Abstract agents for Shared instantiation in ABT

[Research Note]

Marius-Călin Silaghi*
Florida Institute of Technology (FIT)
Melbourne, Florida 32901-6988, USA
msilaghi@cs.fit.edu

Ion Constantinescu
Swiss Federal Institute of Technology (EPFL)
CH-1015 Lausanne, Switzerland
Ion.Constantinescu@epfl.ch

ABSTRACT

Complete asynchronous search algorithms for DisCSPs require an order on participating agents. This order has been often perceived as strongly harming fairness. However, already in the original presentation of asynchronous backtracking (ABT) agents stand for variables. We show here that an additional level of abstraction in ABT, namely replacing ABT agents with a *cooperation Mechanism*, can lead to important shifts in fairness properties. It is remarkable that the result can be achieved without losing completeness and with minimal adaptations to the distributed protocol.

1. INTRODUCTION

Distributed Constraint Satisfaction Problems (DisCSPs) are satisfaction problems where constraints and/or variables are owned by distinct agents that cooperate for finding solutions satisfying all of them. The fact that an agent A owns a constraint can mean that A is interested in enforcing the constraint or simply that he knows it. Owning a variable can mean owning knowledge about its domain (owning a unary constraint), or having the right to propose discussions about possible instantiations for the variable.

In case of conflict in distributed search over DisCSPs, ties have to be broken in a consistent manner [2]. Therefore, complete asynchronous search algorithms for DisCSPs require an order on participating agents. In many types of problems, this order can harm fairness as higher priority agents can propose first their preferred alternatives.

However, already in the original presentation of asynchronous backtracking (ABT) [9] agents stand for variables. We show here that an additional level of abstraction in ABT, namely logically replacing ABT agents with a *cooperation Mechanism*, can lead to important shifts in fairness properties. It is remarkable that the result can be achieved without losing termination and with minimal adaptations to the distributed protocol.

2. RELATED WORK

The first complete asynchronous search algorithm for DisCSPs is the Asynchronous Backtracking (ABT)[9]. The approach in [9] considers that each agent maintains only one variable. More complex definitions were given later [10]. Other definitions of DisCSPs [11, 8, 5] have considered the case where the interest on constraints is distributed among agents. [8] proposes versions that fit the structure of a real problem (the nurse transportation problem). The Asynchronous Aggregation Search (AAS) [5] algorithm is a first extension of ABT to the case where the same variable can be instantiated by several agents. It mainly increases the support for abstractions. A framework for supporting proposals on conflicting resources is defined in [6]. A second technique is then presented in [4] and allows for simple understanding of privacy/efficiency mechanisms. [1] shows how **add-link** messages can be avoided in ABT. [3] uses a consensus algorithm for deciding ordering during search and [7] shows how an abstract agent can be used for modeling the reordering decision of a set of agents with majority voting. No such approach is known for increasing fairness in proposing assignments.

3. ASYNCHRONOUS BACKTRACKING (ABT)

In asynchronous backtracking, the agents run concurrently and asynchronously. Each agent A_i instantiates its variable x_i and communicates the variable value to the relevant agents. [9] assumes FIFO channels. The channels are reliable, delivering messages within a finite delay, τ.

Definition 1 (Assignment) *An assignment for a variable x_i is a tuple $\langle x_i, v\rangle$ where v is a value from the domain of x_i.*

Since the channels deliver messages in FIFO order, the assignments are received in the order in which they are generated. A static order is imposed on agents and we assume that A_i has the i-th position in this order. If $i>j$ then A_i has a *lower priority* than A_j and A_j has a *higher priority* then A_i. A_j can impose first eventual preferences it has on its values

Each constraint C is evaluated by the lowest priority agent whose variable is involved in C. It is denoted *CEA(C)*.

Each agent holds a list of *outgoing links* represented by a set of agent names. Links are associated with constraints. ABT assumes that every link is directed from the value sending agent to the constraint-evaluating-agent.

