Les Cahiers du GERAD ISSN: 0711–2440

An Adaptive Memory Algorithm for the k-Colouring Problem

P. Galinier, A. Hertz
N. Zufferey

G–2003–35

May 2003
Revised: April 2004

Les textes publiés dans la série des rapports de recherche HEC n'engagent que la responsabilité de leurs auteurs. La publication de ces rapports de recherche bénéficie d'une subvention du Fonds québécois de la recherche sur la nature et les technologies.

An Adaptive Memory Algorithm for the k-Colouring Problem

Philippe Galinier
Department of Computer Science
École Polytechnique de Montréal, Canada
Philippe.Galinier@polymtl.ca

Alain Hertz
GERAD and Department of Mathematics and Industrial Engineering
École Polytechnique de Montréal, Canada
Alain.Hertz@gerad.ca

Nicolas Zufferey
Department of Mathematics
Swiss Federal Institute of Technology, Lausanne, Switzerland
Nicolas.Zufferey@epfl.ch

May, 2003
Revised: April, 2004

Les Cahiers du GERAD

G–2003–35

Copyright © 2004 GERAD

Abstract

Let $G = (V, E)$ be a graph with vertex set V and edge set E. The k-colouring problem is to assign a colour (a number chosen in $\{1, \ldots, k\}$) to each vertex of G so that no edge has both endpoints with the same colour. The adaptive memory algorithm is a hybrid evolutionary heuristic that uses a central memory. At each iteration, the information contained in the central memory is used for producing an offspring solution which is then possibly improved using a local search algorithm. The so obtained solution is finally used to update the central memory. We describe in this paper an adaptive memory algorithm for the k-colouring problem. Computational experiments give evidence that this new algorithm is competitive with and simpler and more flexible than the best known graph colouring algorithms.

Keywords: hybrid evolutionary heuristics, adaptive memory algorithms, tabu search, graph colouring.

Résumé

Le problème de la k-coloration d'un graphe consiste à attribuer une couleur (un nombre compris entre 1 et k) à chaque sommet du graphe de telle sorte qu'aucune arête n'ait ses deux extrémités de la même couleur. L'algorithme à mémoire adaptative est une heuristique évolutive hybride qui utilise une mémoire centrale. À chaque itération, l'information contenue dans la mémoire est utilisée dans le but de générer une solution enfant que l'on tente ensuite d'améliorer à l'aide d'un algorithme de recherche locale. La solution résultant de ce processus est utilisée pour mettre à jour la mémoire centrale. Dans cet article nous décrivons un algorithme à mémoire adaptative pour le problème de la k-coloration. Des résultats expérimentaux démontrent que ce nouvel algorithme est compétitif, plus simple et plus flexible que les meilleurs algorithmes de coloration connus à ce jour.

1 Introduction

Evolutionary heuristics encompass various algorithms such as genetic algorithms, scatter search, ant systems and adaptive memory algorithms [3, 12, 13]. They can be defined as iterative procedures that use a central memory where information is collected during the search process. Each iteration, called *generation*, is made of two complementary phases which modify the central memory. In the cooperation phase, a *recombination operator* is used to create new offspring solutions, while in the self-adaptation phase, the new offspring solutions are modified individually. The output solutions of the self-adaptation phase are used for updating the content of the central memory. Termination of the search process may be triggered by reaching a predefined maximum number of iterations, by finding a solution the value of which is considered as good enough, or by satisfying any other stopping condition. The most successful evolutionary heuristics are hybrid algorithms in that sense that a descent method or a more advanced local search technique, called *local search operator* is used during the self-adaptation phase.

The local search operator and the recombination operator are both expected to contribute to the efficiency of a hybrid evolutionary heuristic. The main task of the local search operator is to find rapidly high quality solutions in a particular region of the search space. This operator however encounters two kinds of limitation. First, it can be trapped in a particular area of the search space. Second, as it uses only local information, it is not able to guide the search on long term. The recombination operator should complement the local search operator by diversifying the search, making it possible to explore new regions in the search space. Notice that diversification of the search can be simply implemented by using a restart procedure or by performing random mutations on the individuals in the central memory. However, in addition to diversification, a recombination operator is expected to bring another benefit. As it uses information gathered during the search, it should be able to guide the search on long term by detecting promising new regions. Moreover, it is now established that in order to be efficient, the recombination operator should exploit specific information from the problem at hand. More details on hybrid evolutionary algorithms can be found in [3, 12, 13].

The most famous hybrid evolutionary heuristic is probably the genetic local search algorithm that combines a standard local search (used in the self-adaptation phase) with a standard genetic algorithm. More precisely, the central memory of a genetic local search is made of a population of solutions and the recombination operator is a crossover that produces one or two offspring solutions by using a pair of parent solutions chosen in the population. A more recent hybrid evolutionary heuristic is the *adaptive memory algorithm* [19] that stores pieces of solutions (instead of complete solutions) in the central memory. While two parent solutions are combined to create an offspring in a genetic local search, all pieces of solutions in the central memory can contribute to the creation of an offspring in an adaptive memory algorithm. More details about this technique will be given in Section 2.

In this paper, we describe an adaptive memory algorithm to solve the graph colouring problem where the vertices of a graph must be coloured using as few colours as possible, so that no edge has both endpoints with the same colour. More precisely, the graph colouring problem can be described as follows. Given a graph $G = (V, E)$ with vertex set V and edge set E, and given an integer k, a *k-colouring* of G is a function $c : V \longrightarrow \{1, \ldots, k\}$. The value $c(x)$ of a vertex x is called the *colour* of x. Vertices with a same colour define a *colour class*. If two adjacent vertices x and y have the same colour, then vertices x and y are called *conflicting vertices* and the edge linking x with y is called a *conflicting edge*. A colour class without conflicting edge is called a *stable set*. A k-colouring without conflicting edges is said *legal* and corresponds to a partition of the vertices into k stable sets. The graph colouring problem (GCP for short) is to determine the smallest integer k (called *chromatic number* of G) such that there exists a legal k-colouring of G. The GCP is NP-hard [8]. Exact solution methods [1, 2, 18] can solve problems of relatively small size (no more than 100 vertices). Upper bounds on the chromatic number can be obtained for larger instances by using heuristic algorithms. A survey of some famous heuristic methods for the GCP can be found in [20].

