# A New Ant Algorithm for Graph Coloring

***Alain Hertz*[1] *and Nicolas Zufferey*[2]**

[1] Département de mathématiques et de génie industriel, École Polytechnique de Montréal, Canada, alain.hertz@gerad.ca

[2] Corresponding author, Département d'Opérations et Systèmes de Décision, Faculté des Sciences de l'Administration, Université Laval, Québec (QC), G1K 7P4, Canada, nicolas.zufferey@fsa.ulaval.ca

**Abstract.** *Let $G = (V, E)$ be a graph with vertex set $V$ and edge set $E$. The $k$-coloring problem is to assign a color (a number chosen in $\{1, \ldots, k\}$) to each vertex of $V$ so that no edge has both endpoints with the same color. We describe in this paper a new ant algorithm for the $k$-coloring problem. Computational experiments give evidence that our algorithm is competitive with the existing ant algorithms for this problem, while giving a minor role to each ant. Our algorithm is however not competitive with the best known coloring algorithms*

## 1 Introduction to graph coloring

The graph coloring problem (GCP for short) 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 *$k$-coloring* of $G$ is a function $col : V \longrightarrow \{1, \ldots, k\}$. The value $col(x)$ of a vertex $x$ is called the *color* of $x$. Vertices with a same color define a *color class*. If two adjacent vertices $x$ and $y$ have the same color, then vertices $x$ and $y$ are called *conflicting vertices* and the edge linking $x$ with $y$ is called a *conflicting edge*. A color class without conflicting edge is called a *stable set*. A $k$-coloring without conflicting edges is said to be *legal* and corresponds to a partition of the vertices into $k$ stable sets. The GCP is to determine the smallest integer $k$ (called the *chromatic number* of $G$) such that there exists a legal $k$-coloring of $G$. The GCP is NP-hard [15]. Exact solution methods can solve problems of relatively small size (no more than 100 vertices) [2], [3], [23], [18]. Upper bounds on the chromatic number can be obtained for larger instances by using heuristic algorithms.

The GCP has many practical applications such as the creation of timetables, frequency assignment, scheduling, design and operation of flexible manufacturing systems (e.g. [22], [14], [25]). As a result of its importance and simplicity, it is one of the most studied NP-hard problem, thus many methods have been developed to solve or to approach the GCP.

Given a fixed integer $k$, we consider the optimization problem, called $k$-GCP, which aims to determine a $k$-coloring 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 $k$-coloring. 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 $k$-coloring can be obtained. Many local search methods have been proposed to solve the $k$-GCP. For example, a tabu search is described in [19], and simulated annealing algorithms can be found in [5] and in [20]. Hybrid evolutionary heuristics have also been successfully applied to this problem (e.g., [7], [13], [16]). For a recent survey, the reader is referred to [17].

In this paper, we propose a new kind of ant algorithm for the GCP. Our main goal is to show that even if we give a minor role to each ant, we can still obtain competitive results in comparison with other ant heuristics for the GCP.

## 2 Ant algorithms and two existing adaptations to the GCP

Evolutionary heuristics encompass various algorithms such as genetic algorithms, scatter search, ant systems and adaptive memory algorithms [4]. They can be defined as iterative procedures that use a central memory where information is collected during the search process.

Ant colonies were introduced in [11] and in [8], and are derived from the observation of ants in the nature: ants are able to find a shortest path between a food source and the nest using a trail system (called pheromone). In these methods, the central memory is modeled by a trail system. In the usual *ant system*, a population of ants is used, where each ant is a constructive heuristic able to build a solution step by step. At each step, an ant adds an element to the current partial solution. Each decision or *move* $m$ is based on two ingredients: the greedy force (short term profit for the considered ant) $GF(m)$ and the trails $Tr(m)$ (information obtained from other ants). Let $M$ be the set of all the possible moves. The probability $p_k(m)$ that ant $k$ chooses move $m$ is given by

