Asynchronous Backtracking for Asymmetric DisCSPs

Roie Zivan and Amnon Meisels
{zivanr,am}@cs.bgu.ac.il

Department of Computer Science,
Ben-Gurion University of the Negev,
Beer-Sheva, 84-105, Israel

Abstract. Distributed constraint satisfaction problems (*DisCSP*s) with asymmetric constraints reflect the fact that agents may wish to retain their constraints private. The set of valid pairs of values of every binary constraint is split between the two constrained agents.
An asynchronous backtracking algorithm for asymmetric DisCSPs is presented. The new algorithm is based on asynchronous backtracking (ABT), but, propagates assignments both to lower priority agents and to higher priority agents. The ABT_ASC algorithm is evaluated experimentally on randomly generated asymmetric $DisCSPs$. Its performance is compared to that of the privacy keeping version of ABT, proposed by Brito and Meseguer, which splits the search into two phases. The ABT_ASC algorithm improves the run-time of the 2-phase ABT by a large factor with no additional load on the communication network.

1 Introduction

Distributed constraint satisfaction problems (*DisCSP*s) are composed of agents, each holding its local constraints network, that are connected by constraints among variables of different agents. Agents assign values to variables, attempting to generate a locally consistent assignment that is also consistent with all constraints between agents (cf. [Yok00,SGM96]).

Distributed CSPs are an elegant model for many every day combinatorial problems that are distributed by nature. Take for example a large hospital that is composed of many wards. Each ward constructs a weekly timetable assigning its nurses to shifts. The construction of a weekly timetable involves solving a constraint satisfaction problem for each ward. Some of the nurses in every ward are qualified to work in the *Emergency Room*. Hospital regulations require a certain number of qualified nurses (e.g. for Emergency Room) in each shift. This imposes constraints among the timetables of different wards and generates a complex Distributed CSP [SGM96].

In the hospital example, wards are usually not willing to reveal their constraints to other wards. A better model for a realistic $DisCSP$ can be such that each inter-agent constraint is composed of two disjoint parts, each held by one of the two constraining agents. Each agent holds its part of the constraint data [BM03]. This is in contrast to common assumptions that are used for asynchronous backtracking. Standard ABT assumes a priority order of all agents. Higher priority agents perform assignments and send them via messages to lower priority agents [Yok00]. The standard model of ABT further assumes that every inter-agent constraint can be checked by the lower priority agent that is involved in the constraint [Yok00].

In many real world problems the above assumptions are too strong. When each of the constraining agents holds parts of the constraints privately, checking for consistency has to be performed by both of the constrained agents.

A solution for solving asymmetric DisCSPs was suggested by [BM03], who proposed a model for asymmetric constraints which they term *Partially Known Constraints*

(PKC). In the PKC model each binary constraint is divided between the two constraining agents. In order to solve the resulting $DisCSP$ with asymmetric constraints, a two phase asynchronous backtracking algorithm ($DisFC$-PKC) was proposed [BM03]. Similarly to standard ABT, a static order of priorities is defined among all agents.

In the first phase an asynchronous backtracking algorithm is performed, in which only the constraints held by the lower priority agents are examined. In other words, only one of the two constraining agents in each binary constraint checks its consistency. When a solution is reached, a second phase is performed in which the consistency of the solution is checked again, according to the constraints held by the higher priority agents in each binary constraint. If no constraint is violated, a solution is reported, if there are violated constraints, the first phase is resumed after the necessary $Nogoods$ are recorded [BM03].

The first and immediate drawback of a two-phase algorithm is the effort of producing solutions in each first phase. Since constraints in the opposite direction are not examined, large parts of the search space, which could have been pruned if all constraints were considered, are being exhaustively scanned. The second drawback is the synchronized manner in which the algorithm switches between the two phases. For each such switch among phases, a termination detection mechanism must be performed which is a complicated task in ABT. Furthermore, all agents must be informed about every switch between phases. This requires global monitoring that is in contrast to the independency of agents in a distributed asynchronous system.

