# ABT WITH ASYNCHRONOUS REORDERING

MARIUS-CĂLIN SILAGHI, DJAMILA SAM-HAROUD, AND BOI FALTINGS
*Swiss Federal Institute of Technology Lausanne*
*1015 Ecublens,Switzerland*
{*Marius.Silaghi,Djamila.Haroud,Boi.Faltings*}*@epfl.ch*

Existing Distributed Constraint Satisfaction (DisCSP) frameworks can model problems where a)variables and/or b)constraints are distributed among agents. Asynchronous Backtracking (ABT) is the first asynchronous complete algorithm for solving DisCSPs of type a. The order on variables is well-known as an important issue for constraint satisfaction. Previous polynomial space asynchronous algorithms require for completeness a static order on their variables. We show how agents can asynchronously and concurrently propose reordering in ABT while *maintaining the completeness* of the algorithm with polynomial space complexity.

## 1 Introduction

Distributed combinatorial problems can be modeled using the general framework of Distributed Constraint Satisfaction (DisCSP). A **DisCSP** is defined in [1] as: a set of agents, $A_1, ..., A_n$, where each agent $A_i$ controls exactly one distinct variable $x_i$ and each agent knows all constraint predicates relevant to its variable. The case with more variables in an agent can be obtained quite easily from here. Asynchronous Backtracking (ABT) [1] is the first *complete* and *asynchronous* search algorithm for DisCSPs. A simple modification was mentioned in [1] to allow for a version with polynomial space complexity.

The completeness of ABT is ensured with the help of *a static order* imposed on agents. So far, no asynchronous search algorithm has offered the possibility to perform reordering without losing either the completeness, or the polynomial space property. In this paper we describe a technique that allows the agents to asynchronously and concurrently propose changes to their order. We then prove that, using a special type of markers, the completeness of the search is ensured with polynomial space complexity.

This is the first asynchronous search algorithm that allows for asynchronous dynamic reordering while being complete and having a polynomial space complexity. Here we have built on ABT since it is an algorithm easier to describe than its subsequent extensions. The technique can nevertheless be integrated in a straightforward manner in most extensions of ABT. [2]

## 2 Related Work

The first complete asynchronous search algorithm for DisCSPs is the Asynchronous Backtracking (ABT)[1]. For simplicity, but without severe loss of generality, the approach in [1] considers that each agent maintains only one variable. More complex definitions were given later.[3,4] Other definitions of DisCSPs [5,6,7] have considered the case where the interest on constraints is distributed among agents. [6] proposes versions that fit the structure of a real problem (the nurse transportation problem). The Asynchronous Aggregation Search (AAS) [7] algorithm actually extends ABT to the case where the same variable can be instantiated by several agents and an agent may 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 [8] and allows for simple understanding of the privacy/efficiency mechanisms. The strong impact of the ordering of the variables on distributed search was so far addressed in [9,6,10].

## 3 Asynchronous Backtracking (ABT)

In asynchronous backtracking, the agents run concurrently and asynchronously. Each agent instantiates its variable and communicates the variable value to the relevant agents. Since here we don't assume FIFO channels, in our version a **local counter**, $C_i^{x_i}$, is incremented each time a new instantiation is proposed, and its current value **tags** each generated assignment.

**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.*

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

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

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 containing the* newest *assignments received by $A_i$ for distinct variables.*

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

straints 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?**, **nogood**, and **add-link**. An **ok?** message transports an assignment and is sent to a constraint-evaluating-agent to ask whether a chosen value is acceptable. Each **nogood** message transports a *nogood*. It is sent from the agent that infers a *nogood* $\neg N$, to the constraint-evaluating-agent for $\neg N$. An **add-link** message announces $A_i$ 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 concurently 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 (except for pseudo-code delimited by '*').[4]

**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 invalid** if it contains invalid assignments. The next property is mentioned in [1] and it is also implied by the Theorem 1, presented later.

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

## 4 Histories

Now we introduce a *marking technique* that allows for the definition of a total order among the proposals made concurently and asynchronously by a set of ordered agents on a shared resource (e.g. an order).

**Definition 5** *A* **proposal source** *for a resource* $\mathcal{R}$ *is an entity (e.g. an abstract agent) that can make specific proposals concerning the allocation (or valuation) of* $\mathcal{R}$.

