An Incomplete Method for Solving Distributed Valued Constraint Satisfaction Problems

Michel Lemaître and **Gérard Verfaillie**
ONERA/CERT/DERI
2, avenue Édouard Belin – BP 4025 – 31055 Toulouse cedex 4 – France
{Michel.Lemaitre,Gerard.Verfaillie}@cert.fr

Abstract

This paper sets a model for Distributed Valued Constraint Satisfaction Problems, and proposes an incomplete method for solving such problems. This method is a greedy repair distributed algorithm which extends to the distributed case any greedy repair centralized algorithm. Experiments are carried out on a real-world problem and show the practical interest of this method.

Introduction

The Distributed Constraint Satisfaction Problem formulation (DCSP) is a general framework for modeling situations in which some agents are collectively entrusted with the task of finding a consistent solution to a set of local and inter-agent constraints. This framework has many practical applications such as distributed resource allocation, scheduling, timetabling and concurrent engineering.

Significant research work has been done on this model during last years. Taking a theoretical point of view, (Collin, Dechter, & Katz 1991) demonstrated that "it is generally impossible to guarantee convergence to a consistent solution using a uniform protocol", and thus established a limit of feasibility for algorithms solving such problems. The *distributed asynchronous backjumping* (Sycara *et al.* 1991) is an incomplete algorithm which combines distributed constraint satisfaction with heuristic search ; it has been experimented for job shop scheduling. (Prosser, Conway, & Muller 1992) present a distributed algorithm for maintaining arc-consistency in distributed and dynamic CSPs. The *asynchronous backtracking* and *asynchronous weak-commitment* algorithms (Yokoo *et al.* 1992; Yokoo 1995) are complete algorithms based on *nogood* exchanges between agents. (Solotorevsky & Gudes 1995; Solotorevsky, Gudes, & Meisel 1996) propose complete algorithms exploiting the relative difficulty of one central and many peripheral problems. The *distributed breakout algorithm* (Yokoo & Hiramaya 1996) is an incomplete algorithm for solving distributed CSP instances, based on the modification of the weight of violated constraints in order to escape from local minima.

Few works consider the practical case, yet often found in real-life problems, where instances are overconstrained and no solution exists. However (Yokoo 1993) presents an extended framework in which constraints are given weights reflecting their importance : the *asynchronous incremental relaxation algorithm* is a complete algorithm that relaxes less important constraints in a "lexicographic" order.

This idea of constraint valuation can be generalized, leading to the Valued Constraint Satisfaction Problem (VCSP) formulation (Schiex, Fargier, & Verfaillie 1995). CSP algorithms look for *satisfaction*, whereas VCSP algorithms look for *optimization*.

This paper presents a distributed extension of the VCSP framework, then a distributed algorithm for DVCSP optimization. Taking a practical point of view, and considering that complete algorithms are out of reach of most real-life problems, we deliberately turn towards incomplete algorithms.

Valued Constraint Satisfaction Problems

A Constraint Satisfaction Problem (CSP) instance is defined by a triple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is a set of *variables*, $D = \{d_1, \ldots, d_n\}$ is a set of finite *domains* for the variables, and C is a set of *constraints*. A constraint $c = (X_c, R_c)$ is defined by a subset of variables $X_c \subset X$ on which it holds, and a subset $R_c \subset \prod_{x_i \in X_c} d_i$ of allowed tuples of values. A *solution* of an instance is an assignment of values to all of the variables which satisfies all of the constraints.

Many CSP instances are so constrained that no solution exists. In this case, one can search for a solution minimizing the number of unsatisfied constraints. This is the partial CSP model, introduced by (Freuder &

Wallace 1992). This model can be further generalized by giving a weight or a *valuation* to each constraint, mirroring the importance one gives to its satisfaction. We then search for a solution minimizing an aggregation of the valuations of the unsatisfied constraints. This extension of the CSP model is called the Valued Constraint Satisfaction Problem framework and was introduced by (Schiex, Fargier, & Verfaillie 1995).

A VCSP instance is a quintuple (X, D, C, S, φ) where (X, D, C) is a classical CSP instance, $S = (E, \otimes, \succ)$ is a *valuation structure*, and $\varphi : C \rightarrow E$ is a *valuation function*, giving the valuation of each constraint. E is the set of possible valuations ; $\succ$ is a total order on E ; $\top \in E$ is the valuation corresponding to a maximal dissatisfaction, and $\perp \in E$ is the valuation corresponding to a maximal satisfaction ; $\otimes$, the *aggregation operator*, allows one to aggregate valuations. In order to have a sensible behavior, $\otimes$ must satisfy some properties : commutativity, associativity, monotonicity w.r.t. $\succ$; $\perp$ must be the identity element, and $\top$ an absorbing element.

