ACO for Maximal Constraint Satisfaction Problems

Andrea Roli * † Christian Blum † Marco Dorigo †

* DEIS - Università di Bologna
Viale Risorgimento, 2 - Bologna (Italy)
Email: aroli@deis.unibo.it

† IRIDIA - Université Libre de Bruxelles
Av. Roosvelt 50 - Bruxelles (Belgium)
Email: {aroli, cblum, mdorigo}@ulb.ac.be

1 Introduction

The Ant Colony Optimization metaheuristic (ACO) is a recent metaheuristic [2] that has been successfully applied to a number of combinatorial optimization problems (see [3] for an overview). In this paper we present and discuss an application of ACO to Constraint Satisfaction Problems (CSPs).

1.1 CSPs

A CSP is formally defined as a triple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is the set of variables, $D = \{D_1, \ldots, D_n\}$ is the set of domains which define the values a variable can assume[1] and $C = \{C_1, \ldots, C_m\}$ is the set of constraints among the variables. The CSP is a decision problem: a solution of the problem is a complete assignment which satisfies all the constraints (see [12] for an overview of CSPs). Here we are interested in solving the maximal-CSP, defined as the problem of finding an assignment that satisfies the greatest number of constraints. Often, weights are associated to constraints and the goal is to maximize the sum of the weights belonging to satisfied constraints.

1.2 The Ant Colony Optimization metaheuristic

Ant Colony Optimization is a metaheuristic for designing algorithms for combinatorial optimization problems [2, 4, 3, 11]. In ACO, artificial ants construct a solution by building a path on a *construction graph* $\mathcal{G} = (\mathcal{C}, \mathcal{L})$ where the elements of $\mathcal{L}$ (called *connections*) fully connect $\mathcal{C}$ (set of *components*). Artificial pheromone can be associated either to components (nodes) or connections (edges). The ants behavior is specified by defining *start states* and *termination conditions*, *construction rules*, *pheromone update rules* and *daemon actions* (for a detailed and formal description see [2]). In this paper we use the Ant Colony System algorithm (ACS) [4], a particular instance of ACO (see Fig. 1 for the basic algorithm). In ACS, each ant is initially positioned on a randomly chosen node of $\mathcal{G}$ and builds a solution by applying a probabilistic rule, called *state transition rule*. This probabilistic rule is biased by pheromone values so that the higher the pheromone on a component/connection, the higher the probability it will be selected. While building the solution the ants "eat" some quantity of pheromone on the visited components/connections (this is called *step-by-step pheromone update*). After every ant

[1] In the following we consider only *finite domain problems* (i.e., the cardinality of each domain is finite).

```
procedure ACS
begin
    Initialize
    while stopping criterion not satisfied do
        Position each ant in a starting node
        repeat
            for each ant do
                Choose next node by applying the state transition rule
                Apply step-by-step pheromone update
            end for
        until every ant has built a solution
        Update best solution
        Apply offline pheromone update
    end while
end
```

Figure 1: High-level description of the ACS algorithm.

has completed a solution, the *offline pheromone update* is applied to the components/connections of the best solution found so far, by adding a quantity of pheromone function of the quality of the solution.

2 Where to put pheromone?

In this paper, we adopt the following representation for a CSP: the construction graph $\mathcal{G} = (\mathcal{C}, \mathcal{L})$ (see Fig. 2) is defined such that nodes (components) are the pairs $(variable, value)$ and edges (connections) fully connect the nodes. A solution is a sequence of n nodes, being n the number of variables. Ants construct a solution by probabilistically choosing a node in their feasible neighborhood; in this work, we define the feasible neighborhood of an ant k as the set of pairs $(variable, value)$ such that the variable has not yet been assigned a value. The neighborhood chosen therefore implements the problem constraints that say that nodes associated to the same variable are not in a same solution. A key point for the algorithm designer is the choice of the graph elements (components and connections) to which to associate pheromone. This topic will be discussed in the following.

Pheromone on components

A first possibility is to put pheromone on components. In this case, the amount of pheromone is proportional to the (learned) desirability of having a particular assignment in the solution. With this choice, the ACS state transition rule used by ant k (called *pseudo-random-proportional rule*, see [4])[2] becomes:

$$s = \begin{cases} arg\ max_{u \in J_k}\{\tau(u)\}, & \text{if } q \leq q_0 \\ z, & \text{otherwise} \end{cases} \qquad p_k(z) = \begin{cases} \frac{\tau(z)}{\sum_{u \in J_k} \tau(u)}, & \text{if } z \in J_k \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where s is the next node, $\tau(u)$ is the pheromone value on node u, J_k the set of nodes in the feasible neighborhood of ant k, q is a random variable uniformly distributed in [0,1], q_0 ($0 \leq q_0 \leq 1$) a parameter of the algorithm and z is chosen with probability p_k.

