# Consistency Maintenance for ABT

Marius-Călin Silaghi, Djamila Sam-Haroud, and Boi Faltings

Swiss Federal Institute of Technology (EPFL)
EPFL, CH-1015, Switzerland
{Marius.Silaghi,Djamila.Haroud,Boi.Faltings}@epfl.ch

**Abstract.** One of the most powerful techniques for solving centralized constraint satisfaction problems (CSPs) consists of maintaining local consistency during backtrack search (e.g. [11]). Yet, no work has been reported on such a combination in asynchronous settings[1]. The difficulty in this case is that, in the usual algorithms, the instantiation and consistency enforcement steps must alternate sequentially. When brought to a distributed setting, a similar approach forces the search algorithm to be synchronous in order to benefit from consistency maintenance. Asynchronism [24, 14] is highly desirable since it increases flexibility and parallelism, and makes the solving process robust against timing variations. One of the most well-known asynchronous search algorithms is Asynchronous Backtracking (ABT). This paper shows how an algorithm for maintaining consistency during distributed *asynchronous* search can be designed upon ABT. The proposed algorithm is complete and has polynomial-space complexity. Since the consistency propagation is optional, this algorithms generalizes forward checking as well as chronological backtracking. An additional advance over existing centralized algorithms is that it can exploit available backtracking-nogoods for increasing the strength of the maintained consistency. The experimental evaluation shows that it can bring substantial gains in computational power compared with existing asynchronous algorithms.

## 1 Introduction

Distributed constraint satisfaction problems (DisCSPs) arise when constraints and/or variables come from a set of independent but communicating agents. Successful centralized algorithms for solving CSPs combine search with local consistency. Most local consistency algorithms prune from the domains of variables the values that are locally inconsistent with the constraints, hence reducing the search space. When a DisCSP is solved by distributed search, it is desirable that this search exploits asynchronism as much as possible. Asynchronism gives the agents more freedom in the way they can contribute to search, allowing them to enforce individual policies (on privacy, computation, etc.). It also increases both parallelism and robustness. In particular, robustness is improved by the fact that the search can still detect unsatisfiability even in the presence of crashed agents. Existing work on asynchronous algorithms for distributed CSPs has focused on one of the following types of asynchronism:

[1] A preliminary version of this paper has been presented at the CP2000 Workshop on Distributed CSPs[15]

a) *deciding instantiations* of variables by distinct agents. The agents can propose different instantiations asynchronously (e.g. Asynchronous Backtracking (ABT) [24]).
b) *enforcing consistency*. The distributed process of achieving "local" consistency on the global problem is asynchronous (e.g. Distributed Arc Consistency [25]).

Combining these two techniques is however not as easy as in the synchronous setting. A straightforward mapping of the existing combination scheme cannot preserve asynchronism of type a [21, 4]. The contribution of this work is to consider consistency maintenance as a hierarchical nogood-based inference. This makes it possible to concurrently *i)* perform asynchronous search and *ii)* enforce the hierarchies of consistency, resulting in an *asynchronous* consistency maintenance algorithm. Since the consistency propagation is optional, this algorithms generalizes forward checking as well as chronological backtracking. More general than existing centralized algorithms, our approach can use any available backtracking nogoods to increase the strength of the maintained consistency. As expected from the sequential case, the experiments show that substantial gains in computational power can result from combining distributed search and distributed local consistency.

## 2 Related Work

The first complete asynchronous search algorithm for DisCSPs is the Asynchronous Backtracking (ABT) [23]. The approach in [23] considers that agents maintain distinct variables. Nogood removal was discussed in [8, 14]. Other definitions of DisCSPs have considered the case where the interest on constraints is distributed among agents [25, 20, 14, 7, 5]. [20] proposes algorithms that fit the structure of a real problem (the nurse transportation problem). The Asynchronous Aggregation Search (AAS) [14] family of protocols actually extends ABT to the case where the same variable can be instantiated by several agents (e.g. at different levels of abstraction [12, 16]). An agent may also not know all constraint predicates relevant to its variables. AAS offers the possibility to aggregate several branches of the search. An aggregation technique for DisCSPs was then presented in [10] and allows for simple understanding of privacy/efficiency mechanisms, also discussed in [6]. The use of abstractions, [16], not only improves on efficiency but especially on privacy since the agents need to reveal less their details. A general polynomial space reordering protocol is described in [13] and several heuristics (e.g. weak commitment-like) are discussed in [18]. [3] explains how **add-link** messages can be avoided. A technique enabling parallelization and parallel proposals in asynchronous search is described in [19]. Several algorithms for achieving distributed arc consistency are presented in [9, 25, 2].

## 3 Preliminaries

In this paper we target problems with finite domains (we target problems with numeric domains in [12, 16]). For simplicity, but here without loss of generality, we consider that each agent $A_i$ can propose instantiations to exactly one distinct variable, $x_i$ and

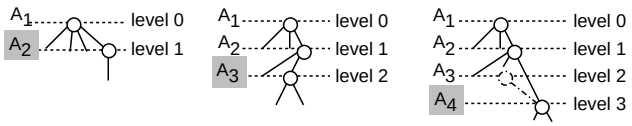



**Fig. 1.** *Distributed search trees in ABT: simultaneous views of distributed search seen by $A_2$, $A_3$, and $A_4$, respectively. Each arc corresponds to a proposal from $A_i$ to $A_j$. Circles show the believed state of an agent. Dashed circle and line show known state that may have been changed.*

knows all the constraints that involve $x_i$. Therefore each agent, $A_i$, knows a local CSP, **CSP($\mathbf{A_i}$)**, with variables **vars($\mathbf{A_i}$)**. We present the way in which our technique can be built on ABT, a simple instance of AAS for certain timings and agent strategies, but it can be easily adapted to more complex frameworks and extensions of AAS. ABT allows agents to asynchronously propose instantiations of variables. In order to guarantee completeness and termination, ABT uses a static order $\prec$ on agents. In the sequel of the paper, we assume that the agent $A_i$ has position $i, i \geq 1$, when the agents are ordered according to $\prec$. If $i{>}j$ then $A_i$ has a *lower priority* than $A_j$ and $A_j$ has a *higher priority* then $A_i$.[2] $A_i$ is then a successor of $A_j$, and $A_j$ a predecessor of $A_i$.