We consider that an order $\prec$ is defined on *proposal sources*. The *proposal sources* with lower position according to $\prec$ have a higher priority. The *proposal source* for $\mathcal{R}$ with position $k$ is noted $P_k^{\mathcal{R}}$, $k \geq x_0^{\mathcal{R}}$. $x_0^{\mathcal{R}}$ is the first position.

**Definition 6** *A* **conflict resource** *is a resource for which several agents can make proposals in a concurent and asynchronous manner.*

Each *proposal source* $P_i^{\mathcal{R}}$ maintains a counter $\mathbf{C_i^{\mathcal{R}}}$ for the *conflict resource* $\mathcal{R}$. The markers involved in our *marking technique for ordered proposal sources* are called **histories**.

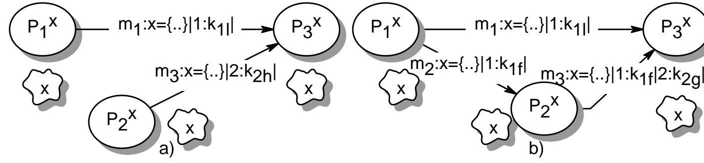


Figure 1. Simple scenarios with messages for proposals on a resource, $x$.

**Definition 7** *A* **history** *is a chain $h$ of pairs, $|a{:}b|$, that can be associated to a proposal for $\mathcal{R}$. A pair $p{=}|a{:}b|$ in $h$ signals that a proposal for $\mathcal{R}$ was made by $P_a^{\mathcal{R}}$ when its $C_a^{\mathcal{R}}$ had the value $b$, and it knew the prefix of $p$ in $h$.*

An order $\propto$ (read "precedes") is defined on pairs such that $|i_1{:}l_1| \propto |i_2{:}l_2|$ if either $i_1{>}i_2$, or $i_1{=}i_2$ and $l_1 < l_2$.

**Definition 8** *A history $h_1$* **is newer than** *a history $h_2$ if a lexicographic comparison on them, using the order $\propto$ on pairs, decides that $h_2$ precedes $h_1$.*

$P_k^{\mathcal{R}}$ builds a history for a new proposal on $\mathcal{R}$ by prefixing to the pair $|k{:}\text{value}(C_k^{\mathcal{R}})|$, the newest history that it knows for a proposal on $\mathcal{R}$ made by any $P_a^{\mathcal{R}}$, $a{<}k$. The $C_a^{\mathcal{R}}$ in $P_a^{\mathcal{R}}$ is reset each time an incoming message announces a proposal with a newer history, made by higher priority *proposal sources* on $\mathcal{R}$. $C_a^{\mathcal{R}}$ is incremented each time $P_a^{\mathcal{R}}$ makes a proposal for $\mathcal{R}$.

**Definition 9** *A history $h_1$ built by $P_i^{\mathcal{R}}$ for a proposal is* **valid** *for an agent $A$ if no other history $h_2$ (eventually known only as prefix of a history $h_2'$) is known by $A$ such that $h_2$ is newer than $h_1$ and was generated by $P_j^{\mathcal{R}}$, $j \leq i$.*

For example, in Figure 1 the agent $P_3^x$ may get messages concerning the same resource $x$ from $P_1^x$ and $P_2^x$. In Figure 1a, if the agent $P_3^x$ has already received $m_1$, it will always discard $m_3$ since the *proposal source* index has priority. However, in the case of Figure 1b the message $m_1$ is the newest only if $k_{1f} < k_{1l}$ and is valid only if $k_{1f} \leq k_{1l}$. In each message, the length of the history for a resource is upper bounded by the number of *proposal sources* for the *conflict resource*.

## 5 Reordering

Now we show how the histories described in the previous section offer during the search a mean for allowing agents to asynchronously and concurrently propose new orders on themselves. In the next subsection we describe a didactic, simplified version that needs additional specialized agents.

### 5.1 Reordering with dedicated agents

Besides the agents $A_1, ..., A_n$ in the DisCSP we want to solve, we consider that there exist $n-1$ other agents, $R^0, ..., R^{n-2}$, that are solely devoted for reordering the agents $A_i$.

**Definition 10** *An* **ordering** *is a sequence of distinct agents $A_{k_0}, ..., A_{k_n}$.*

An agent $A_i$ may receive the position $j, j \neq i$. Let us assume that the agent $A_l$, knowing an ordering $o$, believes that the agent $A_i$, owning the variable $x_i$, has the position $j$. $A_l$ can refer $A_i$ as either $A^j$, $A^j(o)$ or $A_i^j$. The variable $x_i$ is also referred to by $A_l$ as either $x^j$, $x^j(o)$ or $x_i^j$.