Given a fixed integer k, we consider the optimization problem, called k-GCP, which aims to determine a k-colouring of G that minimizes the number of conflicting edges. If the optimal value of the k-GCP is zero, this means that G has a legal k-colouring. The chromatic number of G can be determined by first computing an upper bound on this number (for example by means of a constructive method) and then by solving a series of k-GCPs with decreasing values of k until no legal k-colouring can be obtained. Many local search methods have been proposed to solve the k-GCP. For example, a tabu search is described in [14], and simulated annealing algorithms can be found in [4, 15]. Hybrid evolutionary heuristics have also been successfully applied to this problem (e.g., [6, 7, 9]).

The current most efficient heuristic method for the k-GCP is a genetic local search, that we call GH for short, proposed by Galinier and Hao [9]. As most genetic hybrid evolutionary heuristics for the k-GCP, the GH algorithm uses the TABUCOL algorithm [14] as local search operator. The superiority of the GH algorithm over other hybrid evolutionary heuristics is probably due to the recombination operator. In previous genetic local search algorithms for the k-GCP [7, 9], offspring solutions are obtained by colouring half of the vertices as in one parent solution, and the second half as in the second parent solution. This means that the information exchanged during the cooperation phase is the colour of the vertices. Two k-colourings that are equivalent up to a permutation of the colours will therefore transmit different information while their colour classes are identical. The GH algorithm uses a recombination operator where an offspring is obtained by combining the colour classes of two parent solutions. This recombination operator will be described in more details in Section 2.2.

In this paper we propose an adaptive memory algorithm, called AMACOL, for the solution of the k-GCP. We show that AMACOL is simpler and more flexible than and at least as successful as the GH algorithm. The recombination operator used in AMACOL is based on the same ideas as those used in the GH algorithm. Our algorithm however differs

in several important features. As mentioned above, the GH algorithm creates an offspring by copying colour classes in two parent solutions, which means that the k colour classes of a k-colouring are linked together in the central memory. For comparison, the central memory in AMACOL contains colour classes that are not linked to each other. It may even happen that only a subset of colour classes of a k-colouring are in the central memory, while the other colour classes have been removed. As a consequence, one can for example transform a colour class of a k-colouring into a maximal stable set without taking care of the other colour classes of the k-colouring. More details about AMACOL and its way of handling the central memory will be given in the next section. We show in this paper that AMACOL produces results of as good quality as the GH algorithm while the handling of the central memory is easier and more flexible. Incidentally, this also means that we have identified which features present in the GH algorithm really contribute to its efficiency.

The rest of the paper is organized as follows. Section 2 contains a detailed description of algorithm AMACOL. Computational experiments are reported in Section 3 and concluding remarks are given in Section 4.

2 The AMACOL algorithm

The adaptive memory algorithm was first proposed by Rochat and Taillard in 1995 for solving the vehicle routing problem [19]. It is a hybrid evolutionary heuristic that uses a central memory $\mathcal{M}$ containing pieces of solutions. Its general scheme is summarized in Figure 1.

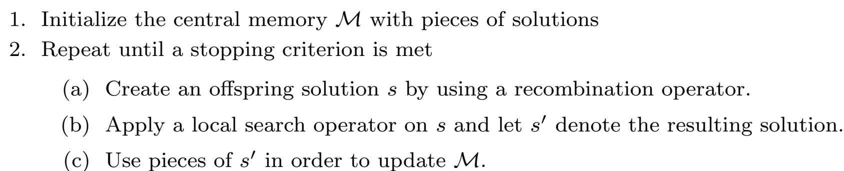


Figure 1: The adaptive memory algorithm

The process consisting of creating an offspring solution s', applying the local search operator on s', and updating the central memory with the resulting solution is called a generation. In order to create an offspring at step 2(a), the recombination operator first chooses pieces of solutions in $\mathcal{M}$. In the context of the k-GCP, we put stable sets in $\mathcal{M}$, which means that the recombination operator selects k stable sets $S_1, \dots, S_k$ in order to build a new k-colouring. The recombination operator cannot create a k-colouring by considering each S_i as the set of vertices with colour i. Indeed, vertices can appear in more than one stable set (such vertices are said *duplicated*) or in none of them (such vertices are said *uncoloured*). We show in Section 2.2 how our recombination operator avoids duplicated and uncoloured vertices. The output of the recombination operator is a k-colouring (not necessarily legal) that is considered as input for the local search operator.

In the following subsections, we give more details on some components of AMACOL such as the structure of the memory, the procedure that initializes it, the recombination operator, the local search operator, and the stopping criterion.

2.1 Memory structure and memory initialization

As already mentioned, the central memory $\mathcal{M}$ contains stable sets of the graph. The number of elements in $\mathcal{M}$ is a multiple of k: $\mid \mathcal{M} \mid = p \cdot k$, where p is a parameter of the method. According to preliminary experiments, and as in [9], we fix $p = 10$.

Before explaining how $\mathcal{M}$ is initialized, we first describe a procedure, called CLEAN, that transforms any subset of vertices into a maximal stable set (inclusion wise). This procedure will be used for filling and updating the central memory.

Procedure CLEAN(S)

Input : a subset S of vertices
Output : a maximal stable set

1. While S has at least one conflicting edge, do
   choose a vertex x in S that has a maximum number of neighbours in S
   (ties are broken randomly), and remove it from S.
2. Let C be the set of vertices that do not belong to S and have no neighbour in S.
3. While C is not empty, do
   choose a vertex x in C that has a minimum number of neighbours in C
   (ties are broken randomly), insert x in S, and remove x and all its neighbours from C.

Figure 2: The CLEAN procedure

The first loop in the CLEAN procedure transforms S into a stable set by removing conflicting vertices. The second loop inserts vertices into S until S becomes a maximal stable set. Ties are broken randomly.

