Applying interchangeability techniques to the distributed breakout algorithm

Adrian Petcu, Boi Faltings
Artificial Intelligence Laboratory (LIA)
Swiss Federal Institute of Technology (EPFL)
IN-Ecublens, CH-1015 Ecublens, Switzerland
{adrian.petcu—boi.faltings}@epfl.ch
http://liawww.epfl.ch
EPFL Technical Report IC/2004/17

Abstract

In this paper, we develop two methods for improving the performance of the standard Distributed Breakout Algorithm (Yokoo *et al.* 1996) using the notion of interchangeability. We study the performance of this algorithm on the problem of distributed sensor networks. In particular, we consider how neighborhood interchangeability and neighborhood partial interchangeability (Freuder 1991) can be used to keep conflicts localized and avoid "chain reactions" where a conflict originating in one part of the problem spreads to neighboring areas.

We see from the experimental results that such techniques 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 for difficult problems. Moreover, the improved algorithms are able to solve a higher proportion of the test problems.

Key words: constraint satisfaction, distributed AI, problem solving, sensor networks.

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.

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

DBA is an extension of the original centralized Breakout Algorithm (Morris 1993). This algorithm is a local search method, with an innovative technique for escaping from local minima: the constraints have weights, which are dynamically increased to force the agents to adjust their values while in a local minimum. During the execution of the algorithm, each agent proposes improvements to the current state by changing its variable value such that the cost of violated constraints is decreased as much as possible, and the one that proposes the largest improvement wins, and can change its value.

While having the interesting properties enumerated above, local search algorithms also have a common drawback: choosing indiscriminately between the possible values of the local variable (only considering the cost of the immediate constraint violations) can lead to "chain-reactions" (one conflict originating in one part of the constraint graph needlessly propagates throughout the whole graph, only to (hopefully) be resolved in a completely different part of the graph).

We analyzed these phenomena, and drew the conclusion that using interchangeability techniques, one can determine what values from the local domain will not cause such conflict propagations, and use one of those values as the next variable assignment. In this way, we look for a "local resolution" to all conflicts, in the sense that we keep them contained as much as possible, and involve "external parties" only when there is no other way.

We see from the experimental results that such techniques 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. Moreover, the improved algorithms are able to solve a higher proportion of the test problems.

As an application domain we considered the sensor allocation problem described in (Gomes *et al.* 2002). It is however worthwhile to note that the techniques described here are not limited to this particular class of problems, or to resource allocation in general, and can be applied wherever Distributed Breakout can be applied.

Preamble

Problem description

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

- a sensor field composed of n sensors: $S = \{s_1, s_2, ..., s_n\}$
- a set of 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 our problem as a DisCSP in two ways:

- one agent represents a target: in this case, the variables are the sensors to be assigned to that target(three variables per agent), and the domain of each variable is the set of sensors that can track the respective target (are within range);
- one agent represents a sensor: in this case, the variables are the targets (one variable per agent), and the domain of each variable is the set of targets within the detection range of the respective sensor;

In our model we chose the first representation because it is a more general model, with multiple variables per agent and both inter and intra agent constraints, and has lower inter-agent communication requirements (minimizing communication is in fact one of the goals in many real world distributed 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 for each target to be tracked. This agent would then have 3 variables to control: x_1, x_2, 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).

Breakout Algorithm

In the distributed version of this algorithm, 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.

Interchangeability background

The concept of interchangeability was first introduced in (Freuder 1991), and informally means equivalence between different values of a CSP variable.

We studied two kinds of interchangeability:

- Neighborhood Interchangeability - NI

  Two values a and b for a variable V_i are *neighborhood* interchangeable if for every constraint involving V_i, for every tuple that admits $V_i = a$ there is an otherwise identical tuple that admits $V_i = b$, and vice-versa.

  Neighborhood interchangeability considers only local interactions and thus can be efficiently computed.