$$p_k(m) = \frac{GF(m)^\alpha \cdot Tr(m)^\beta}{\sum\limits_{m' \in M_k(adm)} GF(m')^\alpha \cdot Tr(m')^\beta},$$

where $\alpha$ and $\beta$ are parameters and $M_k(adm)$ is the set of admissible moves that ant $k$ can perform. When each ant of the population has build a solution, the trails can be updated for example as follows:

$$Tr(m) = \rho \cdot Tr(m) + \Delta Tr(m),\ \forall m \in M,$$

where $0 < \rho < 1$ is a parameter representing the evaporation of the trails, which is generally close or equal to 0.9, and $\Delta Tr(m)$ is a term which reinforces the trails left on move $m$ by the ant population. That quantity is usually proportional to the number of times the ants performed move $m$, and to the quality of the obtained solutions when move $m$ has been performed. More precisely, $\Delta Tr(m) = \sum_k \Delta Tr_k(m)$, where $\Delta Tr_k(m)$ is proportional to the quality of the obtained solutions when performing $m$. In some systems, the trails are updated more often (e.g. each time a single ant has built its solution [10]). In *hybrid ant systems*, the solutions provided by some ants may be improved using a local search technique. In the *max-min ant systems* [26], the authors proposed to normalize $GF(m)$ and $Tr(m)$ in order to better control these ingredients and thus the search process. An overview of ant algorithms can be found in [9].

Two ant heuristics for the GCP already exist in the literature. The first one was proposed in [6]. In their method, each ant is a constructive heuristic derived from *Dsatur* [2] or from

*RLF* [22]. Let us denote these methods by *ANT-DSATUR* and *ANT-RLF*, respectively. A move always consists in selecting a vertex and giving a color to it. The trail system is modeled using a matrix $Tr(x, y)$ proportional to:

- the number of times vertices $x$ and $y$ have the same color in the solutions provided by the ants;
- the quality of the solutions where $col(x) = col(y)$.

Another ant coloring method was recently proposed in [24], where each ant is a local search instead of a constructive heuristic. The authors consider (not necessarily legal) $k$-colorings and try to minimize the number of conflicts. The trail system is modeled, at iteration $t$, by the use of an auxiliary graph $G' = (V, E'(t))$ obtained from the original graph $G = (V, E)$ by adding edges. The edge set $E'(t)$ such that $E \subseteq E'(t)$ is updated at each iteration $t$ in the following way. If many ants give a different color to $x$ and $y$ such that edge $[x, y] \notin E$, then $[x, y]$ is added to $E'$. A move consists in changing the color of a conflicting vertex in $G$, while minimizing the number of conflicts associated with the auxiliary graph $G'$.

## 3 General approach and role of the ants

The new proposed ant coloring method differs from those in [6] and [24] in that the role of each single ant is not to build a whole solution but only to contribute to give a color to a single vertex. Also, on the contrary to the existing ant coloring algorithms, we only make one solution evolve using a population of ants. We deal with (not necessarily legal) $k$-colorings and try to minimize the number $f$ of conflicts. We always associate:

- a color in $\{1, \ldots, k\}$ to each ant;
- $k$ ants to each vertex.

Thus, we use $|V|$ ants of each color. From a distribution of the ants on the vertices, we deduce a coloring of the graph using a procedure, called *Color*, which is derived from *Dsatur* [2] which we now describe.

Let $A$ be a set of vertices which is initialized to $V$. At each step, *Dsatur* colors one vertex in $A$ as follows:

1. select the vertex $x \in A$ with the largest *saturation degree*, which is the number of different colors that are adjacent to $x$; if more than one vertex maximize the saturation degree, the vertex with the largest number of adjacent vertices is chosen (ties are broken randomly);
2. assign the smallest color to $x$ without creating any conflict, and remove $x$ from $A$.