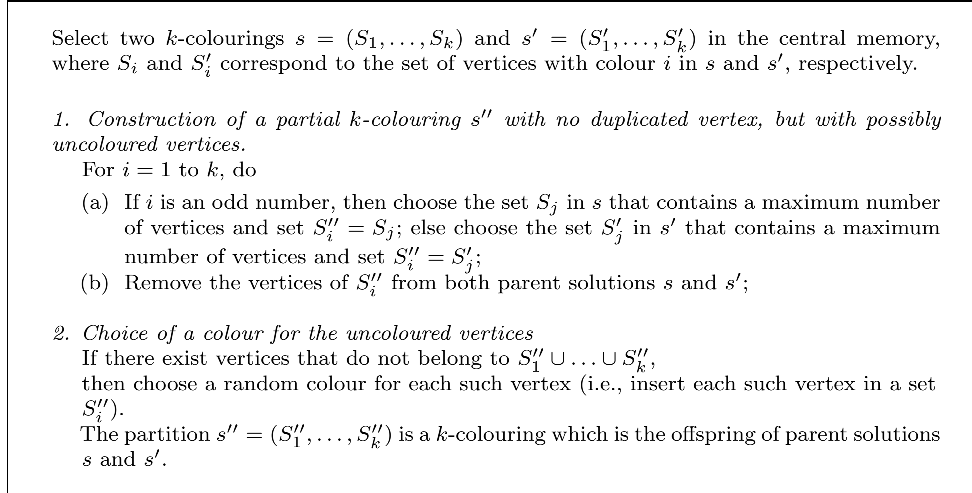
To initialize the central memory, we randomly build p k-colourings that are possibly improved by applying the local search operator. The colour classes of the resulting solutions are then transformed into maximal stable sets by means of procedure CLEAN, and all these stable sets are finally introduced into $\mathcal{M}$. More formally, this is done as follows.

1. Set $\mathcal{M} = \{\}$
2. For $i = 1$ to p, do
   (a) Generate a random k-colouring s by assigning a colour at random to each vertex.
   (b) Apply the local search operator to s for at most It_{max} iterations (a parameter) and let s' be the resulting k-colouring.
   (c) Apply procedure CLEAN on each colour class in s' and introduce each resulting maximal stable set into $\mathcal{M}$.

Figure 3: Memory initialization

2.2 The recombination operator

In order to clearly understand the difference between our proposed recombination operator and the one used in the GH algorithm [9], we first describe the latter one in details.



Select two k-colourings $s = (S_1, \ldots, S_k)$ and $s' = (S'_1, \ldots, S'_k)$ in the central memory, where S_i and S'_i correspond to the set of vertices with colour i in s and s', respectively.

1. Construction of a partial k-colouring s'' with no duplicated vertex, but with possibly uncoloured vertices.
For $i = 1$ to k, do

(a) If i is an odd number, then choose the set S_j in s that contains a maximum number of vertices and set $S''_i = S_j$; else choose the set S'_j in s' that contains a maximum number of vertices and set $S''_i = S'_j$;
(b) Remove the vertices of S''_i from both parent solutions s and s';

2. Choice of a colour for the uncoloured vertices
If there exist vertices that do not belong to $S''_1 \cup \ldots \cup S''_k$,
then choose a random colour for each such vertex (i.e., insert each such vertex in a set S''_i).
The partition $s'' = (S''_1, \ldots, S''_k)$ is a k-colouring which is the offspring of parent solutions s and s'.

Figure 4: The recombination operator of the GH algorithm

To illustrate this operator, assume that G contains 10 vertices $a, b, \ldots, j$ that we try to colour using 3 colours. Let $s = \{\{a, b, c\}, \{d, e, f, g\}, \{h, i, j\}\}$ and $s' = \{\{c, d, e, g\}, \{a, f, i\}, \{b, h, j\}\}$. Since S_2 is the largest colour class in s, S''_1 is set equal to $S_2 = \{d, e, f, g\}$. Vertices d, e, f and g are then removed from s and s' and we therefore get $s = \{\{a, b, c\}, \{h, i, j\}\}$ and $s' = \{\{c\}, \{a, i\}, \{b, h, j\}\}$. S'_3 is now the largest colour class in s' and one therefore set $S''_2 = S'_3 = \{b, h, j\}$. One then get $s = \{\{a, c\}, \{i\}\}$ and $s' = \{\{c\}, \{a, i\}\}$, and S''_3 is therefore set equal to $S_1 = \{a, c\}$. All vertices are now coloured in s'' except vertex i that is placed randomly in one of the sets S''_1, S''_2 or S''_3, say S''_3. The offspring is the k-colouring $s'' = \{\{d, e, f, g\}, \{b, h, j\}, \{a, c, i\}\}$.

The AMACOL uses a recombination operator that differs from the above one on several points. The offspring solution $(S_1, \ldots, S_k)$ is built class by class. Assume that colour classes $S_1, \ldots, S_{i-1}$ are already built. In order to create S_i, the operator first selects at random a sample $\{W_1, \ldots, W_q\}$ of q stable sets in $\mathcal{M}$. Then, the set W_r in this sample with a maximum number of uncoloured vertices is chosen, and S_i is set equal to the set of uncoloured vertices in W_r. This is a simple way to avoid duplicated vertices. Once the k colour classes are built, uncoloured vertices may still exist, and these are dealt in the same way as in the GH algorithm: each uncoloured vertex is inserted in a set S_i chosen at random. Notice that such a random strategy induces a large number of conflicting edges, but these conflicts are immediately handled by the local search operator, and no gain was observed when using a more sophisticated strategy. The recombination operator is formally described in Figure 5.

1. Construction of a partial k-colouring with no duplicated vertex, but with possibly uncoloured vertices

For $i = 1$ to k, do

(a) Select a random sample $W_1, \ldots, W_q$ of q elements in $\mathcal{M}$ (where q is a parameter).
(b) Determine a set W_r with a maximum number of vertices not yet in $S_1 \cup \ldots \cup S_{i-1}$.
(c) Set $S_i = W_r - (S_1 \cup \ldots \cup S_{i-1})$.