The present study proposes a distributed search algorithm, *Asynchronous Backtracking for Asymmetric Constraints* (ABT_ASC), that checks inter-agent constraints asynchronously at both of the constraining agents.

In ABT_ASC, agents send their proposed assignments to all their neighbors in the constraints graph. Agents assign their local variables according to the priority order as in standard ABT, but check the constraints also against the assignment of lower priority agents. When an agent detects a conflict between its own assignment and the assignment of an agent with a lower priority than itself, it sends a $Nogood$ to the lower priority agent *but keeps its assignment*. Agents which receive a *Nogood* from higher priority agents, perform the same operations as if they have produced this $Nogood$ themselves. As in ABT [Yok00,BMBM05], the agents remove their current assignment from their current-domain, store the eliminating $Nogood$ and reassign their variable.

This results in a one phase, correct and complete asynchronous backtracking algorithm, which solves $DisCSPs$ with asymmetric constraints. The asynchronous processing of all the constraints in a single phase generates a much smaller search space. The search space scanned by ABT_ASC is the same size as would have been searched by standard ABT on symmetric $DisCSPs$ with the same constraints. The improvement in efficiency over the two phase algorithm of [BM03] is large.

The same privacy preservation methods suggested in the PKC model for a two-phase algorithm [BM03] can be used in ABT_ASC. Therefore, the large improvement in run-time, is achieved by ABT_ASC without revealing additional information. The experiments presented in the present study, show that this improvement is achieved with no additional load on the communication network.

2 Distributed Asymmetric CSPs

Asymmetric constraints are defined for $DisCSP$s by the PKC model of [BM03]. All constraints are binary constrains. For each pair of agents A_i and A_j, the set C_{ij} includes all constraints between A_i and A_j. The set C_{ij} is divided into two non intersecting

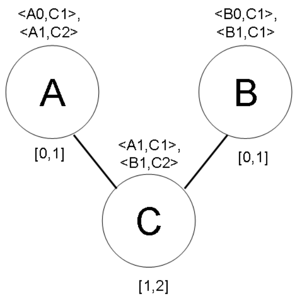


Fig. 1. An Asymmetric DisCSP.

subsets $C_{(i)j}$, which is held by agent A_j, and $C_{i(j)}$ which is held by agent A_i. Consider the example in Figure 1. There are three agents in the constraints network in Figure 1: A, B, C. The domains of the agents are depicted in the figure, A and B have the values 0, 1 and C has 1,2. Each agent holds its part of the constraints. A has forbidden pairs of assignments with C $< A1, C2 >, < A0, C1 >$. B has forbidden pairs of assignments with C $< B0, C1 >, < B1, C1 >$. Agent C holds two binary constraints it is involved in. With agent A it has the forbidden pair of assignments $< A1, C1 >$ and with agent B the pair $< B1, C2 >$.

Running standard ABT on the example in Figure 1, both agents A and B assign the value 0 and send **ok?** messages to agent C. Agent C receives the two **ok?** messages, updates its $AgentView$ and assigns itself the value 1. This value is consistent with both assignments of its higher priority agents (A and B), *according to the part of the constraints held by C*. Clearly, this assignment conflicts with the parts of the constraints held by both A and B. If C sends the solution for checking by higher order agents (i.e. A and B), as it does in a two phase algorithm [BM03], it will fail. A possible solution to the asymmetric $DisCSP$ in Figure 1 is $< A, 0 >, < B, 0 >, < C, 2 >$.

3 Asynchronous Backtracking for Asymmetric Constraints

In order to perform asynchronous backtracking, in a single phase, on asymmetric DisCSPs, all constraints must be satisfied including the constraints held by the higher priority agents. In standard ABT, binary constraints are held completely by the lower priority agent involved in each constraint according to a static priority order among agents [Yok00,BMBM05]. Lower priority agents check consistency of assignments received from higher priority agents via **ok?** messages [Yok00]. In asymmetric $DisCSPs$, constraints are only partially known to each of the participating agents. Consequently, both of the constrained agents need to check the consistency of their assignments against each other. This means that checking consistency of a pair of constrained assignments by the lower priority agent of the two is no longer sufficient.

