_i and V_j be two color classes in a k-coloring c. A Kempe chain for V_i and V_j is a connected component in the subgraph of G induced by $V_i \cup V_j$. Consider a triplet (v, i, j) where V_i and V_j are two disjoint color classes of the current solution c, v is a vertex in V_i, and the subgraph induced by $V_i \cup V_j$ is not connected. A Kempe chain interchange for such a triplet (v, i, j) consists in replacing V_i and V_j by $(V_i - K) \cup (K - V_i)$ and $(V_j - K) \cup (K - V_j)$, where K is the Kempe chain for V_i and V_j that contains v. If the subgraph induced by $V_i \cup V_j$ is connected, then this operation is not performed since it simply swaps two color classes, and is therefore meaningless.

The simplest evaluation function to be minimized is the number of colors used in a solution. However, it is very rare that a move makes it possible to remove a color, and using this function can therefore not really guide the search (the landscape is too flat). A viable option proposed in [32] is to minimize $f(c) = -\sum_{i=1}^{k} |V_i||^2$. This function encourages reducing the size of the smallest color classes, making a class decrease progressively and eventually be removed.

5.2 The k-fixed partial legal strategy

In the k-fixed partial legal strategy, the number k of colors is fixed and the search space S contains all partial legal k colorings. A solution $c \in S$ can be represented as a partition $(V_1, \ldots, V_k, V_{k+1})$ of the vertex set where $V_1, \ldots, V_k$ are k stable sets (i.e. legal color classes) and V_{k+1} is the set of non-colored vertices.

The Kempe chain interchanges described above are still valid here. However, it becomes possible to use a new kind of much simpler moves, namely *i-swaps* [33]. An i-swap consists in moving a vertex v from V_{k+1} to a color class V_i, and by moving to V_{k+1} each vertex in V_i that is adjacent to v. Much simpler kinds of moves can also be used such as moving a vertex v from V_{k+1} to a color class V_i that does not contain any vertex adjacent to v, or removing the color of a colored vertex (i.e., moving a vertex v from a color class V_i to V_{k+1}). However, these kinds of moves are clearly

less attractive than i-swaps.

A possible objective function is the number of uncolored vertices (i.e., $|V_{k+1}|$) [2]. An alternative is to minimize the function $\sum_{v \in V_{k+1}} d(v)$, where $d(v)$ denotes the number of vertices adjacent to v [33].

5.3 The k-fixed penalty strategy

In the k-fixed penalty strategy [4, 27, 24], the number k of colors is fixed and the search space contains all (not necessarily legal) k-colorings. A solution can be represented as a partition $c = (V_1, \ldots, V_k)$ of the vertex set into k subsets. The objective can simply be to minimize the number of conflicting edges, that is $f(c) = \sum_{i=1}^{k} |E_i|$, where E_i is the set of edges with both endpoints in V_i.

Neighbor solutions can be obtained by changing the color of a vertex (i.e. by using 1-moves). It is also possible to use only critical 1-moves, thus forcing the algorithm to deal with the conflicting vertices of a non-legal k-coloring c.

5.4 The penalty strategy

In the penaly strategy, the search space S contains all (not necessarily legal) colorings. Here again, it is possible to use 1-moves. Notice however that a 1-move may consist here in coloring a vertex with a color not yet used in the solution.

The role of the objective function is both to decrease the number of color classes and the number of conflicting edges. This can be done by minimizing the function $f(c) = \sum_{i=1}^{k} 2|V_i||E_i| - \sum_{i=1}^{k} |V_i|^2$, where E_i is the set of edges with both endpoints in V_i. Notice that the second term of this function tends to reduce the number of color classes by reducing the size of the smallest color classes. It is proved in [27] that all local minima of this function correspond to legal colorings.

5.5 Comparisons

The reader now probably wonders which of the presented strategy is most efficient. Such a comparison was performed by Johnson et al. [27] by using SA with the strategies presented in sections 5.1, 5.3 and 5.4. Considering several different types of graphs, the authors concluded that none of the three strategies clearly dominates the two others. However, the legal strategy performs better on large random graphs. We implemented the k-fixed partial legal strategy (see section 5.2) with a simple tabu search strategy. We observed that the best colorings obtained by using this method are generally similar to the ones obtained with *Tabucol*. This remark is consistent with the results presented in [2]. An advantage of permitting non-legal colorings is that it can easily support the adjunction of new constraints. The penalty strategy

has for example been generalized to bandwidth coloring problems, and more broadly to constraint satisfaction problems [16].