*Asynchronous distributed consistency:* Most centralized local-consistency algorithms prune from the domain of variables the values that are locally inconsistent with the constraints. Their distributed counterparts (e.g. [25]) work by exchanging messages on value elimination. The restricted domains resulting from such a pruning are called *labels*. In this paper we will only consider the local consistencies algorithms which work on labels for *individual* variables (e.g. arc-, bound-consistency). Let $P$ be a Distributed CSP with the agents $A_i, i{\in}\{1..n\}$. We denote by $C(P)$ the CSP defined by $\cup_{i\in\{1..n\}}$CSP($A_i$).[3] Let $\mathcal{A}$ be a centralized local consistency algorithm as just mentioned. We denote by DC($\mathcal{A}$) a distributed consistency algorithm that computes, by exchanging value eliminations, the same labels for $P$ as $\mathcal{A}$ for $C(P)$. When DC($\mathcal{A}$) is run on $P$, we say that $P$ becomes DC($\mathcal{A}$) consistent. Generic instances of DC($\mathcal{A}$) are denoted by DC. Typically with DC [25], the maximum number of generated messages is $a^2vd$ and the maximum number of sequential messages is $vd$ ($v$:number of variables, $d$:domain size, $a$:number of agents).

## 4 Asynchronous consistency maintenance

In the sequential/synchronous setting, the view of the search tree expanded by a consistency maintenance algorithm is unique. Each node at depth $k$, corresponds to assigning to the variable $x_k$ a value $v_i$ from its label. Initially the label of each variable is set to its full domain. After each assignment $x_k{=}v_i$, a local consistency algorithm is launched which computes for the future variables the labels resulting from this assignment.

[2] They can impose first eventual preferences they have on their values.

[3] The *union* of two CSPs, $P_1$ and $P_2$, is a CSP containing all the constraints and variables of $P_1$ and $P_2$.

In distributed search (e.g. ABT), each agent has its own perception of the distributed search tree. Its perception on this tree is determined by the proposals received from its predecessors. In Figure 1 is shown a simultaneous view of three agents. Only $A_2$ knows the fourth proposal of $A_1$. $A_3$ has not yet received the third proposal of $A_2$ consistent with the third proposal of $A_1$. However, $A_4$ knows that proposal of $A_2$. In Figure 1 we suppose that $A_4$ has not received anything valid from $A_3$ (e.g. after sending some nogood to $A_3$ which was not yet received). The term *level* in Figure 1 refers to the depth in the (distributed) search tree viewed by an agent.

Let $P$ be a Distributed CSP with the agents $A_i, i{\in}\{1..n\}$, $\mathcal{A}$ be a centralized local consistency algorithm and DC($\mathcal{A}$) one of its distributed counterparts. Suppose that the instantiation order of the variables in $C(P)$ is determined by the order of the agents in $P$. In order to guarantee that with DC($\mathcal{A}$) one maintains for the variables of agents $A_i$ of $P$ the same labels, $\mathcal{L}$, than with $\mathcal{A}$ in $C(P)$, one can simply impose that:

1. $Ai$ must have received the proposals of *all* its predecessors before launching DC($\mathcal{A}$),
2. $Ai$ cannot make any proposal with values outside $\mathcal{L}$, computed by DC($\mathcal{A}$).

This approach [21, 4] is synchronous. Alternatively, we propose to handle consistency maintenance as a hierarchical task. We show that $A_i$ can then benefit from the value eliminations resulting from the proposals of *subsets* of its predecessors, as soon as available. More precisely, if $A_i$ has received proposals from some of its $k$ first predecessors, we say that it can benefit from value elimination (nogoods) of level $k$. Such nogoods are determined by instantiations of $x_t$, $t{\leq}k$ (known proposals), DC process at level $k$ or inherited from DCs at previous levels along the same branch. A DC process of level $k$ is a process which only takes into account the known proposals of the $k$ first agents. The resulting labels are said to be of level $k$. When the nogoods defining labels are classified according to their corresponding levels, and when they are coherently managed by agents as shown here, the instantiation decisions and DCs of levels $k$ can then be performed asynchronously for different $k$ with polynomial space complexity and without loosing the inference power of DC($\mathcal{A}$). Moreover, backtrack-nogoods involving only proposals from agents $A_{i,i\leq k}$ can be used by DC at level $k$. Since the use of most nogoods is optional, many distinct algorithms result from the employment of different strategies by agents.

## 5 The DMAC-ABT protocol

This section presents DMAC-ABT (Distributed Maintaining Asynchronously Consistency for ABT), a complete protocol for maintaining asynchronously consistency. Since it builds on ABT, we start by recalling the necessary background and definitions.

### 5.1 ABT

In asynchronous backtracking, the agents run concurrently and asynchronously. Each agent instantiates its variable and communicates the variable value to the relevant

agents. As described for AAS [14], since we do not assume (generalized) FIFO channels, in the polynomial-space requirements description given here a **local counter**, $C^i_{x_i}$, in each agent $A_i$ is incremented each time a new instantiation is chosen. The current value of $C^i_{x_i}$ **tags** each assignment made by $A_i$ for $x_i$.

