Coloring large graphs based on independent set extraction

Qinghua Wu and Jin-Kao Hao *

LERIA, Université d'Angers
2 Boulevard Lavoisier, 49045 Angers Cedex 01, France

To appear in *Computers and Operations Research*, 2011
DOI:10.1016/j.cor.2011.04.002

Abstract

This paper presents an effective approach (EXTRACOL) to coloring large graphs. The proposed approach uses a preprocessing method to extract large independent sets from the graph and a memetic algorithm to color the residual graph. Each preprocessing application identifies, with a dedicated tabu search algorithm, a number of pairwise disjoint independent sets of a given size in order to maximize the vertices removed from the graph. We evaluate EXTRACOL on the 11 largest graphs (with 1000 to 4000 vertices) of the DIMACS challenge benchmarks and show improved results for 4 very difficult graphs (DSJC1000.9, C2000.5, C2000.9, C4000.5). The behavior of the proposed algorithm is also analyzed.

Keywords: Graph coloring, independent set, preprocessing, tabu search, memetic search.

1 Introduction

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. An independent set is a subset of vertices, $S \subseteq V$, such that no two vertices in S are adjacent. A legal k-coloring of G corresponds to a partition of V into k independent sets (color classes). Graph coloring aims at finding the smallest k for a given graph G (its chromatic number $\chi(G)$) such that G has a legal k-coloring.

* Corresponding author.
Email addresses: wu@info.univ-angers.fr (Qinghua Wu), hao@info.univ-angers.fr (Jin-Kao Hao).

The graph coloring problem is a well-known NP-hard combinatorial optimization problem [16,21]. Prominent applications of graph coloring include crew scheduling [14], computer register allocation [7], timetabling and scheduling [3], radio frequency assignment [30], printed circuit board testing [15] and satellite range scheduling [31]. Exact solution methods can solve problems of relatively small size, whereas heuristics are able to handle larger graphs. A comprehensive survey of the most significant heuristic methods can be found in [12]. We detail some of them in the following.

Local search has been repeatedly applied to graph coloring. Well-known examples include the seminal TabuCOL algorithm [18], Simulated Annealing [20], GRASP [22], Iterated Local Search [5], Neighborhood Search [1], Reactive Partial Tabu Search [2], Variable Space Search [19] and Clustering-Guided Tabu Search [27]. Local search coloring algorithms are usually simple and have achieved satisfactory performance on the standard DIMACS graphs. They are also often used as a key component of more sophisticated hybrid algorithms.

Indeed, it was observed that some graphs, especially large random graphs, cannot be colored efficiently by using pure local search algorithms. Several hybrid approaches have been proposed as a very interesting alternative. Examples of classical hybrid algorithms are presented in [6,8,9,11,25,17]. More recent and powerful algorithms can be found in [13,23,24,28,32].

Another approach for dealing with large graphs is based on a general principle of "reduce-and-solve". This approach consists in applying, prior to the phase of graph coloring, a preprocessing procedure to extract large independent sets from the graph until the remaining graph (called residual graph) becomes sufficiently small and then coloring the residual graph with any coloring algorithm. Typically, large independent sets are extracted and removed from G in an iterative manner by identifying one largest possible independent set each time. This approach was used with success by some prominent classical algorithms [4,9,18,20,25]. Yet surprisingly we observe that it is rarely employed nowadays.

In this paper, we revisit the graph coloring approach using independent set extraction and develop an improved preprocessing procedure. Basically, instead of extracting independent sets one by one, we try to identify, at each preproceesing step, a maximal set of pairwise disjoint independent sets. The rationale behind this approach is that by extracting many pairwise disjoint independent sets each time, more vertices are removed from the initial graph, making, hopefully, the residual graph easier to color. For both tasks of identifying an individual independent set and pairwise disjoint independent sets, we employ a dedicated Adaptive Tabu Search algorithm.

Our large graph coloring algorithm (denoted by EXTRACOL) combines this

improved independent set extraction preprocessing with a recent memetic coloring algorithm (MACOL [23]). We evaluate the performance of EXTRACOL on the 11 largest DIMACS benchmark graphs (with 1000, 2000 and 4000 vertices) and present new results for four very hard instances (DSJC1000.9, C2000.5, C2000.9, C4000.5), with respectively 1, 2, 4 and 11 colors below the currently best colorings.

The rest of this paper is organized as follows. In Section 2, we review the coloring approach based on extraction of large independent sets. In Section 3, we provide a detailed presentation of the proposed algorithm. In Section 4, we show extensive experimental results and comparisons. Section 5 is dedicated to an investigation of the behavior of the proposed algorithm. Conclusions are given in the last section.

2 Review of graph coloring based on maximum independent set

As observed in [18,9,20], it is difficult, if not impossible, to find a k-coloring of a large graph G (e.g. $|V| \geq 1000$) with k close to $\chi(G)$ by applying directly a given algorithm α on G. To color large and very large graphs, a possible approach is to apply a preprocessing to extract i large independent sets from the graph to obtain a much smaller residual graph which is expected to be easier than the initial graph. A coloring algorithm α is then invoked to color the residual graph. Since each of the i independent sets forms a color class, i plus the number of colors needed for the residual graph gives an upper bound on $\chi(G)$.

The conventional methods for the preprocessing phase operate greedily by extracting one independent set each time from the graph G. For instance, in [4,18], the authors uses a tabu search algorithm to identify a large independent set and then remove the vertices of the independent set as well as their incident edges from G. In [9], the authors refine the choice of the independent set to be removed and prefer an independent set that is connected to as many vertices as possible in the remaining graph. The selected independent set can then remove as many edges as possible with the vertices of the independent set, hence, tends to make the residual graph easier to color.

