A Value Ordering Heuristic for Local Search in Distributed Resource Allocation

Adrian Petcu and **Boi Faltings**
{adrian.petcu, boi.faltings}@epfl.ch

Phone: +41-21-6936711, Fax: +41-21-6935225 Artificial Intelligence Laboratory
Swiss Federal Institute of Technology
IN (Ecublens), CH-1015 Lausanne, Switzerland
EPFL Technical Report IC/2004/18

Abstract

In this paper we develop a localized value-ordering heuristic for distributed resource allocation problems. We show how this value ordering heuristics can be used to achieve desirable properties (increased effectiveness, or better allocations). The specific distributed resource allocation problem that we consider is sensor allocation in sensor networks, and the algorithmic skeleton that we use to experiment this heuristic is the distributed breakout algorithm.

We compare this technique with the standard DBA and with another value-ordering heuristic (Petcu, Faltings 2002) and see from the experimental results that it significantly outperforms both of them in terms of the number of cycles required to solve the problem (and therefore improvements in terms of communication and time requirements), especially when the problems are difficult. The resulting algorithm is also able to solve a higher percentage of the test problems.

We show that a simple variation of this technique exhibits an interesting competition behavior that could be used to achieve higher quality allocations of the resource pool. Moreover, combinations of the two methods are possible, leading to interesting results.

Finally, we note that this heuristic is domain, but not algorithm specific (meaning that it could most likely give good results in conjunction with other DisCSP algorithms as well).

Content areas: constraint satisfaction, distributed AI, problem solving

Introduction

Distributed Constraint Satisfaction Problems (DisCSP from now on) are a very powerful paradigm applicable for a wide range of coordination and problem solving tasks in distributed artificial intelligence. An important subclass of these problems is the resource allocation problems, which we consider in this paper.

There is a number of distributed algorithms that were developed for this kind of problems (Yokoo, Hirayama 2000) and (Yokoo, Hirayama 1996) for instance. One of these, the Distributed Breakout Algorithm received quite some interest (for example (Zhang, Wittenburg 2002)) because of a number of interesting properties that this algorithm exhibits (relatively simple, efficient, low overhead, linear memory requirements, good anytime characteristics)

We chose this algorithm as a basis for our work, and as a testbed we considered the sensor allocation problem described in (Gomes *et al.* 2002). With this setup as a starting point, we then studied the effects of different search strategies on the performance of the algorithm.

It has been shown (Petcu, Faltings 2002) that the order in which the agents evaluate the values from their local domains plays an important role in the evolution of the algorithm towards a solution.

Our results show that by using the domain information available from their neighbors, the agents can develop search strategies that avoid resource conflicts with high probability, therefore reaching consistent assignments faster.

We see from the experimental results how one such local value-ordering technique can bring about significant improvements in terms of the number of cycles required to solve the problem (and therefore improvements in terms of communication and time requirements), especially when the problems are very difficult. The resulting algorithm is also able to solve a higher percentage of the test problems.

Moreover, a simple variation of this technique exhibits an interesting behavior that could be used to achieve higher quality allocations of the resource pool.

Problem description

The distributed sensor network problem formalized in (Gomes *et al.* 2002) consists of:

- a sensor field composed of n sensors: $S = \{s_1, s_2, ..., s_n\}$
- m targets that need to be tracked: $T = \{t_1, t_2, ..., t_m\}$

Each sensor has a "range" parameter that expresses the maximum distance that it can cover; in order to successfully track a target, 3 sensors have to be assigned to that target (triangulation can be applied using the data coming from those 3 sensors). However, some restrictions apply:

- the sensors in the field can communicate among themselves, but not necessarily every sensor with every other sensor (the sensor connectivity graph is not fully connected). The 3 sensors tracking a given target must be able to communicate among themselves;
- any one-sensor can only track one target at a time;

Formalization

We can formalize the problem as a DisCSP assigning one agent for each target: the variables are the required sensors (three variables per agent), and the values of each variable are the sensors that can track that target (are within range).

This is a fairly general model, with multiple variables per agent and both inter and intra agent constraints, and has low inter-agent communication requirements (minimizing communication is in fact one of the goals in many real world applications). We will therefore use the terms "agent" and "target" interchangeably for the rest of the paper.