**Definition 1 (Assignment).** *An* assignment *for a variable $x_i$ is a tuple $\langle x_i, v, c\rangle$ where $v$ is a value from the domain of $x_i$ and $c$ is the* tag *value (value of $C^i_{x_i}$).*

Among two assignments for the same variable, the one with the higher *tag* (attached value of the counter) is the **newest**.

**Rule 1 (Constraint-Evaluating-Agent)** *Each constraint $C$ is evaluated by the lowest priority agent whose variable is involved in $C$. This agent is denoted CEA($C$).*

The set of constraints enforced by $A_i$ are denoted **ECSP($\mathbf{A_i}$)** and the set of variables that are involved in ECSP($A_i$) is denoted **evars($\mathbf{A_i}$)**, where $x_i \in$evars($A_i$). Each agent holds a list of **outgoing links** represented by a set of agents. 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, view($A_i$), containing the* newest *assignments received by $A_i$ for distinct variables.*

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

**Definition 3 (Explicit Nogood).** *An* explicit 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 an *explicit nogood*. It is sent from the agent that infers an *explicit 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 (given in [13]).

**Definition 4 (Valid assignment).** *An assignment $\langle x, v_1, c_1\rangle$ known by an agent $A_l$ is valid for $A_l$ as long as no assignment $\langle x, v_2, c_2\rangle$, $c_2 > c_1$, is received.*

A **nogood is valid** if it contains only valid assignments. The next property is a consequence of the fact that ABT is an instance of AAS.

**Property 1** *If only one valid nogood is stored for a value then ABT has polynomial space complexity in each agent, $O(dv)$, while maintaining its completeness and termination properties. $d$ is the domain size and $v$ is the number of variables.*

**when received** *(***ok?**,$\langle x_j, d_j, c_{x_j}\rangle$*)* **do**
  **if**(old $c_{x_j}$) return;
  add($x_j$,$d_j$,$c_{x_j}$) to *agent_view*;
  eliminate invalidated nogoods;
  **check_agent_view**;

**when received** *(***nogood**,$A_j$,$\neg N$*)* **do**
  **when** any $\langle x, d, c\rangle$ in $N$ is invalid (old $c$) **then**
    send (**ok?**,$\langle x_i$, *current_value*, $C^i_{x_i}\rangle$) to $A_j$;
    return;
  **when** $\langle x_k, d_k, c_k\rangle$, where $x_k$ is not connected, is contained in $\neg N$
    send **add-link** to $A_k$;
    add $\langle x_k, d_k, c_k\rangle$ to *agent_view*;
  when $\langle x_i, d, c\rangle \in N$, then put $\neg N$ in *nogood-list* for $x_i$=$d$;
  add other new assignments to *agent_view*;
1.1 eliminate invalidated nogoods;
  *old_value* $\leftarrow$ *current_value*;
  **check_agent_view**;
  **when** *old_value* $=$ *current_value*
1.2   send (**ok?**,$\langle x_i$, *current_value*, $C^i_{x_i}\rangle$) to $A_j$;

**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; $C^i_{x_i}$++;
      send (**ok?**,$\langle x_i, d, C^i_{x_i}\rangle$) to lower priority agents in *outgoing links*;

**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*;
    select ($x_j$,$d_j$,$c$) where $x_j$ has the lowest priority in $V$;
    send (**nogood**,$A_i$,$V$) to $A_j$;
    eliminate invalidated explicit nogoods;
    remove ($x_j$,$d_j$,$c$) from *agent_view*;
  **check_agent_view**;

Algorithm 1: *Procedures of $A_i$ for receiving messages in ABT with nogood removal.*

**when received(propagate,**$A_j$**,**$k$**,**$c^k_{x_v}(j)$**,**$V\rightarrow(x_v \notin l)$**) do**

2.1 **when** have higher tag $c^k_{x_v}(j,i) \geq c^k_{x_v}(j)$ **then** return;
$c^k_{x_v}(j,i) \leftarrow c^k_{x_v}(j)$; **when** any $\langle x, d, c\rangle$ in V is invalid (old $c$) **then** return;
**when** $\langle x_u, d_u, c_u\rangle$, where $x_u$ is not connected, is contained in V
send **add-link** to $A_u$; add $\langle x_u, d_u, c_u\rangle$ to *agent_view*;
2.2 add other new assignments in V to *agent_view*; eliminate invalidated nogoods;
$cn^k_{x_v}(i,j) \leftarrow \{V\rightarrow(x_v \notin l)\}$; **maintain_consistency**(minimal level that is modified);
**check_agent_view**; //only satisfies consistency nogoods of levels t, t<cL$_i$;

**procedure maintain_consistency(***minT***) do**

**if** (minT > cL$_i$) **then** return; //cL$_i$ is the current inconsistent level (initially $i$+1);
2.3 **for** (t←minT; t≤i; t++)
2.4 *new-cn* ← consistency nogood for $x_i$ after local consistency on $P_i(t)$;
**when** (domain wipe out by computing the explicit nogoods $\mathcal{N}$)
**for every** $V \in \mathcal{N}$;
select $\langle x_j, d_j, c_{x_j}\rangle$ where $x_j$ has the lowest priority in $V$;
send (**nogood**,$A_i$,$V$) to $A_j$; eliminate invalidated explicit nogoods;
remove $\langle x_j, d_j, c_{x_j}\rangle$ from *agent_view*;
2.5 cL$_i$ ←t; **break**;
**when** *new-cn* shrinks label of $x_i$ (obtained from $\cup_{k\leq t} cn^k_{x_i}(i,i)$)
2.6 $cn^t_{x_i}(i,i) \leftarrow$ *new-cn*; $C^t_{x_i}$++;
send (**propagate**,$A_i$,k,$C^t_{x_i}$,*new-cn*) to agents $A_j, j\geq t, x_i \in$ **evars**($\mathbf{A_j}$);