Notice that finding maximum independent sets in a graph is itself an NP-hard problem [16] and several heuristics have been used in the context of graph coloring in the literature. For instance, in [4], the authors apply a simple greedy algorithm. In [9,18], the authors propose a dedicated tabu search algorithm. In [20], the authors introduce the XRLF method that combines a randomized greedy search with an exhaustive search.

The performance of this coloring approach depends on the method to find large independent sets and the method to color the residual graph. It also depends significantly on how independent sets are selected by the preprocessing phase.

In the next section, we present our preprocessing method which tries to extract a large number of pairwise disjoint independent sets from the graph at each preprocessing iteration.

3 EXTRACOL: an algorithm for large graph coloring

The proposed EXTRACOL algorithm follows the general schema presented in the previous section and is composed of two sequential phases: a *preprocessing phase* and a *coloring phase*. What distinguishes our work from the existing methods is the way in which the preprocessing phase is carried out. In this section, we explain the basic rationale and illustrate the main techniques implemented. The coloring algorithm applied to the residual graph (MACOL [23]) is also briefly reviewed at the end of the section.

3.1 General procedure

As stated previously, a legal k-coloring of a given graph $G = (V, E)$ corresponds to a partition of V into k independent sets. Suppose that we want to color G with k colors. Assume now that we extract $t < k$ independent sets $I_1, ..., I_t$ from G. It is clear that if we could maximize the number of vertices covered by the t independent sets (i.e. $|I_1 \cup ... \cup I_t|$ is as large as possible), we would obtain a residual graph with fewer vertices when $I_1 \cup ... \cup I_t$ are removed from G. This could in turn help to ease the coloring of the residual graph using $k - t$ colors because there are fewer vertices left in the residual graph.

For this reason, for a given graph G, each time a maximum (or large) independent set is identified we try to maximize the number of *pairwise disjoint* independent sets of that size and then remove all of them from the graph. Our preprocessing phase repeats this process until there are no more than q vertices left in the residual graph, q being a parameter fixed by the user. The value of q depends clearly on the coloring algorithm applied to the residual graph. In our case, q is set to be equal to 800 for all the tested graphs in this paper.

Our preprocessing phase is then described in Algorithm 1. Basically, our preprocessing phase iterates the following 4 steps and stops when the residual

graph contains no more than q vertices.

Algorithm 1 Preprocessing phase: Extraction of pairwise disjoint independent sets

Require: Graph $G = (V, E)$, the size of the residual graph q
Ensure: Residual graph of G

1: **Begin**
2: **while** ($|V| > q$) **do**
3: $I_M = \text{ATS}(G, \_, Iter)$ {Use ATS to find a maximal independent set in G, see Section 3.2}
4: $z_{max} = |I_M|$
5: $M = \{I_M\}$
6: $reapt = 0$
7: **while** ($reapt \leq p_{max}$ AND $|M| \leq M_{max}$) **do**
8: $I = \text{ATS}(G, z_{max}, Iter)$ {Use ATS to find an independent set of size z_{max}}
9: **if** $I \in M$ **then**
10: $reapt = reapt + 1$
11: **else**
12: $M = M \bigcup \{I\}$
13: $reapt = 0$
14: **end if**
15: **end while**
16: Find a maximal number of pairwise disjoint independent sets in M: $\{I_1, ..., I_t\} = arg\ max\{|S| : S \subseteq M, \forall I_x, I_y \in S, I_x \bigcap I_y = \emptyset\}$ {see Section 3.3}
17: **if** $|V| - |I_1 \cup ... \cup I_t| > q$ **then**
18: Remove $I_1, ..., I_t$ from G
19: **else**
20: $p = \lceil \frac{|V|-q}{z_{max}} \rceil$
21: Remove $I_1^{'}, ..., I_p^{'}$ randomly selected from $\{I_1, ..., I_t\}$ from G
22: **end if**
23: **end while**
24: **End**

(1) Apply the Adaptive Tabu Search (ATS) algorithm (see Section 3.2) to identify a maximal independent set I_M in G and set $z_{max} = |I_M|$. Then use I_M to initialize set M which will collect other independent sets of the same size (lines 3-5 in Algorithm 1).
(2) Apply repeatedly ATS to generate as many independent sets of size z_{max} as possible and put them in set M (lines 7-15).
(3) Find as many *pairwise disjoint* independent sets $I_1, ..., I_t$ as possible from M (line 16, see Section 3.3).
(4) Remove all the vertices of $I_1 \cup ... \cup I_t$ from G if the removal leads to a residual graph with at least q vertices. Otherwise, remove randomly independent sets from $\{I_1, ..., I_t\}$ such that the residual graph contains at most q vertices (lines 17-23).

Notice first that this preprocessing procedure is different from conventional methods since steps (2)-(3) are missing from these conventional methods. In our case, after a maximal independent set is identified, we continue to identify as many independent sets of that size as possible with the purpose of extracting a maximal number of pairwise disjoint independent sets.

In Algorithm 1, step (1) identifies a first maximal independent set I_M whose size z_{max} is used at step (2) to build the pool M of independent sets of that size. The search for a new independent set I of size z_{max} stops when the number of independent sets contained in M reaches a desired threshold (M_{max}) or when no new independent set of that size is found after p_{max} consecutive tries. Obviously, both a larger value for M_{max} or p_{max} could include more independent sets in M, thus giving more chance of finding more pairwise disjoint independent sets of size z_{max} in M in step (3). On the other hand, a larger value for M_{max} (or p_{max}) also implies longer computing time for the preprocessing phase. According to our experiments we fix p_{max} equal to 100 and M_{max} equal to $|V| \cdot \rho$, where ρ is the density of the graph. For the tested DIMACS graphs, M_{max} varies from 100 to 2000.