Let $\mathcal{A}$ be an assignment of values to all of the variables, namely a complete assignment. We define the valuation of $\mathcal{A}$ for the constraint c by

$$\varphi(\mathcal{A}, c) = \begin{cases} \perp & \text{if } c \text{ is satisfied by } \mathcal{A} \\ \varphi(c) & \text{otherwise} \end{cases}$$

and the overall valuation of $\mathcal{A}$ by

$$\varphi(\mathcal{A}) = \bigotimes_{c \in C} \varphi(\mathcal{A}, c).$$

By instantiating the valuation structure S one can define specific frameworks such as :

– classical CSP, where E is $\{true, false\}$, $false \succ true$, $\perp$ is *true*, $\top$ is *false*, $\forall c \in C, \varphi(c) = false$ and $\otimes$ is $\wedge$ (boolean *and*)

– additive VCSP, where E is $N \cup \{+\infty\}$, $\succ$ is $>$ (natural order), $\perp$ is 0, $\top$ is $+\infty$ and $\otimes$ is $+$

– max VCSP, or possibilistic VCSP, where E is $[0, 1]$, $\succ$ is $>$, $\perp$ is 0, $\top$ is 1 and $\otimes$ is *max*

– lexicographic CSP, providing a combination of additive and max CSP.

The standard objective is to find a complete assignment with minimal valuation. Two kind of algorithms can be used for this purpose :

– *complete* algorithms which explore systematically and completely the solution space, based on *branch and bound* schemes ; they find an optimal solution, provided they are given enough time

– *incomplete* algorithms which, leaving out completeness, try to find quickly good solutions in an opportunistic way. These methods are generally based on *greedy repair* schemes and randomization techniques (van Laarhoven & Aarts 1987; Selman, Levesque, & Mitchell 1992). Although they often produce very good quality results, they cannot guarantee a distance to the optimal valuation.

A Distributed Model for Valued Constraint Satisfaction Problems

We propose an extension of the VCSP framework to the distributed case. This extension allows one to formalize the following situation. Some agents are entrusted with the task of solving existing local instances of VCSP. However, these local instances appear later to be interconnected by inter-agent constraints (constraints holding on variables of different local instances). Agents are now faced with a distributed VCSP instance. The union of optimal local solutions is not in general an optimal solution of the whole distributed problem instance. Each agent remains responsible for the values assigned to the variables of its local instance, but agents have to cooperate in order to find an optimal, or at least a good solution of the whole problem instance.

Like (Solotorevsky & Gudes 1995), we formalize the distributed constraints by adding an *inter-agent* problem instance to the set of local instances. A Distributed CSP (DCSP) instance will be defined by a couple (P, P_I) :

- $P = \{P_1, \ldots, P_m\}$ is a set of CSP instances, where each $P_i = (X_i, D_i, C_i)$ represents a local problem instance ; $X_i, i = 1 \ldots m$ are disjoint sets of variables
- $P_I = (X_I, D_I, C_I)$ is an inter-agent CSP representing the connection of the local instances, where C_I is the new set of inter-agent constraints, and X_I is the set of inter-agent variables, on which inter-agent constraints hold.

Note that $\{X_1, \ldots X_m\}$ is a partition of all of the variables, $X_I \subseteq \bigcup_{i=1}^{m} X_i$ (an inter-agent variable is also a variable of a local instance), and that $\{C_1, \ldots C_m, C_I\}$ is a partition of all of the constraints.

Similarly, a Distributed VCSP (DVCSP) instance is a couple (P, P_I) where P is a set of local VCSP instances having the same valuation structure. Each local VCSP instance is equipped with its own valuation function φ_i. P_I is the inter-agent problem instance, with the same valuation structure, and equipped with the valuation function φ_I.