So, let's assume that we have one agent A_i for each target T_i to be tracked. This agent would then have 3 variables to control: $A_i(x_1), A_i(x_2), A_i(x_3)$; each of them is one sensor that has to be assigned to track this target. The domain of all the variables for one agent is identical (this is because sensors can be assigned to a target from the same sensor set, namely the set of sensors that can actually "see" the respective target). However, this is a very particular characteristic of the sensor network problem, and we did not make this assumption in our implementation in order to maintain generality.

In this representation of the problem, we have two types of constraints: inter-agent constraints, and intra-agent constraints.

Intra-agent constraints - the constraints within one agent:

- no two variables can be assigned the same value (one agent must have three *different* sensors tracking it)
- there must be a communication link between every two sensors that are assigned to each agent

Inter-agent constraints - the constraints between agents:

- no two variables from any two agents can be assigned the same value (one sensor can track a single target at a given time)

It is interesting to note that all constraints in this problem (except for the "visibility" ones) are constraints of mutual exclusion (typical in resource allocation problems).

Related work

The idea of trying out values for the variables of a CSP in different orders, established based on various criteria, has been present in the AI literature for quite a while - e.g. (Keng, Yun 1989; Minton *et al.* 1992; Frost, Dechter 1995; Sadeh, Fox 1996; Petcu, Faltings 2002; Kask *et al.* 2004).

It has been shown that choosing the values of the variables of a CSP in an informed manner can produce significant improvements in the evolution of the search towards a solution, compared to choosing them in an arbitrary order.

Most of the existing techniques in this area are geared towards centralized mechanisms (e.g. (Frost, Dechter 1995)), where it is possible to achieve a global view of the current state of the problem, and establish the value-ordering based on this information. However, in a distributed setting where we perform local search, it is impossible to work under these assumptions; whatever decisions the agents may take as to the order in which they will try out their values, they must only be based on *local* information.

A further classification of these methods can be made into *static* and *dynamic* w.r.t. to when the ordering of the values is done (only in the beginning, or throughout the whole execution of the algorithm).

Dynamic ordering could in principle be expected to perform better than the static one, since it allows for more informed decisions; however, it also entails a greater runtime overhead. For example, in (Petcu, Faltings 2002) two value-ordering heuristics are presented: a static one (NI-DBA), and a dynamic one (NPI-DBA). The authors observe that NI-DBA does not bring significant performance improvements in dense problems, however, NPI-DBA does. Therefore, we chose to compare our algorithm with NPI-DBA. It should be noted that NPI-DBA is a general-purpose heuristic (works for all types of DisCSP, not only for resource allocation).

Algorithms

Distributed Breakout Algorithm

The Distributed Breakout Algorithm is in fact an extension of the original Breakout Algorithm for solving CSPs in a centralized fashion (Morris 1993). This algorithm is a local search method, with an innovative technique for escaping from local minima: the constraints have weights, and the weights are dynamically increased in order to force the agents to adjust their values while in a condition of local minima.

In the distributed version, agents use *ok?* and *improve* messages for exchanging their local information: an *ok?* message is used to send the current variable value, and an *improve* message is used to send possible improvement in the evaluation of variable value. When receiving *ok?* messages from all neighbors, an agent calculates the evaluation of the current variable value and its possible maximal improvement and sends them to neighbors via *improve* messages. When receiving *improve* messages from all neighbors, an agent compares them with its own improvement. If there is a greater improvement than its own, the agent will not do anything. If there is no possible improvement (all are 0), the agent will increase the weights of the violated constraints. If its improvement is the greatest, the agent will change its variable to the value giving the maximal improvement.

Note that ties in improvement comparison are broken deterministically by comparing agent identifiers. After this step, the agents send *ok?* messages to their neighbors.

When no more constraints are violated, the problem is solved.

Preamble

We assume that the agents representing the targets all know the details of the sensor field: number of sensors, their positions and ranges.