- Neighborhood Partial Interchangeability - NPI

  Two values a and b for a variable V_i are *neighborhood partial interchangeable* (NPI) with respect to a set of variables S if for every constraint between V_i and the neighborhood of the set S, for every tuple that admits $V_i = a$ there is an otherwise identical tuple that admits $V_i = b$, and vice-versa, (where this change can affect the variables from the set S), while the same condition applies also for all the other variables from S.

  NPI is a weaker form of NI, defined for a subset of values from the local domain with respect to a set of neighbors, where the impact of the change of the local variable is limited to the reference set of neighbors.

In the general case, interchangeability classes are computed using *discrimination trees* (Freuder 1991) for NI, or *joint discrimination trees (JDT)* (Choueiry *et al.* 1998) for NPI. However, in our case we have mutual exclusion constraints between the agents; therefore, as shown in (Choueiry *et al.* 1998), the interchangeability sets can be computed easier, by disjunction between sets:

- NI: for two values from the local domain to be neighborhood interchangeable, they have to be absent from the domains of all of the neighbors of the owning agent.
- NPI: for two values from the local domain to be neighborhood partially interchangeable wrt a given set of neighbors, they can only appear in the domains of the variables from the reference set.

It is interesting to note that the techniques described in this paper are not restricted to mutual exclusion constraints: the modifications made to the standard DBA would work similarly in the general case; but then it would be more difficult to compute the NI and NPI sets. In that case, one cannot just do disjunction between sets anymore, but would have to use the general methods with the discrimination trees.

Algorithms

We experimented with two notions of interchangeability: *Neighborhood interchangeability* and *Neighborhood partial interchangeability*. We called the resulting algorithms NI-DB, and NPI-DB. 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's 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:

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 randomly;
  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

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 quite straight-forward, and we don't list them here because of lack of space.

NI-DBA

The idea in the case of NI-DBA is that if we find the NI-sets for the local variables, then we can safely assign to the variables values from those sets, and be certain that this won't cause any conflicts with the neighboring agents.

Example: let's assume that at a certain point in the execution of the algorithm, the assignments are as in the Figure 1. We see that there are conflicts between T_1 - T_3 and T_1 - T_2. At this point, T_2 and T_3 have the possibility to improve the situation by 1. T_2 will have priority, and can choose between

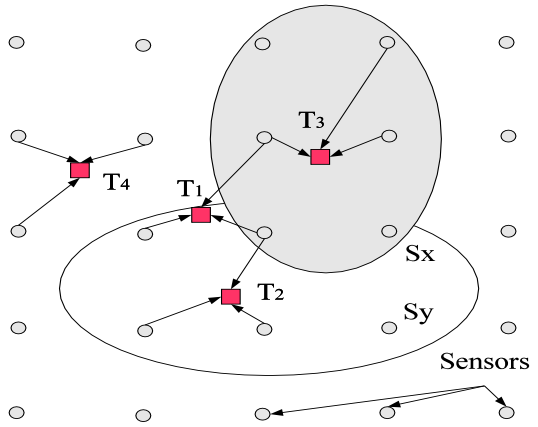

Figure 1: Sensor field - NI

S_x and S_y, both giving the same improvement, 1. However, S_x is not NI (is also present in the domain of T_3), and therefore S_y will be chosen. Standard DBA would have chosen S_x (as the first value that gives maximal improvement). This is important, because we avoid a possible future conflict between T_2 and T_3 that could appear if later on, T_3 would be forced to choose S_x.

The process is as follows:

- initially we assume that the NI-list is composed of all the values from the local domain.
- then, during the preprocessing phase, every time a new domain comes in, we check to see which values from this domain are also in the domain of the receiving agent. These values will not be NI, and we remove them from the NI-list. After the last domain came in, the remaining values from the NI-list are NI.
- we initialize the local variables with random values (in order to be able to compare the algorithms).
- 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 can use the following heuristic: we will first look in the NI-list of that variable; if a value can be found in that list that does not break the local constraints, then we can safely pick that value, knowing that it will not break any external constraints (because of the way we constructed the NI-list)

Formally, the changes to the standard algorithm are described in Algorithm 2.