2. Choice of a colour for the uncoloured vertices

If there exist vertices that do not belong to $S_1 \cup \ldots \cup S_k$,
then choose a random colour for each such vertex (i.e., insert each such vertex in a set S_i).
The partition $s = (S_1, \ldots, S_k)$ is the resulting k-colouring.

Figure 5: The recombination operator of the AMACOL algorithm

In order to enter Step 2 with a minimum number of uncoloured vertices one could think about modifying Step 1 so that it produces a partial k-colouring that maximizes $| S_1 \cup \ldots \cup S_k |$. This is a maximum covering problem that can be solved to optimality if the central memory is of reasonable size. We have however observed that this leads to a too deterministic procedure that tends to produce approximately the same output at each iteration of the adaptive memory algorithm. For this reason, we have decided to implement Step 1 so that it generates "reasonably good" coverings.

Notice that the value of parameter q (the size of the sample) makes it possible to tune the degree of randomization of the procedure. A large value for q (close to $k \cdot p$) transforms the procedure into a pure greedy heuristic where each set S_i is chosen as the stable set in $\mathcal{M}$ with as many vertices as possible not yet in $S_1 \cup \ldots \cup S_{i-1}$. On the opposite, a small value for q offers the opportunity to choose stable sets in $\mathcal{M}$ with a very small number of uncoloured vertices. According to preliminary experiments we have fixed parameter q equal to the number k of colours.

2.3 The local search operator

The offspring solution s produced by the recombination operator is considered as input for the local search operator. This operator tries to improve on s by performing at most It_{max} iterations (where It_{max} is a parameter). We have implemented the same tabu search algorithm as the one used in the GH algorithm. It is an improved version of the TABUCOL algorithm proposed in [14]. While other local search algorithms have been proposed for the k-GCP (see the introduction for references), TABUCOL is simple and efficient as observed by many authors that have embedded a local search operator in an evolutionary heuristic.

TABUCOL can be briefly described as follows. The search space is the set of k-colourings and the objective function f to be minimized is the total number of conflicting edges. A neighbour solution is obtained by modifying the colour of a conflicting vertex. When the colour of a vertex x is modified from i to j, colour i is declared tabu for x for a certain number of iteration (called *tabu tenure*), and all solutions where x has colour i

are called *tabu solutions*. At each iteration, TABUCOL determines the best neighbour s' of the current solution s (ties are broken randomly) such that either s' is a non-tabu solution, or $f(s') = 0$ (i.e., s' is a legal k-colouring). The tabu tenure is set equal to $\text{UNIFORM}(0,9) + 0.6 \cdot NCV(s')$ where $\text{UNIFORM}(a,b)$ returns an integer randomly chosen in $\{a,\ldots,b\}$ and $NCV(s')$ represents the number of conflicting vertices in s' [9].

The local search operator returns the most recent best solution found during the search. Hence, if a solution is encountered with the same value as the current best solution, then the current best solution is replaced by the new one. The reason for choosing the most recent best solution is to possibly contribute (even moderately) to preserve memory diversity (this concept is described in Section 2.5).

2.4 Memory update

The output solution s' of the local search operator is used to update the central memory $\mathcal{M}$. Each colour class of s' is first transformed into a maximal stable set using procedure CLEAN. Then, the resulting k maximal stable sets are introduced into $\mathcal{M}$ in replacement of k elements of $\mathcal{M}$ chosen at random. This procedure is summarized in Figure 6.

1. Choose k elements in $\mathcal{M}$ at random, and remove them from $\mathcal{M}$
2. Apply procedure CLEAN on each colour class S_i of the output solution $s' = (S_1,\ldots,S_k)$ of the local search operator
3. Introduce the resulting maximal stable sets into $\mathcal{M}$.

Figure 6: Update of the central memory

2.5 Stopping criterion

The AMACOL algorithm stops when a legal k-colouring is found. As second stopping criterion, we fix a limit on the total number of iterations performed by the local search operator since the beginning of the AMACOL algorithm. We also consider a third stopping criterion that is related to the notion of diversity in the memory. Indeed, a well-known phenomenon generally observed in evolutionary heuristics is the progressive loss of diversity in the central memory. In AMACOL, this loss of diversity is characterized by k clusters of nearly similar maximal stable sets. It is then clear that the recombination operator will produce a similar output at each iteration, and it is therefore useless to continue the search. To measure the diversity of the memory, we consider $\mid S \cap S' \mid$ as a similarity measure between two elements S and S' of $\mathcal{M}$ and we denote by $N(S)$ the set containing the $\frac{p}{2}$ elements in $\mathcal{M}$ that are the most similar to S. For each element S in $\mathcal{M}$ we then compute the following value, denoted $\sigma(S)$:

$$\sigma(S) = \frac{\sum\limits_{S' \in N(S)} \mid S \cap S' \mid}{\mid N(S) \mid \mid S \mid}$$

It follows that if there are at least $\mid N(S) \mid = \frac{p}{2}$ stable sets in $\mathcal{M}$ that are identical to S, then $\sigma(S) = 1$. We finally compute the diversity measure $D(\mathcal{M})$ as follows:

$$D(\mathcal{M}) = 1 - \frac{1}{\mid \mathcal{M} \mid} \sum_{S \in \mathcal{M}} \sigma(S)$$

Notice that $D(\mathcal{M}) = 1$ in the hypothetical case where $\mathcal{M}$ contains $\mid \mathcal{M} \mid = p \cdot k$ disjoint sets, while $D(\mathcal{M}) = 0$ if $\mathcal{M}$ contains k clusters of p identical stable sets.

3 Computational experiments

3.1 Parameter setting

As mentioned in Section 2, AMACOL uses three parameters p, q and It_{max}. Parameter p is used to fix the size $\mid \mathcal{M} \mid = p \cdot k$ of the central memory, while parameter q fixes the size of the sample chosen in $\mathcal{M}$ by the recombination operator for building a new colour class (see Section 2.2). According to preliminary experiments we have fixed $p = 10$ and $q = k$. Finally, there is still a last parameter in AMACOL which is the number It_{max} of local search iterations performed on each generation by the local search operator. The value of this parameter may have a major effect on the performance of the algorithm. Basically, a large value of It_{max} preserves better the diversity in the central memory than a smaller value. Therefore, solving a difficult instance necessitates using a large value of the parameter because, when using a too small value, the diversity may be exhausted before a legal k-colouring is found by the algorithm. In other words, larger values of the parameter tend to make the algorithm more robust than smaller values. On the other hand, a small value or It_{max} allows the the cost function to decrease more quickly. Finally, it is also important to mention that the "optimal" value for parameter It_{max} when solving a particular k-colouring instance does not only depends on the characteristics of the graph, but also on the value of k.