We call two agents "neighbors" if they share a constraint. In all distributed algorithms it's necessary for each node to be able to identify its neighbors. In some cases this information is considered to be given at startup (for instance from a configuration file), and in others it is learnt at runtime (either

in a "pre-processing" step, or progressively, as the algorithm runs)

In our case, we have an initial "pre-processing/discovery" phase (before we actually start DB):

- each agent determines the set of sensors that can track it (based on its coordinates, and on sensor ranges); this set will be the domain of the three local variables
- each agent sends to all his neighbors the coordinates of its target (this information is sufficient to determine the neighboring)
- upon receiving a target information from another agent, each agent determines if it has any common sensors with the respective target:
  - if so, then the agent that sent this information will be kept as a neighbor, and there will be 9 constraints of mutual exclusion between the two agents (there are 9 possible combinations of variables, and all of them have to be assigned different values)
  - if not, then the agent that sent this information will be removed from the neighbors list, and there will be no other interaction with that agent during the execution of the algorithm
- each agent sends its domain to its neighbors
- alternatively, the first step (target broadcast) could be omitted, and the second (domain broadcast) extended to all the agents: based on the domain information it is also possible to determine the neighboring

Standard Distributed Breakout applied

Here we will present the standard DBA applied to our problem, which will be then used as a skeleton on which we build our improvements. Each agent follows Algorithm 1:

The differences between this version of DBA and the standard one are in the initialization phase(presented in Algorithm 1). There are also some changes in the send* and received* procedures made to accommodate multiple local variables (standard DBA allows only one variable per agent), but they are pretty straightforward, and we don't list them here because of lack of space.

DBA-VO

In the standard version of the DBA, in the initialization phase, each agent randomly assigns values to its variables, and subsequently tries to assign to its variables the first values that produce a conflict reduction. The problem with this approach is that it does not take into account the fact that the initial values that the variables take can actually be very likely to cause a large number of conflicts, and that later on, a large number of cycles would be required to repair those conflicts.

The idea of DBA-VO is that if we take into account the number of times each resource (variable value, in our case) appears in the domain of the neighbors, and then try to assign each variable a value that is the least likely to cause a conflict, then it is possible that we start with an already very good assignment, that would later on require much less effort to fix. Subsequently, while trying to repair the possible conflicts, the agents would pick for their variables the values that appear least in the domains of the neighboring variables, thus reducing the likelihood that a conflict would occur in the future.

Example: let's consider the situation from Figure 1.

We see that there is a sensor S_x which is common among all the 4 targets. An uninformed assignment might look like the one in the figure, thus creating 5 conflicts (between all the agents over S_x, and another one between T_1 and T_3). Resolving these conflicts would then require 4 synchronized steps for T_1, T_2, T_3 and again T_3.

However, if the agents would have observed the fact that S_x is a highly demanded resource and therefore avoided trying to acquire it, they would not have gotten in this situation in the first place. Specifically, T_1 could have used $S_t, T_2 - S_z, T_3 - S_u$, and $T_4 - S_y$.

The information required to make these decisions is available immediately after the initialization phase, and remains valid throughout the whole execution of the algorithm.

The process is as follows:

- during the preprocessing phase, every time a new domain

Algorithm 1: Standard DBA applied to sensor networks

procedure *initialize*;
begin
  load the sensor field ;
  determine sensors "within range" → local domains;
  broadcast domain to all agents ;
  establish neighborhood based on incoming domains;
  initialize local values;
  go to standard send_values from DBA;
end

Following are the rest of the standard DBA procedures:

procedure *send_values*;
begin
  if *my improvement is best* **then** switch value ;
  if *local minima* **then** increase weights ;
  send local_values to neighbors
end
procedure *send_improvements*;
begin
  compute maximal improvement;
  send max_improve, curr_eval and curr_val to neighbors
end
procedure *received_values*;
begin
  add received values to agent_view;
  if *last message received* **then** send_improvements ;
end
procedure *received_improvements*;
begin
  record improvement;
  if *last improvement* **then** go to send_values ;
end

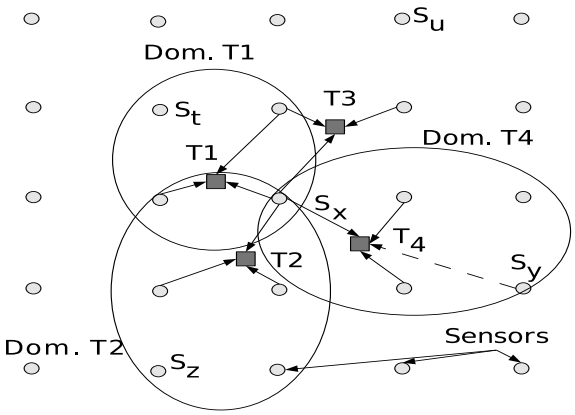

Figure 1: Problem example

comes in, each agent checks to see which values from the local domain are also in the domain of the sending agent. For all such values, increase a counter.

