A Parallel Architecture for Solving Constraint Satisfaction Problems

*Rui Mendes** *Jorge R. Pereira** *José Neves**

* Departamento de Informática
Universidade do Minho
Email: azuki@di.uminho.pt, jrp@omega.di.uminho.pt, jneves@di.uminho.pt

1 Introduction

Real world problems can often be described by a set of constraints to be satisfied, where the goal is to find a feasible solution for the problem. The use of constraints allows one to model a wide variety of problems in a straightforward manner. However, finding a solution that either satisfies all constraints or maximizes some benefit is usually difficult, as existing algorithms for both problems are $\mathcal{NP}$-hard. Furthermore, it is not always possible to satisfy all the constraints. In such cases, the goal is to find a solution that satisfies the maximum number of constraints (*MAX-CSPs*) or one that satisfies the most important ones (*MAX-WCSPs*, where each constraint has a *weight*).

Currently, local search is widely used to tackle these difficult problems. However, those methods are usually incomplete (i.e., they do not guarantee the optimum), and are often mislead by local optima. This situation is usually handled by restarting the search from another starting point.

It is our goal in this paper to present a parallel architecture for solving constraint satisfaction problems (*AntCSP*); i.e., a parallel architecture that combines the stigmergetic capabilities of Ant Colony Optimization (*ACO*) metaheuristics, with local search heuristics to solve *MAX-CSPs* and Constraint Satisfaction and Optimization Problems (*CSOPs*).

One of the main advantages of this approach is that no auxiliary structure is needed. The structure followed by the ants is the proof tree itself, in which the pheromones are laid. This not only provides a very efficient implementation of pheromone updating but also a more general approach to solve *CSPs*.

The other advantage is the use of parallelism both to have a number of ant colonies running in parallel and to have function distribution of local search procedures. This architecture thus provides a powerful tool to tackle these difficult problems by using available processing power.

2 Definitions

A *CSP* is defined by a triple (X, D, C), such that

- $X = \{X_1, X_2, \ldots, X_n\}$ is a finite set of n variables;
- D is a function that maps every variable $X_i \in X$ to its domain $D(X_i)$; i.e., the set of values that can be assigned to X_i;

- C is a set of constraints. A constraint is a relation between variables, which restricts the set of values that can be assigned simultaneously to those variables.

An assignment is defined as the set $\mathcal{A} = \{< X_i, v_i >: X_i \in X \wedge v_i \in D(X_i)\}$. It represents the assignment of the corresponding value v_i to variable X_i for each variable. A partial assignment only assigns values to some of the variables.

The number of conflicts of an assignment $\mathcal{A}$ is defined as the number of constraints unsatisfied by the assignment.

A *Binary CSP* has only binary constraints between variables. A *Random Binary CSP* [7] is of the form $< n, k, p1, p2 >$ where p1 is the probability of connectivity (i.e., having a constraint between two variables) and p2 the probability of tightness (i.e., making a pair of values inconsistent).

3 Local Search

Local search approaches to *CSPs* iteratively construct a complete assignment and then repair it. The local repair phase consists of repeatedly changing the assignment of a conflicting variable (i.e., a variable that is involved in some unsatisfied constraints) until either a solution is found, a maximum number of repairs have been performed or a local optimum has been reached.

Different heuristics exist to choose the variable to be repaired and its new assignment. In *GSAT* [11], the idea is to select the new variable/value assignment which most increases the number of satisfied constraints (ties are broken randomly). In the min-conflict heuristic [9], the idea is to randomly select a conflicting variable, and then choose a new value that minimizes the number of conflicts (ties are broken randomly).

4 Ant Colony Optimization

Ant colony optimization (see [4] for an overview) is a metaheuristic that has been successfully applied to several $\mathcal{NP}$-hard combinatorial optimization problems [3, 2, 5, 8]. In analogy to the biological example, *ACO* is based in artificial pheromone trails as means of communication between simple agents of a colony. The pheromone trails serve as a distributed, numerical information used by the ants to probabilistically construct solutions. They are adapted during the algorithm's execution to reflect the search experience of the ants.

5 The Architecture

5.1 Problem Structure and *ACO* Algorithm

In *AntCSP*, the data structure used by the ants, in their search for solutions, is a direct acyclic graph; i.e. a tree, where nodes correspond to alternative assignments to the variables.

Other approaches involving *ACO* and *CSPs* [12, 10] use a structure based on the *Binary CSP*. For instance, in [12], a *LabelGraph* is used, which consists of a node for each variable/value pair and edges between all pairs of nodes not belonging to the same variable. The approach undertaken in [10] is similar in regard to the effort spent in pheromone updating and in choosing which branch to follow.