Procedure *Color* differs from *Dsatur* as follows. Given a set $A$ of vertices to be colored, and given a $k$-coloring of the induced subgraph $G' = (V' = V - A, E')$ of $G$, the selection of the next vertex $x$ in $A$ to be colored is done as in *Dsatur*, but we assign to $x$ a color that is represented by at least one ant on $x$, and among these colors, we choose the one that minimizes the number of conflicts. If several colors are possible, we give to $x$ the color which is the most represented by the ants on $x$ (ties are broken randomly).

At the beginning of our method, we place one ant of each color on every vertex and we set $A = V$. Thus, the first iteration of our method corresponds to an application of *Dsatur*. Then, we iteratively modify the positions of the ants on the graph and, at the end of each iteration, we recolor a subset $A \subseteq V$ of vertices, using the *Color* procedure.

Note that one might propose, in the *Color* procedure, to always select the next vertex to color as the vertex $x$ in $A$ such that $x$ has the smallest number of colors represented by the ants on it. Suppose for example that only ants of color 2 are on $x$. In this case, it seems appropriate to color $x$ as soon as possible because we can only give color 2 to it. This strategy was tested but its performance was very poor. This is due to the fact that such a strategy is not able to diversify the search process. Indeed, a vertex with very few colors represented on it will always be colored first (or among the first ones), and the *Color* procedure will probably always assign the same color to it.

Let $a_i$ and $a_j$ be two ants with colors $i$ and $j$, respectively. Suppose that $a_i$ is on vertex $x$ and $a_j$ is on vertex $y$. A *move* $m = (x, i) \leftrightarrow (y, j)$ consists in exchanging the positions of $a_i$ and $a_j$ on vertices $x$ and $y$. Thus, $a_i$ is moved from $x$ to $y$ and $a_j$ is moved from $y$ to $x$. At each iteration $t$ of the algorithm, we modify the distribution of the ants on the graph by performing a sequence of such moves, and we reassign a color to some vertices by applying the *Color* procedure. We can already notice that each ant taken individually has an influence only on the vertex on which it is laying. This is not a significant role at all! At the end of each iteration $t$, it is important to reassign a color to:

1. all vertices $x$ with a different set of ants on it (because it is forbidden to give a color to $x$ which is not represented by at least one ant on $x$);
2. all conflicting vertices (because the goal of one iteration is to try to remove some conflicts).

Thus, at iteration $t$, we use the following set $A$ in the *Color* procedure (recall that $A$ is initialized to $V$):

$$\begin{aligned} A = \{ & \text{ vertices involved in a move at iteration } t \, \} \\ \cup \{ & \text{ conflicting vertices at iteration } t - 1 \, \}. \end{aligned}$$

An important issue is to define the set of moves to perform at iteration $t$ (for any fixed $t$). Let $N_l(x, t-1)$ be the number of ants of color $l$ on $x$ at iteration $t-1$. The goal of an iteration $t$ is to change the color of at least one randomly chosen conflicting vertex $x$. Suppose $x$ has color $i$ at iteration $t-1$, thus $N_i(x, t-1) > 0$ (otherwise the *Color* procedure would not be able to assign color $i$ to $x$ at the end of iteration $t-1$). In order to be sure that we change the color of $x$ using *Color* at the end of iteration $t$, we remove all ants of color $i$ from $x$, i.e. we perform a sequence of $N_i(x, t-1)$ moves $m$, chosen in the set

$$M(t) = \{m = (x, i) \leftrightarrow (y, j) \text{ such that } y \neq x,\ N_j(y, t-1) > 0,\ j \neq i\}.$$

Note that in order to reduce the risk of cycling, if we remove all the ants of color $i$ from vertex $x$, then we forbid to put an ant of color $i$ on $x$ during *tab* iterations, where $tab = \text{UNIFORM}(0, 9) + 0.6 \cdot NCV(s)$, where $s$ is the current solution and $NCV(s)$ is the number of conflicting vertices in $s$. The choice of such *tab* value is the one proposed in the efficient tabu search coloring algorithm described in [16], which is derived from the tabu search proposed in [19].

