INTERNATIONAL JOURNAL OF COMPUTERS AND COMMUNICATIONS
DOI: 10.46300/91013.2020.14.8

Volume 14, 2020

Ant Decision Systems for Combinatorial Optimization with Binary Constraints

Nicolas Zufferey
HEC - University of Geneva, Switzerland
Email: nicolas.zufferey-hec@unige.ch

***Abstract*—In this paper is considered a problem (P) which consists in minimizing an objective function f while satisfying a set of binary constraints. Function f consists in minimizing the number of constraints violations. Problem (P) is NP-hard and has many applications in various fields (e.g., graph coloring, frequency assignment, satellite range scheduling). On the contrary to exact methods, metaheuristics are appropriate algorithms to tackle medium and large sized instances of (P). A specific type of ant metaheuristics is designed to tackle (P), where in contrast with state-of-the-art ant algorithms, an ant is a decision helper and not a constructive procedure.**

***Keywords*—Combinatorial optimization, Binary constraints, Ant algorithms, Metaheuristics**

I. Introduction

Consider the problem (P) which consists in minimizing an objective function f while satisfying a set C of binary constraints. Let $s = (s_1, s_2, \ldots, s_n)$ be a solution of problem (P). It is assumed that each constraint is binary: it only involves two variables s_i and s_j and can be formulated as one of the following expressions:

- $|s_i - s_j| \neq \varepsilon_{ij}$,
- $s_i - s_j \geq l_{ij}$,
- $s_i - s_j \leq u_{ij}$.

In addition, the value of each variable s_i must belong to a set D of integer values. A *conflict* occurs between two variables s_i and s_j if their associated binary constraint is violated. The objective function f consists in minimizing the number of conflicts. The density $d(P)$ of (P) is defined as the proportion of pairs (s_i, s_j) linked with a binary constraint, among all the existing pairs (which is $\frac{n\cdot(n-1)}{2}$). For example, if $d(P) = 1$, it means that all pairs of variables are linked with a binary constraint. One can remark that $d(P)$ can be defined as $\frac{2\cdot|C|}{n\cdot(n-1)}$.
Problem (P) is NP-hard and has many applications in graph coloring [1], in frequency assignment in telecommunication networks [2], and in satellite range scheduling problem [3]. From that literature, one can easily deduce that metaheuristics are the most appropriate methods to tackle (P).

An *exact method*, like branch-and-bound or dynamic programming, guarantees the optimality of the provided solution. However, for most real-life optimization problems, such methods need huge computing times to find optimal solutions, because such problems are NP-hard [4]. For these difficult problems, one should prefer to quickly find satisfying solutions, which is the goal of (meta)heuristic solution methods. In contrast with *heuristics* (which are basic solution methods), a *metaheuristic* usually contains refined strategies to guide and control the search process in the solution space. The reader interested in a recent book on metaheuristics is referred to [5], whereas the reader searching for guidelines to efficiently (according to various criteria) adapt a metaheuristic to a problem is referred to [6].

There mainly exist three families of (meta)heuristics: constructive algorithms (where a complete solution is built step by step from scratch, like the greedy algorithm), local search techniques (where one solution is handled and iteratively modified), and evolutionary methods (where a population of solutions is usually managed).
A *local search* method starts with an initial solution and tries to improve it iteratively. At each iteration, a modification (called a *move*) of the current solution is performed in order to generate a neighbor solution. The definition of a move (i.e. the definition of the *neighborhood* structure) depends on the considered problem. Popular local search methods are simulated annealing, tabu search, threshold algorithms, variable neighborhood search, and guided local search.
In *evolutionary algorithms*, a population of solutions (or parts of solutions) is usually handled. At each generation, the population is modified according to two elements: *self-adaptation* and *collaboration*. Well-known evolutionary algorithms are ant algorithms, genetic methods, adaptive memory procedures, and scatter search.