Algorithm 2: *Procedure of $A_i$ for receiving* **propagate** *messages in DMAC-ABT.*

### 5.2 DMAC-ABT

Parts of the content of a message may become *invalid* due to newer available information. We require that messages arrive at destination in finite time after they are sent. The receiver can discard the invalid incoming information, or can reuse invalid nogoods with alternative semantics (e.g. as redundant constraints).

In addition to the messages of ABT, the agents in DMAC-ABT may exchange information about nogoods inferred by DCs. This is done using **propagate** messages as shown in Algorithm 2. Before making their first proposal as in ABT, cooperating agents can start with a call to **maintain_consistency**(0).

**Definition 5 (Consistency nogood).** *A consistency nogood for a level $k$ and a variable $x$ has the form $V\rightarrow(x\in l^k_x)$ or $V\rightarrow\neg(x\in s\backslash l^k_x)$. $V$ is a set of assignments. Any assignment in $V$ must have been proposed by $A_k$ or its predecessors. $l^k_x$ is a label, $l^k_x\neq\emptyset$. $s$ is the initial domain of $x$.*[4]

The **propagate** messages for a level $k$ are sent to all agents $A_i, i\geq k, x_i\in$**evars**($\mathbf{A_i}$). They take as parameters the reference $k$ of a level and a consistency nogood. Each consistency nogood for a variable $x_i$ and a level $k$ is **tagged** with the value of a counter $C^k_{x_i}$ maintained by the sender. The agents $A_i$ use the most recent proposals of the agents $A_j, j\leq k$ when they compute DC consistent labels of level $k$. $A_i$ may receive valid consistency nogoods of level $k$ with assignments for the set of variables $\mathcal{V}$, $\mathcal{V}$ not in

[4] Or a previously known label of $x$ (for AAS).

evars($A_i$). $A_i$ must then send **add-link** messages to all agents $A_{k'}, k' \leq k$ not yet linked to $A_i$ and owning variables in $\mathcal{V}$. In order to achieve consistencies asynchronously, besides the structures of ABT, implementations can maintain at any agent $A_i$, for any level $k$, $k \leq i$:

- The set, $V_k^i$, of the newest valid assignments proposed by agents $A_j, j \leq k$, for each interesting variable.
- For each variable $x$, $x \in$vars($A_i$), for each agent $A_j, j \geq k$, the last consistency nogood (with highest tag) sent by $A_j$ for level $k$, denoted $cn_x^k(i,j)$. $cn_x^k(i,j)$ is stored only as long as it is valid. It has the form $V_{j,x}^k \rightarrow (x \in s_{j,x}^k)$.

$\mathbf{NV_i(V_k^i)}$ is the constraint of coherence of $A_i$ with the view $V_k^i$. Let $cn_x^k(i,.)$ be $(\cup_{t,j}^{t \leq k} V_{j,x}^t) \rightarrow (x \in \cap_{t,j}^{t \leq k} s_{j,x}^t)$. $P_i(k) := \text{CSP}(A_i) \cup (\cup_x cn_x^k(i,.)) \cup \text{NV}_i(V_k^i) \cup \text{CL}_k^i$. $C_{x_i}^k$ is incremented on each modification of $cn_{x_i}^k(i,i)$ (line 2.6).

On each modification of $P_i(k)$, $cn_{x_i}^k(i,i)$ is recomputed by inference (e.g. using local consistency techniques at line 2.4) for the problem $P_i(k)$. $cn_{x_i}^k(i,i)$ is initialized as an empty constraint set. $CL_k^i$ is the set of all nogoods known by $A_i$ and having the form $V \rightarrow C$ where $V \subseteq V_k^i$ and $C$ is a constraint over variables in vars($A_i$). $cn_{x_i}^k(i,i)$ is stored and sent to other agents by **propagate** messages iff its label shrinks and either CSP($A_i$) or $CL_k^i$ was used for its logical inference from $P_i(k)$. This is also the moment when $C_{x_i}^k$ is incremented. The procedure for receiving **propagate** messages is given in Algorithm 2.

We now prove the correctness, completeness and termination properties of DMAC-ABT. We only use DC techniques that terminate (e.g. [25, 2]). By quiescence of a group of agents we mean that none of them will receive or generate any valid nogoods, new valid assignments, **propagate** or **add-link** messages.

**Property 2** *In finite time $t^i$ either a solution or failure is detected, or all the agents $A_j, 0 \leq j \leq i$ reach quiescence in a state where they are not refused a proposal satisfying* ECSP($A_j$)$\cup$NV$_j$(view($A_j$)).

**Proposition 1.** *DMAC-ABT is correct, complete and terminates.*

The proof is given in Annexes. It remains to show the properties of the labels computed by DMAC-ABT at each level of the distributed search tree. If the agents, using DMAC-ABT, store all the valid consistency nogoods they receive, then DCs in DMAC-ABT converge and compute a local consistent global problem at each level (each pair initial_constraint-variable_label is checked by some agent). If on the contrary, the agents do not store all the valid consistency nogoods they receive but discard some of them after inferring the corresponding $cn_x^k(i,i)$, then some valid bounds or value eliminations can be lost when a $cn_x^k(i,i)$ is invalidated. Different labels are then obtained in different agents for the same variable. These differences have as result that the DC at the given level of DMAC-ABT can stop before the global problem is DC consistent at that level.

