Ant Local Search and its Efficient Adaptation to Graph Colouring

Matthieu Plumettaz[1], David Schindl[2] and Nicolas Zufferey[3]

Abstract

In this paper, we propose a new kind of ant algorithm called *Ant Local Search*. In most ant algorithms, the role of each ant is to build a solution in a constructive way. In contrast, we propose to consider each ant as a local search, where at each step and in concordance with all ant algorithms, each ant modifies the current solution by the use of the greedy force and the trail systems. We also propose ways to reduce the computational effort associated with each ant decision. Such a new and general ant methodology is then applied to the well-known k-colouring problem, which is NP-hard. Computational experiments give evidence that our algorithm is competitive with the best colouring methods.

Keywords: heuristics, networks and graphs, artificial life, optimization.

Introduction

Local search heuristics operate in a *search space* S, also called a *solution space*. The elements of this space are called *solutions* even if not all elements are solutions to the initial problem. For every solution $s \in S$, a neighbourhood $N(s) \subset S$ is defined. A local search method starts at an initial solution, and then moves repeatedly from the current solution to a neighbour solution in order

[1] *Department of Industrial Engineering and Operations Research*, Columbia University, New York, United States, mp2761@columbia.edu

[2] Geneva School of Business Administration (HEG), Geneva, Switzerland, david.schindl@hesge.ch

[3] **Corresponding author,** *Faculty of Economics and Social Sciences*, HEC - University of Geneva, Uni-Mail, 1211 Geneva 4, Switzerland, nicolas.zufferey@fsa.ulaval.ca

to try to find better solutions, measured by an appropriate objective function. The passage from one solution to the next is called a *move*. State-of-the-art local search methods are simulated annealing, tabu search and variable neighbourhood search.

Ant colonies were introduced in (Dorigo *et al.*, 1991) and in (Dorigo, 1992), and are derived from the observation of ants in the nature. In these methods, a 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. Often, in order to get competitive results, it is unavoidable to apply a local search method, such as tabu search, to the solutions provided by such constructive ants (Dorigo & Stuetzle, 2002). In such a case, ant colonies and tabu search are successively but not simultaneously used. In contrast, we propose in this paper a new and general ant methodology, called Ant Local Search, where each ant is considered as a local search. At each iteration of Ant Local Search, and as it is the case in all ant algorithms, each decision is based on two elements: the greedy force and the trails.

The proposed Ant Local Search method will be tested on the famous graph colouring problem, which consists in assigning a color to each vertex of the considered graph, such that any two adjacent vertices have different colors, while minimising the number of colors. In such a context, each ant will be a tabu search derived from PartialCol (Bloechliger & Zufferey, 2008), which is a state-of-the-art colouring heuristic. The greedy force will be derived from the objective function used in PartialCol, and the trail system will be built on the following idea: if the ants give the same colour to two vertices, such an information should be transmitted to the ants of the next generations.

The paper is organised as follows. We first briefly describe tabu search and the classical ant algorithms. Secondly, we formally present the Ant Local Search

method, which can be adapted to any optimization problem. Thirdly, we describe the graph colouring problem and mention some well-known methods to solve it. Then, we adapt the proposed Ant Local Search algorithm to the graph colouring problem. Finally, computational experiments are reported and we end up the paper with a general conclusion.

Tabu search and classical ant algorithms

A basic version of tabu search can be described as follows. Let f be an objective function which has to be minimised over the solution space S. First, the method needs an initial solution $s_0 \in S$ as input. Then, the algorithm generates a sequence of solutions $s_1, s_2, \ldots$ in the search space S such that $s_{r+1} \in N(s_r)$. When a move is performed from s_r to s_{r+1}, the inverse of that move is stored in a *tabu list* L. For the following t iterations, where t is the *tabu tenure* (also called *tabu list length*), a move stays *tabu* and cannot be used (with some exceptions) to generate a neighbour solution. The solution s_{r+1} is computed as $s_{r+1} = \arg \min_{s \in N'(s_r)} f(s)$, where $N'(s_r)$ is a subset of $N(s_r)$ containing all solutions s' which can be obtained from s_r either by performing a move that is not in L (i.e. not tabu) or such that $f(s') < f(s^\star)$, where $s^\star$ is the best solution encountered along the search so far. The process is stopped for example when an optimal solution is found (when it is known), or when a fixed number of iterations have been performed. Many variants and extensions of this basic algorithm can be found for example in (Glover & Laguna, 1997).