Definition 2 (Agent_View) *The* agent view *of an agent, A_i, is a set containing the* newest *assignments received by A_i for distinct variables.*

*This work was performed while the first author was working at EPFL, supported by the Swiss National Science Foundation project number 21-52462.97.

when received *(***ok?**,$\langle x_j, d_j \rangle$*)* **do**
add(x_j,d_j) to *agent view*;
check_agent_view;
end do.
when received *(***nogood**,A_j,$\neg N$*)* **do**
when $\langle x_k, d_k \rangle$, *where* x_k *is not connected, is contained in* $\neg N$
send **add-link** to A_k;
add $\langle x_k, d_k \rangle$ to *agent view*;
put $\neg N$ in *nogood-list*;
old_value $\leftarrow$ *current_value*;
check_agent_view;
when old_value = current_value
1 send (**ok?**,$\langle x_i$, *current_value*$\rangle$) to A_j;
end do.
procedure check_agent_view do
when agent view *and* current_value *are not consistent*
if *no value in* D_i *is consistent with* agent view **then**
backtrack;
else
select $d \in D_i$ where *agent view* and d are consistent;
current_value $\leftarrow$ d;
send (**ok?**,$\langle x_i, d \rangle$) to lower priority agents in *outgoing links*;
end
end do.
procedure backtrack do
nogoods
$\leftarrow \{V \mid V =$ inconsistent subset of *agent view*$\}$;
when *an empty set is an element of* nogoods
broadcast to other agents that there is no solution;
terminate this algorithm;
for every $V \in$ nogoods **do**
select $\langle x_j, d_j \rangle$ where x_j has the lowest priority in V;
2 send (**nogood**,A_i,V) to A_j;
remove (x_j,d_j) from *agent view*;
end do
check_agent_view;
end do.

Algorithm 1: Procedures of A_i for receiving messages in ABT.

Based on their constraints, the agents perform inferences concerning the assignments in their *agent view*. By inference the agents generate new constraints called *nogoods*.

Definition 3 (Nogood) *A* nogood *has the form* $\neg N$ *where* N *is a set of assignments for distinct variables.*

The following types of messages are exchanged in ABT:

- **ok?** message transporting an assignment is sent to a constraint-evaluating-agent to ask whether a chosen value is acceptable.
- **nogood** message transporting a *nogood*. It is sent from the agent that infers a *nogood* $\neg N$, to the constraint-evaluating-agent for $\neg N$.
- **add-link** message announcing A_i that the sender A_j owns constraints involving x_i. A_i inserts A_j in its *outgoing links* and answers with an **ok?**.

The agents start by instantiating their variables concurrently and send **ok?** messages to announce their assignment to all agents with lower priority in their *outgoing links*. The agents answer to received messages according to the Algorithm 1 [9].

4. INTRODUCING COORDINATION

In this paper our objective is to explore ways for allowing the process of instantiation a variable to be controlled by two or more agents. For that we propose an extension to the model described above with the concepts of *coAgent* and *coVariable* (cooperating, collaborating). The assignment process itself is going to be exterior to the ABT algorithm and we are going to refer to it as the *coMechanism*. A *coMechanism* for deciding instantiations with interesting behavior could consist of a voting mechanism over the set of values obtained with unanimity when the acceptable sets of all agents are intersected.

First we present a basic set of relations between existing and proposed concepts and then we discuss a number of choices for completely defining this model.

Any *coAgent* controls the instantiation of exactly one *coVariable*. A **Distributed CSP** can have a number of zero or more *coAgent*s / *coVariable*s.

The case of a **Distributed CSP** with zero *coAgent*s is exactly the initial case presented above in Section 3. The case of a **Distributed CSP** with two ore more *coAgent*s can be easily derived from the case of a **Distributed CSP** with one *coAgent*. As such we are going to consider next the case of a **Distributed CSP** with one *coAgent* / *coVariable*.

The agents that participate to the *coMechanism* are said to *support* the *coAgent* or otherwise are called *supporting Agent*s. The agents that participate in the **ABT** are called *ABT Agent*s.

Further defining the newly proposed concepts raises a number of questions:

- 1. what is the intersection set between the *supporting Agent*s and *ABT Agent*s?
- 2. how are the agents controlling the *coMechanism* interacting with the agents participating to the **ABT**?
- 3. is an agent controlling the *coMechanism* and thus participating in the instantiation of the *coVariable* allowed to control also a local variable?