Among the consistency nogoods that an agent computes itself at level $k$ from its constraints, $cn_x^k(i,i)$, let it store only the last one for each variable and only as long as it is valid. Let $A_i$ also store only the last (with highest tag) consistency nogood,

**when received** (**ok?**,$\langle x_j, d_j, c_{x_j}\rangle$) **do**
  **if**(old $c_{x_j}$) return;
3.1 add($x_j$,$d_j$,$c_{x_j}$) to *agent_view*; eliminate invalidated nogoods;
  **maintain_consistency**(j);
  **check_agent_view**; //only satisfies consistency nogoods of levels t, t<cL$_i$;

**procedure check_agent_view do**
  **when** agent_view *and* current_value *are not consistent //cf. nogoods of levels t, t<cL$_i$*
    **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; $C^i_{x_i}$++; **maintain_consistency**(i);
      send (**ok?**,$\langle x_i, d, C^i_{x_i}\rangle$) to lower priority agents in *outgoing links*;

Algorithm 3: *Procedures of $A_i$ for receiving* **ok?** *messages in DMAC-ABT.*

$cn^k_x(i,j)$, sent to it for each variable $x \in$vars($A_i$) at each level $k$ from any agent $A_j$. $cn^k_x(i,j)$ is also stored only as long as it is valid. Each agent stores the highest tag $c^k_x(j)$ for each variable $x$, level $k$ and agent $A_j$ that sends labels for $x$. Then:

**Proposition 2.** *DC($\mathcal{A}$) labels computed at quiescence at any level using* **propagate** *messages are equivalent to $\mathcal{A}$ labels when computed in a centralized manner on a processor. This is true whenever all the agents reveal consistency nogoods for all minimal labels, $l^k_x$, which they can compute* **and when $\mathbf{CL^i_k}$ are not used**.

**Proof.** In each sent **propagate** message, the consistency nogood for each variable is the same as the one maintained by the sender. By checking $c^k_{x_v}(j)$ at line 2.1, the stored consistency nogoods are coherent and are invalidated only when newer assignments are received (event that is coherent) at lines 1.1,2.2,3.1. Any assignment invalid in one agent will eventually become invalid for any agent. Therefore, any such nogood is discarded at any agent, iff it is also discarded at its sender. The labels known at different agents, being computed from the same consistency nogoods, are therefore identical and the distributed consistency will not stop at any level before the global problem is local consistent in each agent. □

Since consistency nogoods are not discarded when nogoods are sent to agents generating their assignments, asynchronism is ensured by temporarily disregarding those consistency nogoods. In Algorithm 3 we only satisfy consistency nogoods at levels lower than the current inconsistent level, cL$_i$ (see line 2.5 in Algorithm 2). Alternatively, such consistency nogoods could be discarded but then, to ensure coherence of labels, agents receiving any nogood should always broadcast assignments with new tags and many nogoods would be unnecessarily invalidated.

ABT may deal with problems that require privacy of domains. For such problems, agents may refuse to reveal labels for some variables, especially since the initial labels at level 0 are given by the initial domains. The strength of the maintained consistency is then function of how many such private domains are involved in the problem. The

**procedure maintain_consistency***(minT)* **do**

**if** (minT $>$ cL$_i$) **then** return;

4.1 **for** (t$\leftarrow$minT; t$\leq$i; t++)

*new-cns* $\leftarrow$ consistency nogoods for all vars($A_i$) after local consistency on $P_i(t)$;

**when** (domain wipe out by computing explicit nogoods *nogoods*)

**for every** $V \in$ *nogoods*;

select $\langle x_j, d_j, c_{x_j} \rangle$ where $x_j$ has the lowest priority in $V$;

4.2 send (**nogood**,$A_i$,$V$) to $A_j$; eliminate invalidated explicit nogoods;

cL$_i \leftarrow$t; remove $\langle x_j, d_j, c_{x_j} \rangle$ from *agent_view*;

**break**;

**forall** *new-cn* $\leftarrow$ consistency nogood for any variable $x_u$ in *new-cns*

**when** *new-cn* shrinks label of $x_u$ (obtained from $\cup_{w,k \leq t} cn^k_{x_u}(i,w)$)

$cn^t_{x_u}(i,i) \leftarrow$ *new-cn*; $c^t_{x_u}(i)$++;

send (**propagate**,$A_i$,t,$c^t_{x_u}$,*new-cn*) to agents $A_j, j \geq t, x_u \in$ **vars**($\mathbf{A_j}$);

Algorithm 4: *Procedure of $A_i$ for receiving* **propagate** *messages in DMAC-ABT1.*

DisCSPs presenting only privacy on constraints, and the corresponding versions and extensions of ABT, suffer less of this problem.

**Proposition 3.** *The minimum space an agent needs with DMAC-ABT for ensuring maintenance of the highest degree of consistency achievable with DC is O($v^2(v+d)$). With bound consistency, the required space is O($v^3$).*

The proof is given in Annexes.

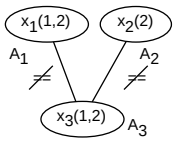
### 5.3 Using available valid nogoods in $P_i(k)$ for maintaining consistency (DMAC-ABT1)

In Algorithm 2, an agent $A_i$ only sends consistency nogoods for the variable $x_i$. However, when the local consistency is computed for $P_i(k)$, new labels are also computed for other variables known by $A_i$.