The number of edges to be updated in a cycle is $\frac{n(n-1)}{2}$ where n stands for the number of variables.

When deciding on a new path, the number of edges consulted is $\frac{k\,n\,(n-1)}{2}$, where k denotes the number of values. Using a tree, n nodes are updated and $n\,k$ branches are consulted when deciding on a new path.

One of the apparent disadvantages of using a tree is that its size increases exponentially as the problem grows in complexity. However, experimental results obtained so far indicate that, due to pheromone evaporation, the number of existing branches with a value greater than the minimum allowed pheromone value (τ_{min}) is minimal. Therefore, in *AntCSP*, only branches with a pheromone value greater than τ_{min} are represented. The nodes reaching τ_{min} are marked for removal but are only effectively removed when storage space becomes an issue. Hence, if they are revisited before effective deletion, their information (i.e., local visibility, available paths) is still available, with no need for further computation.

For instance, in a binary random *CSP* of 50 variables and 5 values for each variable, with connectivity and tightness probabilities of 50% (a tree of size $5^{50} \simeq 10^{80}$) the maximum number of edges in memory was close to 90000, which is a ratio of $\frac{1}{10^{70}}$. In a problem with 200 variables and all other parameters being the same, the maximum number of edges was close to 230000.

AntCSP uses a fixed variable ordering, established by heuristics; e.g., the smallest-domain ordering, one of the most successful for *MAX-CSP* problems. At each node on the tree, the ant has to decide which branch to follow. The decision is taken considering both the pheromone (τ) already present at the node and the local visibility (η) (given by an heuristic). The choice of a branch will determine which value will be assigned to the variable. This choice is made probabilistically, according to the formula:

$$P(v) = \frac{\tau_v^\alpha \eta_v^\beta}{\sum_{w \in D(X)} \tau_w^\alpha \eta_w^\beta}$$

where α and β are parameters that respectively control the relevance of the pheromone (τ) and the local visibility heuristic (η). If the number of branches is too big, candidate lists can be used. A second option that comes naturally from a traversal of the proof tree is the successive partition of the values into smaller sets.

Once the assignment is constructed, a local search procedure is applied in order to improve it. The pheromone updating procedure closely follows the ideas from $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System [14]; i.e., pheromones are only laid on the best path found in the cycle. Also, lower and upper bounds τ_{min} and τ_{max} are imposed on the pheromone trail, such that $0 \leq \tau_{min} \leq \tau_{max}$.

The pheromones are initialized at τ_{max}, to favor a larger exploration of the search tree in the first cycles. By establishing lower and upper bounds, it is possible to prevent some paths from becoming highly attractive, thus avoiding early convergence to a local optima.

The pheromone update is done according to $\tau \leftarrow \rho\,\tau + \Delta_\tau$ if the branch was followed by the best ant, or $\tau \leftarrow \rho\,\tau$ otherwise. ρ stands for the trail persistence parameter, such that $0 \leq \rho \leq 1$ and Δ_τ denotes the quantity of pheromone deposited.

In the case of *MAX-CSPs*, $\Delta_\tau = \frac{1}{1+conflicts(\mathcal{A})}$, where $\mathcal{A}$ denotes the assignment corresponding to the path of the ant. The local visibility (η) is calculated the same way, but with respect to the new conflicts arising from the assignment of the value to the variable.

5.2 Parallelism

As shown in [13], simply running several colonies in parallel, with no communication among them, can be very effective. Several approaches were made to coarse-grained parallelization schemes. It has been shown in [6] that it was better to communicate the best solutions and use them in the pheromone update.

The present approach uses several ant colonies where solutions are exchanged every fixed number of cycles. Furthermore, as the local search procedures can be quite time-consuming in large problems, those tasks are delegated to dedicated processes. It is thus possible to speed up the search considerably and attack large problems.

6 Experimental Results

To have an idea of the number of nodes created with exploration, an experiment was run with a random binary CSP with 100 variables and 20 values. Figure 1 shows the number of nodes created. As one can see, there is an abrupt descent in the number of nodes after the initial iterations. This is due to the pheromone evaporation mechanism.