The pheromone update rules (step-by-step and offline) are:

Step-by-step pheromone update: $\forall$ ant $k, \forall\ s \in S_k : \tau(s) \leftarrow (1-\rho) \cdot \tau(s) + \rho \cdot \tau_0$,

Offline pheromone update: $\forall\ s \in S_{opt} : \tau(s) \leftarrow (1-\alpha) \cdot \tau(s) + \alpha \cdot g(S_{opt})$,

[2]The general rule includes also a heuristic function η. For simplicity we do not introduce it in this discussion; we just observe that instead of $\tau(\cdot)$ we should write $\tau(\cdot)[\eta(\cdot)]^\beta$, where β is a parameter to adjust the relative importance of pheromone values and heuristic information.

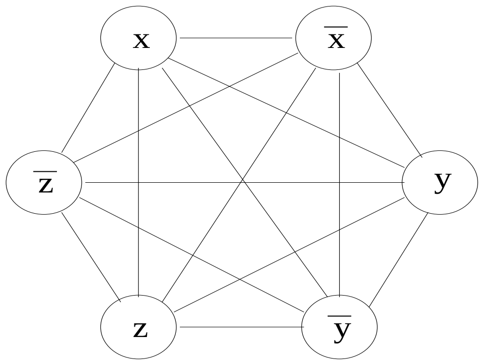

Figure 2: Construction graph for a CSP with binary variables. The correspondences between nodes and assignments are: $(x,1) \leftrightarrow x, (x,0) \leftrightarrow \overline{x}, (y,1) \leftrightarrow y, (y,0) \leftrightarrow \overline{y}, (z,1) \leftrightarrow z, (z,0) \leftrightarrow \overline{z}$.

where S_k is the solution constructed by ant k, S_{opt} is the best solution found so far, $\rho, \alpha \in [0,1]$, $\tau_0 \in \mathbb{R}^+$ are parameters of the algorithm and $0 < g(S_{opt}) < +\infty$ is a nondecreasing monotonic function of the quality of the solution S_{opt}.

Pheromone on connections

An alternative possibility is to associate pheromone with connections. Thus, the quantity of pheromone on the connections between two nodes is proportional to the (learned) benefit of having the two corresponding assignments in the solution. For ant k which moves from node r to node s, rule (1) becomes:

$$s = \begin{cases} arg\ max_{u \in J_k}\{\tau(r,u)\}, & \text{if } q \leq q_0 \\ z, & \text{otherwise} \end{cases} \qquad p_k(r,z) = \begin{cases} \frac{\tau(r,z)}{\sum_{u \in J_k} \tau(r,u)}, & \text{if } z \in J_k \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where $\tau(r,u)$ is the pheromone level on connection (r,u). The pheromone update rules are an obvious extension of the previous ones to pheromone on connections (edges).

Pheromone on connections (with sums)

A last possibility considered in this paper takes into account the dependence of an assignment on those already done. To do this, (1) is changed as follows:

$$s = \begin{cases} arg\ max_{u \in J_k}\{\sum_{w \in S_k'} \tau(w,u)\}, & \text{if } q \leq q_0 \\ z, & \text{otherwise} \end{cases} \qquad p_k(r,z) = \begin{cases} \frac{\sum_{w \in S_k'} \tau(w,z)}{\sum_{u \in J_k} \sum_{w \in S_k'} \tau(w,u)}, & \text{if } z \in J_k \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where S_k' denotes the set of nodes of the current partial path. Formulas (3) consider the sum of pheromone values on the edges which connect the candidate node to all the nodes already in the solution. The rules for pheromone update are changed so that the connection between every couple of nodes in the solution is updated.

Table 1: ACS-comp, ACS-conn and ACS-conn+. For the *jnh* instances we report the average error from the optimal solution (in percentage), the average computational time (in seconds) and the average number of iterations for the best solution found. In the *uuf* instances we reported the best average solution. The averages are evaluated for 100 runs. The algorithms were run on a Pentium II at 400MHz, with 512 MB of RAM and 512 KB of cache memory.

algorithm	instance	average error	std.dev.	average time	std.dev.	average iter.	std.dev.
ACS-comp	jnh1	0.558	0.084	12.811	3.056	76.530	18.193
	jnh201	0.568	0.086	11.940	3.175	74.810	19.882
	jnh301	0.777	0.127	14.809	2.949	81.710	16.299
	uuf250-01	1038.19	2.13	32.56	12.65	63.21	24.46
	uuf250-02	1031.35	2.37	36.01	10.78	70.85	21.23
ACS-conn	jnh1	0.878	0.122	80.849	31.214	64.40	24.879
	jnh201	0.885	0.120	78.676	29.035	66.670	24.517
	jnh301	1.188	0.167	81.245	34.218	61.860	26.109
	uuf250-01	1030.90	2.10	133.35	96.22	40.59	29.31
	uuf250-02	1023.97	1.89	174.03	95.76	51.95	28.68
ACS-conn+	jnh1	0.583	0.077	195.78	52.71	76.63	20.54
	jnh201	0.570	0.118	188.49	50.93	75.09	20.18
	jnh301	0.780	0.139	205.68	38.03	78.86	14.56
	uuf250-01	1037.37	2.47	1843.33	785.08	61.99	26.51
	uuf250-02	1031.11	2.54	2079.18	720.81	70.21	24.49