If in $P_i(k)$ we only use consistency nogoods and initial constraints, the final result of the consistency maintenance is coherent in the sense that at quiescence at any given level, each agent ends knowing the same label for each variable. Namely the new label obtained by $A_i$ for some variable $x_u$ will be computed and sent by $A_u$ after receiving the other labels in consistency nogoods and instantiations that $A_i$ knows and are related to $x_u$.

We propose that agents can use in their $P_i(k)$ valid explicit nogoods that they have received by **nogood** messages or old and invalidated consistency nogoods stored as redundant constraints. In this last case the labels obtained with Algorithm 2 are no longer minimal since an agent $A_u$ does not know all constraints that can be used by $A_i$ locally for computing its version of the label of $x_u$ at level $k$.

In Algorithm 4 we present a version of DMAC-ABT that we call DMAC-ABT1. In DMAC-ABT1, $A_i$ can send consistency nogoods for all variables found in CSP($A_i$). The space complexity for storing the last tags for the consistency nogoods at all levels and coming from all other agents is now O($v^3$) and for DMAC-ABT1 the space




1: $A_1$ ——**ok?**$\langle x_1, 1, 1\rangle$——→ $A_3$
2: $A_2$ –**propagate**($A_2$,0,1,$x_3 \notin \{2\}$)→ $A_1$
3: $A_2$ –**propagate**($A_2$,0,1,$x_3 \notin \{2\}$)→ $A_3$
4: $A_2$ ——**ok?**$\langle x_2, 2, 1\rangle$——→ $A_3$
5: $A_1$ –**propagate**($A_1$,0,1,$x_1 \notin \{1\}$)→ $A_3$
6: $A_1$ ——**ok?**$\langle x_1, 2, 2\rangle$——→ $A_3$
7: $A_3$ –**propagate**($A_3$,0,1,$x_1 \notin \{1\}$)→ $A_1$
8: $A_3$ ——**nogood**$\neg(\langle x_1, 1, 1\rangle \rightarrow A_1$

**Fig. 2.** Simplified example for DMAC-ABT1. Function of the exact timing of the network, some of these messages are no longer generated. Only 2 messages are sequential (half round-trips). ABT needs 4 sequential messages (half round-trips) for the same example (see [23]).

complexity is O($v^3(v+d)$). However, the power of DCs is increased since it can accommodate any available nogood. The number of sequential messages is also reduced since there is no need to wait for $A_u$ to receive the label of $x_i$ before reducing the label of $x_u$. Rather $A_i$ propagates itself the label of $x_u$.

**Proposition 4.** *The minimum space an agent needs with DMAC-ABT1 for ensuring maintenance of the highest degree of consistency achievable with DC is O($v^3(v+d)$). With bound consistency, the required space is O($v^4$).*

The proof is given in Annexes. We denote by DMAC-ABT2 the version of DMAC-ABT where any agent $A_i$ can compute, send and receive labels for variables constrained by their stored nogoods and redundant constraints but not found in **vars**($\mathbf{A_i}$).

## 6 Example

In Figure 2 we show a trace of DMAC-ABT1 for the example described in [23]. Before making its proposal, $A_2$ sends **propagate** messages to announce the consistency nogood $x_3 \notin \{2\}$ of level 0, tagged with $c^0_{x_3}(2) = 1$. These **propagate** messages are sent both to $A_1$ and $A_3$. $A_1$ sends an **ok?** message proposing a new instantiation.

$A_3$ (and $A_1$ when the domain of $x_3$ is public) compute both the consistency nogood $x_1 \notin \{1\}$ at level 0. $A_3$ computes an explicit nogood from consistency at level 1 and sends it to $A_1$. This nogood is invalid since $A_1$ has already changed its instantiation (and a small modification of DMAC-ABT1, for simplicity not given here, can avoid sending it). Then solution and quiescence are reached. The longest sequence of messages valid at their receivers (length 2) consists in messages 2,6. The worst case timing (slow communication channel from $A_2$ to $A_1$ or privacy for the domain of $x_3$) gives the longest sequence 3,7,6 (5 would not be generated). The fact that ABT (as well as any synchronous algorithm) would require at least 4 sequential messages illustrates the parallelism offered by asynchronous consistency maintenance.

## 7 Experiments

We have presented here DMAC-ABT1, an algorithm that allows to maintain consistency in ABT. ABT was chosen since it is simpler to present and explain. Recently we

have presented an extension of ABT that allows several agents to propose modifications to the same variable and allows agents to aggregate values in domains. That extension is called Asynchronous Aggregation Search (AAS) [14]. In [14] is shown that the aggregations bring to ABT improvements of an order of magnitude for versions that maintain a polynomial number of nogoods. Here it is therefore appropriate to test the improvements that our technique for maintaining consistency brings to AAS. The version of DMAC-ABT1 for AAS is denoted DMAC.

We have run our tests on a local network of SUN stations where agents are placed on distinct computers. We use a technique that enables agents to process with higher priority **propagate** and **ok?** messages for lower levels.

