Boosting Local Search with Artificial Ants

Christine Solnon

LISI - University Lyon 1 - 43 bd du 11 novembre
69 622 Villeurbanne cedex, France
csolnon@bat710.univ-lyon1.fr

Local Search: Incomplete approaches for solving CSPs are usually based on local search —or neighborhood search— techniques [4]: the idea is to start from an inconsistent complete assignment of values to the variables, and then gradually and iteratively repair it by changing some variable-value assignments, preferably towards better ones. One of the main problems with local search is that it may get stuck in local optima, i.e., complete assignments that cannot be locally improved by changing one conflicting variable/value assignment, and that are not globally optimal. Therefore, local search has been combined with different meta-heuristics in order to help it escape from local optima, e.g., simulated annealing or tabu search [2]. Local search has proved to be effective and efficient to solve very large CSPs. However, like complete search, it often has more difficulties in solving problems that are within the phase transition region —where the solvable probability is around 50%. Indeed, before the phase transition region, problems are weakly constrained and have many solutions so that local search can usually easily find one. On the other side, beyond the phase transition region, problems are hardly constrained and only have few solutions, but they also have much less local optima so that local search can more easily reach a solution without being trapped in local optima [7]. Between these two "easy" regions, search space landscapes of problems contain more local minima so that local search is more often trapped in these local minima and, even when using some meta-heuristics for escaping from them, local search often "walks" from a local minimum to another without finding a global minimum.

Motivations: A main motivation of the work presented in this paper is to provide a way of taking benefit of the different local minima found by local search in order to guide it towards the most promising states of the search space. Indeed, a study of the shape of the search space of many combinatorial optimization problems has shown a high correlation between the quality of a local minimum and its distance to the closest global minimum, i.e., the better the local minimum, the closer to a global minimum [3]. To perform this task, we use the Ant Colony Optimization (ACO) metaheuristics [1]. The main idea of ACO is to model the problem as the search of a best path in a graph that represents the states of the problem. Artificial ants walk through this graph, looking for good paths. They communicate by laying pheromone trails on edges of the graph, and they choose their path with respect to probabilities that depend on the amount of pheromone previously left. In our context of CSP solving, we

T. Walsh (Ed.): CP 2001, LNCS 2239, pp. 620–624, 2001.

propose to use pheromone to keep track of the best local minima found, in order to guide the search when constructing new assignments to be repaired, as heuristics for choosing values to be assigned to variables. One should remark that this approach is complementary to other meta-heuristics, such as simulated annealing or tabu search, that aim at helping local search to escape from local optima.

Definitions and notations: A *CSP* is defined by a triple (X, D, C) such that X is a finite set of variables, D is a function which maps every variable to its finite domain and C is a set of constraints. A *label*, denoted by $<X_i, v_i>$, is a variable-value pair which represents the assignment of value v_i to variable X_i. A *compound label*, denoted by $\mathcal{A} = \{<X_1, v_1>, \ldots, <X_k, v_k>\}$, is a set of labels and corresponds to the simultaneous assignment of values $v_1, \ldots, v_k$ to variables $X_1, \ldots, X_k$ respectively. A *complete compound label* is a compound label that contains a label for each variable of the CSP. The *valuation* of a compound label $\mathcal{A}$ is defined by the number of violated constraints in $\mathcal{A}$. A *solution of a CSP* is a complete compound label the valuation of which is 0.

Many real-life CSPs are over-constrained, so that no solution exists. Hence, the general CSP framework has been generalized to the partial or maximal constraint satisfaction problem —*max-CSP*. In this case, the goal is no longer to find a consistent solution, but to find a complete assignment which maximizes the number of satisfied constraints. In this paper, we implicitely solve max-CSPs.

Overall description of Ant-solver. We propose to use the ACO meta-heuristics for guiding local search —when constructing a new complete assignment to be repaired— towards the most promising areas of the search space. The algorithm, called Ant-solver, is sketched in figure 1.

```
procedure Ant-solver(X, D, C)
    τ ← InitializePheromoneTrails()
    repeat
        for k in 1..nbAnts do
            A_k ← ∅
            while | A_k |<| X | do
                X_j ← SelectVariable(X, A_k)
                v ← ChooseValue(τ, X_j, D(X_j), A_k)
                A_k ← A_k ∪ {<X_j, v>}
            end while
            A_k ← ApplyLocalSearch(A_k)
        end for
        τ ← UpdatePheromoneTrails(τ, {A_1, ..., A_nbAnts})
    until valuation(A_i) = 0 for some i ∈ {1..nbAnts} or max cycles reached
```

Fig. 1. Ant-solver algorithmic scheme

At each cycle of this algorithm, every ant constructs a complete assignment $\mathcal{A}_k$, i.e., it iteratively selects a variable X_j and chooses a value v for this variable. Then, the constructed assignment is improved by applying some local search techniques. Finally, pheromone trails are updated with respect to the different local minima computed during the current cycle. We shall now briefly describe the pheromone graph on which artificial ants lay pheromone trails, and the different functions used in Ant-solver. More details can be found in [6].