- After the last domain comes in, sort the values from the local domain in the increasing order of the counters. Initialize the local variables with first values from the local domains.
- Afterwards, during the execution of the algorithm, every time we have a constraint violation and we have to try to find another value for the respective variable, we search for the best improving value, in the (now sorted) local domain, and pick that value, knowing that it will be likely to interfere as little as possible with the neighbors.

Intuitively, the heuristic is similar to a "non-competing contract" - agents avoid demanding a resource that they know it's likely to be busy anyway. Formally, the changes to the standard algorithm are described in Algorithm 2. We can easily see that all the overhead that DBA-VO has in addition to Std-DBA is basically the sorting of the domain after all the neighboring has been established (which is a one-time process, and not really expensive: $O(d \times \log d)$)

DBA-VOi

In this section we present a small variation of the previous algorithm. Namely, the heuristic works the same in as far as the domain counters are concerned; however, based on these counters, the domains of the local variables are sorted in the inverse order: the most requested ones first.

The modifications made to Algorithm 2 are presented in Algorithm 3.

The interesting effect that can be noticed after introducing this modification, is that all agents try to acquire the most "popular" resources (the ones that have the highest "demand" counters associated with them). This tendency is two-fold: first, as a result of the initialization (all agents try in the first step to acquire those resources), and second, during the subsequent conflict-repairing rounds, the agents would try to propose improvement values but again, giving preference to the most "popular" resources.

Algorithm 2: DBA-VO

procedure *initialize*
begin
  foreach *local variable* x_i **do**
    initialize the vector dom_cnt(x_i) with 0;
  endforeach
end
procedure *received_domain(dom)*
begin
  foreach *value* v_i *in received_domain* **do**
    foreach *local variable* x_i **do**
      if $v_i \in$ *domain(*x_i*)* **then** dom_cnt(x_i, v_i) ++ ;
    endforeach
  endforeach
  if *dom is last_domain_to_receive* **then** go to initialize_local_values ;
end
procedure *initialize_local_values*
begin
  sort values in dom(x_i) in the ascending order of dom_cnt(x_i);
  initialize local variables with the first values in their domains;
  go to standard send_values from DBA;
end
procedure *send_improvements*
in standard DBA there is a step "find improvement". we redefine this step as follows:
procedure *compute_improvements*
begin
  find local value giving best improvement; the search is done in the ascending order of dom_cnt(x_i);
end

Normally, one cannot expect a gain in the running time of the algorithm when using this heuristic, exactly because of the "competition effect" described above. However, this heuristic is interesting nevertheless because it almost guarantees that a "special" subset of the available resources will be part of the final assignment. This might be important in a setting where we have for instance resources of varying quality, and we would always like to obtain as a final solution an assignment where all the best resources are in use. By simply constraining the best resources by as many consumers as possible and using this heuristic, we are then sure to obtain a final solution that respects this criteria.

Discussion

It is possible to combine the two heuristics in many ways, depending on the requirements of the domain.

If, for instance, the final assignment is important, but we would like to avoid the extra overhead generated by the continuous "fight" of the agents over the same set of "popular" resources, then we could do the initialization according to the DBA-VOi (domains sorted in the descending order of the

Algorithm 3: DBA-VOi

procedure *initialize_local_values*
begin
sort values in dom(x_i) in the descending order of dom_cnt(x_i);
initialize local variables with the first values in their domains;
go to standard send_values from DBA;
end
procedure *send_improvements*
in standard DBA there is a step "find improvement". we redefine this step as follows:
procedure *compute_improvements*
begin
find local value giving best improvement; the search is done in the descending order of dom_cnt(x_i);
end

domain counters), and continue the search in the subsequent improvement steps according to the DBA-VO heuristic (domains sorted in the ascending order of the domain counters).

Another possibility is if the time-to-solution is important, and good anytime characteristics are required. In that case, we could do the inverse: the initialization according to the DBA-VO, to start with an assignment that is as close to a solution as possible, and continue the search with DBA-VOi to go towards a solution that uses as many qualitative resources as possible.

We could even imagine a probabilistic combination of the two heuristics: for instance, while doing the initial assignments, choose for each variable a value which corresponds with high probability to the DBA-VO order, but with a small probability, choose a value corresponding to the DBA-VOi order. In this way, we would end up with a balanced initial assignment that would also have a high overall probability of making use of the qualitative resources.