5.6 LS coloring algorithms that use more than one neighborhood

While standard LS metaheuristics (such as SA or tabu search) use a single neighborhood, a promising alternative is to use several neighborhoods. For example, the *iterated local search* consists in alternating a steepest descent that uses a simple neighborhood with jumps performed by means of more complex neighborhoods. Mladenovic and Hansen [31] have proposed recently the *variable neighborhood search* (VNS for short) that is based on this idea. Given a set of neighborhoods and an incumbent c, a jump to another solution c' is obtained by means of one of the neighborhoods, and an LS method (that uses a simple neighborhood) is then applied on c' in order to get a local optimum c". If c" is better than c, then c" becomes the new incumbent; otherwise, a different neighborhood is considered in order to try to improve upon solution c. This process is repeated until no neighborhood leads to an improvement of the incumbent.

Several neighborhoods (in addition to those described in the previous sections) have been proposed and investigated for graph coloring. One of them, called *s-chain*, is a generalization of Kempe chain interchanges to more than two classes. This kind of moves transforms a legal coloring into another legal coloring, as Kempe change interchanges. Therefore, it is applicable to the four strategies presented above. It is defined as follows. Let $(v, i_0, \ldots, i_{s-1})$ be an ordered $(s+1)$-tuple such that $1 \leq i_r \leq k$ for all $r = 0, \ldots, s-1$, $i_r \neq i_s$ if $r \neq s$, and v is a vertex in V_{i_0}. Consider the directed graph H with vertex set $W = V_{i_0} \cup \ldots \cup V_{i_{s-1}}$ and in which there is an arc from a vertex u to a vertex w if and only if u and w are adjacent in G and there exists an index $r \in \{1, \ldots, s\}$ such that $u \in V_{i_r}$ and $w \in V_{i_{(r+1) \bmod s}}$. Let $R(v)$ denote the set of vertices $w \in W$ such that there exists a directed path from v to w in H. An s-chain interchange for an $(s+1)$-tuple $(v, i_0, \ldots, i_{s-1})$ consists in moving each vertex in $R(v) \cap V_{i_r}$ to $V_{i_{(r+1) \bmod s}}$. If $R(v) = W$ then such an s-chain interchange is a relabeling of color classes $V_{i_0}, \ldots, V_{i_{s-1}}$, and is therefore meaningless. For $s = 2$, an s-chain interchange is equivalent to a Kempe chain interchange. Morgenstern used i-swaps as a basic neighborhood and s-chain interchanges for making jumps in the search space [33] while using a simplified SA. He obtained many of the best results on several DIMACS instances [11].

Avanthay et al. [1] have designed twelve neighborhoods that they use within a VNS algorithm. They use *Tabucol* as LS method.

6 Efficient metaheuristics for coloring large graphs

The algorithms described in the previous sections can be considered as pure LS methods. They generate a series $s_0, s_1, \ldots$ of solutions such that each s_{i+1} is obtained from s_i by using one or several neighborhoods. Although these pure LS algorithms produce remarkable results on many medium-sized graphs, the most efficient heuristics for coloring large graphs combine a pure LS method with additional features. Two efficient approaches have been proposed so far for coloring large graphs. The first one was proposed as early as 1987 in [4] and it is now routinely used for coloring large graphs [4, 24, 27, 9, 14, 33]. The idea is to use a preprocessing algorithm that extracts stable sets until the residual graph is small enough to be handled by a pure LS method. The second approach is to use a population based algorithm that combines a pure LS method with a recombination operator.

6.1 Extraction of stable sets

Consider the following two-phases algorithm:

Input : a graph $G = (V, E)$ and an integer q.
Output : a coloring c.

Phase 1
$i := 0$;
While $|V| > q$ do
  Set $i := i + 1$;
  Find a large stable set V_i in G;
  Remove all vertices of V_i from G;

Phase 2
Use an LS method to color the residual graph G;
Let $(W_1, \ldots, W_p)$ be the coloring returned by the LS method;
Set $c = (V_1, \ldots, V_i, W_1, \ldots, W_p)$;

Finding a large stable set V_i in Phase 1 can be done by different means. A simple greedy technique was used in [4]. However, it is more efficient to use an LS algorithm dedicated to the maximum stable set problem [24, 14]. In XRLF, each stable set is build by using a technique that combines a randomized greedy search with retrials and an exhaustive search [27]. Costa et al. [9] have proposed a slightly different idea which is to build several stable sets at the same time. More precisely, a number r of stable sets are removed by means of a GLS that produces a partial legal r-coloring with as many colored vertices as possible.