In order to determine an appropriate value for this parameter, we use the following strategy. At the beginning, we fix It_{max} equal to the number of vertices to be coloured. If the diversity measure $D(\mathcal{M})$ becomes smaller than 0.1, then we multiply It_{max} by $\sqrt{2}$ as soon as 100 generations of AMACOL are performed without improvement of the best solution s^*. The algorithm stops if

1. a legal k-colouring has been discovered, or
2. the local search operator has performed a total number $TotalIt_{max}$ of iterations (without taking into account the initialization of the central memory) since the beginning of the search. We set $TotalIt_{max} = 250$ millions.

Notice that better results can probably be obtained by better tuning the parameters for each instance.

3.2 Algorithms used for comparisons

In order to demonstrate the importance of the recombination operator, we report on results obtained using TABUCOL with multiple restarts. More precisely, we first run TABUCOL 50 times with $It_{max} = 100'000$. If no legal k-colouring is found, we then run TABUCOL 30 times with $It_{max} = 1.5$ millions. If we are not successful we then run TABUCOL 10 times with $It_{max} = 10$ millions. Finally, if needed, we run TABUCOL 5 times with $It_{max} = 20$ millions. This gives a total number of 250 millions of iterations, as for AMACOL. This procedure will be called Long_TABU as opposed to Short_TABU that runs TABUCOL 5 times with $It_{max} = 100'000$.

We have also implemented the greedy algorithm DSATUR [1]. The four algoritms DSATUR, Short_TABU, Long_TABU and AMACOL were tested on all instances of the COLOR04 benchmarks graphs (see http://mat.gsia.cmu.edu/COLOR04/) as well as on the "flat" DIMACS benchmark graphs taken from [5], these latter being particularly challenging graphs. All results are presented in Tables 1, 2 and 3.

3.3 Experimental results

We first give detailed results for three random graphs of density 0.5: DSJC250.5, DSJC500.5 and DSJC1000.5. We run TABUCOL and AMACOL on these graphs with different values of k. Each algorithm was run three times on each graph and with each value of k. Instead of using the strategy described in Section 3.1 for tuning parameter It_{max}, we run AMACOL with different values of this parameter ($It_{max} = 500, 1000, 2000, 4000$, etc.) and we report results obtained with the best value. The two first columns of Table 1 indicate the name of the graph and the value of k. Each run of TABUCOL and AMACOL was given a maximum number of local search iterations that is indicated in the third column of Table 1 (in millions). The five next columns report the results obtained with AMACOL. We indicate the value of parameter It_{max}, the average number of local search iterations (in thousands), the average number of crossovers, the average computing time (in seconds) and the number of successful runs. The three following columns display the average number of iterations (in thousands), the average computing time (in seconds) and the number of successful runs for TABUCOL. We can observe from Table 1 that AMACOL was able to find k-colourings with less colours than TABUCOL. Moreover, when both TABUCOL and AMACOL produce a k-colouring with the same value k, AMACOL is faster.

Table 2 reports the results on some of the seemingly most challenging graphs. Each line in the table corresponds to a particular graph. The first columns indicate the name, the number of vertices, the number of edges and the density of the considered graph. The two following columns display the chromatic number of the graph (when it is known) and the smallest k for which an algorithm reported in the literature has been able to exhibit a k-colouring of the graph. Notice that some graphs with known chromatic number have never been coloured optimally, to the best of our knowledge (e.g., graphs flat300_28_0 and flat1000_76_0). The next four column, labelled DS, ST, LT and AMA report the best number of colours obtained with DSATUR, Short_TABU, Long_TABU and AMACOL, respectively. We finally indicate the value of It_{max} with which AMACOL has obtained the

best colouring, the number $NBiter$ of iterations performed by the local search operator (in thousands) with this value of It_{max}, and the corresponding computing time (in seconds). So, for example, for DSJC250.5, AMACOL has found a 29-colouring using TABUCOL for 3943 thousands of iterations since It_{max} was set equal to 1292. We have also tested the GH algorithm with a similar procedure for tuning It_{max} as for AMACOL. We got exactly the same results as AMACOL, except for the graph flat1000_76_0 where GH was able to exhibit a 83-colouring.

			AMACOL					TABUCOL		
Graph	k	Max	It_{max}	Iter	Cross.	Time	Succ.	Iter	Time	Succ.
DSJC250.5	28	20	4000	1348	337	47	3/3	5985	189	1/3
	29	10	500	43	86	4	3/3	611	19	3/3
	30	10	500	26	53	3	3/3	127	4	3/3
DSJC500.5	48	50	32000	14304	447	999	3/3	-	-	-
	49	10	4000	728	182	90	3/3	-	-	0/3
	50	10	2000	298	149	48	3/3	1574	173	3/3
	51	10	1000	86	86	23	3/3	139	15	3/3
DSJC1000.5	84	100	64000	38464	601	7554	2/3	-	-	-
	85	50	32000	16960	530	3384	3/3	-	-	-
	86	20	16000	5243	327	1508	3/3	-	-	-
	87	10	8000	1672	209	627	3/3	-	-	-
	88	10	4000	2375	593	609	3/3	-	-	0/3
	89	10	2000	864	432	340	3/3	7453	2534	1/3
	90	10	2000	269	134	139	3/3	3324	1130	3/3