Step (3) aims at finding a maximum number of *pairwise disjoint* independent sets from the set of collected independent sets of size z_{max}. As we explain in Section 3.3, this problem is in fact equivalent to the maximum set packing problem which itself can be approximated by our Adaptive Tabu Search for the maximum independent set problem.

In the next Section, we present the ATS algorithm which is the key element of our preprocessing phase.

3.2 Adaptive Tabu Search for maximum independent set

The Adaptive Tabu Search algorithm is designed to find an independent set of given size k within a graph $G = (V, E)$ (see Algorithm 2). If k is not specified, the ATS algorithm just returns the largest possible independent set that can be identified.

ATS explores a search space composed of subsets of V of size k (k-subsets) which is formally identified by: $\Omega = \{S \subset V : |S| = k\}$. For any candidate solution $S \in \Omega$, the evaluation function $f(S)$ counts the number of edges in the subgraph of G induced by S. In other words, let $G_S = (S, E_S)$ such that $E_S = \{\{u, v\} \in E : u, v \in S\}$, than $f(S) = |E_S|$. Obviously, if $f(S) = 0$, any two vertices of S are not adjacent and S is an independent set. Otherwise, S is not an independent set of G. The objective of our ATS algorithm is to find a solution S^* such that $f(S^*)$ reaches its minimum value $f(S^*) = 0$.

A neighbor of a given solution S could be obtained by simply swapping a vertex of S with another vertex of $V\backslash S$. However, such a move is not focused and cannot guide the search process to efficiently explore the search space. In what follows, we introduce a *constrained neighborhood* which is both more focused and smaller-sized.

Let $S \in \Omega$ be a candidate solution (k-subset). For each vertex $v \in V$, let $d(v)$ denotes the degree of v relative to the subset S:

$d(v) = |\{i \in S : \{i, v\} \in E\}|$

Let *tabu_list* be the tabu list containing the vertices that are currently forbidden for migration. Let $MaxInS = \max\{d(u) : u \in S,\ u \notin tabu\_list\}$ and $MinOutS = \min\{d(v) : v \in V\backslash S,\ v \notin tabu\_list\}$.

Define:

$A = \{u \in S : u \notin tabu\_list,\ d(u) = MaxInS\}$.

$B = \{\ v \in V\backslash S : v \notin tabu\_list,\ d(v) = MinOutS\}$.

$T = \{(u, v) : u \in A, v \in B, \{u, v\} \in E\}$.

In other words, set A identifies the vertices of the current solution S that have the highest number of adjacent vertices in S. Set B identifies the vertices of $V - S$ that have the smallest number of adjacent vertices in S. To obtain a neighboring solution S' from S, one swaps one vertex $u \in A$ and another

Algorithm 2 Pseudo-code of ATS for maximal independent set

```
1: Input: Graph G, Integer z, Integer L
2: Output: independent set of size z if found
3: Begin
4: N_I = 0; {N_I is the consecutive iterations during which f(S) is not improved}
5: S = Random_Initialize ()
6: S* = S
7: while (N_I < L and f(S) > 0) do
8:    Find the best allowed move (u, v) ∈ A × B {See comments in the text}
9:    S = S ∪ {v}\{u}, u ∈ S, v ∈ V\S
10:   Mark u and v tabu for the next T_u and T_v iterations respectively
11:   N_I = N_I + 1;
12:   if f(S) < f(S*) then
13:      S* = S
14:      N_I = 0
15:   end if
16: end while
17: Return (independent set S* of size z or report failure)
18: End
```

vertex $v \in B$. This transition (from S to S') can be conveniently characterized by a move denoted by $swap(u, v)$ and written formally as: $S' = S \oplus swap(u, v)$ or equivalently $S' = S \backslash \{u\} \cup \{v\}$. For a given $swap(u, v)$, the move gain Δ_{uv} can be conveniently computed by: $\Delta_{uv} = f(S') - f(S) = d(v) - d(u) - e_{uv}$ where e_{uv} is equal to 1 if $\{u, v\} \in E$, 0 otherwise.

Now, to obtain the best neighbor solution, we use the following strategy. If T is not empty, then one pair (u, v) from T is randomly selected for swap. Otherwise, vertex u is randomly selected from A and v is randomly selected from B. We prefer $(u, v) \in T$ since it can produce a smaller move gain value $(MinOutS - MaxInS - 1)$.

For the tabu list, once a $swap(u, v)$ move is performed, u and v are marked tabu for the next T_u and T_v iterations respectively. To be effective, the tabu tenures T_u and T_v are dynamically and adaptively adjusted: $T_u = f(S) + Random(4)$ and $T_v = 0.6 \cdot T_u$, where $Random(4)$ returns randomly a number in $\{1, \ldots, 4\}$.

3.3 Finding maximal pairwise disjoint independent sets

As mentioned before, given the set of independent sets $M = \{I_1, ..., I_n\}$ of G which are found by ATS, we want to determine from M as many pairwise disjoint sets as possible. In fact, this is a typical maximum set packing problem (MSPP), which itself is equivalent to the maximum clique (thus independent set) problem [16].