Let $\mathcal{A}$ be a complete assignment, and $\mathcal{A}_i$ (resp. $\mathcal{A}_I$) the projection of $\mathcal{A}$ over the variables in X_i (resp. X_I). $\mathcal{A}_i$ is the agent i's local assignment. We now define the valuation of $\mathcal{A}$ for the constraint c by

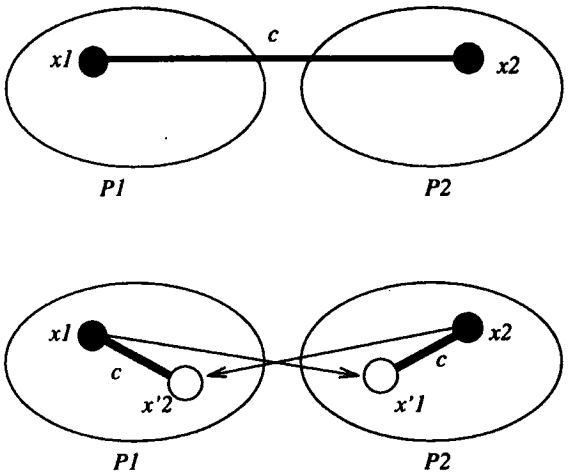

Figure 1: Taking into account an inter-agent constraint.

$$\varphi(\mathcal{A}, c) = \begin{cases} \varphi(\mathcal{A}_i, c) & \text{if } c \in C_i \\ \varphi(\mathcal{A}_I, c) & \text{if } c \in C_I. \end{cases}$$

The valuation of a complete assignment is just

$$\varphi(\mathcal{A}) = \bigotimes_{c \in C} \varphi(\mathcal{A}, c)$$

where $C = (\bigcup_{i=1}^{m} C_i) \bigcup C_I$ is the set of all constraints.

We must now consider the way agents manage inter-agent constraints. (Solotorevsky & Gudes 1995) suggest that a new agent is entrusted with the task of solving the inter-agent problem instance. But it seems preferable not to add a new agent and to distribute the inter-agent problem instance over existing agents. Each agent must know all inter-agent constraints holding on its variables. Suppose (figure 1, top) a binary inter-agent constraint c connecting variables x_1 and x_2 of local instances P_1 and P_2. Following (Prosser, Conway, & Muller 1992), we add to P_2 a new variable x'_1 which is a remote copy of x_1 (figure 1, bottom). Similarly, we create x'_2 on P_1. We now replace the original constraint c by two copies on each instance, and we add two new specialized inter-agent constraints, represented by oriented edges (x_1, x'_1) and (x_2, x'_2). These new constraints will capture the fact the same value must always be assigned to a variable and its remote copies. A simple way to satisfy these last equality constraints is to consider that remote copies of variables must be and can only be assigned by the agent which owns the original variable. This will be our hypothesis in the following section. Communications are supposed to be realized via message-passing.

An Incomplete Method

It is easy to show that the decision problem associated with the VCSP optimization problem is NP-complete, by restriction to MAXIMUM 2-SAT (Garey & Johnson 1979, LO5). Therefore, it is highly improbable that there could exist a polynomial time algorithm for this problem. In practice, all known algorithms present an exponential time behavior in the general case. Relatively small instances having 50 variables with 10 values per domain are generally out of reach of complete algorithms. The situation is even worse for distributed complete algorithms, due to communication overheads. For example, a simple forward-checking mechanism, which is a minimal improvement for a branch and bound scheme, would entail an enormous amount of messages between agents. The way (Yokoo 1995) tackles the problem in the pure satisfaction case, namely by local accumulation of ***nogoods***, is difficult to extend to the valued case. We conclude that complete algorithms are generally not relevant in the context of VCSP instances with real-world size. Therefore, we have to turn toward incomplete methods.

We present a greedy repair distributed method which extends to the distributed case any greedy repair centralized algorithm. The major difficulty with distributed systems is that no agent can know precisely the state of other agents at any instant. The main idea of our method lies is the following statements :

– a "leader agent" can know the valuation of a complete distributed assignment, not at any instant, but at precise instants of time

– suppose a stable complete distributed assignment $\mathcal{A}$; if an agent modifies $\mathcal{A}$ by assigning new values to its variables, then the resulting ***variation*** of the valuation of $\mathcal{A}$ can be computed by this agent, provided that it is the only one to change the assignment.

This last property is easily verified. Here is a sketch of the proposed method :

- during the initialization stage of a try, each agent initializes its local variables, and sends messages to initialize remote variable copies ; this corresponds to a complete initial assignment, made of the union of local assignments ; a leader agent (which can be one of the existing agents) gathers information from the others in order to compute the valuation of this complete initial assignment
- next, each agent in turn is allowed to modify its local assignment, using locally a greedy repair algorithm ; during an agent's turn, other agents must not change their local assignment
- at the end of its turn, an agent sends messages to update remote copies of its variables ; it sends also the

induced variation of the valuation of the complete assignment, computed locally, to the leader agent, allowing it to update the valuation of the complete distributed assignment

- the end of a try is decided by the leader agent ; several tours may be necessary to converge toward a good assignment.