Pheromone graph: The pheromone graph associates a vertex with each variable/value pair $<X_i, v>$ such that $X_i \in X$ and $v \in D(X_i)$. There is a non oriented edge between any pair of vertices corresponding to two different variables. The amount of pheromone laying on an edge $(<X_i, v>, <X_j, w>)$ is noted $\tau(<X_i, v>, <X_j, w>)$. Intuitively, this amount of pheromone represents the learned desirability of assigning simultaneously value v to variable X_i and value w to variable X_j.

SelectVariable($X, \mathcal{A}_k$): This function returns a variable $X_j \in X$ that is not yet assigned in $\mathcal{A}_k$. This choice can be performed randomly, or with respect to some commonly used variable ordering, such as the *smallest-domain ordering*, which selects a variable that has the smallest number of consistant values with respect to some given partial consistency.

ChooseValue($\tau, X_j, D(X_j), \mathcal{A}_k$): This function returns a value $v \in D(X_j)$ to be assigned to X_j. The choice of v is done with respect to a probability $p(v, \tau, X_j, D(X_j), \mathcal{A}_k)$ which depends on two factors: the pheromone factor $\mathcal{P}$ — which evaluates the learned desirability of v— and the quality factor $\mathcal{Q}$ —which evaluates the number of conflicts of v with the already assigned variables:

$$p(v, \tau, X_j, D(X_j), \mathcal{A}_k) = \frac{[\mathcal{P}(\tau, \mathcal{A}_k, X_j, v)]^\alpha [\mathcal{Q}(\mathcal{A}_k, X_j, v)]^\beta}{\sum_{w \in D(X_j)} [\mathcal{P}(\tau, \mathcal{A}_k, X_j, w)]^\alpha [\mathcal{Q}(\mathcal{A}_k, X_j, w)]^\beta}$$

where α and β are two parameters which determine the relative importance of pheromone and quality factors; the pheromone factor $\mathcal{P}(\tau, \mathcal{A}_k, X_j, v)$ corresponds to the sum of all pheromone trails laid on all edges between $<X_j, v>$ and the labels in $\mathcal{A}_k$, i.e., $\mathcal{P}(\tau, \mathcal{A}_k, X_j, v) = \sum_{<X_l, m> \in \mathcal{A}_k} \tau(<X_l, m>, <X_j, v>)$; and the quality factor $\mathcal{Q}(\mathcal{A}_k, X_j, v)$ is inversely proportional to the number of new violated constraints when assigning value v to variable X_j, i.e., $\mathcal{Q}(\mathcal{A}_k, X_j, v) = 1/(1+valuation(\{<X_j, v>\} \cup \mathcal{A}_k) - valuation(\mathcal{A}_k))$.

ApplyLocalSearch($\mathcal{A}_k$): This function allows one to improve the constructed assignment $\mathcal{A}_k$ by performing some local search, i.e., by iteratively changing some variable-value assignments. Different heuristics can be used to choose the variable to be repaired and the new value to be assigned to this variable (see, e.g., [2] for an experimental comparison of some of these heuristics). The approach proposed in this paper can be applied to any local search algorithm for solving CSP and is independant from heuristics used to select the repair to be performed.

UpdatePheromoneTrails($\tau, \{\mathcal{A}_1, \ldots, \mathcal{A}_{nbAnts}\}$): This function updates the amount of pheromone laying on each edge according to the ACO meta-heuristics, i.e., all pheromone trails are uniformely decreased —in order to simulate some kind of evaporation that allows ants to progressively forget worse paths— and then pheromone is added on edges participating to the construction of the best local minimum —in order to further attract ants towards the corresponding area of the search space. Hence, at the end of each cycle, the quantity of pheromone laying on each edge (i, j) is updated as follows:

$\tau(i,j) \leftarrow \rho * \tau(i,j)$
if $i \in \mathcal{A}_{Best}$ and $j \in \mathcal{A}_{Best}$ then $\tau(i,j) \leftarrow \tau(i,j) + \frac{1}{valuation(\mathcal{A}_{Best})}$
if $\tau(i,j) < \tau_{min}$ then $\tau(i,j) \leftarrow \tau_{min}$
if $\tau(i,j) > \tau_{max}$ then $\tau(i,j) \leftarrow \tau_{max}$

where ρ is the trail persistence parameter such that $0 \leq \rho \leq 1$, $\mathcal{A}_{Best}$ is the best assignment of $\{\mathcal{A}_1, \ldots, \mathcal{A}_{nbAnts}\}$, and τ_{min} and τ_{max} are bounds, such that $0 \leq \tau_{min} \leq \tau_{max}$.