In ant algorithms, the self-adaptation ingredient, modeled by the *greedy force*, is the short-term profit that each ant has to select a specific decision (independently from the other ants), and the collaboration is managed with a *trail* system represented by a central memory (build by ants from previous generations), which contains information from the past of the search process. Based on the trail system, an ant is likely to select the same decisions as other ants, especially if such decisions were performed by *many* ants in *good* solutions generated in the past. As pointed out in [7], there exist various possibilities to design an ant algorithm, depending on the *role* assigned to each ant. In *Constructive Ant Systems* (CAS), an ant is a constructive heuristic. In *Ant Local Search* (ALS) methods, an ant is a local search technique. In *Ant Decision Systems* (ADS), an ant is a decision helper within a local search framework.

The goal of this paper is to generalize the ant decision systems proposed in [8] and [7] for combinatorial optimization problems with binary constraints. Among the motivations of this work, it is important to mention that an ADS approach was already successfully adapted to the graph k-coloring problem (see again [8] and [7]), which is a reduced version of (P) as the binary constraints have always the form $|s_i - s_j| \neq 0$ for adjacent vertices. It was showed that ADS significantly outperforms CAS for that problem, for various densities $d(P)$ of (P) ranging from 0.1 to 0.97. The remaining part of this paper is organized as follows. The ant decision systems are discussed in Section II, and positioned according to the standard constructive ant systems. Then, an ADS approach is designed to tackle problem (P) in Section III. The paper ends up with a conclusion in Section IV, along with avenues of research.

II. Ant algorithms: CAS and ADS approaches

As described 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* $GF(m)$ (short-term profit for the considered ant, also called *visibility* or *heuristic information*) and the *trail* $Tr(m)$ (information obtained from other ants). The probability $p_i(m)$ that ant i chooses decision m is given by Equation (1).

$$p_i(m) = \frac{GF(m)^\alpha \cdot Tr(m)^\beta}{\sum\limits_{m' \in M_i} GF(m')^\alpha \cdot Tr(m')^\beta} \tag{1}$$

where α and β are parameters (which strongly depend on the considered problem and algorithm), and M_i is the set of admissible decisions that ant i can perform at that time. Let M be the set of all possible decisions. When each ant of the population has built a solution, the trails are generally updated as presented in Equation (2).

$$Tr(m) = \rho \cdot Tr(m) + \Delta Tr(m), \; \forall m \in M \tag{2}$$

where $\rho \in]0,1[$ is a parameter representing the evaporation of the trails, which is usually close to 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 the reinforcement term can be set as indicated in Equation (3).

$$\Delta Tr(m) = \sum_{i=1}^{N} \Delta Tr_i(m), \tag{3}$$

where $\Delta Tr_i(m)$ is proportional to the quality of the solution provided by ant i if it has performed decision m. The pseudo-code of such constructive ant systems (CAS) is given in Algorithm 1. A *generation* consists in performing steps (1) to (2). A stopping condition can be a maximum number of generations or a maximum time limit. For a recent survey on ant algorithms, the reader is referred to [9].

Instead of being a constructive heuristic, a single ant can help to select a decision within a procedure which makes only one solution evolve. More precisely, if one uses the commonly used local search terminology, each ant helps to move from a *current* solution to a *neighbor* solution by performing minor modifications on the current solution. Again and by definition of an ant algorithm, the motor of each ant is based on the greedy force and the trail. The general method, called *Ant Decision System* (ADS), is summarized in Algorithm 2.

Algorithm 1 Constructive Ants System (CAS)

```
While no stopping condition is met, do:
    1) for $i = 1$ to $N$, do:
        a) ant $i$ builds a solution $s_i$ step by step based
           on Equation (1);
        b) locally update the trails by the use of $s_i$
           (optional);
    2) globally update the trails by the use of a subset of
       $\{s_1, \ldots, s_N\}$;
Output: best encountered solution during the search.
```

Algorithm 2 Ant Decision System (ADS)

```
Initialization:
    1) generate (randomly or greedily) an initial solution $s$;
    2) set $s^\star = s$ and $f^\star = f(s)$ (best encountered solution);
While no stopping condition is met, do:
    1) some ants modify the solution $s$ (let $B$ be the set of
       the associated decisions);
    2) globally update the trails based on the set $B$;
    3) if $f(s) < f^\star$, set $s^\star = s$ and $f^\star = f(s)$;
Output: best encountered solution $s^\star$.
```