In most ant algorithms (Dorigo & Stuetzle, 2002) (also called *Ant Colony Optimization*), the role of each ant is 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 $GF(m)$ (short term profit for the considered ant) and the trails $Tr(m)$ (information obtained from other ants). Let M be the set of all possible decisions. The probability $p_i(m)$ that

ant i chooses decision m is given by $p_i(m) = \frac{GF(m)^\alpha \cdot Tr(m)^\beta}{\sum_{m' \in M_i(adm)} GF(m')^\alpha \cdot Tr(m')^\beta}$, where α and β are parameters and $M_i(adm)$ is the set of admissible decisions that ant i can perform. When each ant of the population has built a solution, the trails are generally updated 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 usually close or equal to 0.9, and $\Delta Tr(m)$ is a term which reinforces the trails left on decision m by the ant population. That quantity is usually proportional to the number of times the ants performed decision m, and to the quality of the obtained solutions when decision m was performed. More precisely, let N be the number of ants, then: $\Delta Tr(m) = \sum_{i=1}^{N} \Delta Tr_i(m)$, where $\Delta Tr_i(m)$ is proportional to the quality of the solution provided by ant i if it has performed decision m. Note that in some *elitist ant algorithms*, the summation is only performed over the $N^\star$ best ants, where $N^\star \leq N$ is a parameter. In some systems, the trails are updated more often (e.g. each time a single ant has built its solution (Dorigo & Gambardella, 1997)). In *hybrid ant systems*, the solutions provided by some ants may be improved using a local search technique. In the *max-min ant systems* (Stuetzle & Hoos, 1997), the authors proposed to normalise $GF(m)$ and $Tr(m)$ in order to better control these ingredients and thus the search process. Recent overviews of ant algorithms can be found in (Blum, 2005) and (Dorigo *et al.*, 2006).

The Ant Local Search algorithm

As previously mentioned, in most ant algorithms, the role of each ant is to build a solution from scratch. At each step of the construction, the considered ant takes a decision according to the greedy force and the trails. Often, in order to get competitive results, it is unavoidable to apply a local search method (e.g. a descent method, tabu search, simulated annealing) to the solutions provided by such constructive ants (Dorigo & Stuetzle, 2002). In contrast, we propose to give a more important role to each ant by considering each of them as a local search.

More precisely, at each step of the local search and as in every ant algorithm, the considered ant takes a decision according to the greedy force and the trails. The resulting general method is called Ant Local Search and is summarised in Algorithm 1, where N is the number of used ants. The main difference between the proposed heuristic and the classical ant algorithms relies in step (1.a): in our algorithm, each ant makes one solution evolve by performing an arbitrary number of modifications on it, instead of only building step by step a single solution from scratch. Notice that the idea of considering each ant as a local search first appears in (Zufferey, 2002) and (Shawe-Taylor & Zerovnik, 2002), where there is however no generalization of their ant colouring methods which could lead to a new and general Ant Local Search algorithm able to tackle other optimization problems.

Algorithm 1 Ant Local Search Algorithm

While no stopping condition is met, **do**:

1. **for** $i = 1$ **to** N, **do**:
   (a) apply the local search associated with ant i;
   let s_i be the resulting solution;
   (b) *locally* update the trails by the use of s_i (optional);
2. *globally* update the trails by the use of a subset of $\{s_1, \ldots, s_N\}$;
3. perform some *actions* that can not be performed by the ant system (optional);

We can remark that such an algorithm could be easily parallelised by the use of N processors, where each processor would perform the job of a single ant. A *generation* consists in performing steps (1) to (3). Thus, a stopping condition can be a maximum number of generations or a maximum time limit. Note that in step (3), local search can for example be applied on some solutions provided by the ants of the current generation, or a specific diversification/intensification mechanism can be triggered. Such a heuristic can of course be adapted to any optimization problems. In order to do it, we have to:

- choose the nature of the local search associated with each ant (e.g. tabu search, simulated annealing, hill climbing) and a stopping condition for the chosen local search (e.g. a maximum number of iterations);
- define the neighbourhood structure(s) used within the local search, i.e. the nature of the moves which can be performed during the local search;
- define the key parameters of the considered local search; for example, if each ant is a tabu search procedure, we have to define the tabu list structure and the way to manage it;
- define the greedy force $GF(m)$ and the trail $Tr(m)$ of any move m;
- define the way to use the greedy force and the trails in order to select a move at each iteration of the local search;
- define a way to update the trails: this is always performed at the end of each generation (global update of the trails), and optionally each time a single ant provides its solution (local update of the trails).

All these elements will be defined below for the k-colouring problem. However, we would like to mention here that in most ant algorithms, it is very time consuming to make a single decision according to the probability distribution. Therefore, we propose the following general way to select a move based on the greedy forces and the trails. At each iteration, let A be the set of moves with the largest greedy force (resp. trail) value. Then, the selected and performed move is the one in A with the largest trail (resp. greedy force) value (we break ties randomly). Of course, this process is only interesting if $\mid A \mid > 1$ (i.e. if we can have an interesting number of ties), otherwise the trails (resp. greedy forces) will have no impact on the heuristic. Such a way of selecting each move at each iteration leads to several advantages over most classical ant algorithms: it is not required anymore to: (1) compute the trails (resp. greedy forces) of all possible moves; (2) normalise the greedy forces and the trails of the possible moves; (3) compute the probability $p_i(m)$ associated with each possible move m; (4) consider the parameters α and β. In other words, in contrast with most

classical ant algorithms, we successively (instead of jointly) use the greedy forces and the trails to make a decision. Therefore, a lot of computing time is saved, and the tuning phase of the heuristic is significantly reduced. We would like to mention that this is only a possible way to reduce the computing time needed for an ant to make a decision. One can of course propose other ways to manage the selection of a decision.