Phase 1 stops when the residual graph has at most q vertices, where q is chosen so that the residual graph is small enough to be colored efficiently by the LS algorithm. A typical value for q is 300 or 500. However, in XRLF, q is set equal to 70, making it possible to color the residual graph with an exact method.

6.2 Hybrid evolutionary algorithms

An LS algorithm is generally very efficient in discovering good solutions in a particular region of the search space. The idea of a hybrid algorithm is to combine the power of an LS algorithm with the use of a central memory in order to guide the search on long term. The most common hybrid algorithms are genetic local search algorithms (GLS for short). A GLS algorithm uses a population of solutions and a crossover operator, and it is very similar to a genetic algorithm (GA for short) except that mutation is replaced by an LS operator - the LS operator can be a steepest descent or a more sophisticated LS algorithm such as tabu search.

The performance of a GLS algorithm depends on the way the central memory is managed, on the LS operator, and most importantly on the crossover operator. There are two major efficient families of crossover operators for coloring graphs: *assignment-based* and *partition-based* crossovers [17].

An assignment-based crossover operator builds an offspring by assigning to each vertex the color it has in one of the two parents. For example, it is possible to imitate the uniform crossover used in the standard GA as follows. Suppose a solution c is a (not necessarily legal) k-coloring represented by the vector $c = (c(v_1), \cdots, c(v_n))$ of size $n = |V|$, each $c(v_i)$ corresponding to the color assigned to v_i in c. Given two parent solutions $c = (c(v_1), \cdots, c(v_n))$ and $c' = (c'(v_1), \cdots, c'(v_n))$, an offspring $o = (o(v_1), \cdots, o(v_n))$ is built by setting $o(v_i)$ equal to $c(v_i)$ or $c'(v_i)$, each value being chosen with probability 1/2.

Instead of transmitting color assignments, the principle of a partition-based crossover operator is to combine the color classes of two parents in order to build the color classes of the offspring. For example, Galinier and Hao's GLS algorithm uses the k-fixed penalty strategy combined with the following partition-based crossover operator. Consider two parent solutions c and c'. The operator first builds a partial legal k-coloring $(W_1, \ldots, W_k)$ such that about half of the color classes W_i are subsets of classes of c, and the other half are subsets of color classes of c'. In addition, the subsets W_i are built with the objective of maximizing $\sum_{i=1}^{k} |W_i|$. The partial coloring is then completed by inserting the remaining vertices at random into the k color classes. More precise information about this crossover can be found in [17]. The crossover operator proposed in [12] is also partition-based, although it is based upon a different principle.

Instead of storing solutions in the central memory, one can simply store color classes and try to combine them in order to create offspring solutions. This idea was

recently exploited within an adaptive memory algorithm [18].

7 Overall Comparison

In order to illustrate the performance of the algorithms presented in this paper, we indicate the number of colors of the best colorings found by each algorithm on two famous graphs : DSJC500.5 and DSJC1000.5. These two graphs are two special random graphs proposed by Johnson et al. [27]. They have been obtained by linking a pair (v, w) of vertices by an edge with probability p, independently for each pair. These two graphs have 500 and 1000 vertices, respectively, and they both belong to the set of DIMACS benchmark problems [11]. The best colorings found for these two graphs have been reported for all algorithms mentioned in this paper. Notice however that the results reported for references [4] and [24] do not correspond to these two graphs but rather to two similar random graphs.

All algorithms use a local search, except two of them, namely DSATUR [3] and XRLF [27]. For each algorithm, we recall the search strategy, the LS method that is used, and eventually other features of the method. We indicate the number of colors of the best coloring found by each algorithm. All results are taken from the papers which reference appears in the second column.

We first observe that the greedy algorithm is not competitive with the LS and hybrid methods. Among the pure LS algorithms, the most efficient one uses a legal strategy. However, it should be noticed that it is not the case for other classes of graphs, as shown in [27]. Making jumps in the search space by means of additional neighborhoods can be beneficial, but it is possible to get much better results (especially on large graphs) by using a stable sets extraction preprocessing or a hybrid algorithm doted with a partition-based crossover operator.