NPI-DBA

The idea in the case of NPI-DBA is that if we find the NPI-sets for the local variables with respect to the set of the neighbors that we have constraint violations with, then if we assign to the variables values from those sets, we will not risk causing conflicts with other neighbors. This way, when we have a conflict, we try to keep it from spreading around, and try to solve it locally.

Example: let's assume that at a certain point in the execution of the algorithm, the assignments are as in the Figure 2. We see that there are conflicts between T_1 - T_3 and T_1 - T_2. At this point, T_1 has the possibility to switch to either S_x or S_y, both giving the same improvement, 1. Seeing that it has constraint violations with T_2 and T_3, T_1 determines the NPI set wrt to S=$\{T_2, T_3\}$. S_x is not in the NPI set, while S_y is. Therefore S_y will be chosen. Standard DBA would have chosen S_x (as the first value that gives maximal improvement). This is important, because we avoid a possible future conflict between T_1 and T_4 that could appear if later on, T_4 would be forced to choose S_x.

The process is like this:

- the initialization phase remains the same as in standard-DBA;
- afterwards, during the execution of the algorithm, at every step, suppose we have constraint violations with a set N_k of k neighboring nodes;
- then we determine the NPI-sets for local variables that have conflicts with respect to the set N_k
- when we try to find the improvements for the conflicting variables, we will first look in the NPI-list of the conflicting variables. If a value that does not break the local constraints can be found there, then we can safely pick that value, knowing that it will not move the conflict from where it is now (between a local variable and one from the N_k set) to another place (between a local variable and some other agent)

Algorithm 2: NI-DBA

procedure *initialize*
begin
  foreach *local variable* x_i **do**
    NI-list(x_i) $\leftarrow$ Domain(x_i);
  endforeach
end
procedure *received_domain*
begin
  foreach *value* v_i *in received_domain* **do**
    foreach *local variable* x_i **do**
      if v_i *in domain(*x_i*)* **then** remove v_i from NI-list(x_i) ;
    endforeach
  endforeach
end
procedure *send_improvements*
in standard DBA there is a step "find improvement". we redefine this step as follows:
procedure *compute_improvements*
begin
  find improvement; give preference to NI values;
end

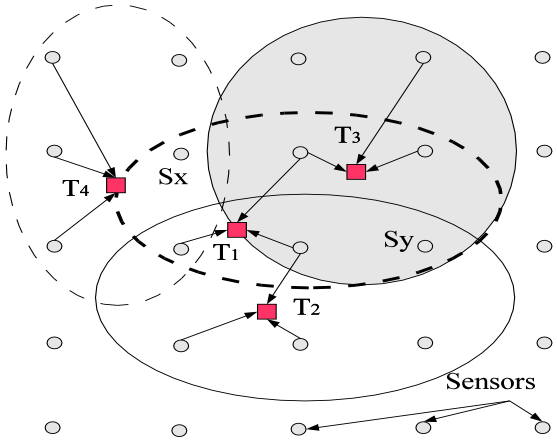

Figure 2: Sensor field - NPI

Formally, the changes to the standard algorithm are described in Algorithm 3

procedure *send_improvements*
in standard DBA there is a step "find improvement". we redefine this step as follows:
procedure *compute_improvements*
begin
  foreach *local variable* x_i *that has a conflict* **do**
    let N_k be the set of external variables that conflict with x_i;
    NPI-list(x_i) $\leftarrow$ those values in domain(x_i) that are NPI wrt N_k;
  endforeach
  find improvement, giving preference to values from the NPI-lists;
end

Algorithm 3: NPI-DBA

Evaluation

We made our evaluations with the following settings: the sensor field was a square network (20 by 20 $\rightarrow$ 400 sensors in total), and we experimented with 40, 60, 80, 100, 110, 115, 120, 125 and 130 simultaneous targets.