The graph colouring problem

The graph colouring 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-colouring* of G is a function $col : V \longrightarrow \{1, \ldots, k\}$. The value $col(x)$ 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 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 and denoted by $\chi(G)$) such that there exists a legal k-colouring of G. The GCP is NP-hard (Garey & Johnson, 1979). Currently, no known exact solution method is able to solve all instances with up to 100 vertices (e.g. (Mehrotra & Trick, 1996) and (Herrmann & Hertz, 2002)). For larger instances, upper bounds on the chromatic number can be obtained by using heuristic algorithms. The GCP and the k-GCP (described below) have many practical applications such as the creation of timetables, frequency assignment, scheduling, design and operation of flexible manufacturing systems, bag rationalisation for food manufacturers (e.g. (Leighton, 1979), (Stecke, 1985), (Gamst & Rave, 1992), (Glass, 2002), (Zufferey *et al.*, 2008)).

Given a fixed integer k, we consider the optimization problem, called k-GCP,

which aims to determine a legal k-colouring of G by minimizing the number of conflicting edges in the k-colouring solution space. If the optimal value of the k-GCP is zero, this means that G has a legal k-colouring. Starting at most with $k = |V|$, an upper bound on the chromatic number of G can be determined by solving a series of k-GCPs with decreasing values of k until no legal k-colouring can be obtained. Many local search methods were proposed to solve the k-GCP: tabu search ((Hertz & de Werra, 1987), (Bloechliger & Zufferey, 2008)), simulated annealing ((Chams *et al.*, 1987), (Johnson *et al.*, 1991)), and variable space search ((Hertz *et al.*, 2008)). In (Glass & Prugel-Bennett, 2005), the authors proposed, for the GCP, a first adaptation of a new kind of local search where a polynomial time sub-procedure is used for finding an optimal neighbour in an exponential sized neighborhood. Hybrid evolutionary heuristics were also successfully applied to this problem (e.g., (Fleurent & Ferland, 1996), (Galinier & Hao, 1999)). For a recent survey, the reader is referred to (Galinier & Hertz, 2006). Note that some ant heuristics for the GCP already exist in the literature ((Costa & Hertz, 1997), (Shawe-Taylor & Zerovnik, 2002), (Hertz & Zufferey, 2006)). Their performances are however very poor when compared with the best colouring algorithms. We describe these ant colouring algorithms below.

The first ant colouring algorithm was proposed in (Costa & Hertz, 1997). In their method, each ant is a constructive heuristic derived from $Dsatur$ (Brélaz, 1979) or from RLF (Leighton, 1979). A move always consists in selecting a vertex and giving a colour to it. The trail system is modeled using a matrix $Tr(x, y)$ proportional to: (1) the number of times vertices x and y have the same colour in the solutions provided by the ants; (2) the quality of the solutions where $col(x) = col(y)$.

Another ant colouring method was proposed in (Shawe-Taylor & Zerovnik, 2002), where each ant can be considered as a heuristic close to the one proposed in (Petford & Welsh, 1989). The trail system is modeled by the use of an auxiliary graph $G' = (V, E')$ obtained from the original graph $G =$

(V, E) by adding edges. The set of added edges can evolve during the search and is based on the following idea: if many ants give a different colour to x and y such that edge $[x, y] \notin E$, then $[x, y]$ is added to E'. Note that the authors mainly tested their method on a few benchmark instances from ftp://dimacs.rutgers.edu/pub/challenge/graph/, namely $le450\_5a$, $le450\_5b$, $le450\_5c$, $le450\_5d$, $le450\_15a$ and $le450\_15b$, known to be optimally colourable 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 result is given for standard random graphs.

A third ant colouring heuristic was recently proposed in (Hertz & Zufferey, 2006). In this method, the role of each single ant is minor: it only helps choosing a colour for a single vertex. Only one solution evolves using a population of ants. A colour in $\{1, \ldots, k\}$ is associated with each ant and k ants are associated with each vertex. From a distribution of the ants on the vertices, a k-colouring of the graph is built, using a greedy procedure which is derived from *Dsatur* (Brélaz, 1979). At each step of the method, the distribution of the ants over the graph is modified, and therefore the associated k-colouring is modified.