Our approach exploits the strict relation between the MSPP and the maximum independent set problem. More precisely, we transform the MSPP into the maximum independent set problem and then determine a maximum independent set in the transformed graph. Given n independent sets $\{I_1, ..., I_n\}$, we define a new maximum independent set instance $G' = (V', E')$ as follows. Define V' by $\{1, ..., n\}$ and define the edge matrix by:

$$e_{ij} = \begin{cases} 0, \text{ if } I_i \cap I_j = \emptyset, i, j \in \{1, ..., n\} \\ 1, \text{ otherwise.} \end{cases}$$

Note that the order n of graph G' is bounded by $M_{max} = |V| \cdot \rho$ (see Section 3.1). For the graphs used in this paper, n varies from several to 2000.

Now it is straightforward to see that if $\{i_1, ..., i_r\}$ is an independent set in G', $\{I_{i_1}, ..., I_{i_r}\}$ is a pairwise disjoint set in $\{I_1, ..., I_n\}$. Consequently, to obtain a maximum pairwise disjoint set in $\{I_1, ..., I_n\}$, we can determine a maximum independent set in G'.

For this purpose, we can approximate the problem by applying directly our ATS algorithm described in Section 3.2 or use an exact algorithm. To make the decision, we implement the exact algorithm described in [26]. Experiments with this algorithm and comparisons with ATS lead to two clear observations. First, if G' is solved optimally by the exact algorithm, then ATS is able to find a solution (independent set) of the same size (thus an optimal solution). Second, many G' graphs cannot be solved by the exact algorithm. In fact, for most of the 11 tested graphs, there is always at least one G' that cannot be solved by the exact algorithm. For these graphs, ATS still provides approximate solutions. Based on this experiment, we decide to use our ATS to handle the underlying MSPP.

3.4 Coloring the residual graph

To color the residual graph, one can apply any graph coloring algorithm. For instance, the authors of [18] use a tabu search algorithm (TabuCOL). In [9], the residual graph is colored by a genetic local search procedure. In [20], independent sets are removed until the residual graph can be colored by a branch-and-bound algorithm. In this work, we employ MACOL which is a recent and effective Memetic Algorithm for graph coloring [23].

As a population-based hybrid algorithm, MACOL uses a tabu search procedure to ensure local optimization and an adaptive multi-parent crossover operator (AMPaX) to generate new solutions. AMPaX is an extension of the greedy partition crossover (GPX) presented in [11] and employs m ($m \geq 2$) parents for solution recombinations. The AMPaX operator builds one by one the color classes of an offspring solution by taking each time the largest class among the parents. To maintain population diversity, MACOL implements a distance-and-quality based replacement strategy for pool updating. The performance of MACOL on the DIMACS graphs is quite competitive compared to other state-of-art coloring algorithms. More details about MACOL can be found in [23].

4 Experimental Results

In this section, we assess the performance of the proposed EXTRACOL algorithm. For this purpose, we present computational results on the 11 largest benchmark graphs from the well-known DIMACS graph coloring Challenge [21]. Comparisons are also reported with respect to the underlying MACOL algorithm and 7 other top-performing coloring algorithms from the literature.

Test instances. As EXTRACOL is designed to color large graphs, we only consider graph instances with at least 1000 vertices. These graphs are known to be difficult and represent a real challenge for coloring algorithms. These graphs belong to four families [1] .

- Three large random graphs (DSJC1000.1, DSJC1000.5, DSJC1000.9). The first and second number in the name of each graph represent respectively the number of vertices and the edge density in the graph. The chromatic numbers of these graphs are unknown.
- Three large flat graphs (flat1000_50_0, flat1000_60_0, flat1000_76_0). They are structured graphs with known chromatic number (respectively 50, 60 and 76).
- Two large random geometric graphs (R1000.1c, R1000.5). These graphs are generated by picking random points (vertices) in a plane and by linking two points situated within a certain geometrical distance. The chromatic number is unknown for R1000.1c and is equal to 234 for R1000.5.
- Three very large random graphs (C2000.5, C2000.9, C4000.5). The chromatic numbers of these graphs are unknown. Given the size and difficulty of these graphs, they are not always used in computational experiments in the literature.

Parameter. Our EXTRACOL algorithm is programmed in C and compiled using GNU GCC on a PC with 2.8 GHz CPU and 2G RAM. To obtain our computational results, each instance is solved 20 times independently with different random seeds (the very large instances are solved 5 times). For each run, the time limit is set to 5 CPU hours except for the three very large graphs C2000.5, C2000.9 and C4000.5, for which a time limit of 120 hours (5 days) is allowed. Notice that these time conditions are comparable with those used in the most recent works like [13,24,28,23].

To run EXTRACOL, we need to fix q, the number of vertices left in the residual graph. We tested different values and choose the value 800 for all of our experiments. Let us notice that the choice of the value of q mainly depends on the performance of the coloring algorithm applied to the residual graph. In addition to q, MACOL requires also several parameters. In our case, we adopt those used in the original paper [23].

[1] `http://www.info.univ-angers.fr/pub/porumbel/graphs/index.html`.

Table 1
Computational results of EXTRACOL (and MACOL) on the set of 11 largest DIMACS benchmark graphs. The four improved results are indicated in bold.