Evaluation

We made our evaluations with the same settings as in (Petcu, Faltings 2002): the sensor field was a network of 400 sensors, and we experimented with 110 to 130 agents (as shown there, this is the "phase transition" area of this problem type, therefore, this is where the most difficult problems can be found).

Since every target has to have three associated sensors, this means that in total, our experiments ran with 330 to 390 variables respectively. Obviously, the problems were increasingly difficult, not only because the number of agents increased, but also because the number of "required" sensors approached the number of available sensors. This made the allocation increasingly difficult, and for the 130-targets problem (which is very close to the maximum size possible), almost impossible.

For small numbers of targets, all tested algorithms performed well; the differences start to appear only when the problems become difficult. Therefore, on the curves that we present, we show the results only from the most interesting tests, with 110 targets and more.

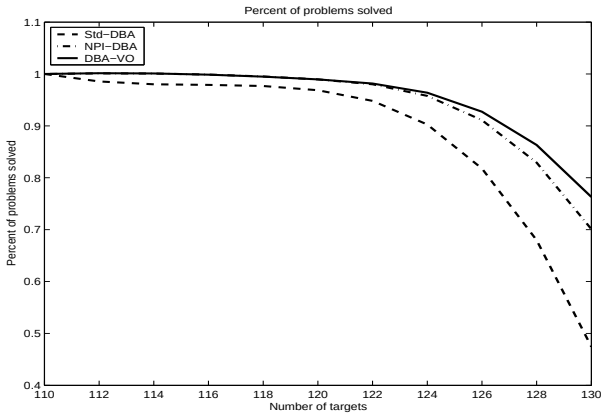

Figure 2: Percent of problems actually solved

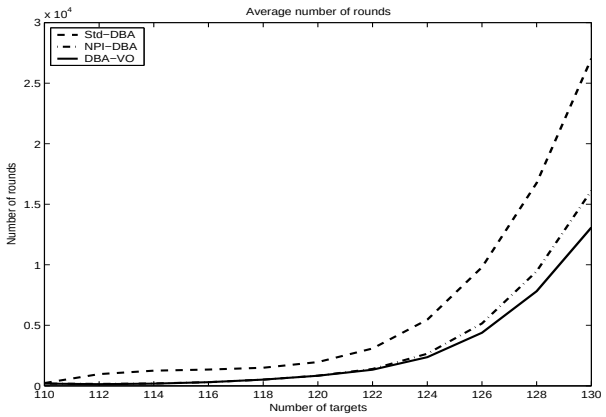

Figure 3: Average number of rounds

The problems were randomly generated, in such a way that they were solvable. However, DBA being incomplete, not all of them were actually solved. We set the maximum number of iterations that DB goes through to 50000, after which the problem was declared unsolvable.

We logged the time spent to solve each problem, the number of cycles required, and whether the problem was solved or not. We developed a visual interface that allows us to monitor the solving process.

We can see what percent of the problem instances were solved by different search strategies in Figure 2. The average number of rounds is shown in Figure 3. The average time spent for each problem size by each method is shown in Figure 4.

We define an empirical parameter "problem density" ρ as follows $\rho = \frac{number\_of\_targets \times 3}{number\_of\_sensors}$ This parameter will vary with the number of targets from 0 (for 0 targets) to almost 1 (for the maximum number of targets that in this case is 133)

We can clearly see in all the curves that the methods are quite similar in performance for smaller values of ρ, up to a point where ρ approaches 1. We can say that for ρ close to 1,

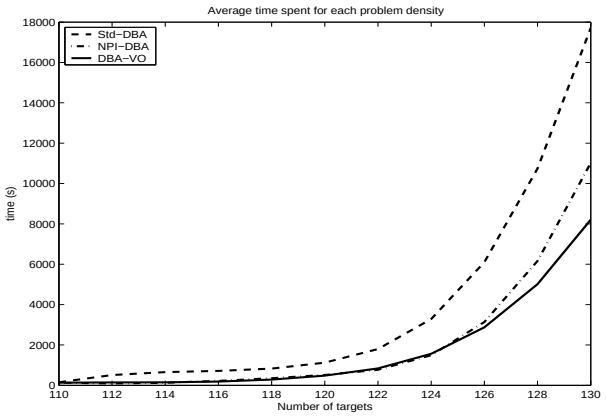

Figure 4: Time spent for each problem size

we have a "phase transition" phenomenon in this case. Figure 2 shows that there is a steep decrease in the percentage of the problems solved by all algorithms, but DBA-VO performs best in that area (manages to solve most of the problems), followed by NPI-DBA (about 70%) , and standard DBA (less than half of the problems solved).