Since every target has to have three associated sensors, this means that in total, our experiments ran with 120, 180, 240, 300, 330, 345, 360, 375 and 390 total 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 the 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.

The problems were randomly generated, in such a way that they were solvable. We set a maximum limit of 50000 for the number of iterations that DB goes through. If, after this threshold was reached, the problem was still not solved (some constraints were still violated), then we declared it unsolvable.

However, even if the problems were solvable, not always was the case that they were actually solved. This is due to the fact that the Distributed Breakout algorithm is incomplete, and happened for particularly difficult instances.

We then collected the results in the form of time spent to solve the problem, number of cycles required, and a boolean value indicating if the problem was solved or not.

We developed a visual interface that allows us to monitor the solving process. We could observe clearly that using the strategies based on NI/NPI greatly inhibits the propagation of changes around the constraint graph.

An important aspect is that during the preprocessing phase of NI-DB we make the variable initializations randomly, as in standard DB. We chose to do this kind of suboptimal initialization in order to keep the algorithms comparable, and see the improvements that the *search strategy* brings. If, instead, we assign values to the variables from the determined NI-sets, then the improvements are more substantial. Based on these results, we can conclude that both the "informed" initialization of the variables and the subsequent search strategy play a role in the performance of the algorithm.

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)

Mean values

We can see what percent of the problem instances were solved by different search strategies in Figure 3. The average number of rounds is shown in Figure 4. The average time spent for each problem size by each method is shown in Figure 5.

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. For values of ρ close to 1, we observe a phenomenon similar to a phase transition (moving from easy problems to a phase where most of the problems are not solved)

Figure 3 shows that there is a steep decrease in the percentage of the problems solved by all algorithms, but NPI-DBA performs best in that area (manages to solve most of the problems), followed by NI-DBA that is comparable with standard DBA (less than half of the problems solved).

In figure 4 we see that the average number of rounds for standard DBA is bigger than for NPI-DBA, and close to the one of NI-DBA.