Ant Local Search applied to the k-GCP

The proposed Ant Local Search colouring method is derived from the quick and efficient tabu search algorithm proposed for the k-GCP in (Bloechliger & Zufferey, 2008), where the authors consider the set of *partial legal k-colourings* which are defined as legal k-colouring of a subset of vertices of G. Such colourings can be represented by a partition of the vertex set into $k + 1$ subsets $V_1, \ldots, V_{k+1}$, where $V_1, \ldots, V_k$ are k disjoint stable sets (i.e. legal colour classes) and V_{k+1} is the set of non coloured vertices. The objective is to minimise the number of vertices in V_{k+1}. A neighbour solution s' can be obtained from the current solution s by moving a vertex v from V_{k+1} to a colour class V_c, and by moving to V_{k+1} each vertex in V_c that is adjacent to v. Such a move m is

denoted $m = (v; V_c)$. When it is performed, it is then tabu to move v back to V_{k+1} during $w_{nc} \cdot NC(s) + \text{RANDOM}(i_1, i_2)$ iterations, where $NC(s)$ is the number of non colored vertices in solution s, and $\text{RANDOM}(i_1, i_2)$ (with i_1, i_2 integers such that $i_1 < i_2$) is a function providing a uniform random integer in the set $\{i_1, i_1 + 1, \ldots, i_2 - 1, i_2\}$. Preliminary experiments confirmed that $(w_{nc}, i_1, i_2) = (0.6, 0, 9)$ is a reasonable parameter setting (which confirms the parameter setting proposed in (Galinier & Hao, 1999) and (Bloechliger & Zufferey, 2008)). In our experiments, $NC(s)$ is initially the number $|V|$ of vertices of the considered graph (no vertex is coloured in the initial solution), and then decreases very quickly to a much smaller value, generally below $\frac{|V|}{10}$ after less than 3000 iterations (which corresponds to a few seconds of computation). Then, the procedure needs more effort to assign a colour to every vertex. Such a behaviour is illustrated in Figure 1, where the variation of of the objective function is showed on an experiment performed on graph DSJC1000.5 with 86 colors.

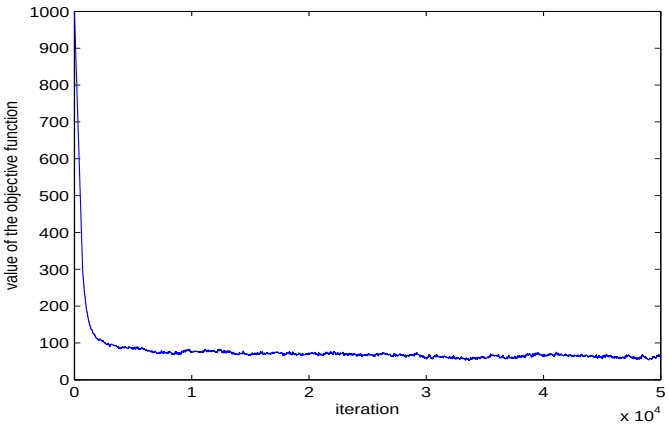

Figure 1: Variation of the objective function on graph DSJC1000.5 with 86 colors

In our Ant Local Search algorithm for the k-GCP (denoted ALS-COL), we propose to define a move m as above. Hence, it is straightforward to define the greedy force $GF(m)$ of a move $m = (v; V_c)$ as the inverse of the number of

adjacent vertices to v that are in colour class V_c. A main challenge is of course to define a relevant trail value $Tr(m)$ associated with move m. We propose to build and use a global trail system as follows. Let x and y be two vertices, and let $s_i = (V_1, \ldots, V_k; V_{k+1})$ be a solution provided by a single ant i of the population at a specific generation. If ant i gives the same colour c to x and y in solution s_i (i.e. $x, y \in V_c \neq V_{k+1}$), such an information should be transmitted to the ants of the next generation, and this information should be more important if x and y are in a large colour class. Formally, let

$$\Delta Tr_i(x,y) = \begin{cases} |V_c|^2 & \text{if } x \text{ and } y \text{ have the same colour } c \text{ in } s_i; \\ 0 & \text{if } x \text{ and } y \text{ have different colours in } s_i. \end{cases}$$

Then, as in many ants algorithms, we set $\Delta Tr(x,y) = \sum_{i=1}^{N} \Delta Tr_i(x,y)$. At the end of each generation and as in any classical ant algorithm, we globally update the trails as follows: $Tr(x,y) = \rho \cdot Tr(x,y) + \Delta Tr(x,y)$, where $\rho = 0.9$. We are now able to define the trail of a single move $m = (v; V_c)$ as follows: $Tr(v; V_c) = \sum_{x \in V_c} Tr(v,x)$.
In contrast with the classical ant algorithm, we propose to manage an iteration in order to avoid too much computational effort. At each iteration, the considered ant first chooses the set A of non tabu moves which have the largest greedy force values. Because of the definition of the greedy force, A will very often contain more than one element (this was confirmed by preliminary experiments). The chosen move m is then the one in A which has the largest trail value Tr. Ties are broken randomly.

We have now all the ingredients to formulate ALS-COL in Algorithm 2. Note that each ant i starts with the solution where all the vertices are in V_{k+1}, i.e. no vertex is coloured.