InitializePheromoneTrails(): Pheromone trails can be initialized to a constant value, e.g., $\tau_{\max}$, as proposed in [5]. However, Ant-solver can be boosted by introducing a preprocessing step. The idea is to collect a significant number of local minima by performing "classical" local search, i.e., by iteratively constructing complete compound labels —without using pheromone— and repairing them. For easy problems, that are far enough from the phase transition region, local search usually quickly find solutions, so that this preprocessing step stops iterating on a success, and the whole algorithm terminates. However, for harder problems within the phase transition region, local search may be successively trapped in local minima without finding a solution. In this case, the goal of the preprocessing step is to collect a representative set of local minima, thus constituting a kind of sampling of the search space. Then, we select from this sample set the best local minima and we use them to initialize pheromone trails.

Experiments on Random Binary CSPs: Ant-solver has been implemented in C++. For all experiments reported below, we have used the smallest-domain ordering as the variable selection rule, and the min-conflict heuristics [4] for the local search procedure. Parameters have been setted to $\tau_{min} = 0.01$, $\tau_{max} = 4$, $\alpha = 3$, $\beta = 10$, $\rho = 0.98$ and $nbAnts = 8$. Figure 2 reports experimental results for solving random binary CSPs with 100 variables, 8 values in each variable domain, a connectivity (p_1) successively equals to 0.05 and 0.14 and different tightness values (p_2) around the phase transition. For each tightness, we report average results on 100 feasible problem instances. We successively display, for Local Search (LS) and Ant-Solver (AS), the success rate within a same limit of time (300s for $p_1 = 0.05$ and 1000s for $p_1 = 0.14$), the CPU time (in seconds) spent to find a solution and the corresponding number of repairs (in thousands of repairs).

Problems with connectivity $p_1 = 0.05$							Problems with connectivity $p_1 = 0.14$						
	Succ rate		CPU Time		Nb of repairs			Succ rate		CPU Time		Nb of repairs	
p_2	LS	AS	LS	AS	LS	AS	p_2	LS	AS	LS	AS	LS	AS
0.38	100	100	0.1	0.1	9K	11K	0.19	100	100	0.7	0.8	20K	24K
0.40	100	100	2.5	2.2	123K	102K	0.20	100	100	17.6	14.6	424K	291K
0.42	100	100	4.0	4.9	177K	170K	0.21	86	100	41.5	17.9	926K	309K
0.44	88	100	39.9	9.6	1 824K	313K	0.22	54	99	107.3	40.5	2 566K	491K
0.46	57	100	55.7	14.1	2 604K	402K	0.23	28	97	183.5	62.1	4 060K	665K
0.48	50	100	43.0	10.8	2 044K	376K	0.24	34	98	141.3	63.8	3 593K	661K
0.50	82	100	38.8	9.6	1 948K	419K	0.25	40	99	117.4	37.1	2 681K	513K
0.52	87	100	20.4	8.4	989K	412K	0.26	47	100	69.4	22.4	1 621K	414K
0.54	94	100	14.7	3.9	789K	216K	0.27	69	100	30.5	12.2	746K	290K

Fig. 2. Experimental results on $< 100, 8, p_1, p_2 >$ binary random CSPs

One can remark that, on the easiest problems, that are far enough from the phase transition region Ant-solver and local search have comparable results: for these problems, solutions are nearly always found during the preprocessing step, after the computation of very few complete compound labels. However, on the hardest problems that are within the phase transition region, Ant-solver is always much more successful and efficient than local search, showing that ACO actually allows one to boost the resolution.

References

1. M. Dorigo, G. Di Caro and L. M. Gambardella. Ant Algorithms for Discrete Optimization. *Artificial Life*, 5(2):137–172, 1999
2. J.K. Hao and R. Dorne. Empirical studies of heuristic local search for constraint solving. In *Proceedings of CP'96, LNCS 1118, Springer Verlag*, pages 194–208, 1996
3. P. Merz and B. Freisleben. Fitness landscapes and memetic algorithm design. In D. Corne and M. Dorigo and F. Glover, editors, *New Ideas in Optimization*, pages 245–260. McGraw Hill, UK, 1999
4. S. Minton, M.D. Johnston, A.B. Philips and P. Laird. Minimizing Conflicts: a Heuristic Repair Method for Constraint Satistaction and Scheduling Problems. *Artificial Intelligence*, 58:161–205, 1992
5. T. Stutzle and H.H. Hoos. MAX-MIN Ant System. *Journal of Future Generation Computer Systems*, 16:889–914, 2000
6. C. Solnon. Boosting Local Search with Artificial Ants (long paper). Research Report, LISI, 15 pages, 2001
7. M. Yokoo. Why adding more constraints makes a problem easier for hill-climbing algorithms: analyzing landscapes of CSPs. In *Proceedings of CP'97, LNCS 1330, Springer Verlag*, pages 356–370, 1997