By tuning the parameters of local greedy searches, one has to find a compromise between

– short local greedy searches, with many tours, allowing more inter-agent variables to change in a given lapse of time, at the expense of increased communication overheads

– longer local greedy searches, with less tours, decreasing communications.

In the first case, it is expected that agents will converge sooner toward a good solution, whereas in the second case the convergence will be slower.

It is important to observe that, in both cases, main communications are reduced to updating remote variables copies, after an agent's turn. This communication time can very small when compared with typical local search times.

Let us go into details. Here is first the pseudo-code of a general centralized greedy repair method for VCSP optimization, adapted from (Selman, Levesque, & Mitchell 1992) :

```
GreedyRepair(φ0)
1 A* ← InitialAssignment()
2 for e = 1 to MaxTries
3     A ← InitialAssignment()
4     while not Finished()
5         if φ(A) ⪯ φ0 then return A
6         if φ(A) ≺ φ(A*) then A* ← A
7         V ← Neighborood(A)
8         A ← Choice(V)
9 return "not found better than" : A*
```

The parameter φ_0 is a lower bound of the valuation searched for : the search will stop as soon as an assignment whose valuation is less or equal to φ_0 is found.

InitialAssignment : returns an initial assignment, generally randomly chosen.

Finished : returns **true** when a stopping criteria is satisfied, for example no improvement of $\varphi(\mathcal{A})$ has been obtained within a fixed number of changes (flips) of the current assignment.

Neighborood($\mathcal{A}$) : returns a set of assignments close to $\mathcal{A}$, for example all assignments obtained by changing the value of one variable.

Choice($\mathcal{V}$) : picks an assignment in $\mathcal{V}$, for example one with a minimal valuation.

Here is now the main procedure of the distributed greedy repair method, executed by the leader agent :

```
DistributedGreedyRepair(φ0)
1  φ* ← ⊤
2  for e = 1 to MaxTries
3      φ ← DistributedInitialization()
4      i ← 1
5      while not Finished()
6          if φ ⪯ φ0 then Stop()
7          if φ ≺ φ* then Memorize()
8          Δφ ← AgentGreedyRepair-i()
9          φ ← φ ⊗ Δφ
10         i ← NextAgent(i)
11 End()
```

DistributedInitialization : activates the initialization step of a try – as explained in the above sketch of the method – and returns the valuation of an initial distributed assignment.

Stop : signals to each agent that local assignments form together a complete assignment satisfying the required lower bound, so that the search is achieved.

Memorize : asks each agent to memorize its local assignment, because it participates in the best complete assignment found so far.

End : ends the search ; the best assignment is the one memorized by the last call to Memorize.

Note that AgentGreedyRepair-i returns the variation of the valuation which results from agent i's turn, which can be computed locally.

At the end, here is the pseudo-code executed by each agent in turn :

```
AgentGreedyRepair-i()
1 A_i ← current agent i 's local assignment
2 A'_i ← InitialAssignment-i()
3 while not FinishedAgent-i()
4     V_i ← AgentNeighborood-i(A'_i)
5     A'_i ← AgentChoice-i(V_i)
6 UpdateRemoteCopies-i()
7 return Δφ(A → A')
```

Experimental results

The goal of our experiments was to compare the performance of the proposed distributed method and of a centralized greedy repair. Since we address real-world problems, we choose for this experiment a Radio Link Frequency Assignment Problem (RLFAP) real instance