Another important issue is to define the way to select a move. As in most ant algorithms, we choose at iteration $t$ a move $m$ according to the greedy force $GF(m,t)$ and the trail $Tr(m,t)$. Given $\alpha$ and $\beta$, we propose to always choose the move $m$ that maximizes

$$p(m,t) = \alpha \cdot GF(m,t) + \beta \cdot Tr(m,t),$$

where $GF(m,t)$ and $Tr(m,t)$ are normalized in $[0;1]$. We use normalized quantities in order to better control the weights of these ingredients during the process, as in some *max-min ant systems* [26].

### 3.1 Definition of the greedy force

Recall that the greedy force represents the short-term profit of a single ant. At each iteration, the short term profit consists in removing some conflicts. In order to remove a conflict, we should remove some ant-conflicts, where an *ant-conflict* occurs when two ants of the same color $c$ are placed on two adjacent vertices $x$ and $y$. Remember that in this case, the *Color* procedure may give the same color $c$ to $x$ and $y$, because color $c$ is represented on $x$ and $y$. Thus, a move with a large greedy force value should have the potential to significantly reduce the number of ant-conflicts, and consequently the number of conflicts. Suppose we aim to change the current color $i$ of vertex $x$. We thus have to remove all the ants of color $i$ from vertex $x$. In order to do that, we perform a sequence of moves of type $m = (x,i) \leftrightarrow (y,j)$. For such a move $m$, we define the greedy force $GF(m,t)$ by setting

$$GF(m,t) = Adv(m,t) - Disadv(m,t),$$

where $Adv(m,t)$ and $Disadv(m,t)$ are respectively the advantage and disadvantage of performing move $m$ (i.e. exchanging ants $a_i$ and $a_j$) at iteration $t$. Note that we always normalize $Adv$ and $Disadv$ in interval $[0;1]$.

Let $S(x,y,i,t)$ be the number of ants of color $i$ on vertices, different from $y$, which are adjacent to $x$. An ant $a_i$ placed on vertex $x$ is attracted by vertex $y$ if:

1. there are several ants of color $i$ on $y$;
2. there are several ants of color $i$ on vertices, different from $y$, which are adjacent to $x$.

Similar considerations hold for the ant $a_j$ placed on vertex $y$ which is candidate to be placed on vertex $x$ instead of $a_i$. Thus, we can set

$$\begin{aligned} Adv(m,t) = {} & N_i^2(y,t-1) + S(x,y,i,t-1) \\ & + N_j^2(x,t-1) + S(y,x,j,t-1). \end{aligned}$$

Note that we respectively use $N_i^2(y,t-1)$ and $N_j^2(x,t-1)$ instead of $N_i(y,t-1)$ and $N_j(x,t-1)$ in order to give more importance to the fact that ants with the same color should be grouped together. In addition, preliminary experiments showed us that such a formula leads to better results.

An ant $a_j$ placed on vertex $y$ is not attracted by vertex $x$ if:

1. there are several ants of color $j$ on $y$;
2. there are several ants of color $j$ on vertices, different from $y$, which are adjacent to $x$.

As we know that all the ants of color $i$ will be removed from $x$, vertex $x$ is not attracted by vertex $y$ only if there are several ants of color $i$ on vertices, different from $x$, which are adjacent to $y$. Thus, we can set

$$Disadv(m,t) = N_j^2(y,t-1) + S(x,y,j,t-1) + S(y,x,i,t-1).$$

## 3.2 Definition of the trail system

We first define the way to update the trail left by ants of color $c$ on vertex $v$ at the end of iteration $t$:

$$tr(v,c,t) = \rho \cdot tr(v,c,t-1) + \Delta tr(v,c,t),$$

where $0 < \rho < 1$ is an evaporation parameter and the reinforcement value of color $c$ on vertex $v$ is $\Delta tr(v,c,t)$. Note that evaporation and reinforcement may simultaneously occur, as in any ant system.