Table 1: Detailed results on three random graphs

Graph	$\|V\|$	$\|E\|$	d	χ	Best	DS	ST	LT	AMA	It_{max}	NBiter	Time
DSJC250.5	250	15668	0.5	-	28	38	30	29	28	1292	3943	64
DSJC250.9	250	27897	0.89	-	72	91	73	72	72	353	6289	2604
DSJC500.1	500	12458	0.1	-	12	16	13	13	12	1000	550	9
DSJC500.5	500	62624	0.5	-	48	67	53	50	48	22627	4682	326
DSJC500.9	500	224874	0.9	-	126	161	133	130	126	45254	9454	1710
DSJR500.1c	500	121275	0.97	-	85	87	87	86	85			
DSJR500.5	500	58862	0.47	-	123	130	130	128	125			
DSJC1000.1	1000	49629	0.1	-	20	26	22	22	20	2828	4739	969
DSJC1000.5	1000	249826	0.5	-	83	114	95	89	84	128000	43269	9235
DSJC1000.9	1000	449449	0.9	-	224	297	248	245	224	32000	10342	4937
latin_square_10	900	307350	0.76	98	-	126	113	106	104	1272	2265	852
le450_15c	450	16680	0.16	15	15	24	21	16	15	10182	27	2

... continued on next page

Graph	$\|V\|$	$\|E\|$	d	χ	Best	DS	ST	LT	AMA	It_{max}	NBiter	Time
le450_15d	450	16750	0.17	15	15	24	22	16	15	5091	47	4
le450_25c	450	17343	0.17	25	25	29	27	26	26	1800	1719	93
le450_25d	450	17425	0.17	25	25	28	27	27	26	450	299	10
flat300_26_0	300	21633	0.48	26	26	41	33	27	26	95	848	4
flat300_28_0	300	21695	0.48	28	28	41	33	31	31	677	677	15
flat1000_50_0	1000	245000	0.49	50	50	112	93	92	50	1195	1000	1423
flat1000_60_0	1000	245830	0.49	60	60	113	95	93	60	1638	4000	1950
flat1000_76_0	1000	246708	0.49	76	76	114	95	88	84	10050	32000	11968

Table 2: Results obtained by DSATUR (DS), Short_TABU (ST), Long_TABU (LT) and AMACOL (AMA) on challenging graphs

From Table 2, we observe that in 16 out of 20 cases, AMACOL finds better colourings than Long_TABU. This demonstrates that the recombination operator is essential for the success of the evolutionary heuristic. We also observe that the quality of the solutions found by AMACOL is generally close to the optimum or the best known solution for the considered graph. In particular, AMACOL performs well on random DSJC graphs with density 0.5 which are the most studied graphs in the literature. Indeed, AMACOL reaches the best known results for the graphs having 250 vertices (28 colours) and 500 vertices (48 colours). However, the algorithm finds a legal 84-colouring for DSJC1000.5, while the best known solution for this graph has 83 colours.

We report in Table 3 the solutions found by DSATUR, Short_TABU, Long_TABU and AMACOL on all tested graphs. Notice first that many of these benchmark problems can be coloured optimally by a greedy algorithm such as DSATUR. Moreover we see from Table 3 that the four tested algorithms can be ordered as DSATUR $<$ Short_TABU $<$ Long_TABU $<$ AMACOL in that sense that each algorithm has produced on all instances an at least as good solution as its predecessors. To better analyse the results of Table 3, we propose to classify the instances into three groups. In the first group labelled "$k = \chi$", we include all benchmark graphs for which AMACOL was able to exhibit an colouring in $\chi(G)$ colours. In the second group labelled "k ? χ", we put all graphs G for which AMACOL has produced the best known upper bound k on $\chi(G)$, while it is not known whether k equals $\chi(G)$ or not. The last group contains all other graphs. It is labelled "$k > \chi$" since we know that the best k found by AMACOL is strictly larger than the chromatic number. For the first group "$k = \chi$", we indicate in the first row of Table 4 that 49 graphs can be coloured optimally using DSATUR (and hence also any of the three others algorithms). Twenty additional graphs were coloured optimally by Short_TABU (see the second row of Table 4). No other graph was coloured optimally by Long_TABU while 5 graphs were coloured optimally only using AMACOL. An analogous classification is indicated in Table 4 for the groups "k ? χ" and "$k > \chi$".

name	$\|V\|$	$\|E\|$	d	χ	DS	ST	LT	AMA
DSJC125.1	125	736	0.09	5	6	5	5	5
DSJC125.5	125	3891	0.49	-	21	17	17	17
DSJC125.9	125	6961	0.88	-	50	44	44	44
DSJC250.1	250	3218	0.1	-	10	8	8	8
DSJC250.5	250	15668	0.5	-	38	30	29	28
DSJC250.9	250	27897	0.89	-	91	73	72	72
DSJC500.1	500	12458	0.1	-	16	13	13	12
DSJC500.5	500	62624	0.5	-	67	53	50	48
DSJC500.9	500	224874	0.9	-	161	133	130	126
DSJR500.1	500	3555	0.03	-	12	12	12	12
DSJR500.1c	500	121275	0.97	-	87	87	86	86
DSJR500.5	500	58862	0.47	-	130	130	128	125
DSJC1000.1	1000	49629	0.1	-	26	22	22	20
DSJC1000.5	1000	249826	0.5	-	114	95	89	84
DSJC1000.9	1000	449449	0.9	-	297	248	245	224
fpsol2.i.1	496	11654	0.09	65	65	65	65	65
fpsol2.i.2	451	8691	0.09	30	30	30	30	30
fpsol2.i.3	425	8688	0.1	30	30	30	30	30
inithx.i.1	864	18707	0.05	54	54	54	54	54
inithx.i.2	645	13979	0.07	31	31	31	31	31
inithx.i.3	621	13969	0.07	31	31	31	31	31
latin_square_10	900	307350	0.76	-	126	113	106	104
le450_15a	450	8168	0.08	15	16	15	15	15
le450_15b	450	8169	0.08	15	16	15	15	15
le450_15c	450	16680	0.16	15	24	21	16	15
le450_15d	450	16750	0.17	15	24	22	16	15
le450_25a	450	8260	0.08	25	25	25	25	25
le450_25b	450	8263	0.08	25	25	25	25	25
le450_25c	450	17343	0.17	25	29	27	26	26
le450_25d	450	17425	0.17	25	28	27	27	26
le450_5a	450	5714	0.06	5	10	5	5	5
le450_5b	450	5734	0.06	5	9	5	5	5
le450_5c	450	9803	0.1	5	6	5	5	5
le450_5d	450	9757	0.1	5	11	5	5	5
mulsol.i.1	197	3925	0.2	49	49	49	49	49
mulsol.i.2	188	3885	0.22	31	31	31	31	31