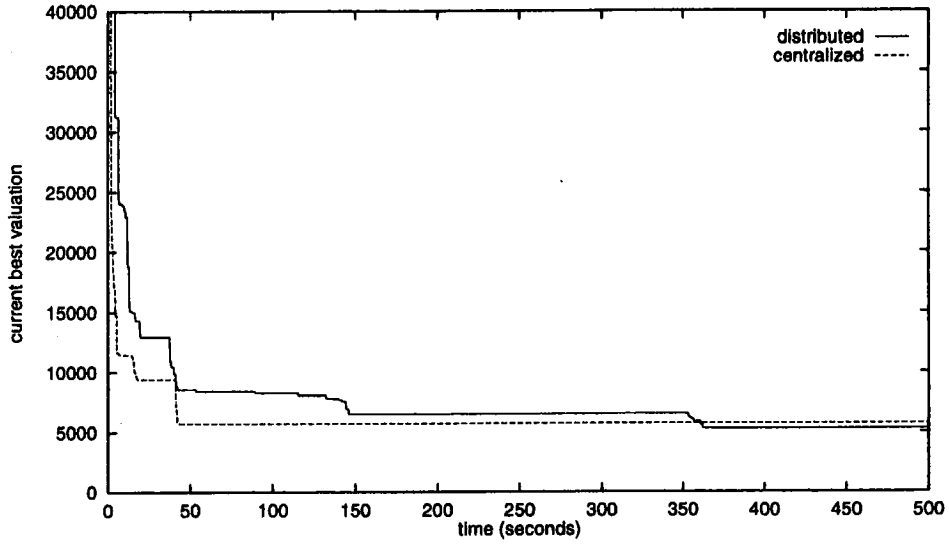

Figure 2: Typical executions of the distributed and centralized greedy repair algorithms, on RLFAP instance # 6.

from the French CELAR[1]. This additive instance has 100 variables[2], 44 values per domain, and 1222 constraints weighted from 1 to 1000. Variables have been arbitrarily distributed over 5 agents.

The greedy repair method used for both candidates is a simple hill-climbing local search : the CHOICE function returns an assignment randomly chosen from those which do not increase the valuation. The distributed method was simulated using the greedy algorithm, modified in such a way that only variables of the active agent are allowed to change. Hence we don't take into account communication overheads. Recall however that they are very limited in the proposed method.

Figure 2 shows typical valuation/time profiles for distributed and centralized searches. In this first experiment, each method was given a 500 seconds maximum time. During this amount of time, the distributed algorithm did 3 tries, each one composed of 20 complete tours, that is to say about 300 agent local searches, each one consuming 1 to 2 seconds of CPU time. It can be observed on these particular cases a general behavior : with the centralized algorithm, the current valuation decreases more rapidly, but both algorithms converge toward valuations close to each other. In figure 2, the distributed algorithm appears slightly better in the long run : this is not a general behavior, but it happens sometimes.

To have a better idea of the performance of the distributed method compared to the centralized one, we need more accurate comparisons. Our main experimental results are reported Table 1. The *MaxTries* parameter was fixed to 1 for each method, in order to get a more precise measure of the dispersion of results. The FINISHEDAGENT-i test was just a maximum number of flips test. Note that this is not the best way to use greedy repair methods, but these settings allow us to obtain a more accurate comparison. Each line of the table reports statistics obtained for 500 tries. Let us compare, lines A and B of the table, the centralized and distributed algorithms when the maximum number of flips is set to 500,000[3]. We observe

- a relatively large dispersion of the valuations
- that average values are increased by 8% from the centralized to the distributed case : this is the price to be paid for the distribution.

The line C shows that this loss of performance can be overcome by more computational work : doubling the number of flips by try allows one to recover and even to improve the performance level of the centralized algorithm.

The optimal valuation for this instance is not known. The best known today (to our knowledge) is 3389. Table 1 shows results which appear far from this. But recall that this is only a coarse experiment, the aim of which is to compare distributed and central-

[1]This benchmark and others are available in a CSP archive maintained by P. Eaton, `ftp://ftp.cs.unh.edu/pub/csp/archive/code/benchmarks`.

[2]Actually, the original problem instance has 200 variables and 1322 constraints, but by exploiting the structure of imperative constraints, it can be reduced to 100 variables while preserving the optimal valuation.

[3]Leading to an average CPU time of 250 seconds per try.

		# tours	# flips	min	max	med.	aver.	st. dev.
A	centralized		0,5M	4169	14353	8115	8184	1789
B	distributed 1	50	0,5M	4471	17351	8804	8851	2020
C	distributed 2	100	1M	4116	14067	7815	7917	1729

Table 1: Simulation results for RLFAP instance #6, giving statistics on valuations obtained for 500 tries.

ized methods on fair grounds, in order to get a precise idea of the loss of performance due to the tour mechanism. Better results for the distributed method could almost certainly be obtained by more sophisticated local search algorithms such as simulated annealing or taboo search, including a better control of tries (FINISHEDAGENT-i test). Note also that the best known valuation (3389) was obtained by a specific and very sophisticated method[4]. So we should rather compare it to the minimum valuation (4116) given in Table 1 line C.