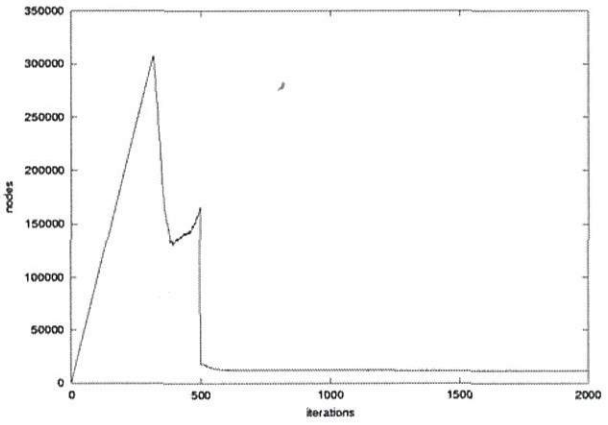

Figure 1: Evolution of the number of nodes with the number of iterations

The architecture was tested with several instances of random binary CSPs with 50 variables and 5 values per variable. The tightness and density of the constraints were set to 50%. The ACO parameters chosen were $\alpha = 1$, $\beta = 5$, $\rho = 0.98$ and $n = 10$. The parallel experiments used 10 colonies, exchanging solutions every 10 iterations. The maximum number of iterations was 500. The sequential experiments used the same parameters, with a maximum number of iterations of 5000. The results were obtained over an average of 100 runs. Table 1 illustrates the results, using one problem. Checkpoints were made at several points in the sequential system to compare the performance.

Algorithm/Iterations	Average	Error
Parallel	389.3	2.15
Sequential: 500	393.3	2.72
Sequential: 1000	391.8	1.61
Sequential: 2000	390.1	2.02
Sequential: 3000	389.8	1.94
Sequential: 4000	389.8	1.94
Sequential: 5000	389.7	1.99

Table 1: Results of AntCsp versus the sequential system

Looking at the results, one can see that the parallel system shows speedups close to the number of processors used. Even after running for 5000 iterations, the sequential system was still unable to reach the solutions obtained by the parallel one.

7 Conclusions and Further Work

The present architecture is capable of searching solutions for large problems. The parallelization of colonies and local search processes allows for considerable speedups. It can use the processing power of a workstation network to parallelize the necessary tasks. It also provides an excellent framework to investigate the use of different local search procedures in combination with the *ACO* metaheuristic.

Further research is in progress to develop a learning system capable of choosing the best local search procedure for each problem, as well as a way of estimating values for *ACO* parameters.

We are also combining *AntCSP* with a constraint logic programming environment, *ECLiPSe* [1], to provide a way to describe *CSPs* in a high level language. This will allow for a high level description of the problems to be solved.

References

[1] Abderrahamane Aggoun, David Chan, Pierre Dufresne, Eamon Falvey, Hugh Grant, Warwick Harvey, Alexander Herold, Geoffrey Macartney, Micha Meier, David Miller, Shyam Mudambi, Stefano Novello, Bruno Perez, Emmanuel van Rossum, Joachim Schimpf, Kish Shen, Periklis Andreas Tsahageas, and Dominique Henry de Villeneuve. *ECLiPSe user manual*, 2000.

[2] G. Di Caro and M. Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research (JAIR)*, 6:791–812, dec 1998.

[3] M. Dorigo and L.M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[4] Marco Dorigo and Gianni Di Caro. The ant colony optimization meta-heuristic. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.

[5] L.M. Gambardella and M. Dorigo. HAS-SOP: Hybrid Ant System for the Sequential Ordering Problem. Technical Report IDSIA 11-97, IDSIA, Lugano, Switzerland, 1997.

[6] F. Krüger, D. Merkle, and M. Middendorf. Studies on a Parallel Ant System for the BSP Model. Unpublished manuscript.

[7] Ewan MacIntyre, Patrick Prosser, Barbara M. Smith, and Toby Walsh. Random constraint satisfaction: Theory meets practice. In *Principles and Practice of Constraint Programming*, pages 325–339, 1998.

[8] R. Michel and M. Middendorf. An Island Based Ant System with Lookahead for the Shortest Common Supersequence Problem. In *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498, pages 692–708. Springer Verlag, 1998.

[9] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence*, 58:161–205, 1992.

[10] Luk Schoofs and Bart Naudts. Ant colonies are good at solving constraint satisfaction problems. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 1190–1195, Piscataway, NJ, 2000. IEEE Service Center.

[11] B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of AAAI92*, 1992.

[12] C. Solnon. Ants can solve Constraint Satisfaction Problems. *Submitted to IEEE Transactions on Evolutionary Computation*, feb 2001.

[13] T. Stützle. Parallelization Strategies for Ant Colony Optimization. In Agoston E. Eiben, Thomas Bäch, Mark Schoenauer, and Hans-Paul Schwefel, editors, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of Lecture Notes in Computer Science, pages 722–731, Berlin, Germany, 1998. Springer Verlag.

[14] T. Stützle and H.H. Hoos. $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. *Journal of Future Generation Computer Systems*, 16:889–914, 2000.