Algorithm 2 Ant Local Search for the k-GCP (ALS-COL)

While a maximum time limit is not reached, **do**:

1. **for** $i = 1$ **to** N, **do**:
   (a) apply the tabu search associated with ant i during I iterations; let s_i be the resulting solution;
2. update the trails by the use of $\{s_1, \ldots, s_N\}$;

Obtained results

Our algorithm was implemented in C++ and run on a computer with the following properties: Processor Intel Core2 Duo Processor E6700 (2.66GHz, 4MB Cache, 1066MHz FSB), RAM 2GB DDR2 667 ECC Dual Channel Memory (2x1GB). Two important parameters of ALS-COL are N (number of ants) and I (number of iterations performed by each single ant). Preliminary experiments showed that $N \in \{5; 10\}$ and $I \in \{50,000; 500,000\}$ are reasonable values. Usually, the best results are obtained with $(N; I) = (10; 500,000)$, but there are some exceptions: $N = 5$ on graphs with 1000 vertices (i.e. the large graphs), and $I = 50,000$ on the Leighton graphs and the graphs with 1000 vertices and a density of 0.5. It is probably possible to choose a better setting of parameters for each graph, but our goal is to have generic parameters which use only general characteristics of the graphs, and not to propose a specific set of parameters for each instance.

We made two series of tests with two maximal computational times, which are 1 hour and 10 hours. We ran our algorithm on 16 graphs from the DIMACS Challenge (see ftp://dimacs.rutgers.edu/pub/challenge/graph/). After a preliminary set of experiments, and in line with the literature (e.g. (Bloechliger & Zufferey, 2008), (Galinier *et al.*, 2008)), we selected those graphs because they are the most challenging ones. The considered graphs are described below.

- Six DSJC$n.d$ graphs: the DSJC's are random graphs with n vertices and

a density of $\frac{d}{10}$. It means that each pair of vertices has a probability of $\frac{d}{10}$ to be adjacent. We use the DSJC's graphs with $n \in \{500, 1000\}$ and $d \in \{1, 5, 9\}$.

- Two DSJR$n.r$ graphs: the DSJR's are geometric random graphs. They are constructed by randomly choosing n points in the unit square and two vertices are connected if they are distant by less than r. Graphs with an added end letter 'c' are the complementary graphs. We use two graphs with $n = 500$ and, respectively, $r = 1$ and $r = 5$.
- Four flatn_χ_0 graphs: the flat graphs are constructed graphs with n vertices and a chromatic number χ. The end number '0' means that all vertices are incident to the same number of vertices.
- Four le$n_\chi x$ graphs: the Leighton graphs have n vertices and a chromatic number χ equal to the size of the largest clique (i.e., the largest number of pairwise adjacent vertices). The end letter 'x' stands for different graphs with similar settings.

We first report the results obtained by using ALS-COL on these 16 graphs, and then compare our algorithm with TabuCol (Hertz & de Werra, 1987), PartialCol (Bloechliger & Zufferey, 2008), as well as with three graph colouring algorithms which are among the most effective today: the GH algorithm in (Galinier & Hao, 1999), the MOR algorithm in (Morgenstern, 1996), and the MMT algorithm in (Malaguti *et al.*, 2008). GH, MOR and MMT are all population based methods which use local search procedures. GH uses TabuCol to improve offspring solutions, whereas MMT uses a procedure close to PartialCol. MOR works in the same search space as PartialCol, but uses simulated annealing instead of tabu search, and much more sophisticated moves. Note that the three ant colouring heuristics previously described will not be considered because they are not competitive at all with the best colouring methods.

Table 1 reports the results obtained with ALS-COL with a time limit of one hour. The first two columns respectively indicate the name and the number of

vertices of the graph. The third column contains two numbers, the first one being the chromatic number (we put a "?" when it is not known), and the second one the best upper bound ever found by a heuristic. We ran ALS-COL 10 times on each graph and each value of k. The fourth column reports various values of k ranging from the smallest number for which we had at least one successful run, to the smallest number for which we had 10 successful runs. The next columns respectively contain the number of successful runs and the number of tries, the average number of iterations in thousands (i.e., the total number of performed moves divided by 1000) on successful runs, and the average CPU-time used (in seconds).

Table 2 gives the same information as for ALS-COL, but for PartialCol (Bloechliger & Zufferey, 2008). The authors of PartialCol provided us with the code of their algorithm and we were able to run it on our computer, thus there may be a few very minor differences with the results described in (Bloechliger & Zufferey, 2008). Note that we omit some results for some values of k which are not relevant for the comparison.

Table 3 gives the same information as for PartialCol, but for PartialCol which is restarted from scratch every I iterations (using the same I as the one used in ALS-COL for the same graph). This will allow to better measure the contribution of the ingredients added to PartialCol to derive ALS-COL.