Variants

This distributed method allows for several variants and adaptations.

First, local optimization turns could use any algorithm, possibly a complete algorithm, if the size of local instances permits it.

Second, it could be worth taking into account possible structural properties of the inter-agent constraint graph. In this respect, the suggested method presents some similarities with the work of (Kask & Dechter 1996), which is not in the context distributed optimization, but concerns only centralized satisfaction : variables are partitioned into two subsets, according to the structure of the constraint graph, and distinct procedures – one of which is a greedy repair, the other is a specialized complete algorithm for tree-like graphs – are applied in turn on each subset.

Third, when there exist imperative constraints (with cost $\top$), one could use a distributed 2-consistency polynomial algorithm (Prosser, Conway, & Muller 1992) in order to eliminate some values, as a pre-processing before tackling the main optimization problem.

Finally, the proposed method is open to some degree of parallelization :

– agents which do not share constraints can be activated simultaneously

– m tries (m is the number of agents) can be executed in a network of parallel pipelines : during the first turn, agent i starts the try number i ; then during the second turn, agent $i+1$ mod m continues the try number i, and so on. This scheme implies that all turns spend the same time, and that data for each pending try is memorized by agents

– a parallel variant inspired by (Yokoo & Hiramaya 1996) consists in running all agents simultaneously during a try (without exchanging values) ; one agent, among those which bring the best improvement of the global valuation, is allowed to exchange its values.

[4]A mix of a genetic algorithm and integer programming, developed by Antoon Kolen from Limburg, The Netherlands.

Conclusion

We presented in this paper an incomplete method for solving Distributed Valued Constraint Satisfaction Problems. This method is a greedy repair distributed algorithm which extends to the distributed case any greedy repair centralized algorithm. It is quite simple, based on successive greedy repairs computed in turn by agents. It induces a low rate of inter-agent communications. Finally, it can be tuned such that the handicap due to the distribution is overcome by more computational work. Experiments have been conducted on a real-world size problem instance and demonstrate the practical interest of this method.

Acknowledgments

This work was supported in part by French Délégation Générale à l'Armement, under contract DRET 94/002 BC 47. We thank Lionel Lobjois for his support in experimenting greedy repair methods.

References

Collin, Z.; Dechter, R.; and Katz, S. 1991. On the feasibility of distributed constraint satisfaction. In *Proceedings of IJCAI'91*, 318–324.

Freuder, E., and Wallace, R. 1992. Partial Constraint Satisfaction. *Artificial Intelligence* 58:21–70.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability, a guide to the theory of NP-completeness.* Freeman.

Kask, K., and Dechter, R. 1996. A Graph-Based Method for Improving GSAT. In *Proc. of AAAI-96*, 350–355.

Prosser, P.; Conway, C.; and Muller, C. 1992. A distributed constraint maintenance system. In *Proc. of the 12 th Int. Conf. on AI*, 221–231.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued Constraint Satisfaction Problems : Hard and Easy Problems. In *Proc. of IJCAI-95*, 631–637.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A New Method for Solving Hard Satisfiability Problems. In *Proc. of AAAI-92*, 440–446.

Solotorevsky, G., and Gudes, E. 1995. Algorithms for solving distributed constraint satisfaction problems. Ben-Gurion University of the Negev, Israel.

Solotorevsky, G.; Gudes, E.; and Meisel, A. 1996. Modeling and solving distributed constraint satisfaction problems. In *Principles and Practice of Constraint Programming - CP'96*, 561–562.

Sycara, K.; Roth, S.; Sadeh, N.; and Fox, M. 1991. Distributed constraint heuristic search. *IEEE Trans. on Systems, Man, and Cybernetics* 21(6):1446–1461.

van Laarhoven, P. J. M., and Aarts, E. H. L. 1987. *Simulated Annealing : Theory and Applications.* D. Reidel Publishing Company.

Yokoo, M., and Hiramaya, K. 1996. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Second International Conference on Multiagent Systems (ICMAS-96).*

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1992. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proc. of the 12th IEEE Int. Conf. on Distr. Comp. Syst.*, 614–621.

Yokoo, M. 1993. Constraint relaxation in distributed constraint satisfaction problems. In *Proc. of the 5th IEEE Int. Conf. on Tools with AI*, 56–63.

Yokoo, M. 1995. Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In *Principles and Practice of Constraint Programming - CP'95*, 88–102.