3 Results

We applied ACS to the MAX-SAT problem, a typical NP-hard max-CSP. MAX-SAT is the maximization version of the Satisfiability Problem (SAT) [6]. The following definition is one of the most common ones: given a set of clauses, each of which is the logical disjunction of $k > 2$ *literals* (i.e., a variable negated or not), we ask whether an assignment to the variables exists that satisfies all the clauses. In MAX-SAT the problem is to find an assignment that satisfies the greatest number of clauses, or, in the weighted version, the assignment that maximizes the sum of weights of the satisfied clauses. We implemented three algorithms: ACS-comp (pheromone on components), ACS-conn (pheromone on connections) and ACS-conn+ (pheromone on connections with sums). The parameter settings were: $\rho = \alpha = 0.01$, $\beta = 1$, $\tau_0 = 0.1$, $q_0 = 0.8$ and 10 ants; we stopped the algorithms after 100 iterations. We tested the algorithms on weighted and unweighted MAX-SAT instances[3]. The average quality of solutions produced by ACS-comp and ACS-conn+ is very similar (see Table 1), while ACS-conn generates on average slightly worse solutions. The behavior of ACS-conn suggests that considering only pairs of assignments can be misleading. The average number of iterations does not change much among the algorithms (ACS-conn shows an average slightly lower than the other algorithms). If we consider the average computation time we note that the use of a matrix of pheromone values causes a high load in the computation. The fact that ACS-conn+ does not perform significantly better than ACS-comp suggests that, for random instances, learning inter-variable correlations does not seem advantageous.

We also tested ACS using local search as a daemon action. As local search we developed a MAX-SAT version of GSAT, a local search algorithm for SAT [10]. Local search is applied to each solution generated by the ants. The introduction of local search boosts the performance of the algorithms (they find better solutions than the algorithms without local search in the same amount of execution time) and flattens the differences in the quality of solutions produced (the average error from the optimal solution is statistically equivalent for the three algorithms).

[3]The weighted instances (2nd DIMACS benchmark set) can be retrieved from: `http://www.research.att.com/~mgcr/data/maxsat.tar.gz`.
Unweighted instances (SATLIB): `http://www.intellektik.informatik.tu-darmstadt.de/SATLIB`.

4 Related work and future directions

The first, non implemented, description of an application of ACO to CSPs was presented in [1]. An approach that fits the one described in Section 2 has been introduced in [8]. That work is an implementation of an Ant System [5] with pheromone on connections (with sums) and dynamic heuristic; ants first select a variable (randomly), then they apply the state transition rule to select the value to assign. The way of assigning values to variables used in [8] can be covered by our approach if we change the definition of feasible neighborhood for the ants. Different approaches which do not exactly fit into the ACO paradigm are presented in [9, 7]. Future work is directed to the extension of the presented representations to other combinatorial optimization problems with a CSP formulation and to problems with a 0-1 formulation (like the knapsack problem). Furthermore, we are currently implementing ACS in a *reduced graph representation*, suited for problems with binary variables, whereby a node in the construction graph represents the binding ($variable$,1).

Acknowledgments: This work was supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, contract HPRN-CT-1999-00106. Andrea Roli acknowledges support from the CEC through a "Marie Curie Training Site" fellowship, contract HPMT-CT-2000-00032. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication. Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a Senior Research Associate.

References

[1] M. Dorigo. *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, DEI, Politecnico di Milano,Italy, 1992. pp. 140.

[2] M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999.

[3] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Art. Life*, 5(2):137–172, 1999.

[4] M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evol. Comp.*, 1(1):53–66, 1997.

[5] M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: Optimization by a colony of cooperating agents. *IEEE Tran. Sys., Man and Cyb. - Part B*, 26(1):29–41, 1996.

[6] M. R. Garey and D. S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[7] A. Løkketangen. Satisfied ants. In *Abst. Proc. ANTS'2000*, 2000.

[8] S. Pimont and C. Solnon. A generic ant algorithm for solving constraint satisfaction problems. In *Abst. Proc. ANTS'2000*, 2000.

[9] L. Schoofs and B. Naudts. Solving CSPs with ant colonies. In *Abst. Proc. ANTS'2000*, 2000.

[10] B. Selman, H. J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proc. 10th Nat. Conf. Art. Int.*, pages 440–446, Menlo Park, California, 1992. American Association for Artificial Intelligence, AAAI Press.

[11] T. Stützle and M. Dorigo. ACO algorithms for the quadratic assignment problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 33–50. McGraw-Hill, 1999.

[12] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1991.