If we have a look at Tables 1, 2 and 3, we can first easily remark that PartialCol with restarts is not competitive at all when compared to the usual PartialCol and to ALS-COL. Thus, it confirms that we add relevant ingredients to PartialCol to derive ALS-COL. There is now no need to focus farther on this method. If we compare ALS-COL with PartialCol, we observe that ALS-COL finds a smaller number of colors on 8 graphs (namely DSJC1000.1, DSJC1000.5, DSJC1000.9, DSJC500.5, DSJR500.5, flat1000_76_0, le450_25c, and le450_25d), the same number on 7 graphs (namely DSJC500.1, DSJC500.9, DSJR500.1c, flat1000_50_0, flat1000_60_0, le450_15c, and le450_15d), and a larger number on

one graph (namely flat300_28_0). This is denoted by $Score$(ALS-COL; PartialCol) = $(8, 7, 1)$. For the 7 graphs for which both algorithms find similar k-colourings, we can observe that: (1) ALS-COL has a better success rate on 2 graphs (namely DSJC500.9 and DSJR500.1c), the same success rate on 4 graphs (namely DSJC500.1, flat1000_50_0, flat1000_60_0, and le450_15c), but a worse success rate on 1 graph (namely le450_15d); (2) on the graphs where the success rate is the same, both algorithms quickly find the associated k-colourings. Therefore, we can securely conclude that ALS-COL has a better performance than PartialCol.

In Table 4, we compare ALS-COL with PartialCol, TabuCol, GH, MMT and MOR. For every algorithm, we report the smallest k such that a legal k-colouring could be found. The results for GH, MMT and MOR are taken from (Bloechliger & Zufferey, 2008). Comparisons must therefore be done carefully because the conditions of experimentation are not the same. For example, our algorithm has a 1 hour time limit, while MMT uses a limit of 100 minutes. In addition, the performances of the computers are different, and on the contrary to GH and MMT, we do not adjust the parameters of ALS-COL on each instance.
We can observe that $Score$(ALS-COL; TabuCol) = $(7, 9, 0)$, $Score$(ALS-COL; MOR) = $(7, 6, 3)$, $Score$(ALS-COL; MMT) = $(1, 10, 5)$, and $Score$(ALS-COL; GH) = $(1, 9, 4)$. In summary, if we allow a time limit of one hour to ALS-COL, it is better than PartialCol and MOR, but not as good as MMT and GH, which are the current best graph colouring heuristics. The two latter methods are based on a specialized recombination operator, which builds any offspring solution colour class by colour class, while trying at each step to put as many vertices as possible in any colour class. Such a way of transmitting the information during the search seems to be slightly better than using the trail system we have proposed in this paper. However, ALS-COL is the first ant algorithm which is competitive with the best colouring algorithms!
We finally report in Table 5 the obtained results with a time limit of 10 hours

(with the use of $(N; I) = (10; 500,000)$, except for the Leighton graphs where $I = 50,000$), but only for graphs for which ALS-COL could find better colourings when compared to Table 1. We can observe that we save one colour on graphs DSJC1000.9, DSJC500.9 and le450_25d, and two colours on graphs DSJC1000.5 and flat1000_76_0. Therefore, with such a time limit, the gap between ALS-COL and MMT and GH is significantly reduced.

Conclusion

We have proposed a new general optimization methodology, called Ant Local Search, that, in contrast with the other ant algorithms, uses each ant as a local search procedure, and where each decision made by each ant can be quickly computed. We have also presented an adaptation of the Ant Local Search to the $k - GCP$. The computational experiments, carried out on a set of the most challenging DIMACS graphs, show that ALS-COL is more effective than PartialCol, which is the local search from which we derive ALS-COL by adding a powerful trail system, as well as a quick and efficient way to make a choice between the possible moves at each iteration. ALS-COL appears to be also competitive with the current best hybrid evolutionary graph colouring algorithms. Therefore, we think that the Ant Local Search methodology is a new interesting and challenging approach for the solution of complex optimization problems.

References

Bloechliger, I., & Zufferey, N. 2008. A Graph Coloring Heuristic Using Partial Solutions and a Reactive Tabu Scheme. *Computers & Operations Research*, **35**, 960 – 975.

Blum, C. 2005. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, **2(4)**, 353–373.

Brélaz, D. 1979. New Methods to Color Vertices of a Graph. *Communications of the Association for Computing Machinery*, **22**, 251–256.

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

Costa, D., & Hertz, A. 1997. Ants can colour graphs. *Journal of the Operational Research Society*, **48**, 295–305.

Dorigo, M. 1992. *Optimization, learning and natural algorithms (in italian).* Ph.D. thesis, Politecnico di Milano, Dipartimento di Elettronica, Italy.

Dorigo, M., & 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.

Dorigo, M., & Stuetzle, T. 2002. *Handbook of metaheuristics.* In F. Glover and G. Kochenberger (Eds). Chap. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances, pages 251 – 285.

Dorigo, M., Maniezzo, V., & Colorni, A. 1991. *Positive feedback as a search strategy.* Tech. rept. 91-016. Politecnico di Milano, Dipartimento di Elettronica, Italy.

Dorigo, M., Birattari, M., & Stuetzle, T. 2006. Ant colony optimization – artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine.*

Fleurent, C., & Ferland, J. A. 1996. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, **63**(3), 437–461.