Instance	n	*dense*	k^*	EXTRACOL[a]					MACOL [23]		
				k_{best}	*Succ*	$T_{extr}(m)$	*iter*	$T(m)$	k_{macol}	*Succ*	*iter*
DSJC1000.1	1000	0.10	20	20	20/20	11	3.1×10^7	93	20	20/20	3.5×10^7
DSJC1000.5	1000	0.50	83	83	20/20	3	2.0×10^8	132	83	20/20	2.2×10^8
DSJC1000.9	1000	0.90	223	**222**	3/20	8	5.4×10^8	258	223	18/20	4.5×10^8
R1000.1c	1000	0.97	98	101	18/20	10	6.4×10^5	18	98	20/20	7.5×10^5
R1000.5	1000	0.48	234	250	11/20	12	8.8×10^8	183	245	13/20	1.2×10^9
flat1000_50_0	1000	0.49	50	50	20/20	8	2.9×10^5	12	50	20/20	3.2×10^5
flat1000_60_0	1000	0.49	60	60	20/20	7	5.1×10^5	15	60	20/20	6.3×10^5
flat1000_76_0	1000	0.49	82	82	20/20	3	6.7×10^7	79	82	20/20	7.2×10^7
C2000.5	2000	0.50	148	**146**	5/5	145	1.7×10^8	253	148	1/5	8.2×10^8
C2000.9	2000	0.90	413[b]	**409**	2/5	118	4.5×10^8	329	413[b]	2/5	7.5×10^8
C4000.5	4000	0.50	271	**260**	4/5	5186	1.8×10^8	5298	272	3/5	1.2×10^9

a. See `http://www.info.univ-angers.fr/pub/hao/extracol.html` for the coloring result of each graph.
b. This result, which is not reported in the original paper [23], is obtained by running MACOL ourself.

4.2 Computational results

4.2.1 Comparison with the current best results

Our first experiment aims to evaluate EXTRACOL with the current best results on our benchmark graphs. Table 1 summarizes the computational statistics. Columns 2-3 give the features of the graphs: the number of vertices (n) and the density of each graph (*dense*). Column 4 shows the current best known result k^* reported in the literature. In columns 5-9, the computational statistics of our EXTRACOL algorithm are given, including the number of colors obtained (k_{best}), the success rate (*Succ*), the average computation time in minutes required by the preprocessing procedure (T_{extr}), the average number of iterations required by MACOL for coloring the residual graph (*iter*) and the total time required in minutes (T). The results of MACOL are also reproduced in Table 1 (columns 10-12) and we discuss these results in the next section.

From Table 1, we observe that, except for the two R1000 graphs, the results obtained by EXTRACOL are very competitive when compared to the current best known results reported in the literature.

For two of the three large random graphs (DSJC1000.1 and DSJC1000.5) and the three *flat* graphs (flat1000_x_0,x=50,60,76), EXTRACOL can easily reach the previous best known result within no more than 2.5 hours and with a success rate of 100%. More importantly, for DSJC1000.9, our EXTRACOL algorithm obtains for the first time a new 222-coloring, improving thus the previous best-known 223-coloring solution that has been obtained very recently in [24,32,28,23].

Table 2
Detailed information of EXTRACOL and MACOL on C4000.5 and C2000.5 regarding the number of independent sets (color classes) of different sizes.

Size of independent set	C2000.5		C4000.5	
	EXTRACOL	MACOL	EXTRACOL	MACOL
18	0	0	63	1
17	0	0	57	29
16	53	11	42	79
15	18	43	21	67
14	14	36	15	42
13	14	26	22	20
12	16	10	12	16
11	10	8	10	9
10	15	8	6	3
9	4	3	5	4
8	2	1	6	1
7	0	2	1	1
Total	146	148	260	272

For the three very large random graphs with 2000 and 4000 vertices, EXTRACOL shows even better performance. Indeed, the previous best known coloring requires respectively $k^* = 148$, 413 and 271 for these graphs, which have been found very recently in [28,23]. It is interesting to observe that EXTRACOL is able to color these graphs with $k = 146$, 409 and 260, leading to a gain of 2, 4 and 11 colors respectively. In Section 5.1, we show some elements to explain why EXTRACOL performs so well on these graphs.

However, for the two random geometric graphs (R1000.1c and R1000.5), our algorithm performs poorly. In Section 5.2, we show an analysis of EXTRACOL on these two graphs to try to understand why this happens.

4.2.2 Comparison between EXTRACOL and MACOL

Since EXTRACOL uses MACOL in its coloring phase to color the residual graphs, it is interesting to compare the results of EXTRACOL against those of MACOL. The results of MACOL without the preprocessing phase are listed in Table 1 (columns 10–12). Note that for both algorithms, the time limit is set to be 5 CPU hours for graphs of 1000 vertices and 5 days for graphs of 2000 and 4000 vertices.

As one can observe in Table 1, for the three very large graphs (C2000.5, C2000.9, C4000.5), EXTRACOL is able to find much better solutions than MACOL with respectively 2, 4 and 12 fewer colors. In order to get some insight about this difference, we show in Table 2 more detailed information about the computational results on C2000.5 and C4000.5. For each graph, columns 2-3 and columns 4-5 show the number of independent sets of size $|S|$ for the results obtained by EXTRACOL and MACOL respectively on these graphs. Now recall that to generate a new offspring solution, MACOL operates by transmitting large color classes (independent sets in legal k-colorings) from parents to offspring. However, from Table 2, we observe that for these two

graphs, it is very difficult for MACOL to generate very large color classes when it is directly applied to the initial graphs. For instance, if we consider C4000.5, we see that EXTRACOL obtains 63 color classes of size 18 while MACOL obtains only one class of this size. We can make the same remark on C2000.5 for which EXTRACOL and MACOL obtain respectively 53 and 11 color classes of size 16. More generally, for large graphs, it seems wise to use a dedicated algorithm (as EXTRACOL's preprocessing phase) to find large independent sets and remove them from the graph.