In Asynchronous Backtracking for Asymmetric Constraints (ABT_ASC) each agent checks its constraints with all constraining agents (i.e. neighbors in the constraint graph). This includes higher priority, as well as lower priority constraining agents. Agents hold in their $Agent\_Views$ assignments of agents with higher and lower priorities. Values from the domain of agents are eliminated *only if they violate constraints with higher priority agents*. After a new assignment is found to be consistent with all assignments of higher priority agents in the $Agent\_view$, the selected assignment is checked against the assignments of lower priority agents. If a conflict is detected, the agent keeps its assignment and sends a $Nogood$ including its own assignment and the conflicting assignment to the lower priority agent that owns the conflicting assignment. An agent

```
when received (ok?, $(x_j, d_j)$) do:
1.  add $(x_j, d_j)$ to $Agent\_View$;
2.  remove inconsistent nogoods;
3.  if (conflicting $(x_i, d_i)$ and $(x_j, d_j)$)
4.      if ($x_j$ has lower priority than $x_i$)
5.          $nogood \leftarrow \{(x_i, d_i)(x_j, d_j)\}$;
6.          send (nogood, $(x_i, nogood)$) to $x_j$;
7.      else
8.          $nogood \leftarrow (x_j, d_j)$;
9.          store $nogood$;
10.         check_agent_view;

when received (nogood, $x_j, nogood$) do:
1.  if ($nogood$ consistent with $Agent\_View$
            and $current\_assignment$)
2.      store $nogood$;
3.      if ($nogood$ contains an agent $x_k$
                that is not a neighbor of $x_i$)
4.          add $x_k$ to neighbor list;
5.          request $x_k$ to add $x_i$ as a neighbor;
6.          add $(x_k, d_k)$ to $Agent\_View$;
7.      check_agent_view;
8.  else
9.      send (ok?, $(x_i, current\_value)$) to $x_j$;

procedure check_agent_view
1.  if(no value in $D_i$ is consistent with higher
            priority assignments in $Agent\_View$)
2.      backtrack;
3.  else
4.      select $d \in D_i$ which is consistent with higher
                priority assignments in $AgentView$;
5.      $current\_value \leftarrow d$;
6.      send (ok?,$(x_i, d)$) to $neighbors$;
7.      for each (conflicting lower priority
                assignment $(x_j, d_j)$)
8.          $nogood \leftarrow \{(x_i, d_i)(x_j, d_j)\}$;
9.          send (nogood, $(x_i, nogood)$) to $x_j$;

procedure backtrack
1.  $nogood \leftarrow resolve\_inconsistent\_subset$;
2.  if ($nogood$ is empty)
3.      broadcast to other agents that there
                is no solution;
4.      stop;
5.  select $(x_j, d_j)$ where $x_j$ has the lowest
            priority in $nogood$;
6.  send (nogood, $x_i$, $nogood$) to $x_j$;
7.  remove $(x_j, d_j)$ from $Agent\_View$;
8.  check_agent_view;
```

Fig. 2. ABT_ASC algorithm

which receives a $Nogood$ acts the same with $Nogoods$ received from higher and lower priority agents.

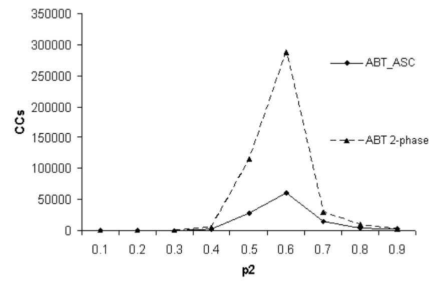
Figure 2 presents the code of ABT_ASC. When an **ok?** message is received, the $Agent\_View$ is updated with the received assignments and non consistent $Nogoods$ are eliminated (lines 1,2). A conflicting assignment is treated according to the priority of the sending agent. If its priority is lower than the receiving agent, the receiving agent keeps its assignment and sends a $Nogood$ to the lower priority sender which contains the received and the current assignment (lines 4-6). If the priority of the sending agent is higher than the receiver, the receiving agent stores the $Nogood$ containing the received assignment and calls procedure **check_agent_view** in order to find a different assignment, consistent with the updated $Agent\_View$ (lines 8-10).