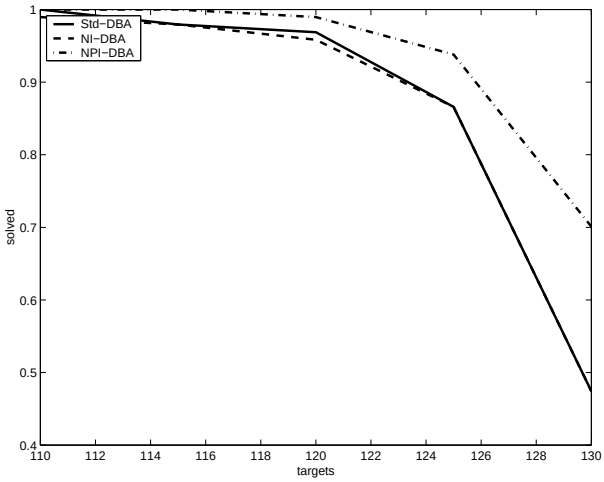

Figure 3: Solving rates for different strategies

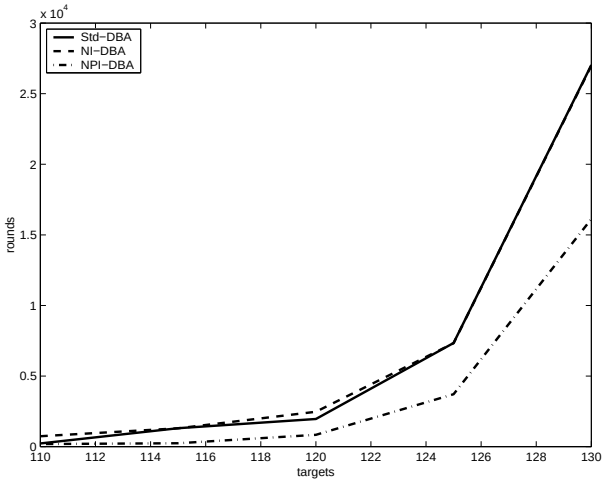

Figure 4: Number of rounds for different strategies

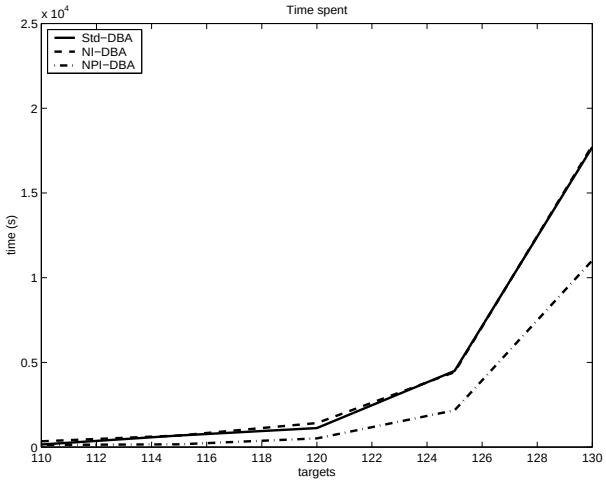

Figure 5: Time spent for each problem size by different strategies

We also recorded the time required to solve each problem by the different methods, having in mind the fact that determining the NI and NPI sets is a computational overhead that the standard DBA does not have. However, similar to the number of rounds, we can see in figure 5 that this overhead pays off eventually, and we have better results than standard DBA.

Variances

We compute the variance of the same parameters (solvability rate, number of cycles and time) for the entire set of test problems. The formula employed is the most common one:

$$Var = \sqrt{\frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N-1}}$$

In this formula, N is the number of problems, x_i is the parameter measured for the i-th problem, and μ is the mean value of the parameter throughout the whole test set.

Figure 6 shows the evolution of the variance of the solving rate when the density ρ increases. For small problem densities, the algorithms are in general capable to solve most if not all of the test problems. This means that there is little variance throughout the testing set in this respect, resulting in small values for the variance of the solvability rate.

As ρ increases, the problems become increasingly difficult, and so higher proportions become unsolvable, culminating with about 50-50 solvability for 130 targets for the standard breakout. That in turn, leads to the maximum standard deviation of about 0.25, making the variance close to 0.5.

Similar results can be observed for the number of rounds, and the time spent for solving the test problems.

Overall evaluation

Overall, our results have shown that the method based on the NPI-sets is much better than the one based on NI sets. This is due to the fact that in dense problems, usually there