For DSJC1000.9, EXTRACOL improves on the result of MACOL with one fewer color. For the two random graphs (DSJC1000.x, x=1,5) and the three flat graphs (flat1000_x_0,x=50,60,76), EXTRACOL and MACOL achieve the same results in terms of the number of colors used. However, EXTRACOL requires less search effort and finds these solutions more quickly in terms of the number of iterations. This observation remains in fact valid for most of the tested graphs. This tends to show that EXTRACOL's preprocessing makes the residual graph easier to color.

Finally, for the two random geometric graphs (R1000.1c and R1000.5), the results of EXTRACOL are inferior to those obtained with MACOL. We investigate this phenomenon in Section 5.2.

4.2.3 Comparison with other algorithms

Now we compare EXTRACOL with 7 other state-of-art heuristics published in the literature [28,32,24,13,10,25,9]. One notices that all of them but one [10] are population-based hybrid algorithms.

For this comparison, we are mainly interested in solution quality in terms of the number of colors needed to color a graph. Table 3 presents the comparative results on the set of the 11 graphs. Columns 3 and 4 recall the previous best known k^* and the best results obtained by EXTRACOL (k_{best}). Columns 5-11 present the best results obtained by these reference algorithms.

From Table 3, we observe that recent hybrid algorithms like [28,32,24] show globally very good or excellent performance, in particular on graphs with no more than 1000 vertices. However, none of them can find a 222-coloring for DSJC1000.9 which is achieved by EXTRACOL. This difference seems even more pronounced when the three very large graphs are considered. Indeed, to color C2000.5 and C4000.5, the reference algorithms require at least 148 and 271 colors respectively with the best results obtained recently by Evo_Div [28] using a time limit of 5 and 30 days while EXTRACOL requires only 146 and 260 colors. For C2000.9 which is very dense (and hard), few results are available in the literature. On the other hand, we observe that EXTRACOL performs poorly on R1000.1c and R1000.5 with respect to most of the reference

Table 3
Comparison of EXTRACOL with 7 best performing coloring algorithms. Most of them are population-based hybrid algorithms.

Instance	n	k^*	k_{best}	7 reference graph coloring algorithms						
				[28]	[32]	[24]	[13]	[10]	[25]	[9]
DSJC1000.1	1000	20	20	20	-	20	20	21	-	-
DSJC1000.5	1000	83	83	83	84	83	84	88	89	84
DSJC1000.9	1000	223	**222**	223	223	225	224	228	-	-
R1000.1c	1000	98	101	98	-	98	-	98	98	99
R1000.5	1000	234	249	238	-	234	-	237	241	268
flat1000_50_0	1000	50	50	50	50	50	50	50	50	84
flat1000_60_0	1000	60	60	60	60	60	60	60	60	84
flat1000_76_0	1000	82	82	82	83	82	84	87	89	84
C2000.5	2000	148	**146**	148	150	-	-	162	165	153
C2000.9	2000	413	**409**	-	-	-	-	-	-	-
C4000.5	4000	271	**260**	271	-	-	-	301	-	280

algorithms.

5 Analysis and insights

5.1 Influence of preprocessing

The EXTRACOL's preprocessing phase uses a heuristic method to extract at each iteration as many pairwise disjoint independent sets as possible. Compared to the conventional preprocessing which extracts greedily independent sets one by one, our preprocessing is able to cover more vertices than the greedy extraction approach. In other words, in order to cover the same number of vertices (nearly $|V|-q$), our preprocessing needs fewer independent sets and thus fewer colors for these vertices.

To highlight the difference of these two preprocessing methods, we show a detailed comparison by considering two large graphs (C2000.5 and C4000.5). We recall that these two graphs are very difficult if they are directly colored without a preprocessing phase. Indeed, even the most recent hybrid algorithms can only find 148-coloring for C2000.5 and 271-coloring for C4000.5.

To solve these two instances, we apply both the conventional greedy preprocessing method and our preprocessing method to extract independent sets until there are at most 800 vertices left in the residual graph, which is then colored by MACOL. For both algorithms, each instance is solved 5 times with a time limit of 5 days per run. The results are summarized in Table 4 (C2000.5) and Table 5 (C4000.5). In the table, we show the results obtained respectively with our preprocessing (EXTRACOL) and the conventional greedy preprocessing method (OBOCOL). Columns 2 and 5 show, in the form of $x \times y$, the number y of independent sets of size x extracted by the preprocessing phase

Table 4
Effect of our preprocessing (EXTRACOL) and conventional preprocessing (OBOCOL) on C2000.5.

	EXTRACOL			OBOCOL		
	$\|S\|\times$No. of sets of $\|S\|$	No. of colored vertices	No. of colors used	$\|S\|\times$No. of sets of $\|S\|$	No. of colored vertices	No. of colors used
Independent sets obtained by preprocessing	16×53 15×18 14×6	1202	77	16×36 15×27 14×16	1205	79
Independent sets (color classes) from MACOL	14×8 13×14 12×16 11×10 10×15 9×4 8×2 7×0	798	69	14×5 13×20 12×14 11×13 10×6 9×7 8×3 7×1	795	69
Total		2000	146		2000	148

Table 5
Effect of the preprocessing (EXTRACOL) and conventional greedy preprocessing (OBOCOL) on C4000.5.