When a **Nogood** is received, the agents act exactly the same as in the standard ABT algorithm. Procedure **backtrack** is also the same as in standard ABT [Yok00,BMBM05].

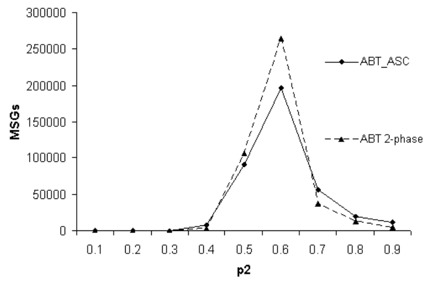
The main difference between ABT_ASC and ABT is in procedure **check_agent_view**. After an assignment, which is consistent with all assignments of agents with higher priority in the $Agent\_View$ is selected (lines 1-5), it is checked against the assignments of lower priority agents. For each violation of a constraint between the selected assignment and an assignment of a lower priority agent in the $Agent\_View$, a $Nogood$ containing both conflicting assignments is sent to the lower priority agent (lines 7-9). As in the case when a conflicting **ok?** message from a lower priority agent is received, a $Nogood$ is sent but the current assignment is not changed.

4 Experimental Evaluation

The common approach in evaluating the performance of distributed algorithms is to compare two independent measures of performance - time, in the form of steps of com-

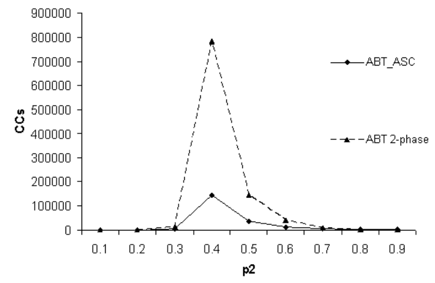


(a)

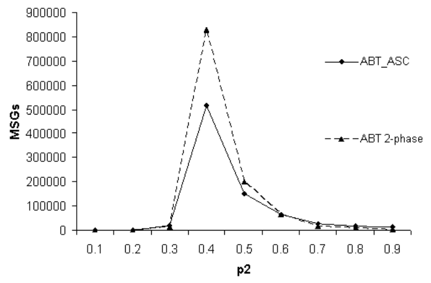


(b)

Fig. 3. Non concurrent constraints checks performed (a) and Total number of messages sent (b), by $ABT\_2\text{-}Phase$ and ABT_ASC ($p_1 = 0.4$).



(a)



(b)

Fig. 4. Non concurrent constraints checks performed (a) and Total number of messages sent (b), by $ABT\_2\text{-}Phase$ and ABT_ASC ($p_1 = 0.7$).

putation [Lyn97,Yok00], and communication load, in the form of the total number of messages sent [Lyn97]. Comparing the number of non-concurrent steps of computation of search algorithms on DisCSPs, measures the time of run of the algorithms.

Non concurrent computation steps, are counted by the method of [MRKZ02]. In order to take into account the local computational effort of agents in each step, the number of non concurrent constraints check performed ($NCCC$s) is counted instead of computation steps [MRKZ02].

All experiments were conducted on random networks with 15 agents ($n = 15$) each holding exactly one variable, 10 values for each variable ($k = 10$). Two values of constraints density were used $p_1 = 0.4$ and $p_1 = 0.7$ The tightness value p_2, is varied between 0.1 and 0.9, to cover all ranges of problem difficulty. For every two agents A_i and A_j, the set of illegal pairs of values C_{ij}, was randomly split among the two agents and each part was uniquely assigned to one of the agents involved. For each pair of fixed density and tightness (p_1, p_2), 50 different random problems were solved by each algorithm and the results presented are an average of these 50 runs.