III. ADS for (P)

Remind that the value of each s_i should belong to set D of integers. Consider an ant as a possible value of D. Initially, q (positive and integer parameter) ants of each value of D is associated with each variable s_i. Let $G_i^{(t)}$ be the group of ants associated with s_i at the end of iteration t. Note that the index (t) referring to the iteration t will be often ignored to simplify the text. Several ants with the same value can belong to each G_i. At iteration t, let $ASSIGN(U_t)$ be a procedure able to assign a value in $G_i^{(t)}$ to each s_i, for all the s_i's belonging to the set U_t. At each iteration t, the following steps are performed:

1) modify the distribution of the ants over some decision variables (i.e. modify some G_i's), based on the greedy forces and the trail system;
2) determine the set U_t of decision variables for which the value has to be recomputed by procedure $ASSIGN$;
3) perform $ASSIGN(U_t)$;
4) evaluate the resulting neighbor solution with the objective function f.

At the above second step, it is important to reassign a value to the following decision variables: (1) all the decision variables s_j with a different group of ants on it (because it is forbidden to give a value to s_j which is not represented by at least one ant on s_j); (2) all the conflicting decision variables (because the final goal is still to try to remove some conflicts). More precisely, at iteration t, the set U_t provided to the $ASSIGN$ procedure is given in Equation (4).

$$\begin{aligned} U_t &= \{\text{decision variables involved in a move at iteration } t\} \\ &\cup \{\text{conflicting decision variables at iteration } t-1\} \end{aligned} \tag{4}$$

Note that U_0 is initialized to $\{s_1, \ldots, s_n\}$. The procedure $ASSIGN$ is performed in the following way. First, the value of each decision variable belonging to U_t is temporarily removed. Then, a value is given to the decision variables of U_t as follows. At each step, select the decision variable $s_i \in U_t$ with the largest *saturation* (see its definition below), and give the best value to s_i, which is the one in $G_i^{(t)}$ minimizing the augmentation of the number of conflicts. If there are several possibilities, chose the most represented value in $G_i^{(t)}$ (ties are broken randomly).

The saturation $sat(s_i)$ of a decision variable s_i is now defined. Let $A(s_i)$ denote the set of decision variables which are linked to s_i with a constraint. In addition, let $v(s_i)$ be the value assigned to s_i. Further, let $V(s_i)$ be the set of values defined as $\bigcup_{s_j \in A(s_i)}\{v(s_j)\}$. Note that the same value cannot appear more than one time in $V(s_i)$. The saturation of s_i is defined as $sat(s_i) = |V(s_i)|$. One can remark that the larger $sat(s_i)$ is, the less values are available for s_i, and thus the more saturated is s_i.

At each iteration, the goal is to change the value of a randomly chosen conflicting variable s_i. For illustration purpose, let v_1 and v_2 be two ants (i.e. two integer values). Suppose that v_1 is on the conflicting variable s_i (i.e. $v_1 \in G_i$), and v_2 is on variable s_j (conflicting or not). A *move* $m = (s_i, v_1) \leftrightarrow (s_j, v_2)$ consists in switching the ants v_1 and v_2 on variables s_i and s_j. In other words, v_2 (resp. v_1) is moved from G_j (resp. G_i) to G_i (resp. G_j). Suppose that the value $v(s_i)$ of s_i is v_1 because p ants of value v_1 are on variable s_i. In order to be sure to remove the conflicting value v_1 from variable s_i (at the considered iteration), a sequence of p moves has to be performed: all the ants of value v_1 have to be replaced by ants of other values. After such an iteration, in order to help to avoid cycling (i.e. coming back to an already visited solution), a *tabu* status can be put on the pair (s_i, v_1): it is forbidden to put v_1 in G_i for tab (parameter) iterations. This kind of feature reminds the well-known tabu search algorithm. For more information on tabu search, the reader is referred to [10].