Our first proposed model gives the following responses:

- 1. only one agent acts as both a *supporting Agent* and a *ABT Agent* (the cardinality of the intersection set from 1. is one). This is actually going to be the *coAgent*.
- 2. the *coAgent* is going to act as a gateway running the **ABT** and also the *coMechanism*
- 3. the *coAgent* doesn't control a local variable

The **ABT** algorithm is going to be different in the case of the *coAgent* in the following aspects: for selecting a value $d \in D_i$ for the *coVariable* the *coAgent* iteratively checks if the value is consistent with the *agent view* but also runs the *coMechanism* for the value.

As such the **check_agent_view** procedure of the **ABT** algorithm is going to look as following:

procedure check_agent_view do
  when agent view *and* current_value *are not consistent*
    for every *value* $d \in D_i$ *consistent with* agent view **do**
      if *d consistent with* coMechanism **then**
        current_value $\leftarrow$ d;
        send (**ok?**,$\langle x_i, d \rangle$) to lower priority agents in *outgoing links*;
        return ;
      end
    end do
    if *no value d found* **then**
      backtrack;
    end
end do.

Algorithm 2: The **check_agent_view** in the case of the *coAgent*

Figure 1 shows an example of running the above algorithm with *coAgent*A_2.

In step (a) agent A_1 instantiate his variable to 1 and sends a **ok?** message to the *coAgent* A_2 . Upon receiving the **ok?** message agent A_2 chooses a possible value for his local variable - in our case 2 - and initiates the *coMechanism* with this variable. The *coMechanism* returns successfully and as such agent A_2 assigns the results to his current_value (*current_value* = 2).

In step (b) A_2 sends an **ok?** message to A_3 with the newly assigned value.

In step (c) A_3 is not able to instantiate his local variable due to incompatibility with the current value of the higher priority agent A_2 so it *backtrack* by sending back to A_2 a **nogood** message.

In step (d) A_2 has his local variable incompatible with his nogood-list and is also not able to find a new assignment. As such it is also forced to backtrack and sends back to A_1 a **nogood** message.

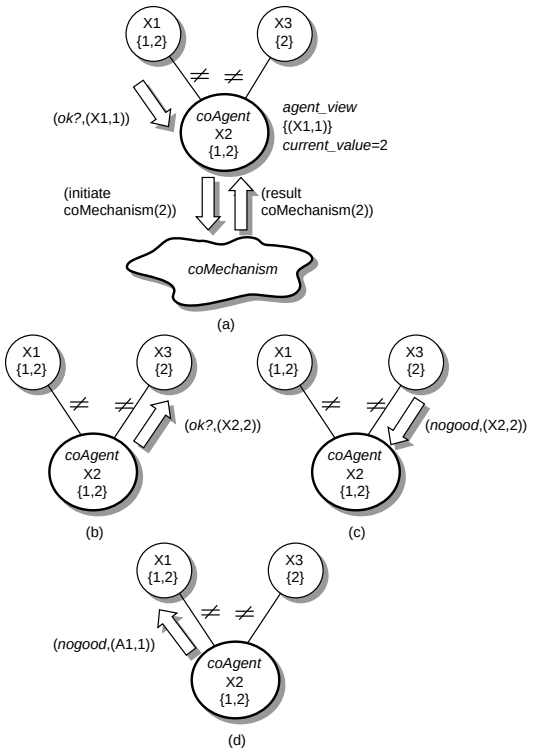

Figure 1: ABT algorithm with *coAgent* and *coMechanism*

5. ABSTRACT AGENTS

As noticed in the examples, the *coAgent*s in the previous algorithm can become bottlenecks in the interaction between ABT and *coMechanism*. We propose now a simpler formalism, ShABT, that modifies ABT but removes the bottlenecks and ensures the needed fairness. A total order is defined on the public variables of the DisCSP. ABT is modified as follows.

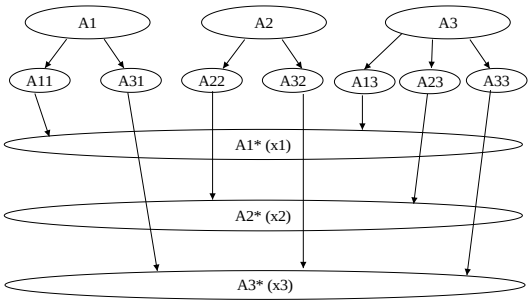

Figure 2: ShABT agent hierarchy.

