An Adaptive Memory Algorithm for the k-Colouring Problem

Philippe Galinier[1] Alain Hertz[2] Nicolas Zufferey[3]

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.

1. Introduction

Evolutionary heuristics encompass various algorithms such as genetic algorithms, scatter search, ant systems and adaptive memory algorithms [3, 11, 12]. 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

[1]Department of Computer Science, Ecole Polytechnique de Montréal and CRT, Canada, Philippe.Galinier@polymtl.ca

[2]Department of Mathematics and Industrial Engineering, Ecole Polytechnique de Montréal and GERAD, Canada, Alain.Hertz@gerad.ca

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

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, 11, 12].

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* [18] 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

[7]. Exact solution methods [1, 2, 17] 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 [19].

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 greedy 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 [13], and simulated annealing algorithms can be found in [4, 14]. Hybrid evolutionary heuristics have also been successfully applied to this problem (e.g., [5, 6, 8]).

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 [8]. As most genetic hybrid evolutionary heuristics proposed in the literature for the k-GCP, the GH algorithm uses TABUCOL (a tabu search algorithm) [13] as local search operator. The superiority of the GH algorithm over other hybrid evolutionary heuristics is due to the recombination operator. In previous genetic local search algorithms for the k-GCP [5, 6, 8], 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. Unlike the other existing hybrid evolutionary heuristics proposed for this problem, the central memory in AMACOL does not store solutions produced at previous generations but rather colour classes extracted from these solutions. The way we recombine colour classes in AMACOL is based on the same ideas as those used in the GH algorithm. However, an offspring solution can now be produced by using the colour classes originating from more than two solutions. Moreover, the handling of the central memory is made easier and more flexible. We show in this paper that AMACOL is simpler than and as successful as the GH algorithm.

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 [18]. 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.

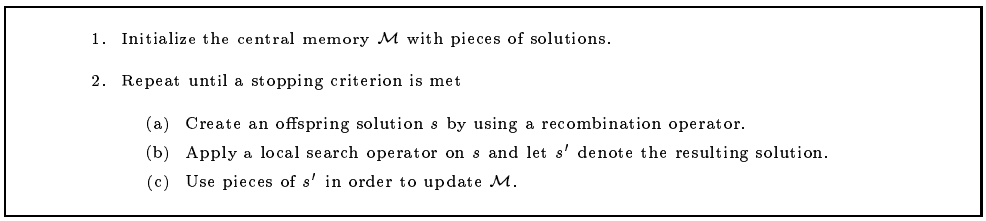


Figure 1 : The adaptive memory algorithm

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 $S_1, \ldots, S_k$ 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 [8], 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 recombination operator and the one used in the GH algorithm [8], 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 maximises $\mid S_1 \cup \ldots \cup S_k \mid$. 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 [13]. 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 UNIFORM$(0, 9) + 0.6 \cdot NCV(s')$ where UNIFORM(a, b) returns an integer randomly chosen in $\{a, \ldots, b\}$ and $NCV(s')$ represents the number of conflicting vertices in s' [8].

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 the fact that $\mathcal{M}$ contains 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 $\sigma(S)$:

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

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.

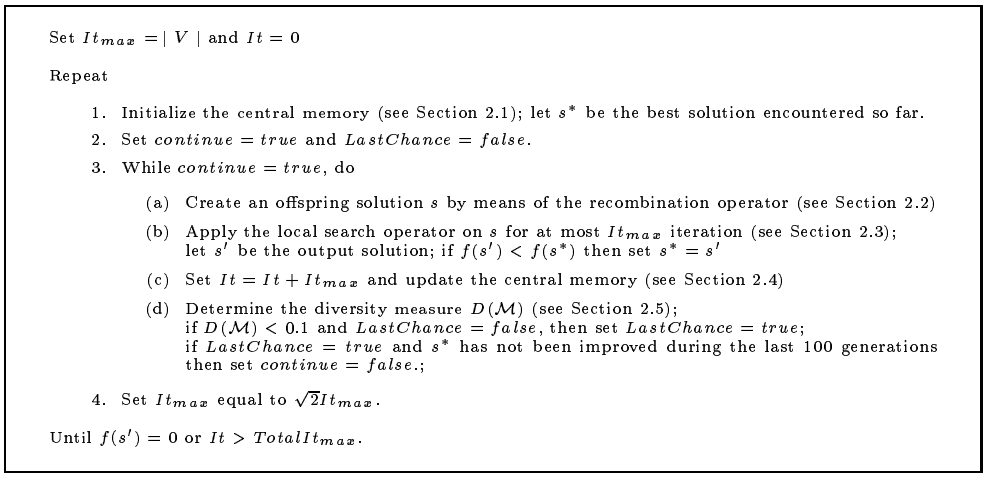
3. Computational experiments

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). As already mentioned, we have fixed $p = 10$ and $q = k$, according to preliminary experiments.