We consider several cases. Suppose that during iteration $t$, the following events occurred:

1. $x$ loses $p = N_i(x,t-1)$ ants of color $i$;
2. we put $p$ ants on $x$, and the $p$ ants have colors in the subset $C(x)$ of $\{1,\dots,k\}$;
3. the $p$ new ants on $x$ came from a subset of vertices $P(x)$ of $V$.

A good trail system has to incorporate the following information:

1. if an ant of color $c$ leaves vertex $v$, then $tr(v,c,t)$ should be evaporated (this is especially true for $v = x$ and $c = i$) and not reinforced;
2. if no ant of color $c$ leaves a vertex $v \in P(x)$, then $tr(v,c,t)$ should be evaporated and slightly reinforced;
3. if an ant of color $c$ arrives on vertex $v$, then $tr(v,c,t)$ should be evaporated and reinforced, and the reinforcement should be especially large if the generated solution $s'$ from $s$ is better than $s$, or better than $s^*$, which is the best solution visited so far during the search process;
4. if nothing occurs relatively to a vertex $v$ and a color $c$, then $tr(v,c,t)$ should be slightly evaporated and not reinforced.

Preliminary experiments showed that the following parameter setting is appropriate.

- $\rho = 0.9$ and $\Delta tr(v,c,t) = 2 \cdot \delta$ if $v = x, c \in C(x)$;
- $\rho = 0.8$ and $\Delta tr(v,c,t) = 0$ if $v = x, c = 1$;
- $\rho = 0.9$ and $\Delta tr(v,c,t) = 0$ if $v = x, c \notin C(x) \cup \{i\}$;
- $\rho = 0.9$ and $\Delta tr(v,c,t) = 0$ if $v \in P(x), c$ such that there is an ant $a_c$ which left $v$;
- $\rho = 0.9$ and $\Delta tr(v,c,t) = \delta$ if $v \in P(x)$ and $c = i$;
- $\rho = 0.9$ and $\Delta tr(v,c,t) = \frac{\delta}{2}$ if $v \in P(x)$ and $c$ such that there is no ant $a_c$ which left $v$;
- $\rho = 0.99$ and $\Delta tr(v,c,t) = 0$ if $v \notin \{x\} \cup P(x)$ and $c \in \{1,\dots,k\}$;

$$\text{where } \delta = \begin{cases} 0.1 \text{ if } f(s) - f(s') \geq 0 \\ 0.2 \text{ if } f(s) - f(s') < 0 \\ 0.4 \text{ if } f(s) - f(s^*) < 0 \end{cases}$$

Remember that $f(s)$ is the number of conflicts in solution $s$. We can now define the trail of a move $m$ at iteration $t$ as follows:

$$Tr[m = (x,i) \leftrightarrow (y,j), t] = tr(x,j,t) + tr(y,i,t) - tr(x,i,t) - tr(y,j,t).$$

Note that we initialize $tr(x,j,0) = 1$, $\forall x \in \{1,\ldots,n\}$, $\forall j \in \{1,\ldots,k\}$, and we always normalize the $tr$ values in interval $[0;1]$.

# 4 General algorithm

We have now all the ingredients necessary to formulate a new ant heuristic for the $k$-GCP. The main loop of our method, called $ANTCOL$, stops when a maximum number $MaxIter$ of iterations without improvement of the best solution $s^*$ encountered so far have been performed. The algorithm is the following:

1. set $t = 0$;
2. place one ant of each color on each vertex;
3. initialize the trails of each color on each vertex to 1;
4. apply the $Color$ procedure with $A = V$; let $s$ be the so obtained solution;
5. set $t = 1$, $t' = 1$, $A = V$, $f^* = f(s)$, and $s^* = s$;
6. **while** $t' < MaxIter$ **and** $f^* > 0$**, do**
   (a) determine the set $M(t)$ of the possible non tabu moves for iteration $t$;
   (b) compute the greedy force $GF(m,t)$ and the trail $Tr(m,t)$, $\forall \in M(t)$;
   (c) normalize the greedy forces and the trails in $[0,1]$;
   (d) perform the non tabu move $m \in M(t)$ maximizing $p(M,t)$; let $m = (x,i) \leftrightarrow (y,j)$ be such a move;
   (e) while there is at least one ant of color $i$ on $x$, perform the move $m$ maximizing $p(M,t)$ among the moves in $\{m = (x,i) \leftrightarrow (y,j) \in M(t) \mid y \neq x$ and color $j$ is represented on $y\}$;
   (f) update the trails $tr(v,c)$ for each vertex $v$ and each color $c$, then normalize those quantities;
   (g) update the tabu status;
   (h) set $A = \{$ vertices involved in a move performed at iteration $t\,\} \cup \{$ conflicting vertices at iteration $t-1\,\}$;
   (i) update the colors of the vertices in $A$ using the $Color$ procedure; let $s$ be the so obtained solution;
   (j) if $f(s) < f^*$, set $s^* = s$, $f^* = f(s)$, and $t' = -1$;
   (k) set $t = t+1$ and $t' = t'+1$;

Note that preliminary experiments showed that the use of $MaxIter = 5000$ is appropriate: the method is usually not able to improve the solution $s^*$ further if we use a larger value. Also, preliminary experiments showed that $\alpha = 1$ and $\beta = 5$ are appropriate values in the following formula: $p(m,t) = \alpha \cdot GF(m,t) + \beta \cdot Tr(m,t)$. Thus, as we use normalized values for the trails and the greedy forces, it is better to give more importance to the trails. The reader interested in having more details on the parameter settings is referred to [27].

## 5 Obtained results and conclusion

We will compare our ant coloring algorithm with some other coloring heuristics. We choose to consider the following methods:

- *Dsatur* [2] which is a constructive algorithm,
- the ant algorithms *ANT-DSATUR* and *ANT-RLF* [6] (see Section 2);
- *Tabucol*, which is a very well-known tabu search algorithm originally proposed in [19] and improved in [16];
- *GH*, which is an hybrid genetic algorithm proposed in [16], and is now widely considered as the best coloring heuristic; such a method uses *Tabucol* as intensification procedure.

Note that the ant coloring heuristic proposed in [24] will not be considered because of the following reasons. First, the authors mainly tested their method on very easy instances, namely *le*450_5*a*, *le*450_5*b*, *le*450_5*c*, *le*450_5*d*, *le*450_15*a* and *le*450_15*b*, which are well-known benchmark instances [21]. But it is known that these instances can be optimally colored by exact algorithms within a few seconds! Then, they tested their algorithm on non standard random graphs they generated with a very small number of edges. However, no results is given for standard random graphs, and we have not been able to re-implement their algorithm. Finally, notice that in their method, each ant is a whole local search coloring heuristic, which means that, on the contrary to our method, an ant has a very significant role. Consequently, it is not relevant to compare their hybrid algorithm with our method. However, we compare our heuristic with another type of local search, namely *Tabucol*, which is considered as one of the most simple and efficient local search coloring algorithm [17].

The *density* of a graph is the average number of edges between two vertices. We compare the above methods on several random graphs of size $|V| \in \{100, 300, 500, 1000\}$ and density $d = 0.5$. These graphs are obtained by linking a pair of vertices by an edge with probability 0.5, independently for each pair. It is known in the graph coloring community that random graphs with $d = 0.5$ are hard to color. While the results obtained on these graphs are representative, the reader interested in having more details on the results obtained by *ANTCOL* and many other coloring heuristics on other instances is referred to [27].