Galinier, P., & Hao, J.K. 1999. Hybrid Evolutionary Algorithms for Graph Coloring. *Journal of Combinatorial Optimization*, **3(4)**, 379–397.

Galinier, P., & Hertz, A. 2006. A Survey of Local Search Methods for Graph Coloring. *Computers & Operations Research*, **33**, 2547–2562.

Galinier, P., Hertz, A., & Zufferey, N. 2008. An Adaptive Memory Algorithm for the Graph Coloring Problem. *Discrete Applied Mathematics*, **156**, 267 – 279.

Gamst, A., & Rave, W. 1992. On the frequency assignment in mobile automatic telephone systems. *In: Proceedings of GLOBECOM'92*.

Garey, M., & Johnson, D.S. 1979. *Computer and Intractability: a Guide to the Theory of NP-Completeness*. San Francisco: Freeman.

Glass, C. 2002. Bag rationalisation for a food manufacturer. *Journal of the Operational Research Society*, **53**, 544 – 551.

Glass, C.A., & Prugel-Bennett, A. 2005. A polynomial searchable exponential neighbourhood for graph colouring. *Journal of the Operational Research Society*, **56**, 324 – 330.

Glover, F., & Laguna, M. 1997. Tabu search. *Kluwer Academic Publishers, Boston, MA*.

Herrmann, F., & Hertz, A. 2002. Finding the chromatic number by means of critical graphs. *ACM Journal of Experimental Algorithmics*, **7(10)**, 1–9.

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

Hertz, A., & Zufferey, N. 2006. A new ant colony algorithm for the graph coloring problem. *Pages 51 – 60 of: Proceedings of the Workshop on Nature Inspired Cooperative Strategies for Optimization, NICSO 2006, June 29 – 30*.

Hertz, A., Plumettaz, M., & Zufferey, N. 2008. Variable space search for graph coloring. *Discrete Applied Mathematics*, **156**, 2551 – 2560.

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

Leighton, F. T. 1979. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau Standard*, **84**, 489–505.

Malaguti, E., Monaci, M., & Toth, P. 2008. A Metaheuristic Approach for the Vertex Coloring Problem. *INFORMS Journal on Computing*, **20 (2)**, 302 – 316.

Mehrotra, A., & Trick, M.A. 1996. A column generation approach for graph coloring. *INFORMS Journal on Computing*, **8**, 344–354.

Morgenstern, C. 1996. Distributed Coloration Neighborhood Search. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, **26**, 335–357.

Petford, A., & Welsh, D. 1989. A randomised 3-colouring algorithm. *Discrete Mathematics*, **74**, 253–261.

Shawe-Taylor, J., & Zerovnik, J. 2002. Ants and graph coloring. *Pages 593–597 of: Proceedings of ICANNGA'01.*

Stecke, K. 1985. Design planning, scheduling and control problems of flexible manufacturing. *Annals of Operations Research*, **3**, 3–12.

Stuetzle, T., & Hoos, H. 1997. *Improving the ant system: a detailed report on the max-min ant system.* Tech. rept. Department of Computer Sciences - Intellectics Group, Technical University of Darmstadt.

Zufferey, N. 2002. *Heuristiques pour les Problèmes de la Coloration des Sommets d'un Graphe et d'Affectation de Fréquences avec Polarités.* Ph.D. thesis, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland.

Zufferey, N., Amstutz, P., & Giaccari, P. 2008. Graph Colouring Approaches for a Satellite Range Scheduling Problem. *Journal of Scheduling*, **11 (4)**, 263 – 277.

Graph	$\|V\|$	χ, $k^\star$	k	succ./run	10^3 iter.	Time
DSJC1000.1	1000	?,20	20	2/10	226284	3056
			21	10/10	388	5
DSJC1000.5	1000	?,83	86	1/10	41734	1535
			87	8/10	34951	1231
			88	10/10	37156	1286
DSJC1000.9	1000	?,224	225	1/10	93617	2759
			226	8/10	70203	2442
			227	10/10	42044	1591
DSJC500.1	500	?,12	12	10/10	45964	304
DSJC500.5	500	?,48	48	8/10	103496	1177
			49	10/10	61934	675
DSJC500.9	500	?,126	127	9/10	63364	802
			128	10/10	9027	149
DSJR500.1c	500	?,85	85	4/10	39787	372
			86	10/10	18778	180
DSJR500.5	500	?,122	125	2/10	184949	1934
			126	7/10	331381	1543
			127	9/10	27177	313
			128	10/10	16010	181
flat1000_50_0	1000	50,50	50	10/10	113	15
flat1000_60_0	1000	60,60	60	10/10	383	46
flat1000_76_0	1000	76,82	85	2/10	40966	1487
			86	5/10	42707	1473
			87	9/10	27060	927
			88	10/10	26716	897
flat300_28_0	300	28,28	29	1/10	2931	30
			30	1/10	6602	57
			31	10/10	17423	111
le450_15c	450	15,15	15	10/10	390	7
le450_15d	450	15,15	15	9/10	669	10
			16	10/10	320	5
le450_25c	450	25,25	26	10/10	220007	1847
le450_25d	450	25,25	26	9/10	204560	1728
			27	10/10	32469	253