Each physical agent A_i participates under the role/name A_{k_i} in the instantiation of each of its variables x_k and communicates the variable value to other interested agents. A_{k_i} is referred to as a *coAgent* and enforces the constraints of A_i involving x_k and higher priority variables. The abstract agent for x_i is denoted A_{i_*}. The abstraction hierarchy for

the previous problem is shown in Figure 2.

Each *coAgent* holds a list of *outgoing links* represented by a set of physical agent names. ShABT assumes that every link is directed from the value sending agent to the constraint-evaluating-agent. Each physical agent owns an *agent view* that can be accessed by all its *coAgent*s.

Based on their constraints, the *coAgent*s perform inferences concerning assignments in their *agent view* and explanations of disagreement of other *coAgent*s, received via *coMechanism*. To send a message to the abstract agent A_{i_*}, it is sent to a *coAgent* A_{i_t}, for some t.

The following types of messages are exchanged in ShABT:

- **ok?** message transporting an assignment is sent to a nogood owner physical agent to ask whether a chosen value is acceptable. The reception event is triggered for all *coAgent*s.
- **nogood** message transporting a *nogood*. It is sent from the *coAgent* that infers a *nogood* $\neg N$, to a constraint-evaluating-*coAgent* for $\neg N$.
- **add-link** message announcing A_{i_k} that the sender A_j owns constraints involving x_i. A_{i_k} inserts A_j in its *outgoing links* and answers with an **ok?**.
- **coMechanism** message announcing A_{i_t} that the sender A_{i_j} runs the *coMechanism* for the assignment of x_i. It transports the current round ID of the *coMechanism* on the variable, as well as the data needed for the *coMechanism*.
- **nogoodlist** messages are exchanged within the *coMechanism* for composing *nogood-lists* after failure.

The agents start by launching *coMechanism* for instantiating their variables concurrently. The agents answer to received messages according to the Algorithm 3. *coMechanism* rounds are tagged with monotonically increasing IDs. **coMechanism** messages carry the round ID and the local data needed for *coMechanism*. E.g., for the *coMechanism* mentioned in the previous section, the data can consists of the set of available values and a preference (e.g. a number from 0 to 1) for each of them. The winning assignment will belong to the intersection of the available values received for the current round from all *coAgent*s. It can be chosen by optimizing a function (e.g. sum) over the preferences of all agents. For ensuring consensus, ties can be broken with an order on agents.

Theorem 1 *ShABT is correct, complete and terminates.*

Proof The proof is identical with the proof of ABT in [9], where the reasoning is performed on variables.

5.1. POLYNOMIAL SPACE SHABT

ShABT has exponential space requirements, but a polynomial space version called pShABT can be obtained as follows:

when received *(***ok?**,$\langle x_j, d_j\rangle$*)* **do**
  add(x_j,d_j) to *agent view*;
  check_agent_view;
end do.
when received *(***nogood**,A_{j_t},$\neg N$*)* **do**
  Integrate-nogood($\neg N$);
  old_value ← *current_value*;
  check_agent_view;
  when old_value = current_value
1 send (**ok?**,$\langle x_i$, *current_value*$\rangle$) to A_t;