	EXTRACOL			OBOCOL		
	$\|S\|\times$No. of sets of $\|S\|$	No. of colored vertices	No. of colors used	$\|S\|\times$No. of sets of $\|S\|$	No. of colored vertices	No. of colors used
Independent sets obtained by preprocessing	18×63 17×57 16×42 15×21 14×8	3202	191	18×40 17×66 16×40 15×32 14×17	3200	195
Independent sets (color classes) from MACOL	14×7 13×22 12×12 11×10 10×6 9×5 8×6 7×1	798	69	14×7 13×20 12×11 11×16 10×5 9×5 8×4 7×1	800	69
Total		4000	260		4000	264

(upper part of each table) or established by MACOL (lower part of each table). Columns 3 and 6 give the number of vertices covered by the independent sets. Columns 4 and 7 indicate the number of these independent sets (color classes).

From Table 4, we observe that for C2000.5, our preprecessing identifies 77 independent sets (of sizes 16, 15, 14) covering 1202 vertices while the conventional preprecessing removes 1205 vertices, but using 79 colors. We notice also that our preprocessing extracts more independent sets of the largest size than the conventional preprecessing (53 against 36). For both residual graphs, they are colored by MACOL with 69 colors. This leads to a difference of 2 colors in favor of our preprocessing.

Similarly, from Table 5, we observe that for C4000.5, the maximal independent set that is identified is of size 18. The conventional preprocessing can only extract 40 independent sets of this size while our preprocessing extracts 63. Moreover, our preprocessing removes 191 large independent sets which

cover 3202 vertices, while the conventional preprocessing extracts 4 more independent sets (i.e. 195) which cover only 3200 vertices. To color both residual graphs, MACOL requires additional 69 colors. Once again, this leads to a better solution with our preprocessing method with 4 colors in less with respect to the conventional preprocessing.

Finally, if we compare the coloring results of Tables 4 and 5 with those reported in Table 3, we observe that for these two very large random graphs, even the conventional preprocessing method leads to better results than the direct coloring approach.

5.2 Limitation of preprocessing

As previously observed, EXTRACOL performs poorly on the two geometric graphs R1000.1c and R1000.5 although it shows excellent performance on all the other large graphs. It is then interesting to investigate what happens on these graphs. Clearly, it would be very difficult to provide a formal justification. Still, empirical observations would contribute to some extent to the understanding of EXTRACOL's counter-performance on these graphs.

Notice first that the geometric graphs are constructed in a special way [29]. For a graph R$N.d$, the set of N points (nodes) are randomly scattered in an 1 by 1 square. Two nodes are adjacent if their geometric distance is smaller than d. So contrary to standard random graphs, geometric graphs have special structures in terms of node degrees, independent sets and cliques. These structures may imply a particular relation between the number of color classes of a given size in an optimal coloring (call this number A) and the number of independent sets (potential color classes) of the same size existing in a graph (call this number B). To illustrate this point, we sampled a set of optimal 65-colorings for geometric graph r250.5. Inspecting these optimal solutions shows the number of color classes of size 6 (the largest possible size for this graph) within a 65-coloring varies between 2 and 6 while it is easy to extract 12 pairwise disjoint independent sets of this size, i.e. $A << B$. In other words, more than half of the extracted independent sets of size 6 are not part of an optimal coloring and using these independent sets as color classes would increase the number of needed colors with respect to the chromatic number.

To complement this explanation, we show in Table 6 some statistics obtained on R1000.5 with the preprocessing phase.

R1000.5 has a known chromatic number of 234. For this graph, it is quite easy for our ATS algorithm to find maximum cliques of size 234. Consequently, for the preprocessing to be helpful, each time an independent set is removed from the graph, the maximum clique size for the residual graph should decrease by

Table 6
Detailed results on R1000.5 showing the number of independent sets extracted (N_1), the maximum clique size of the residual graph (N_2), and the lower bound on the needed colors after the extraction ($N_3 = N_1 + N_2$).

size of independent set	EXTRACOL			OBOCOL		
	N_1	N_2	N_3	N_1	N_2	N_3
7	12	225	≥ 237	8	229	≥ 237
6	20	209	≥ 241	24	211	≥ 243

1. However, we observe from Table 6 that this is not always the case when the preprocessing is applied to this graph.

For instance, the first iteration of our preprocessing extracts 12 pairwise disjoint independent sets of size 7. However, one observes that at least 3 out of these 12 independent sets cannot be part of an optimal solution and are wrongly extracted. Indeed, the residual graph after extracting these 12 independent sets contains cliques of size 225, implying that we need at least 237 (12 plus 225) colors for the initial graph which is 3 colors above the chromatic number. The situation becomes even worse when 20 pairwise disjoint independent sets of size 6 are additionally extracted because the lower bound of the needed colors becomes now 241. The same observation can be made with the conventional greedy preprocessing which leads to even worse results (see OBOCOL results). We conclude that for this graph, some large independent sets are not part of any optimal 234-coloring and extracting such independent sets cannot help decrease the number of colors needed for the whole graph. The analysis realized on R1000.1c leads to the same conclusion.

This analysis shows the limit of the preprocessing approach which is basically due to its greedy nature. Indeed, if a mistake is made during the preprocessing phase, the mistake cannot be repaired. To remedy this difficulty, one possibility would be to allow the coloring procedure to integrate the extracted independent sets during its search.

6 Conclusion

In this paper, we revisited the graph coloring approach based on independent set extraction and proposed an effective algorithm (EXTRACOL) for coloring large graphs. The proposed preprocessing used by EXTRACOL distinguishes itself from the conventional greedy extraction methods. Instead of extracting one independent set each time, the proposed preprocessing method tries to extract many pairwise disjoint sets. Such a preprocessing maximizes the number of vertices covered by the extracted independent sets, hopefully making the residual graph easier to color. To identify large independent sets and

pairwise disjoint sets, we devised an adaptive Tabu Search algorithm which employs among other things a focused neighborhood and a dynamic tabu list management strategy.