Table 1: Detailed results of ALS-COL with a time limit of 1 hour

Graph	$\|V\|$	χ, $k^\star$	k	succ./run	10^3 iter.	Time
DSJC1000.1	1000	?,20	21	10/10	388	1
DSJC1000.5	1000	?,83	89	5/10	141919	2047
DSJC1000.9	1000	?,224	226	3/10	59994	1572
			227	9/10	67305	1629
			228	10/10	16988	422
DSJC500.1	500	?,12	12	10/10	41404	104
DSJC500.5	500	?,48	50	10/10	22748	164
DSJC500.9	500	?,126	127	3/10	169134	2033
			128	10/10	5957	67
DSJR500.1c	500	?,85	85	1/10	32789	249
			86	3/10	39701	323
DSJR500.5	500	?,122	126	8/10	126638	888
			127	10/10	121043	931
			128	10/10	5957	67
flat1000_50_0	1000	50,50	50	10/10	113	10
flat1000_60_0	1000	60,60	60	10/10	383	30
flat1000_76_0	1000	76,82	88	5/10	89058	1278
flat300_28_0	300	28,28	28	4/10	83895	624
			29	8/10	317044	2056
			30	10/10	205835	1153
			31	10/10	141841	685
le450_15c	450	15,15	15	10/10	287	1
le450_15d	450	15,15	15	10/10	3659	9
			16	10/10	1521	4
le450_25c	450	25,25	27	10/10	14509	5
le450_25d	450	25,25	27	10/10	8643	3

Table 2: Detailed results for PartialCol with a time limit of 1 hour

Graph	$\|V\|$	χ, $k^\star$	k	succ./run	10^3 iter.	Time
DSJC1000.1	1000	?,20	21	10/10	361	2
DSJC1000.5	1000	?,83	93	1/10	110645	3180
DSJC1000.9	1000	?,224	233	1/10	78975	3138
DSJC500.1	500	?,12	12	10/10	45964	1928
DSJC500.5	500	?,48	50	9/10	142833	1147
DSJC500.9	500	?,126	128	3/10	109894	1500
DSJR500.1c	500	?,85	85	10/10	129355	1178
			86	10/10	11922	108
DSJR500.5	500	?,122	128	8/10	194655	1943
flat1000_50_0	1000	50,50	50	10/10	113	10
flat1000_60_0	1000	60,60	60	10/10	383	30
flat1000_76_0	1000	76,82	92	1/10	263001	3231
flat300_28_0	300	28,28	28	5/10	124130	931
			29	6/10	242786	1598
			30	8/10	202857	1166
			31	10/10	209899	1038
le450_15c	450	15,15	15	10/10	1786	16
le450_15d	450	15,15	15	10/10	3660	10
			16	10/10	1521	4
le450_25c	450	25,25	27	10/10	24034	117
le450_25d	450	25,25	27	10/10	36665	178

Table 3: Detailed results for PartialCol with restarts with a time limit of 1 hour

Graph	$\|V\|$	χ, $k^\star$	ALS-Col	PartialCol	TabuCol	MMT	GH	MOR
DSJC1000.1	1000	?,20	20	21	20	20	20	21
DSJC1000.5	1000	?,83	86	89	89	83	83	88
DSJC1000.9	1000	?,224	225	226	227	226	224	226
DSJC500.1	500	?,12	12	12	12	12	12	12
DSJC500.5	500	?,48	48	49	49	48	48	49
DSJC500.9	500	?,126	127	127	127	127	126	128
DSJR500.1c	500	?,85	85	85	85	85	-	85
DSJR500.5	500	?,122	125	126	126	122	-	123
flat1000_50_0	1000	50,50	50	50	50	50	50	50
flat1000_60_0	1000	60,60	60	60	60	60	60	60
flat1000_76_0	1000	76,82	85	88	88	82	83	89
flat300_28_0	300	28,28	29	28	31	31	31	31
le450_15c	450	15,15	15	15	16	15	15	15
le450_15d	450	15,15	15	15	15	15	15	15
le450_25c	450	25,25	26	27	26	25	26	25
le450_25d	450	25,25	26	27	26	25	26	25

Table 4: Comparisons between ALS-COL and state-of-the-art algorithms

Graph	$\|V\|$	χ, $k^\star$	k	succ./run	10^3 iter.	Time
DSJC1000.5	1000	?,83	84	9/10	567131	16059
			85	10/10	461330	12734
DSJC1000.9	1000	?,224	224	1/10	119838	4292
DSJC500.9	500	?,126	126	1/10	1112960	12681
flat1000_76_0	1000	76,82	83	10/10	643137	17130
			84	10/10	526522	14341
le450_25d	450	25,25	25	2/10	85487	22908

Table 5: Additional results of ALS-COL with a time limit of 10 hours