An Efficient Population-based Extension of the PartialCol Algorithm

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

Abstract

The *PartialCol* algorithm is a tabu search method that has shown to be among the state-of-the-art graph coloring heuristics. In this paper, we use PartialCol as a subprocedure of a population-based algorithm, namely an ant colony system. In contrast with usual ant methods, we consider each ant as a local search guided by greedy forces and a trail system. Computational experiments show that this extension to PartialCol leads to better performances.

Keywords: graph coloring, local search, ant colony

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 they do not satisfy all problem constraints. For every solution $s \in S$, a neighborhood $N(s) \subset S$ is defined. A local search method starts at an initial solution, and then moves repeatedly from the current solution to a neighbor solution in order to try to find better solutions, measured by an appropriate objective function. The passage from one solution to the next solution is called a *move*. State-of-the-art local search methods are simulated annealing, tabu search and variable neighborhood search.

Population-based methods deal with a set of solutions that cooperate in order to generate new improved solutions. Ant colonies methods, introduced in [4], belong to this class 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. In contrast, we consider each ant as a local search. At each step of the local search and as in every ants algorithm,

[1] *Institut de mathématiques*, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland, matthieu.plumettaz@epfl.ch

[2] **Corresponding author,** *Département de mathématiques et de génie industriel*, École Polytechnique de Montréal, Canada, david.schindl@gerad.ca

[3] *Faculté des Sciences de l'Administration*, Université Laval, Québec (QC), G1K 7P4, Canada, nicolas.zufferey@fsa.ulaval.ca

the considered ant takes a decision according to the greedy force and the trails.

The paper is organized as follows. We first briefly describe tabu search and the classical ants algorithms. Secondly, we define the graph coloring problem and mention some well-known heuristic approaches. Then we describe the PartialCol heuristic, and how we incorporated it in an ant colony method to obtain our algorithm, called *Ant-PartialCol* (APC for short). Finally, computational experiments are reported, and we end up the paper with a conclusion containing some ongoing research directions.

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 minimized 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 neighbor solution. The solution s_{r+1} is computed as $s_{r+1} = \arg \min_{s \in N'(s_r)} f(s)$, where $N'(s)$ is a subset of $N(s)$ containing all solutions s' which can be obtained from s 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 [7].

In most ant algorithms, 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 (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 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 generally 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 obtained by: $\Delta Tr(m) = \sum_{i=1}^{N} \Delta Tr_i(m)$, where $\Delta Tr_i(m)$ is growing with the quality of the solution s_i provided by ant i if it has performed decision m. Recent overviews

of ant algorithms can be found in [2] and [3].

The graph coloring problem

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 and denoted by $\chi(G)$) such that there exists a legal k-coloring of G. The GCP is NP-hard [6]. Current exact solution methods can only solve problems of relatively small size (roughly not more than 100 vertices) [8].

Given a fixed integer k, we consider the decision problem, called k-GCP, which aims to determine whether or not G admits a legal k-coloring. This problem can be approached with an optimization heuristic by minimizing the number of conflicting edges in the k-coloring solution space. If the value of the obtained solution is zero, this means that G has a legal k-coloring. An upper bound on the chromatic number of G can then be obtained by finding a smallest possible number k such that our heuristic could obtain a legal k-coloring. For a recent survey, the reader is referred to [5].

The Ant-PartialCol algorithm

The proposed APC coloring method is derived from the quick and efficient PartialCol algorithm proposed for the k-GCP in [1], where the authors consider the set of *partial legal k-colorings* which are defined as legal k-colorings of a subset of vertices of G. Such colorings 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 color classes) and V_{k+1} is the set of non colored vertices. The objective it to minimize the number of vertices in V_{k+1}. A neighbor solution s' can be obtained from the current solution s by moving a vertex v from V_{k+1} to a color 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, as proposed in [1], it is then tabu to move v back to V_{k+1} during $0.6 \cdot n_c + \text{RANDOM}(0, 9)$ iterations, where n_c is the number of vertices in V_{k+1} of s, and $\text{RANDOM}(0, 9)$ is a function providing a random integer in the set $\{0, 1, \ldots, 9\}$.