According to (P), a consistent trail system should incorporate the following information. On the one hand, if an ant of value v just leaves variable s_i (i.e. one value v is removed from G_i), the other ants of value v will be poorly attracted by variable s_i. On the other hand, if an ant of value v just arrives on variable s_i (i.e. one value v is added to G_i), then other ants of value v will be attracted by the group G_i. Formally, let $tr_v^{(t)}(s_i)$ be the trail left by ant of value v on decision variable s_i at the end of iteration t. Such values are updated at the end of an iteration as in state-of-the-art ant algorithms as presented in Equation (5).

$$tr_v^{(t)}(s_i) = \rho \cdot tr_v^{(t-1)}(s_i) + \Delta tr_v^{(t)}(s_i) \tag{5}$$

In addition, the trail $Tr(m)$ of a move $m = (s_i, v_1) \leftrightarrow (s_j, v_2)$ can be computed in a straightforward fashion as proposed in Equation (6).

$$Tr(m) = tr_{v_2}^{(t)}(s_i) + tr_{v_1}^{(t)}(s_j) - tr_{v_1}^{(t)}(s_i) - tr_{v_2}^{(t)}(s_j) \tag{6}$$

The main challenge is now to design relevant ways of determining the greedy force (i.e. the short-term profit). At each iteration, the greedy force should focus on removing some conflicts (if possible). In order to remove a conflict, one should remove some ant-conflicts, where an *ant-conflict* occurs when two ants of conflicting value v_1 and v_2 are respectively placed on two variables s_i and s_j involved in the same constraint. Note that such an ant-conflict occurs even if $v(s_1) \neq v_1$ and $v(s_2) \neq v_2$, because as long as v_1 and v_2 are respectively represented on s_i and s_j, the procedure $ASSIGN$ might give the value v_1 to s_i and v_2 to s_j. Therefore, a move with a large greedy force value should have the ability to reduce the number of ant-conflicts, and consequently the number of conflicts. A generic way of determining the greedy forces is now depicted.

Suppose that the goal of iteration t consists in changing the value $v(s_i)$ of the conflicting decision variable s_i. All the ants of value $v(s_i)$ have thus to be removed from G_i. Therefore, a sequence of moves of type $m = (s_i, v(s_i)) \leftrightarrow (s_j, v)$ (with $i \neq j$ and $v \neq v(s_i)$) has to be performed. For such a kind of move m, the greedy force $GF^{(t)}(m)$ at iteration t can be defined by setting $GF^{(t)}(m) = A^{(t)}(m) - D^{(t)}(m)$, where $A^{(t)}(m)$ and $D^{(t)}(m)$ are respectively the advantage and disadvantage of performing move m at iteration t. Note that at each iteration t, the $A^{(t)}(m)$'s and $D^{(t)}(m)$'s can be easily normalized in interval $[0, 1]$ in order to avoid the situation where one of these components dominates too much the other.

Additional notation is now introduced to formally define $A^{(t)}(m)$ and $D^{(t)}(m)$. For iteration t, it is helpful to define the following quantities.

- $N_v^{(t)}(s_i)$: number of ants of value v in $G_i^{(t)}$;
- $G_{k,i}^{(t)}(v)$: number of ants in $G_k^{(t)}$ that are in ant-conflict with ant v belonging to $G_i^{(t)}$ (with $k \neq i$);
- $N_v^{(t)}(s_i, s_j)$: number of ants associated with decision variables – different from s_j – which are in ant-conflict with ant of value v belonging to $G_i^{(t)}$.