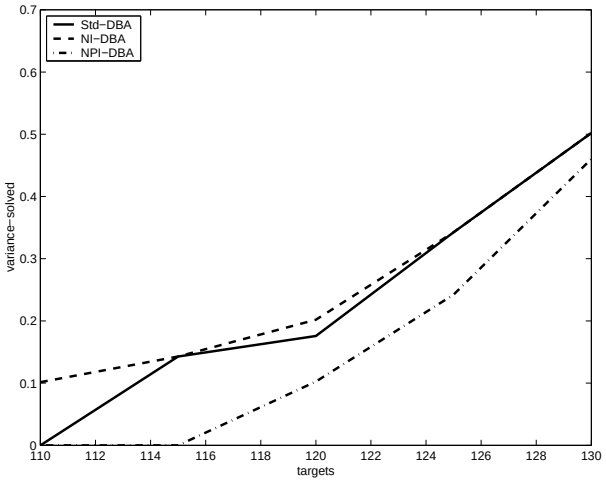

Figure 6: Variance of the solving rate

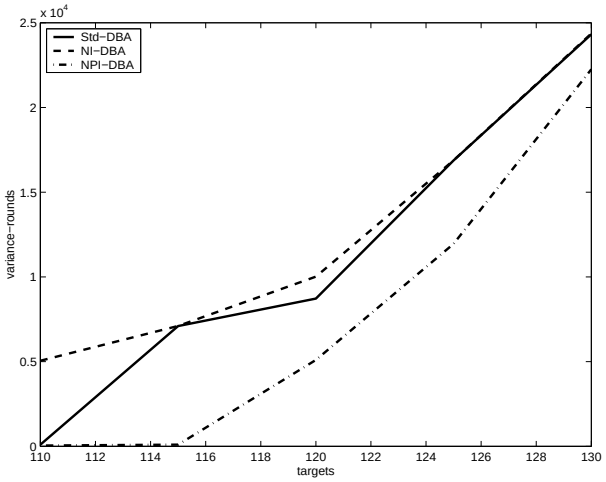

Figure 7: Variance of the number of rounds

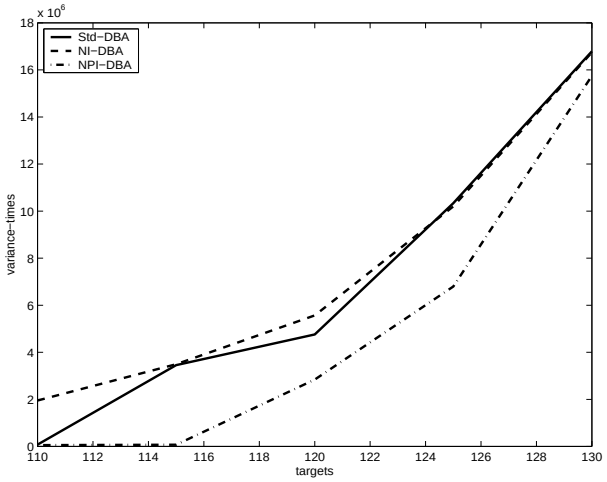

Figure 8: Variance of the solving time

is little or no NI at all, whereas NPI, being a weaker form of NI is computable also in denser problem instances.

Conclusions and future work

We presented two methods for improving the performance of the Distributed Breakout Algorithm, based on computing interchangeability sets. Our results have shown that standard DBA can be improved by using interchangeability techniques.

These techniques help containing conflicts and avoiding chain-reactions triggered by isolated conflicts that propagate uncontrollably through the constraint graph. We believe that these techniques are easy to generalize beyond mutual exclusion constraints, and therefore can be applied to a much wider range of problems, beyond resource allocation problems.

In our tests, NPI-DBA clearly outperforms standard DBA, and NI-DBA has comparable performances for particularly difficult problems. The method based on the NPI-sets is much better than the one based on NI sets because in dense problems, usually there is little or no NI at all, whereas NPI can still be computed.

We kept the initialization phase similar for all the studied methods in order to be able to compare the improvements brought by the *search strategies*. In addition to these improvements, further speedups were obtained with an "informed" initialization, based on the data available after the preprocessing phase.

It would be interesting to study in more detail the performance improvements brought by interchangeability techniques when the problem size increases, in terms of two dimensions:

- when the size of the sensor field increases, then also the maximum number of targets increases
- when the sensor ranges increases, then the size of the domains increases.

Further improvements could be obtained by:

- allowing multiple simultaneous changes of the local variables at each step
- trying a hierarchical approach to the problem, where certain agents are delegated as a "local authority" for solving a particularly difficult local problem

References

Berthe Y. Choueiry and Guevara Noubir: *On the Computation of Local Interchangeability in Discrete Constraint Satisfaction Problems*. In Proc. of AAAI-98, pages 326-333, Madison,Wisconsin, 1998.

Eugene C. Freuder *'Eliminating interchangeable values in Constraint Satisfaction Problems'* In Proc. of AAAI 1991, p 227-231

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

Nicoleta Neagu and Boi Faltings *Exploiting Interchangeabilities for Case Adaptation* In Proceedings of ICCBR'01

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

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, 3(2), 189-211

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