We measure the computational effort of AMACOL by counting the total number of iterations performed by the local search operator. Indeed, the computing time used by the recombination operator can be considered as negligible. Parameter It_{max} used by the local search operator has an important effect on the behaviour or the algorithm. Suppose that the total number of local search iterations in AMACOL is fixed. A small value for It_{max} indicates that the recombination operator is called more frequently than with a large value. We have observed that the value of the objective function decreases more rapidly when It_{max} has a small value while a large value for It_{max} better preserves the diversity of the central memory. The most appropriate value for It_{max} depends both on the graph instance and on the number k of colours. Instead of fine tuning this parameter for each particular instance, we have decided, for convenience, to apply the following strategy in our experiments.

We perform successive runs of AMACOL with increasing values for It_{max}, starting with a small value (equal to the number of vertices in the considered graph). When the diversity measure $D(\mathcal{M})$ becomes smaller than 0.1, the algorithm is given a maximum number of 100 generations without improvement of the best solution s^*. Then, the algorithm is reinitialized with a larger value for It_{max} (It_{max} is multiplied by $\sqrt{2}$). This process is applied until a legal k-colouring is found, or a total number $TotalIt_{max}$ (equal to 250 millions) of local search iterations (without taking into account the initialization of the central memory) have been performed since the beginning of the search process. This procedure is summarized in Figure 7.


Set $It_{max} = \mid V \mid$ and $It = 0$

Repeat

1. Initialize the central memory (see Section 2.1); let s^* be the best solution encountered so far.
2. Set $continue = true$ and $LastChance = false$.
3. While $continue = true$, do
   (a) Create an offspring solution s by means of the recombination operator (see Section 2.2)
   (b) Apply the local search operator on s for at most It_{max} iteration (see Section 2.3); let s' be the output solution; if $f(s') < f(s^*)$ then set $s^* = s'$
   (c) Set $It = It + It_{max}$ and update the central memory (see Section 2.4)
   (d) Determine the diversity measure $D(\mathcal{M})$ (see Section 2.5); if $D(\mathcal{M}) < 0.1$ and $LastChance = false$, then set $LastChance = true$; if $LastChance = true$ and s^* has not been improved during the last 100 generations then set $continue = false$.;
4. Set It_{max} equal to $\sqrt{2}It_{max}$.

Until $f(s') = 0$ or $It > TotalIt_{max}$.

Figure 7 : The AMACOL algorithm

3.2 Algorithms used for comparisons

In order to make comparisons we have implemented the GH algorithm by modifying the algorithm of Figure 7 as indicated below.

- Step 1: the initialization of the central memory is done as follows : generate p random k-colourings, apply the local search operator for at most It_{max} iterations on each of them, and introduce the p resulting k-colourings into $\mathcal{M}$.

- Step 3(a): use the recombination operator proposed by Galinier and Hao (see Section 2.2) instead of the AMACOL recombination operator.
- Step 3(c): when two parent solutions are combined to create an offspring s', the offspring is introduced in the central memory and the worse parent is removed.

Also, 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 and the GH algorithm.

3.3 Experimental results

All algorithms are tested on the 16 instances proposed at the COLOURING02 workshop (see http://mat.gsia.cmu.edu/COLOURING02/). These instances encompass all large and difficult instances of the famous DIMACS benchmarks [15]. Results are reported in Table 1. For each instance we indicate the number $\mid V \mid$ of vertices, the edge density d, and the chromatic number $\chi(G)$ when known. For TABUCOL we indicate the smallest k for which a legal k-colouring was found (first line) and the total number (in thousands) of iterations that were needed to obtain such a colouring (second line). For the GH and AMACOL algorithms, we also give the smallest k for which a legal k-colouring was found (first line). In addition we indicate the value of It_{max} for which such a colouring was found (number in parenthesis in the second line) and the number of iterations performed by the local search operator (in thousands) with this value of It_{max} before the legal k-colouring was found (first number in the second line). So, for example, for DSJC500.1, AMACOL has found a 12-colouring using TABUCOL for 550 thousands of iterations since It_{max} was set equal to 1'000.

It clearly appears in Table 1 that GH and AMACOL outperform TABUCOL. Indeed, except for le450.25c, flat300.20 and flat300.28, both hybrid algorithms find a solution with less colours than TABUCOL. This demonstrates that the recombination operator is essential for the success of both hybrid evolutionary heuristics. By comparing GH with AMACOL, we see that both algorithms find colourings with the same number of colours on all instances except on flat1000.76. Moreover, both GH and AMACOL have found an optimal colouring on 6 among the 10 instances for which the chromatic number is known.

If we now concentrate on the 15 first instances for which GH and AMACOL find legal colourings with the same number of colours, we can observe that AMACOL uses a larger value of It_{max} than GH on 9 cases. This means that with the chosen

values for parameters p and q, the loss of diversity in the central memory is faster for AMACOL than for GH on these 9 instances. However, by fixing It_{max} equal to the values given in parenthesis in Table 1, we can observe that AMACOL is faster than GH for 9 of the 15 instances (i.e., AMACOL performs less iterations of the local search operator than GH). In summary, there is no clear evidence that one of these two algorithms is better than the other one.