In figure 3 we see that on average, DBA-VO does less than half of the rounds of Std-DBA, and about 25% less rounds than NPI-DBA.

Based on these results, we can conclude that both the "informed" initialization of the variables and the subsequent search strategy plays a role in the performance of the algorithm.

We also recorded the time required to solve each problem by the different methods, having in mind the fact that as little as may be, there is an overhead in DBA-VO that standard DBA does not have. However, similar to the number of rounds, we can see in figure 4 that this overhead pays off eventually, and we achieve better results.

Overall, we see that DBA-VO outperforms its counterparts in all the three considered measures.

Conclusions and future work

We presented a value-ordering heuristic for improving the performance of the Distributed Breakout Algorithm applied on distributed resource allocation problems.

We compared this technique with the standard DBA and with another value-ordering heuristic (Petcu, Faltings 2002) and saw from the experimental results that it outperforms both of them in terms of the number of cycles required to solve the problem (and therefore improvements in terms of communication and time requirements), especially for difficult problems. The resulting algorithm is also able to solve a higher percentage of the test problems.

Moreover, a simple variation of this technique exhibits an interesting behavior that could be used to achieve higher quality allocations of the resource pool (ensuring that a certain subset of the resources is allocated in a final assignment). Interesting combinations of the two techniques are possible, giving desirable properties of the allocation algorithm.

Further improvements could be obtained by allowing multiple simultaneous changes of the local variables at each step, or by trying a hierarchical approach to the problem, where certain agents are delegated as a "local authority" for solving a particularly difficult local problem.

It would be interesting to study in more detail the performance gains brought by combinations of these techniques when the problem size increases, in terms of two dimensions: the size of the sensor field (thus also the maximum number of targets), and the sensor ranges (thus the size of the domains)

References

Frost, D., and Dechter, R. *"Look-ahead value ordering for constraint satisfaction problems."*. In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada, August 1995, pp. 572-578.

Carla Gomes, Cesar Fernandez, Ramon Bejar and Bhaskar Krishnamachari *"Communication and Computation in DisCSP Algorithms"*. In Proceedings of CP-2002, Ithaca, New York, USA

Kask,Kalev and Rina Dechter and Vibhav Gogate *"New Look-Ahead Schemes for Constraint Satisfaction"*. In The Eighth International Symposium on Artificial Intelligence and Mathematics, 2004, forthcoming

Keng, N., Yun, D.Y.Y. *"A planning/scheduling methodology for the constrained resource problem"*. Proceedings of the 11'th IJCAI, 998-1003, 1989

Minton,S and M.D. Johnston, A.B. Philips, and P. Laird *"Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems"*. Artificial Intelligence, vol. 58, nos. 1 3, pp. 161 205, 1992. of the IEEE.

Morris, P. *"The breakout method for escaping from local minima"*. In Proceedings of the Eleventh National Conference on Artificial Intelligence, 40-45

Petcu, Adrian and Faltings, Boi *"Applying interchangeability techniques to the distributed breakout algorithm"*. In Proceedings of the International Joint Conference on Artificial Intelligence 2003

N.M. Sadeh and M.S. Fox. *"Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem"*. Artificial Intelligence 86(1):1-41, 1996.

Makoto Yokoo *"The Distributed Constraint Satisfaction Problem: Formalization and Algorithms"*. IEEE Transactions on Knowledge and Data Engineering 10(5), 673-685.

Makoto Yokoo and Katsutoshi Hirayama *"Distributed Breakout Algorithm for Solving Distributed Constraint Satisfaction Problems"*. In Proc. of the Second International Conference on Multiagent Systems 1996

Makoto Yokoo and Katsutoshi Hirayama *"Algorithms for distributed constraint satisfaction: A review"* In Proc. of Autonomous Agents and Multi-agent Systems, 189-211

Weixiong Zhang and Lars Wittenburg *"Distributed Breakout Revisited"*. In Proceedings of AAAI 2002