All the experiments were done on a computer *Silicon Graphics Indigo2 (195 MHz, IP28 processor)*. We will not present the detailed CPU times for each graph because of their large variability. However, in order to give an idea of these CPU times, we mention that the time needed by *ANTCOL* to find a legal coloring varies from a few seconds to a few minutes if $|V| = 100$, from a few minutes to an hour if $|V| = 300$, from one to three hours if $|V| = 500$, and from three to eight hours if $|V| = 1000$. These computing times are comparable with the ones needed by the other methods. Once again, the reader interested in getting more detailed is referred to [27].

The results associated with *ANT-DSATUR* and *ANT-RLF* are taken from [6]. For the other methods, we generated four graphs for each considered value of $|V|$ and performed four runs on each graph. The results are summarized in Table 1. For each size $|V|$, we give the "most likely" number of colors used by each method to generate legal colorings. By "most likely", we mean the number of colors for which the success rate was at least 50%. In brackets, we mention the smallest number of colors found by each method on at least one graph. For example, for $|V| = 300$, $ANTCOL$ will most likely build legal 39-colorings, and will sometimes generate legal colorings using only 37 colors.

| $\|V\|$ | ANT-RLF | ANT-DSATUR | DSATUR | Tabucol | GH | ANTCOL |
|---|---|---|---|---|---|---|
| 100 | 16 (15) | 16 (15) | 19 (18) | 14 | 14 | 16 (16) |
| 300 | 36 (35) | 39 (38) | 43 (42) | 33 | 33 | 39 (37) |
| 500 | 56 (55) | 68 (67) | 66 (65) | 49 | 48 | 59 (57) |
| 1000 | 111 (111) | 121 (121) | 115 (114) | 89 | 84 | 106 (105) |

Table 1: Comparison of $ANTCOL$ with five other heuristics

We can first observe that $ANTCOL$ is much better than $Dsatur$. However, both algorithms are based on the same coloring strategy, which is to iteratively select a vertex with a large saturation degree, and to assign to it the best possible color. This means that the ingredients we add to $Dsatur$ in order to elaborate $ANTCOL$ are useful. The same conclusion holds when comparing $Dsatur$ with *ANT-DSATUR*

Then, we can remark that as soon as $|V| > 100$, $ANTCOL$ is much better than *ANT-DSATUR*, which is another ant algorithm also based on $Dsatur$. This probably indicates that it is better to make one solution evolve instead of performing several multi-starts. The same kind of remark holds if we compare $ANTCOL$ with *ANT-RLF* on graphs with $|V| = 1000$.

Finally, we see that $ANTCOL$ is not competitive at all with $Tabucol$, and thus with $GH$ which uses $Tabucol$ as intensification procedure. We think that this is mainly due to the fact that in $ANTCOL$, too many ingredients are used to define the greedy force and the trail system. Thus, the algorithm is very slow because lots of computation is needed to only change the color of a small number of vertices. This kind of conclusion also holds for the two other existing ant heuristics for the GCP [6], [24].

Remember that in [6], an ant is a constructive heuristic, and in [24], an ant is a local search heuristic. But in contrast, in $ANTCOL$, each individual ant only helps giving a color to a single vertex, which is not a significant role. Hence, we have shown that we can obtain competitive results in comparison with other ant coloring methods by giving a minor role to each ant.

## References

1. Bloechliger, I., and Zufferey, N. 2004. A Reactive Tabu Search using Partial Solutions for the Graph Coloring Problem, Technical Report, Institute of Mathematics, Swiss Federal Institute of Technology, Lausanne
2. Brélaz, D. 1979. New Methods to Color Vertices of a Graph, *Communications of ACM* ***22***: 251–256
3. Brown, J.R. 1972. Chromatic Scheduling and the Chromatic Number Problem, *Management Science* ***19***: 456–463
4. Calegari, P., Coray, C., Hertz, A., Kobler, D., and Kuonen, P. 1999. A Taxonomy of Evolutionary Algorithms in Combinatorial Optimization, *Journal of Heuristics* ***5***: 145–158