Graph	$\mid V \mid$	d	$\chi(G)$	TABUCOL	GH algorithm	AMACOL
DSJC500.1	500	0.1	?	**13** 4	**12** 1'956 (4'000)	**12** 550 (1000)
DSJC500.5	500	0.5	?	**50** 1'238	**48** 3'692 (8'000)	**48** 4'682 (22'627)
DSJC500.9	500	0.9	?	**130** 32	**126** 17'857 (32'000)	**126** 9'454 (45'254)
DSJC1000.1	1000	0.1	?	**22** 2	**20** 14'029 (5'657)	**20** 4'739 (2'828)
DSJC1000.5	1000	0.5	?	**89** 7'822	**84** 18'113 (22'627)	**84** 43'269 (128'000)
DSJC1000.9	1000	0.9	?	**245** 31	**224** 39'275 (128'000)	**224** 10'342 (32'000)
le450.15c	450	0.16	15	**16** 23	**15** 41 (900)	**15** 27 (10'182)
le450.15d	450	0.16	15	**16** 27	**15** 54 (1'272)	**15** 47 (5'091)
le450.25c	450	0.17	25	**26** 31	**26** 735 (900)	**26** 1'719 (1'800)
le450.25d	450	0.17	25	**27** 81	**26** 377 (636)	**26** 299 (450)
flat300.20	300	0.47	20	**20** 3	**20** 6 (300)	**20** 2 (300)
flat300.26	300	0.48	26	**27** 30	**26** 35 (424)	**26** 95 (848)
flat300.28	300	0.48	28	**31** 789	**31** 369 (1'697)	**31** 677 (6'788)
flat1000.50	1000	0.49	50	**92** 27	**50** 718 (2'000)	**50** 1'195 (1'000)
flat1000.60	1000	0.49	60	**93** 25	**60** 1'842 (2'828)	**60** 1'638 (4'000)
flat1000.76	1000	0.49	76	**88** 6'496	**83** 42'171 (64'000)	**84** 10'050 (32'000)

Table 1: Comparison of the different algorithms

4. Final remarks

In this paper, we have proposed an adaptive memory algorithm, called AMACOL for the solution of the k-GCP. The recombination operator in AMACOL is specialized for the k-GCP and inspired from the crossover operator used in the genetic local search algorithm GH proposed in [8]. The basic idea of our recombination operator is to use colour classes produced in previous generations in order to build a partial

legal k-colouring with a large number of coloured vertices, and to complete this partial legal k-colouring in order to get a k-colouring. Notice that the recombination operator does not construct an optimal partial legal colouring (i.e., a partial legal colouring with as many coloured vertices as possible) since such a deterministic strategy reduces the diversity of the central memory too rapidly.

The above idea is implemented in AMACOL in the simplest possible way. Indeed, the central memory stores colour classes (and not k-colourings), 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, and a random strategy is used to update the central memory. Notice that we do not use any elitist strategy to update the central memory as neither the quality of an offspring k-colouring, nor the size of the colour classes to be removed or inserted into $\mathcal{M}$ are taken into account. In summary, AMACOL is one of the simplest hybrid evolutionary heuristics for the k-GCP, and the results provided in this paper show that AMACOL outperforms local search algorithms (when used alone, without recombination), and is competitive with the best known hybrid evolutionary heuristics on large difficult benchmark problems.

Although GH and AMACOL are based on similar ideas, AMACOL is simpler both conceptually and technically. This is not surprising since the idea of using colour classes as building blocks is more consistent with an adaptive memory approach than with a genetic one. The genetic approach used in GH requires the handling of complete k-colourings and this makes things more complicated and less natural. For example, GH chooses the colour classes of an offspring solution *alternatively* in each of the two parents, and this is necessary for balancing the role or the two parents. More importantly, the handling of the central memory is easier and more flexible in AMACOL than in GH. 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.

The following additional features can be considered for possibly improving AMACOL, while they can hardly be implemented in GH. The aim of these features is to carefully manage the contents of the central memory in order to preserve diversity. As first possible extension, we observe 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 such a case, one can decide not to introduce such colour classes in $\mathcal{M}$. Another idea is to take into account the individual participation $\sigma(S)$ of each colour class to the diversity measure $D(\mathcal{M})$ (see Section 2.5). For example, one could decide to favour colour classes S in $\mathcal{M}$ with a small value $\sigma(S)$, since such colour classes contribute highly to the total diversity.

Notice finally that this study also contributes to a better understanding of the key features that are essential for the success of GH. Indeed, in AMACOL, offspring

solutions are created by combining colour classes originating from more than two parents, the central memory does not consider any link between the colour classes of a k-colouring, and no elitist strategy is used to update the memory. It clearly appears in Table 1 that these modifications do not seem to have any negative impact on the performance of GH since they have not led to any observable loss of quality.

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] 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

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

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

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

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

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

[11] A. Hertz, *A Colourful Look on Evolutionary Techniques*, Belgian Journal of Operation Research 35, 23-39, 1997

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

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

[14] 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

[15] 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

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

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

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

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