. . . continued on next page

name	$\|V\|$	$\|E\|$	d	χ	DS	ST	LT	AMA
mulsol.i.3	184	3916	0.23	31	31	31	31	31
mulsol.i.4	185	3946	0.23	31	31	31	31	31
mulsol.i.5	186	3973	0.23	31	31	31	31	31
school1	385	19095	0.26	14	17	14	14	14
school1_nsh	352	14612	0.24	-	25	14	14	14
zeroin.i.1	211	4100	0.18	49	49	49	49	49
zeroin.i.2	211	3541	0.16	30	30	30	30	30
zeroin.i.3	206	3540	0.17	30	30	30	30	30
anna	138	493	0.05	11	11	11	11	11
david	87	406	0.11	11	11	11	11	11
homer	561	1629	0.01	13	13	13	13	13
huck	74	301	0.11	11	11	11	11	11
jean	80	254	0.08	10	10	10	10	10
games120	120	638	0.09	9	9	9	9	9
miles1000	128	3216	0.39	42	42	42	42	42
miles1500	128	5198	0.63	73	73	73	73	73
miles250	128	387	0.05	8	8	8	8	8
miles500	128	1170	0.14	20	20	20	20	20
miles750	128	2113	0.26	31	31	31	31	31
queen5_5	25	160	0.49	5	5	5	5	5
queen6_6	36	290	0.44	7	9	7	7	7
queen7_7	49	476	0.39	7	10	7	7	7
queen8_12	96	1368	0.29	12	13	12	12	12
queen8_8	64	728	0.35	9	12	9	9	9
queen9_9	81	2112	0.64	10	14	10	10	10
queen10_10	100	2940	0.58	11	13	11	11	11
queen11_11	121	3960	0.54	11	15	11	11	11
queen12_12	144	5192	0.5	12	15	13	13	13
queen13_13	169	6656	0.46	13	17	14	14	14
queen14_14	196	8372	0.43	14	18	15	15	15
queen15_15	225	10360	0.41	-	19	16	16	16
queen16_16	256	12640	0.38	-	21	17	17	17
myciel3	11	20	0.3	4	4	4	4	4
myciel4	23	71	0.26	5	5	5	5	5
myciel5	47	236	0.21	6	6	6	6	6
myciel6	95	755	0.17	7	7	7	7	7
myciel7	191	2360	0.13	8	8	8	8	8

. . . continued on next page

name	$\|V\|$	$\|E\|$	d	χ	DS	ST	LT	AMA
mugg88_1	88	146	0.04	4	4	4	4	4
mugg88_25	88	146	0.04	4	4	4	4	4
mugg100_1	100	166	0.03	4	4	4	4	4
mugg100_25	100	166	0.03	4	4	4	4	4
abb313GPIA	1557	53356	0.04	-	11	11	11	11
ash331GPIA	662	4185	0.02	-	5	5	5	5
ash608GPIA	1216	7844	0.01	-	5	5	5	5
ash958GPIA	1916	12506	0.01	-	6	6	6	6
will199GPIA	701	6772	0.03	-	7	7	7	7
1-Insertions_4	67	232	0.1	4	5	4	4	4
1-Insertions_5	202	1227	0.06	-	6	6	6	6
1-Insertions_6	607	6337	0.03	-	7	7	7	7
2-Insertions_3	37	72	0.1	4	4	4	4	4
2-Insertions_4	149	541	0.05	4	5	4	4	4
2-Insertions_5	597	3936	0.02	-	6	6	6	6
3-Insertions_3	56	110	0.07	4	4	4	4	4
3-Insertions_4	281	1046	0.03	-	5	5	5	5
4-Insertions_3	79	156	0.05	-	4	4	4	4
4-Insertions_4	475	1795	0.02	-	5	5	5	5
1-FullIns_3	30	100	0.22	4	4	4	4	4
1-FullIns_4	93	593	0.14	5	5	5	5	5
1-FullIns_5	282	3247	0.08	6	6	6	6	6
2-FullIns_3	52	201	0.15	5	5	5	5	5
2-FullIns_4	212	1621	0.07	6	6	6	6	6
2-FullIns_5	852	12201	0.03	7	7	7	7	7
3-FullIns_3	80	346	0.11	-	6	6	6	6
3-FullIns_4	405	3524	0.04	7	7	7	7	7
4-FullIns_3	114	541	0.08	7	7	7	7	7
4-FullIns_4	690	6650	0.03	8	8	8	8	8
5-FullIns_3	154	792	0.07	8	8	8	8	8
wap01	2368	110871	0.04	-	46	46	45	45
wap02	2464	111742	0.04	-	45	45	44	44
wap03	4730	286722	0.03	-	54	54	53	53
wap04	5231	294902	0.02	-	48	48	48	48
wap05	905	43081	0.11	-	50	50	50	50
wap06	947	43571	0.1	-	46	46	44	44
wap07	1809	103368	0.06	-	46	46	45	45

. . . continued on next page

name	$\|V\|$	$\|E\|$	d	χ	DS	ST	LT	AMA
wap08	1870	104176	0.06	-	45	45	45	45
qg.order60	3600	212400	0.03	60	62	60	60	60
qg.order100	10000	990000	0.02	100	103	100	100	100
flat300_20_0	300	21375	0.47	20	40	20	20	20
flat300_26_0	300	21633	0.48	26	41	33	27	26
flat300_28_0	300	21695	0.48	28	41	33	31	31
flat1000_50_0	1000	245000	0.49	50	112	93	92	50
flat1000_60_0	1000	245830	0.49	60	113	95	93	60
flat1000_76_0	1000	246708	0.49	76	114	95	88	84