5. Chams, M., Hertz, A., and de Werra, D. 1987. Some Experiments with Simulated Annealing for Coloring Graphs, *European Journal of Operational Research* ***32***: 260–266
6. Costa, D., and Hertz, A. 1997. Ants can colour graphs, *Journal of the Operational Research Society* ***48***: 295–305
7. Costa, D., Hertz, A., and Dubuis, O. 1995. Embedding of a Sequential Algorithm within an Evolutionary Algorithm for Coloring Problems in Graphs, *Journal of Heuristics 1* ***1***: 105–128
8. Dorigo, M. 1992. Optimization, learning and natural algorithms (in Italian), Unpublished doctoral dissertation, Politecnico di Milano, Dipartimento di Elettronica, Italy
9. Dorigo, M., Di Caro, G., and Gambardella, L.M. 1999. Ant algorithms for discrete optimization, *Artificial Life* ***5***: 137–172
10. Dorigo, M., and Gambardella, L.M. 1997. Ant Colony System: a Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Transactions on Evolutionary Computation 1* ***1***: 53–66
11. Dorigo, M. Maniezzo, V., and Colorni A. 1991. Positive feedback as a search strategy, Technical Report 91-016, Politecnico di Milano, Dipartimento di Elettronica, Italy
12. Dorigo, M. Maniezzo, V., and Colorni A. 1996. The Ant System: Optimization by a Colony of Cooperating Agents, *IEEE Transactions on Systems, Man, and Cybernetics 26* ***1***: 1–13
13. Fleurent, C., and Ferland, J. A. 1996. Genetic and Hybrid Algorithms for Graph Coloring, *Annals of Operations Research* ***63***: 437–461
14. Gamst, A. and Rave, W. 1992. On the frequency assignment in mobile automatic telephone systems, Proceedings of GLOBECOM'92
15. Garey, M., and Johnson, D. 1979. Computers and Intractability: a Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York
16. Galinier, P., and Hao, J.K. 1999. Hybrid Evolutionary Algorithms for Graph Coloring, *Journal of Combinatorial Optimization* ***3***: 379–397
17. Galinier, P., and Hertz, A. 2006. A Survey of Local Seach Methods for Graph Coloring, *Computers & Operations Research*, to appear.
18. Herrmann, F., and Hertz, A. 2002. Finding the chromatic number by means of critical graphs, *ACM Journal of Experimental Algorithmics* ***7/10***: 1–9
19. Hertz, A., and de Werra, D. 1987. Using Tabu Search Techniques for Graph Coloring, *Computing* ***39***: 345–351
20. Johnson, D. S, Aragon, C. R., McGeoch, L. A., and Schevon, C. 1991. Optimization by Simulated Annealing: An Experimental Evaluation, Part II; Graph Coloring and Number Partitioning, *Operations Research* ***39***: 378–406
21. Johnson, D. S., and Trick, M. A. 1996. Proceedings of the 2nd DIMACS implementation challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Sciences, *American Mathematical Society* ***26***
22. Leighton, F. T. 1979. A graph coloring algorithm for large scheduling problems, *Journal of Research of the National Bureau Standard* ***84***: 489–505
23. Peemöller, J. 1983. A Correction to Brélaz's Modification of Brown's Coloring Algorithm, *Communications of ACM* ***26***: 593–597
24. Shawe-Taylor, J., Zerovnik, J. 2002. Ants and Graph Coloring, Proceedings of ICANNGA'01: 593–597
25. Stecke, K. 1985. Design planning, scheduling and control problems of flexible manufacturing, *Annals of Operations Research* ***3***: 3–12
26. Stuetzle, T., and Hoos, H. 1997. Improving the Ant System: a Detailed Report on the Max-Min Ant System, Technical Report, Department of Computer Sciences - Intellectics Group, Technical University of Darmstadt
27. Zufferey, N. 2002. Heuristiques pour les Problèmes de la Coloration des Sommets d'un Graphe et d'Affectation de Fréquences avec Polarités, PhD. Thesis, École Polytechnique Fédérale de Lausanne, Switzerland