The smaller is $N_v^{(t)}(s_i, s_j)$, the better an ant of value v feels in G_i, as it generates a small number of ant-conflicts. Formally, $N_v^{(t)}(s_i, s_j)$ can be computed as proposed in Equation (7).

$$N_v^{(t)}(s_i, s_j) = \sum_{k | s_k \in A(s_i) - \{s_j\}} G_{k,i}^{(t)}(v) \tag{7}$$

Considering move $m = (s_i, v_i) \leftrightarrow (s_j, v_j)$, an ant of value $v_i \in G_i$ is attracted by decision variable s_j if: (1) there are several ants of value v_i in G_j; (2) there are several ants associated with decision variables – different from s_j – which are in ant-conflict with ant of value v_i belonging to G_i. Symmetric considerations are true for the ant of value v_j belonging to G_j, which is candidate to be put in G_i instead of ant of value v_i. Thus, one can define the advantage $A^{(t)}(m)$ associated with move m as expressed in Equation (8), where the components could be weighted by parameters if necessary.

$$\begin{aligned} & A^{(t)}(m) \\ = \; & N_{v_i}^{(t-1)}(s_j) + N_{v_i}^{(t-1)}(s_i, s_j) \\ + \; & N_{v_j}^{(t-1)}(s_i) + N_{v_j}^{(t-1)}(s_j, s_i) \end{aligned} \tag{8}$$

In contrast with the previous paragraph, when considering a move $m = (s_i, v_i) \leftrightarrow (s_j, v_j)$, an ant with value v_j belonging to G_j is not attracted by decision variable s_i if: (1) there are several ants of value v_j in G_j; (2) there are several ants associated with decision variables – different from s_j – which will be in ant-conflict with an ant of value v_j if it is added to G_i. Thus, one can define the disadvantage $D^{(t)}(m)$ as expressed in Equation (9), where the components could again be weighted by parameters if relevant. Note that the term $N_{v_i}^{(t-1)}(s_i)$ is not taken into account, as all ants of value s_i will be removed from G_i at iteration t.

$$D^{(t)}(m) = N_{v_j}^{(t-1)}(s_j) + N_{v_j}^{(t-1)}(s_i, s_j) + N_{v_i}^{(t-1)}(s_j, s_i) \quad (9)$$

IV. Conclusion

In this paper is designed ADS (Ant Decision System), a new type of ant algorithm, specifically dedicated to combinatorial optimization with binary constraints. As in every ant algorithm, the selection of a decision relies on two ingredients: the greedy force on the one hand, and the trail system on the other hand. The former element represents the self-adaptation ability of an ant, whereas the latter element is a central memory containing relevant information on the history of the search process. In ADS, in contrast with most of the state-of-the-art ant algorithms, the role of each ant is limited to help to assign a value to a single decision variable. Relevant formulas are proposed to design the greedy force and the trail of a decision. Among the future works in this area, one could develop ADS approaches for specific combinatorial optimization problems in telecommunication networks, for which the frequency constraints are often binary. Another avenue of research relies in the generalization of the ADS approach for ternary constraints.

References

[1] E. Malaguti and P. Toth, "A survey on vertex coloring problems," *International Transactions in Operational Research*, vol. 17 (1), pp. 1 – 34, 2010.

[2] K. I. Aardal, S. P. M. van Hoesel, A. M. C. A. Koster, C. Mannino, and A. Sassano, "Models and Solution Techniques for Frequency Assignment Problems," *4OR*, vol. 1:4, pp. 261 – 317, 2003.

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

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

[5] M. Gendreau and J.-Y. Potvin, *Handbook of Metaheuristics*, ser. International Series in Operations Research & Management Science. Springer, 2010, vol. 146.

[6] N. Zufferey, "Metaheuristics: some Principles for an Efficient Design," *Computer Technology and Applications*, vol. 3 (6), pp. 446 – 462, 2012.

[7] ——, "Optimization by ant algorithms: Possible roles for an individual ant," *Optimization Letters*, vol. 6 (5), pp. 963 – 973, 2012.

[8] A. Hertz and N. Zufferey, "A New Ant Colony Algorithm for Graph Coloring," in *Proceedings of the Workshop on Nature Inspired Cooperative Strategies for Optimization, NICSO 2006, June 29–30*, Pelta and Krasnogor, Eds., Granada, Spain, 2006, pp. 51–60.

[9] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *Theoretical Computer Science*, vol. 344 (2-3), pp. 243–278, 2005.

[10] F. Glover, "Tabu search - part I," *ORSA Journal on Computing*, vol. 1, pp. 190–205, 1989.