end do.
procedure *Integrate-nogood(*$\neg N$*)* **do**
  when $\langle x_k, d_k\rangle$, *where* x_k *is not connected, is contained in* $\neg N$
    send **add-link** to A_{k_*};
    add $\langle x_k, d_k\rangle$ to *agent view*;
  put $\neg N$ in *nogood-list*;
2 add other new assignments to *agent view*//pShABT;
end do.
procedure check_agent_view do
  when agent view *and* current_value *are not consistent*
3 select *d=coMechanism*(++lcm,*agent view*,D_i,*nogood-list*);
    if $d = \emptyset$ **then**
      wait and get **nogoodlist**(nl);
      foreach *(*$n \in nl$*)* **do**
        Integrate-nogood(n);
      end
      backtrack;
    else
      current_value ← d;
      send (**ok?**,$\langle x_i, d\rangle$) to lower priority agents in *outgoing links* and all $A_{i_f}, f > z$;
    end
end do.
procedure coMechanism*(cm,*agent view,D_i,nogood-list,*data)* **do**
  lcm ← max(cm,lcm);
  discard *coMechanism* data for rounds less than lcm;
  launch d=*coMechanism* by sending (*coMechanism*,A_{i_h},cm,local-data) to all agents A_{i_t};
  block waiting for *coMechanism* to end;
  compute and return d;
end do.
when received *(***coMechanism**,A_{i_h},*cm,data)* **do**
  d = *coMechanism*(cm,*agent view*,D_i,*nogood-list*);
  if *(*$d \neq \emptyset$*)* **then**
    current_value ← d;
    send (**ok?**,$\langle x_i, d\rangle$) to lower priority agents in *outgoing links*;
  else
4 send (**nogoodlist**,A_{i_h},*nogood-list*);
  end
end do.
procedure backtrack do
  nogoods ← $\{V \mid V =$ inconsistent subset of *agent view*$\}$;
  when *an empty set is an element of* nogoods
    broadcast to other agents that there is no solution;
    terminate this algorithm;
  for every $V \in$ nogoods **do**
    select $\langle x_j, d_j\rangle$ where x_j has the lowest priority in V;
5 send (**nogood**,A_{i_z},V) to A_{j_*};
    remove (x_j,d_j) from *agent view*;
  end do
  check_agent_view;
end do.

Algorithm 3: Procedures of A_{i_z} for receiving messages in ShABT.

- Instantiations have to be tagged (e.g. with the ID of *coMechanism*). Only the most recent is retained and old ones are invalidated.
- The line 2 can and has to be called.

- Only one valid nogood has to be stored for a value.

Theorem 2 *pShABT is correct, complete, terminates and has polynomial space requirements.*
Proof The proof is identical with the proofs for polynomial space versions of ABT, where the reasoning is performed on variables.

5.2. OPTIMIZATIONS

A lot of optimizations are possible in both ShABT and pShABT. For example, the *coMechanism* call at line 3 can be avoided if a nogood can be locally inferred. Especially for ShABT, but also for pShABT, at line 4, *coAgent*s can send a resume nogood (e.g. as the conflict list CL in AAS). Otherwise, for ShABT it becomes important to mark already sent nogoods and to avoid sending them again to the same target.

6. CONCLUSIONS

We have presented a way of modeling DisCSP agents by three levels of abstraction. This allows for running search protocols initially developed for ABT with totally ordered agents, while proposed assignments of variables are decided by majority voting of interested agents. We have applied these concepts to the basic ABT, obtaining the ShABT algorithm. While no experiments are available, the new approach clearly provides new fairness properties when compared to other existing approaches.

REFERENCES

[1] C. Bessière, A. Maestre, and P. Meseguer. Distributed dynamic backtracking. In *Proc. IJCAI DCR Workshop*, pages 9–16, 2001.

[2] Z. Collin, R. Dechter, and S. Katz. On the feasibility of distributed constraint satisfaction. In *Proceedings of IJCAI 1991*, pages 318–324, 1991.

[3] Amnon Meisels and Igor Razgon. Distributed forward checking with dynamic ordering. In *CP01 COSOLV Workshop*, pages 21–27, Paphos,Cyprus, December 2001.

[4] P. Meseguer and M. A. Jiménez. Distributed forward checking. In *CP DCS Workshop*, 2000.

[5] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *Proc. of AAAI2000*, pages 917–922, Austin, August 2000.

[6] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. ABT with asynchronous reordering. In 2^{nd} *A-P Conf. on Intelligent Agent Technology*, pages 54–63, Maebashi, Japan, October 2001.

[7] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Hybridizing abt and awc into a polynomial space, complete protocol with reordering. Technical Report #01/364, EPFL, May 2001.

[8] G. Solotorevsky, E. Gudes, and A. Meisels. Algorithms for solving distributed constraint satisfaction problems (DCSPs). In *Proceedings of AIPS96*, 1996.

[9] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *ICDCS*, pages 614–621, June 1992.

[10] M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Proceedings of 3rd ICMAS'98*, pages 372–379, 1998.

[11] Y. Zhang and A. K. Mackworth. Parallel and distributed algorithms for finite constraint satisfaction problems. In *Proc. of Third IEEE Symposium on Parallel and Distributed Processing*, pages 394–397, 1991.