The computational results obtained on the 11 largest DIMACS benchmark graphs with 1000, 2000 and 4000 vertices show, except for the two geometric random graphs, remarkable performance. In particular, for four very hard instances (DSJC1000.9, C2000.5, C2000.9, C4000.5), EXTRACOL is able to improves on the previous best known results reported in the literature by finding solutions with 222, 146, 409 and 260 colors respectively, implying a gain of 1, 2, 4 and 11 colors with respect to the current best colorings for these graphs.

We also presented detailed analyses of the preprocessing phase in hope of getting some insights about the advantages and limitations of the proposed approach. This approach could be further improved by allowing the coloring algorithm to reconsider the independent sets extracted during the preprocessing phase. Such an improvement can be for instance achieved within the multi-level optimization paradigm.

Acknowledgment

We are grateful to the referees for their comments and questions which helped us to improve the paper. The work is partially supported by the Pays de la Loire Region (France) within the RaDaPop (2009-2013) and LigeRO (2010-2013) projects.

References

[1] C. Avanthay, A. Hertz, N. Zufferey, A variable neighborhood search for graph coloring. European Journal of Operational Research 151(2)(2003) 379–388.

[2] I. Blöchliger, N. Zufferey, A graph coloring heuristic using partial solutions and a reactive tabu scheme. Computers and Operations Research 35(3) (2008) 960-975.

[3] E.K. Burke, B. McCollum, A. Meisels, S. Petrovic, R. Qu, A graph-based hyper heuristic for timetabling problems. European Journal of Operational Research 176 (2007) 177–192.

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

[5] M. Chiarandini, T. Stützle, An application of iterated local search to graph coloring. In: D.S. Johnson, A. Mehrotra, M. Trick, editors, Proc. of the Computational Symposium on Graph Coloring and its Generalizations, Ithaca, New York, USA, 112–125, 2002.

[6] D Costa, A Hertz, O Dubuis, Embedding of a sequential procedure within an evolutionary algorithm for coloring problem in graphs. Journal of Heuristics 1 (1995) 105-28.

[7] D. de Werra, C. Eisenbeis, S. Lelait, B. Marmol, On a graph-theoretical model for cyclic register allocation. Discrete Applied Mathematics 93(2-3) (1999) 191–203.

[8] R. Dorne, J.K. Hao, A new genetic local search algorithm for graph coloring. Lecture Notes in Computer Science 1498 (1998) 745–754.

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

[10] N. Funabiki, T. Higashino, A minimal-state processing search algorithm for graph coloring problems. IEICE Transaction Fundamentals E83-A (2000) 1420-1430.

[11] P. Galinier, J.K. Hao, Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization 3(4) (1999) 379-397.

[12] P. Galinier, A. Hertz, A survey of local search methods for graph coloring. Computers and Operations Research 33(9) (2006) 2547–2562.

[13] P. Galinier, A. Hertz, N. Zufferey, An adaptive memory algorithm for the K-colouring problem. Discrete Applied Mathematics 156(2) (2008) 267–279.

[14] M. Gamache, A. Hertz, J.O. Ouellet, A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. Computers and Operations Research 34(8) (2007) 2384–2395.

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

[16] M.R Garey, D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness. W.H. Freeman and Company, San Francisco, 1979.

[17] J.P. Hamiez, J.K. Hao, Scatter search for graph coloring, Lecture Notes in Computer Science 2310 (2002) 168–179, Springer-Verlag.

[18] A. Hertz, D. de Werra, Using tabu search techniques for graph coloring. Computing 39 (1987) 345-351.

[19] A. Hertz, M. Plumettaz, N. Zufferey, Variable space search for graph coloring. Discrete Applied Mathematics 156(13) (2008) 2551–2560.

[20] 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(3) (1991) 378-406.

[21] D.S. Johnson and M.A. Trick (Eds.), Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, volume 26 of DIMACS series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996.

[22] M. Laguna, R. Martí, A GRASP for coloring sparse graphs. Computational optimization and applications 19(2) (2001) 165–178.

[23] Z. Lü and J.K. Hao, A memetic algorithm for graph coloring. European Journal of Operational Research 200(1) (2010) 235-244.

[24] E. Malaguti, M. Monaci, P. Toth, A metaheuristic approach for the vertex coloring problem. INFORMS Journal on Computing 20(2) (2008) 302–316.

[25] C. Morgenstern, Distributed coloration neighborhood search. In [21], pages 335-357.

[26] P.R. Östergard, A fast algorithm for the maximum clique problem. Discrete Applied Mathematics 120(1–3) (2002) 197–207.

[27] D.C. Porumbel, J.K. Hao, P. Kuntz, A search space cartography for guiding graph coloring heuristics. Computers and Operations Research 37(4) (2010) 769–778.

[28] D.C. Porumbel, J.K. Hao, P. Kuntz, An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. Computers and Operations Research 37(10) (2010) 1822-1832.

[29] E.C. Sewell, An improved algorithm for exact graph coloring. In [21], pages 359–376.

[30] D.H. Smith, S. Hurley, S.U. Thiel, Improving heuristics for the frequency assignment problem. European Journal of Operational Research 107(1) (1998) 76–86.

[31] N. Zufferey, P. Amstutz, P. Giaccari, Graph colouring approaches for a satellite range scheduling problem. Journal of Scheduling 11(4) (2008) 263–277.

[32] X.F. Xie, J. Liu, Graph coloring by multiagent fusion search. Journal of Combinatorial Optimization 18(2) (2009) 99–123.