To conclude, we observe that almost all efficient heuristic algorithms for graph coloring use a local search, and many of them are based on a tabu search. In particular, *Tabucol* is very popular, the reason being probably that it is a very simple algorithm that is easy to implement. In addition, *Tabucol* offers a good compromise between solution quality and computational effort.

References

[1] C. Avanthay, A. Hertz and N. Zufferey, *A variable neighborhood search for graph coloring*, European Journal of Operational Research 151, 379-388, 2003.

[2] I. Bloechliger and N. Zufferey, *A reactive tabu search using partial solutions for the graph coloring problem*, Research Report, Institut de Mathématiques, Ecole Polytechnique Fédérale de Lausanne, 2003.

Technique		Reference	Search Strategy	LS method	Other features	$G_{500,0.5}$	$G_{1000,0.5}$
Greedy algorithm		Brélaz (1979)	-	-	DSATUR	67	113
Pure LS	One neighborhood	Chams et al. (1987)	*k*-fixed penalty	SA	-	54	98
		Hertz and de Werra (1987)	*k*-fixed penalty	Tabucol	-	51	93
		Johnson et al. (1991)	*k*-fixed penalty	SA	-	51	91
		Johnson et al. (1991)	legal	SA	-	49	87
		Johnson et al. (1991)	penalty	SA	-	50	91
		Galinier and Hao (1999)	*k*-fixed penalty	Tabucol	-	49	89
		Bloechliger and Zufferey (2003)	*k*-fixed partial legal	Tabucol	Reactive tabu search	49	89
	Several neighborhoods	Morgenstern(1996)	*k*-fixed partial legal	Simplified SA	*s*-chains	49	88
		Avanthay et al. (2003)	*k*-fixed penalty	Tabucol	VNS	49	90
LS methods with stable sets extraction preprocessing		Chams et al. (1987)	*k*-fixed penalty	SA	Greedy algorithm for extracting stable sets	54	91
		Hertz and de Werra (1987)	*k*-fixed penalty	Tabucol	Tabu search for extracting stable sets	50	87
		Johnson et al. (1991)	-	-	XRLF	50	86
		Fleurent and Ferland (1996)	*k*-fixed penalty	Tabucol	Tabu search for extracting stable sets	49	84
Population based algorithms	with stable sets extraction preprocessing	Costa et al. (1995)	*k*-fixed penalty	Steepest descent	GLS + extraction of *r*-stable sets	49	85
		Fleurent and Ferland (1996)	*k*-fixed penalty	Tabucol	GLS + tabu search for extracting stable sets	49	84
		Morgenstern(1996)	*k*-fixed partial legal	Simplified SA	*s*-chains + XRLF	48	84
	without stable sets extraction preprocessing	Dorne and Hao (1998)	*k*-fixed penalty	Tabucol	GLS with partition-based crossover	48	83
		Galinier and Hao (1999)	*k*-fixed penalty	Tabucol	GLS with partition-based crossover	48	83
		Glass and Pruegel-Bennett (2003)	*k*-fixed penalty	Steepest descent	GLS with partition-based crossover	48	83
		Galinier et al. (2003)	*k*-fixed penalty	Tabucol	Adaptive memory	48	84

Figure 2: Comparative results on two random graphs.

[3] D. Brélaz, *New methods to color the vertices of a graph*, Communications of the ACM 22, 251-256, 1979.

[4] M. Chams, A. Hertz and D. de Werra, *Some experiments with simulated annealing for coloring graphs*, European Journal of Operational Research 32, 260-266, 1987.

[5] G.J. Chaitin, M. Auslander, A.K. Chandra, J. Cocke, M.E. Hopkins and P. Markstein, *Register allocation via coloring*, Computer Languages 6, 47-57, 1981.

[6] M. Chiarandini and T. Stuetzle, *An application of iterated local search to graph coloring*, in Proceedings of the Computational Symposium on Graph Coloring and its Generalizations (Editors: D.S. Johnson, A. Mehrotra and M.A. Trick) Ithaca, New York, USA, 112-125, 2002

[7] M. Chiarandini, I. Dumitrescu and T. Stuetzle, *Local search for the graph colouring problem. A computational study*, Technical Report AIDA-03-01, FG Intellektik, TU Darmstadt, January 2003.

[8] F.C. Chow and J.L. Hennessy, *The priority-based coloring approach to register allocation*, ACM Transactions on Programming Languages and Systems 12, 501-536, 1990.