The DC used in our experimental evaluation maintains bound-consistency. In each agent, computation at lower levels is given priority over computations at higher levels. We generated randomly problems with 15 variables of 8 values and graph density of $20\%$. Their constraints were randomly distributed in 20 subproblems for 20 agents. Figure 3 shows their behavior for variable tightness (percentage of feasible tuples in constraints), averaged over 500 problems per point. We tested two versions of DMAC, A1 and A2. A1 asynchronously maintains bound consistency at all levels. A2 is a relaxation where agents only compute consistency at levels where they receive new labels or assignments, not after reduction inheritance between levels. A2 is obtained in Algorithm 4 by performing the cycle starting at line 4.1 only for $t = k$, where $k$ is the level of the incoming **ok?** or **propagate** message triggering it. In both cases, the performance of DMAC is significantly improved compared to that of AAS. Even for the easy points where AAS requires less than 2000 sequential messages, DMAC proved to be more than 10 times better in average. A2 was slightly better than A1 on average (excepting at tightness $15\%$). In these experiments we have stored only the minimal number of nogoods. The nogoods are the main gain of parallelism in asynchronous distributed search. Storing additional nogoods was shown for AAS to strongly improve performance of asynchronous search. As future research topic, we foresee the study of new nogood storing heuristics [8, 24, 22, 18, 6].

## 8 Conclusion

Consistency maintenance is one of the most powerful techniques for solving centralized CSPs. Bringing similar techniques to an asynchronous setting poses the problem of how search can be asynchronous when instantiation and consistency enforcement steps are combined. We present a solution to this problem. A distributed search protocol which allows for *asynchronously maintaining* distributed consistency with polynomial space complexity is proposed. DMAC-ABT builds on ABT, the basic asynchronous search technique. However, DMAC-ABT can be easily integrated into more complex versions of ABT (combining it with AAS and using abstractions [16], one can use complex splitting strategies [17] to deal efficiently with numeric DisCSPs [12]). Another original feature of DMAC is its capability of using backtrack nogoods to increase the strength of the maintained consistency.[5] The experiments show that the overall performance of

[5] Since this paper was submitted, [1] presents an algorithm reusing some backtrack nogoods in MAC. That algorithm can be proven to behave as a centralized instance of DMAC.

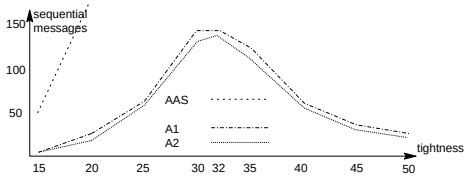



**Fig. 3.** *Results averaged over 500 problems per point.*

asynchronous search with consistency maintenance is significantly improved compared to that of asynchronous search that does not maintain consistency.

## Annexes (Proof)

**Property 2** *In finite time $t^i$ either a solution or failure is detected, or all the agents $A_j, 0 \leq j \leq i$ reach quiescence in a state where they are not refused a proposal satisfying* ECSP($A_j$)$\cup$NV$_j$(view($A_j$)).

**Proof.** The proof is by induction on i. Let this be true for the agents $A_j, j<i$. Let $\tau$ be the maximum time taken by a message. After $t^{i-1} + \tau$, $A_i$ no longer receives **ok?** messages. $A_i$ receives the last valid **ok?** message at time $t^i_o \leq t^{i-1} + \tau$. $\exists t^i_v$, $t^{i-1} + \tau \geq t^i_v$ such that after $t^i_v$, view($A_i$) and all $V^u_k, k<i$ of any agent $A_u$ are no longer modified.
The set of disabled tuples in CL$^u_k, k<i$ can contain only a bounded number of elements for each agent $A_u$ and they cannot be invalidated after $t^i_o$. CL$^u_k, k<i$ cannot be invalidated after $t^i_v$. Since DCs were assumed to terminate, they terminate after each modification of a CL$^u_k$. Since the number of such modifications that can generate a new consistency nogood after $t^i_v$ is bounded, after a finite time no consistency nogood is received any longer by $A_i$ for levels $k<i$.
Since the domains are finite, $A_i$ can make only a finite number of different proposals satisfying view($A_i$). Once any of them is sent, the total number of consistency nogoods that can be received before the proposal is modified is finite (this results by induction to levels $k \leq i$ of the reasoning for $k<i$ in the previous paragraph since after $v\tau$, $A_i$ can receive only valid nogoods: valid explicit nogoods trigger the modification of the instantiation of $A_i$ so that they can arrive only in finite time; if valid explicit nogoods are not received and no instantiation modification is done in finite time, no **ok?** is sent any longer by $A_i$, and the number of valid consistency nogoods at level $i$ is limited as in the previous paragraph).
Only one valid explicit nogood can be received for a proposal since the proposal is immediately changed on such an event. Invalid nogoods can be received only within $v\tau$ time delay after a proposal is made. Therefore, there is a finite number of nogoods that can be received by $A_i$ for any of its proposals made after $t^i_v$ (and after $t^i_o$).

1. If one of the proposals is not refused by incoming nogoods, and since the number of received nogoods is finite, the induction step is correct.

2. If all proposals that $A_i$ can make after $t^i_o$ are refused or if it cannot find any proposal, $A_i$ has to send according to rules inherited from ABT a valid explicit nogood

$\neg N$ to somebody. $\neg N$ is valid since all the assignments of $A_k, k < i$ were received at $A_i$ before $t_o^i$.

2.a) If $N$ is empty, failure is detected and the induction step is proved.

2.b) Otherwise $\neg N$ is sent to a predecessor $A_j, j<i$. Since $\neg N$ is valid, the proposal of $A_j$ is refused, but due to the premise of the inference step, $A_j$ either

2.b.i) finds an assignment and sends **ok?** messages or

2.b.ii) announces failure by computing an empty nogood (induction proven).

In the case (i), since $\neg N$ was generated by $A_i$, $A_i$ is interested in all its variables, and it will be announced by $A_j$ of the modification by an **ok?** messages.