Figure 3(a) presents the computational effort in number of $NCCCs$ to find a solution for *Asymmetric DisCSPs*. ABT_ASC is compared to the 2-phase version of ABT proposed by [BM03]. As can be seen in Figure 3(a), ABT_ASC performs much better than *2-phase ABT*. On the hardest instances ($p_2 = 0.6$), the factor of improvement is 6.

Figure 3(b) presents the total number of messages sent by both algorithms. The load on the network is very similar for both algorithms. It is important to note that these results do not include the overhead caused in the two phase version by informing agents that the second phase begins, and a need for multiple idle detection.

Figures 4(a) and 4(b) present similar results for high density *Asymmetric DisCSPs*. While the runtime improvement is similar to the improvement in low density $DisCSPs$ the results in network load are conclusive in favor of ABT_ASC.

5 Discussion

During the algorithm run, $explicit\ Nogoods$ are sent by higher priority agents to lower priority agents. Although *2-phase* ABT records similar $Nogoods$ it does so only when it examines solutions of the first phase during the second phase [BM03].

To enhance privacy preservation of constraints, one can use two approaches. The first can be used when the agents know the content of the initial domain of their neighboring agents. If this is the case, the algorithm is performed using the method of [BM03] ($DisFC$). Instead of including the selected assignment in an **ok?** message, agents send to their neighbors the subset of the neighbor's domain, which is consistent with the assignment, in the message. Agents hold in their *AgentViews* a counter of the number of changed assignments received from each agent. This way, agents generate *Nogoods* which include the counter values in their *AgentViews* i.e. the indices of the assignments instead of the assignments themselves. This of course conceals the constraints which would have bean revealed by the $explicit\ Nogoods$.

If domains are not known, agents can conceal the identity of the message sender in $Nogood$ messages. This way the receiver of a $Nogood$ will not know if it is an $explicit\ Nogood$ sent by a higher priority agent or a $regular\ Nogood$ sent by a lower priority agent. This idea has one flaw. In ABT when a $Nogood$ is not accepted, an **ok?** message must be sent back to the sending agent to inform that the receiver keeps its assignment [Yok00,BMBM05]. If the sender is unknown, an agent which discards a received $Nogood$ must send an **ok?** message to all its neighbors. This will increase the number of messages sent by the algorithm.

The ABT_ASC algorithm, proposed in the present study, performs a single phase asynchronous backtracking algorithm to solve *Asymmetric DisCSPs*. All constraints are processed asynchronously in a single phase and the resulting search space explored is the same as in standard ABT. The experimental results show a clear advantage of the proposed ABT_ASC algorithm over the 2-phase ABT ($DisFC$-PKC) algorithm of [BM03]. This advantage is achieved without additional network load.

References

[BM03] I. Brito and P. Meseguer. Distributed forward checking. In *Proc. CP-2003*, pages 801–806, September, Ireland, 2003.

[BMBM05] C. Bessiere, A. Maestre, I. Brito, and P. Meseguer. Asynchronous backtracking without adding links: a new member in the abt family. *Artificial Intelligence*, 161:1-2:7–24, January 2005.

[Lyn97] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Series, 1997.

[MRKZ02] A. Meisels, I. Razgon, E. Kaplansky, and R. Zivan. Comparing performance of distributed constraints processing algorithms. In *Proc. AAMAS-2002 Workshop on Distributed Constraint Reasoning DCR*, pages 86–93, Bologna, July 2002.

[SGM96] G. Solotorevsky, E. Gudes, and A. Meisels. Modeling and solving distributed constraint satisfaction problems (dcsps). In *Constraint Processing-96*, pages 561–2, New Hamphshire, October 1996.

[Yok00] M. Yokoo. Algorithms for distributed constraint satisfaction problems: A review. *Autonomous Agents & Multi-Agent Sys.*, 3:198–212, 2000.