Table 3: Results obtained by DSATUR (DS), Short_TABU (ST), Long_TABU (LT) and AMACOL (AMA) on the COLOR04 benchmark and on the flat graphs

$k = \chi$	DSATUR (49 graphs)	fpsol2 (3 graphs), inithx (3 graphs), le450_25a/b mulsol (5 graphs), zeroin (3 graphs), anna, david, homer, huck, jean games120, miles (5 graphs), queen5_5, myciel (5 graphs), mugg (4 graphs) Fullins (10 graphs), 2-Insertions_3, 3-Insertions_3
	Short_TABU (20 graphs)	DSJC125.1, le450_15a/b, le450_5a/b/c/d, school1 queen6_6, queen7_7, queen8_12, queen8_8, queen9_9, queen10_10 queen11_11, 1-Insertions_4, 2-Insertions_4, qg.order (2 graphs), flat300_20_0
	Long_TABU	(none)
	AMACOL (5 graphs)	le450_15c/d, flat300_26_0, flat1000_50_0, flat1000_60_0
k ? χ	DSATUR (16 graphs)	DSJR500.1, GPIA (5 graphs), Insertions (6 graphs), 3-FullIns_3 Wap04/05/08
	Short_TABU (6 graphs)	DSJC125.5, DSJC125.9, DSJC250.1 school1_nsh, queen15_15, queen16_16
	Long_TABU (7 graphs)	DSJC250.9, DSJR500.1c, wap01/02/03/06/07
	AMACOL (7 graphs)	DSJC250.5, DSJC500.1, DSJC500.5, DSJC500.9, DSJR500.5, DSJC1000.1 DSJC1000.9
$k > \chi$	DSATUR	(none)
	Short_TABU (3 graphs)	queen12_12, queen13_13, queen14_14
	Long_TABU (2 graphs)	le450_25c, flat300_28_0
	AMACOL (5 graphs)	DSJR500.5, DSJC1000.5, latin_square_10, le450_25d, flat1000_76_0

Table 4: Classification of the graphs

4 Final remarks

In this paper, we have developed an adaptive memory algorithm, called AMACOL for the solution of the k-GCP.

AMACOL is based on principles that are simple both conceptually and technically. Indeed, the recombination operator builds each colour class by performing a greedy choice in a sample of elements that is randomly chosen in the central memory. Moreover, a random strategy is used to update the central memory. Another important feature of AMACOL is the non-existence of any link between the colour classes stored in the central memory. An offspring solution is created by combining stable sets that may originate from k different k-colourings. This makes the recombination operator versatile and easy to adapt. For example, the colour classes of an offspring solution can be transformed, without taking care of the other colour classes of the k-colouring, before being inserted in the central memory. This is exactly what we did with procedure CLEAN that first removes conflicting edges in a colour class, and then eventually add vertices in order to obtain a maximal stable set.

As a conclusion, we present some additional features that can be considered for possibly improving AMACOL. The first possible extension is based on the observation that it often happens that an offspring solution contains colour classes that already belong to $\mathcal{M}$. It is then clear that this duplication of information is useless. In order to preserve the diversity of the memory, one can simply decide not to introduce such colour classes in the memory, without taking care of the other colour classes of the offspring. Another way to preserve diversity could be to create offsprings by combining elements S of the central memory with a small value $\sigma(S)$ (which measures how S is similar to the other elements in $\mathcal{M}$, see Section 2.5).

References

[1] D. Brélaz, *New Methods to Color Vertices of a Graph*, Communications of ACM 22, 251–256, 1979

[2] J.R. Brown, *Chromatic Scheduling and the Chromatic Number Problem*, Management Science 19, 456–463, 1972

[3] P. Calegari, C. Coray, A. Hertz, D. Kobler, P.Kuonen, *A Taxonomy of Evolutionary Algorithms in Combinatorial Optimization* Journal of Heuristics 5, 145–158, 1999

[4] M. Chams, A. Hertz, D. de Werra, *Some Experiments with Simulated Annealing for Coloring Graphs*, European Journal of Operational Research 32, 260–266, 1987

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

[6] D. Costa, A. Hertz, O. Dubuis, *Embedding of a Sequential Algorithm within an Evolutionary Algorithm for Coloring Problems in Graphs*, Journal of Heuristics 1, p. 105–128, 1995

[7] C. Fleurent, J.A. Ferland, *Genetic and Hybrid Algorithms for Graph Coloring*, Annals of Operations Research 63, 437–461, 1996

[8] M. Garey, D. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979

[9] P. Galinier, J.K. Hao, *Hybrid Evolutionary Algorithms for Graph Coloring*, Journal of Combinatorial Optimization 3, 379–397, 1999

[10] F. Glover, *Tabu Search - Part I*, ORSA J. Comput. 1, 190–206, 1989

[11] F. Glover, *Tabu Search - Part II*, ORSA J. Comput. 2, 4–32, 1990

[12] A. Hertz, *A Colorful Look on Evolutionary Techniques*, Belgian Journal of Operation Research 35, 23–39, 1997

[13] A. Hertz, D. Kobler, *A Framework for the description of Evolutionary algorithms*, European Journal of Operational Research 126, 1–12, 2000

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

[15] D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, *Optimization by Simulated Annealing: An Experimental Evaluation, Part II; Graph Coloring and Number Partitioning*, Operations Research 39, 378–406, 1991

[16] D.S. Johnson, M.A. Trick, *Proceeding of the 2nd DIMACS Implementation Challenge*, Volume 26 of *DIMACS* Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996

[17] S. Kirkpatrick, C.D. Gelatt, M. Vecchi, *Optimization by Simulated Annealing*, Science 220, 671–680, 1983

[18] J. Peemöller, *A Correction to Brélaz's Modification of Brown's Coloring Algorithm*, Communications of ACM 26, 593–597, 1983

[19] Y. Rochat, E. Taillard, *Probabilistic Diversification and Intensification in Local Search for Vehicle Routing*, Journal of Heuristics 1, 1995

[20] D. de Werra, *Heuristics for Graph Coloring*, Computing 7, 191–208, 1990