Case 2.b.i contradicts the assumption that the last **ok?** message was received by $A_i$ at time $t_o^i$ and the induction step is therefore proved for all alternative cases. The property can be attributed to an empty set of agents and it is therefore proved by induction for all agents. □

**Proposition 1.** *DMAC-ABT is correct, complete and terminates.*

**Proof.** **Completeness:** All the nogoods are generated by logical inference from existing constraints. Therefore, if a solution exists, no empty nogood can be generated.
**No infinite loop:** The result follows from Property 2.
**Correctness:** All valid proposals are sent to all interested agents and stored there. At quiescence all the agents know the valid interesting assignments of all predecessors. If quiescence is reached without detecting an empty nogood, then all the agents agree with their predecessors and their intersection is nonempty and correct. □

**Proposition 3.** *The minimum space an agent needs with DMAC-ABT for ensuring maintenance of the highest degree of consistency achievable with DC is O($v^2(v + d)$). With bound consistency, the required space is O($v^3$).*

**Proof.** $d$-maximal domain size;$v$-number of variables. The space required for storing all valid assignments is O($v$) for values and O($v$) for the corresponding counters. The agents need to maintain at most $v$ levels, each of them dealing with maximum $v$ variables, for each of them having at most 1 last consistency nogood. Each consistency nogood refers at most $v$ assignments in premise and stores at most $d$ values in label. The stack of labels requires therefore O($v^2(v + d)$). The space required by the algorithm for solving the local problem depends on the corresponding technique (e.g. chronological backtracking requires O($v$)). The stored explicit nogoods require O($dv$) as mentioned in Property 1. In DMAC-ABT are also stored O($v^2$) tags for consistency nogoods. □

**Proposition 4.** *The minimum space an agent needs with DMAC-ABT1 for ensuring maintenance of the highest degree of consistency achievable with DC is O($v^3(v + d)$). With bound consistency, the required space is O($v^4$).*

**Proof.** The agents need to maintain at most $v$ levels, each of them dealing with maximum $v$ variables, for each of them having at most $v$ last consistency nogoods. Each consistency nogood refers at most $v$ assignments in premise and stores at most $d$ values in label. The stack of labels requires therefore O($v^3(v + d)$). DMAC-ABT1 also stores O($v^3$) tags for consistency nogoods. The other structures are identical as for DMAC-ABT. □

## References


1. J.-F. Baget and Y.S. Tognetti. Backtracking through biconnected components of a constraint graph. In *Proc. of IJCAI-01*, pages 291–296, 2001.
2. B. Baudot and Y. Deville. Analysis of distributed arc-consistency algorithms. Technical Report RR-97-07, U. Catholique Louvain, 97.
3. C. Bessière, A. Maestre, and P. Meseguer. Distributed dynamic backtracking. In *Proc. IJCAI DCR Workshop*, pages 9–16, 2001.
4. Z. Collin, R. Dechter, and S. Katz. Self-stabilizing distributed constraint satisfaction. *Chicago Journal of Theoretical Computer Science*, 2000.
5. J. Denzinger. Tutorial on distributed knowledge based search. IJCAI-01, August 2001.
6. E.C. Freuder, M. Minca, and R.J. Wallace. Privacy/efficiency tradeoffs in distributed meeting scheduling by constraint-based agents. In *Proc. IJCAI DCR Workshop*, pages 63–72, 2001.
7. M. Hannebauer. On proving properties of concurrent algorithms for distributed csps. In *Proc. of CP-01 DisCS Workshop*. EPFL, 2000.
8. W. Havens. Nogood caching for multiagent backtrack search. In *Proc. AAAI'97 Constraints and Agents Workshop*, '97.
9. S. Kasif. On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks. *Artificial Intelligence*, 45(3):275–286, October 90.
10. P. Meseguer and M. A. Jiménez. Distributed forward checking. In *Proceedings of the International Workshop on Distributed Constraint Satisfaction*. CP'00, 2000.
11. D. Sabin and E. C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings ECAI-94*, pages 125–129, 94.
12. M.-C. Silaghi, Ş. Sabău, D. Sam-Haroud, and B.V. Faltings. Asynchronous search for numeric DisCSPs. In *Proc. of CP'2001*, Paphos,Cyprus, 2001.
13. M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. ABT with Asynch. Reordering. In *IAT*, 01.
14. M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *Proc. of AAAI2000*, pages 917–922, 2000.
15. M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Maintaining hierarchical distributed consistency. In *Proc. of CP-00 Workshop on DisCS*, 2000.
16. M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Multiply asynchronous search with abstractions. In *IJCAI-01 DCR Workshop*, pages 17–32, Seattle, August 2001.
17. M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Search techniques for non-liniar constraint satisfaction problems with inequalities. In *Proc. of AI2001*, Otawa, June 2001.
18. M.-C. Silaghi, D. Sam-Haroud, and B.V. Faltings. Hybridyzing ABT and AWC into a polynomial space, complete protocol with reordering. Technical Report #364, EPFL, May 2001.
19. M.C. Silaghi and B. Faltings. Parallel proposals in asynchronous search. Technical Report #371, EPFL, August 2001.
20. G. Solotorevsky, E. Gudes, and A. Meisels. Distributed Constraint Satisfaction Problems - a model and application. Preprint: http://www.cs.bgu.ac.il/~am, 97.
21. G. Tel. *Multiagent Systems, A Modern Approach to Distributed AI*, chapter Distributed Control Algorithms for AI, pages 539–580. MIT Press, 99.
22. E. H. Turner and J. Phelps. Determining the usefulness of information from its use during problem solving. In *Proceedings of AA2000*, pages 207–208, 2000.
23. M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *ICDCS'92*, pages 614–621, June 92.
24. M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The Distributed CSP: Formalization and algorithms. *IEEE Trans. on KDE*, 10(5):673–685, 98.
25. 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, 91.