[9] D. Costa, A. Hertz and O. Dubuis, *Embedding of a sequential procedure within an evolutionary algorithm for coloring problem in graphs*, Journal of Heuristics 1, 105-128, 1995.

[10] L. Davis *Order-based genetic algorithms and the graph coloring problem*, in Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 72-90, 1991.

[11] DIMACS graph coloring instances, http:mat.gsia.cmu.edu/COLOR/instances.html.

[12] R. Dorne and J.-K Hao, *A new genetic local search algorithm for graph coloring*, Lecture Notes in Computer Science 1498, Springer Verlag, Berlin, Germany, 745-754, 1998.

[13] E. Falkenauer, *A Hybrid Grouping Genetic Algorithm for Bin Packing*, Journal of Heuristics 2, 5-30, 1996.

[14] C. Fleurent and J.A. Ferland *Genetic and hybrid algorithms for graph coloring*, Annals of Operations Research 63, 437-461, 1996.

[15] C. Friden, A. Hertz and D. de Werra, *Stabulus: a technique for finding stable sets in large graphs with tabu search*, Computing 42, 35-44, 1989.

[16] P. Galinier and J.-K. Hao, *Tabu search for maximal constraint satisfaction problems*, in Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP'97) (Editor: G. Smolka), LNCS 1330, Springer Verlag, Berlin, Germany, 196-208, 1997

[17] P. Galinier and J.-K. Hao, *Hybrid evolutionary algorithms for graph coloring*, Journal of combinatorial optimization 3, 379-397, 1999.

[18] P. Galinier, A. Hertz and N. Zufferey, *An adaptive memory algorithm for the k-colouring problem*, Les cahier du GERAD G-2003-35, GERAD, Montral, 2003.

[19] A. Gamst, *Some lower bounds for a class of frequency assignment problems*, IEEE Transactions of Vehicular Echnology 35, 8-14, 1986.

[20] M.R. Garey, D.S. Johnson and H.C. So, *An application of graph coloring to printed circuit testing*, IEEE Transactions on Circuits and Systems 23, 591-599, 1976.

[21] C. Glass and A. Prügel-Bennett, *Genetic algorithm for graph colouring: exploration of Galinier and Hao's algorithm* Journal of Combinatorial Optimization 7 , 229-236, 2003.

[22] F. Glover, *Future paths for Integer Programming and Links to Artificial Intelligence*, Computers and Operations Research 13/5, 533-549, 1986.

[23] F. Glover, *Tabu Search (Part I)*, ORSA Journal on Computing 1, 190-206, 1989.

[24] A. Hertz and D. de Werra, *Using Tabu Search Techniques for Graph Coloring*, Computing 39, 345-351, 1987.

[25] A. Hertz , *Application des métaheuristiques la coloration des sommets d'un graphe*, Chapter 1 in *Résolution de problémes de RO par les métaheuristiques* (Editors: M. Pirlot and J. Teghem), Hermes Science Publication, Paris, 21-48, 2003.

[26] A. Johri and D.W. Matula, *Probabilistic bounds and heuristic algorithms for coloring large random graphs*, Southern Methodist University, Dallas, Texas, 1982.

[27] D.S. Johnson, C.R. Aragon, L.A. McGeoch and C. Shevon, *Optimization by simulated annealing: an experimental evalutation; part II, graph coloring and number partitioning*, Operations Research 39, 378-406, 1991.

[28] D.S. Johnson and M.A. Trick, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge* DIMACS series in Discrete Mathematics and Theoretical Computer Science, vol 26, The American Mathematical Society, Providence, RI, 1996.

[29] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, *Optimization by simulated annealing*, Science 220, 671-680, 1983.

[30] F.T. Leighton, *A graph coloring algorithm for large scheduling problems*, Journal of Research of the National Bureau of Standards 84, 489-503, 1979.

[31] N. Mladenovic and P. Hansen, *Variable neighborhood search*, Computers and Operations Research 24, 1097-1100, 1997.

[32] C. Morgenstern and H. Shapiro, *Chromatic Number Approximation Using Simulated Annealing*, Unpublished manuscript, 1986.

[33] C. Morgenstern, *Distributed Coloration Neighborhood Search*, in [28], 335-357, 1996.

[34] Y. Rochat and E. Taillard, *Probabilistic diversification and intensification in local search for vehicle routing*, Journal of Heuristics 1, 147-167, 1995.

[35] D. de Werra, *An introduction to timetabling*, European Journal of Operational Research 19, 151-162, 1985.