In APC, 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 vertices adjacent to v that are in color 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 color 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 in addition this information should be more important if x and y are in a large color class. Formally, let

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

Then, as in lots of ants algorithms, we set $\Delta Tr(x,y) = \sum_{i=1}^{N} \Delta Tr_i(x,y)$, where N is the number of ants. 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: $Tr(v; V_c) = \sum_{x \in V_c} Tr(v,x)$.
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 APC in Algorithm 1.

Algorithm 1 Ant-PartialCol for the k-GCP (APC)

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

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

Note that each ant i starts with the solution where all the vertices are in V_{k+1}, i.e. no vertex is colored.

Obtained results

Our algorithm was implemented in C++ and run on a 2GHz Pentium 4 with 512MB of RAM. Two important parameters of APC are N (number of ants)

and I (number of iterations performed by each single ant). Preliminary experiments lead us to the following parameter setting: $N = 10$ and $I = 500'000$. 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.

The time limit has been fixed to one hour. After a preliminary set of experiments, and in adequation with the literature (e.g. [1]), we selected the 16 graphs we considered the most challenging, from the DIMACS Challenge. These graphs as well as their description can be found at [10].

Name	$\|V\|$	d	k(opt)	k(best)	APC	PartialCol
DSJC500.1	500	0.1	?	12	12	12
DSJC500.5	500	0.5	?	48	48	49
DSJC500.9	500	0.9	?	126	127	127
DSJC1000.1	1000	0.1	?	20	20	20
DSJC1000.5	1000	0.5	?	83	88	89
DSJC1000.9	1000	0.9	?	224	228	228
DSJR500.1c	500	0.9	?	85	85	85
DSJR500.5	500	0.5	?	122	125	126
flat300_28_0	300	0.5	28	28	31	28
flat1000_50_0	1000	0.5	50	50	50	50
flat1000_60_0	1000	0.5	60	60	60	60
flat1000_76_0	1000	0.5	76	82	87	88
le450_15c	450	0.17	15	15	15	15
le450_15d	450	0.17	15	15	15	15
le450_25c	450	0.17	25	25	26	27
le450_25d	450	0.17	25	25	26	27

Table 1: Ant-PartialCol versus PartialCol.

For APC as well as for PartialCol, we performed 10 runs for each considered value of k on each instance. Table 1 reports the best results obtained by APC and PartialCol. The first three columns indicate respectively the name of the graph, the number of vertices and edge density. The fourth column contains the chromatic number (we put a "?" when it is not known) and the fifth one contains the best upper bound ever found by a heuristic. The last two columns contain the best coloring obtained by APC and PartialCol, respectively.

As can be seen, APC is strictly better on 6 instances and worse on only one, and both methods provide the same results on the rest. Concerning the differences between the column "k(best)" and "APC", we observe that APC reaches the best known value on half of the instances. However, there are some where the difference is quite important.

Conclusion and perspectives

The computational experiments, carried out on a set of the most challenging DIMACS graphs [10] show that APC, obtained by adding to PartialCol a powerful trail system, with a quick and efficient way to make a choice between the

possible moves at each iteration, is in average more efficient than PartialCol alone.

Our current research directions are the following. First, as mentioned above, there are several instances where APC failed to reach the chromatic number or the best known bounds. Thus, it may be interesting to analyze its behavior with larger time limits. Second, in order to improve APC, other trail systems and greedy forces should be developed. Third, we shall compare APC with all state-of-the-art coloring heuristics. Finally, the idea of considering each ant as a local search can lead to a new and general ant algorithm applicable to further optimization problems [9].

References

[1] I. Bloechliger and N. Zufferey. A Graph Coloring Heuristic Using Partial Solutions and a Reactive Tabu Scheme. *Computers & Operations Research (to appear)*, 2007.

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

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

[4] M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical Report Technical Report 91-016, Politecnico di Milano, Dipartimento di Elettronica, Italy, 1991.

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

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

[7] F. Glover and M. Laguna. Tabu search. *Kluwer Academic Publishers, Boston, MA*, 1997.

[8] F. Herrmann and A. Hertz. Finding the chromatic number by means of critical graphs. *ACM Journal of Experimental Algorithmics*, 7/10:1–9, 2002.

[9] M. Plumettaz, D. Schindl, and N. Zufferey. Ant local search for graph coloring. In *"Graph and Optimization VI", Cademario, Switzerland*, 2007.

[10] DIMACS Website. ftp://dimacs.rutgers.edu/pub/challenge/graph/.