We propose to consider the ordering on agents as a *conflict resource.* We attach to each ordering a *history* as defined in the previous section. The *proposal sources* for the ordering on agents are the agents $R^i$, where $R^i \prec R^j$ if $i<j$ and $x_0^{\text{order}}=0$. $R^i$ is the *proposal source* that when knowing an ordering, $o$, can propose orderings that reorder only agents on positions $p$, $p > i$.

**Definition 11 (Known order)** *An ordering known by $R^i$ (respectively $A^i$) is the order $o$ with the newest history among those proposed by the agents $R^k, 0 \leq k < i$ and received by $R^i$ (respectively $A^i$). $A^i$ has the position $i$ in $o$. This order is referred to as the* known order of $R^i$ *(respectively $A^i$).*

**Definition 12 (Proposed order)** *An ordering, $o$,* proposed by $R^i$ *is such that the agents placed on the first $i$ positions in the* known order of $R^i$ *must have the same positions in $o$. $o$ is referred to as the* proposed order of $R^i$.

Let us consider two different orderings, $o_1$ and $o_2$, with their corresponding histories: $O_1 = \langle o_1, h_1 \rangle$, $O_2 = \langle o_2, h_2 \rangle$; such that $|h_1| \leq |h_2|$. Let $p_1^k = |a_1^k{:}b_1^k|$ and $p_2^k = |a_2^k{:}b_2^k|$ be the pairs on the position $k$ in $h_1$ respectively in $h_2$.

**Definition 13 (Reorder position)** *Let $u$ be the lowest position such that $p_1^u$ and $p_2^u$ are different and let $v = |h_1|$. The* **reorder position** *of $h_1$ and $h_2$ is either $min(a_1^u, a_2^u) + 1$ if $u > v$, or $a_2^{v+1} + 1$ otherwise. This is the position of the highest priority reordered agent between $h_1$ and $h_2$.*

New optional messages for reordering are: **heuristic** messages for heuristic dependent data, and **reorder** messages announcing a new ordering, $\langle o, h \rangle$.

An agent $R^i$ announces its proposed order $o$ by sending **reorder** messages to all agents $A^k(o)$, k>i, and to all agents $R^k$, k>i. Each agent $A^i$ and each agent $R^i$ has to store a set of orderings denoted $C^{ord}$. $C^{ord}$ contains the ordering with the newest history that was received from each $R^j$, j<i (if that history is valid).[a] By the history of $C^{ord}$ we refer the newest history in $C^{ord}$. For allowing asynchronous reordering, each **ok?** and **nogood** message receives

[a] Typically $C^{ord}$ is completely described by the ordering with the newest received history.

as additional parameter an order and its history (see Algorithm 1). The **ok?** messages hold the newest *known order of* the sender. The **nogood** messages hold the order in the $C^{ord}$ at the sender $A^j$ that $A^j$ believes to be the newest *known order of* the receiver, $A^i$. This ordering consists of the first $i$ agents in the newest ordering known by $A^j$ and is tagged with a history obtained from the history of its $C^{ord}$ by removing all the pairs $|a{:}b|$ where $a{\geq}i$.[b]

When a message is received which contains an order with a history $h$ that is newer that the history $h^*$ of $C^{ord}$, let the *reordering position* of $h$ and $h^*$ be $I^r$. The assignments for the variables $x^k$, $k \geq I^r$, are invalidated.[c]

The agents $R^i$ modify the ordering in a random manner or according to special strategies appropriate for a given problem.[d] Sometimes it is possible to assume that the agents want to colaborate in order to decide an ordering.[e] The **heuristic** messages are intended to offer data for reordering proposals. The parameters depend on the used reordering heuristic. The **heuristic** messages can be sent by any agent to the agents $R^k$. **heuristic** messages may only be sent by an agent to $R^k$ within a bounded time, $\mathbf{t_h}$, after having received a new assignment for $x^j, j{\leq}k$. Agents can only send **heuristic** messages to $R^0$ within time $t_h$ after the start of the search. Any **reorder** message is sent within a bounded time $\mathbf{t_r}$ after a **heuristic** message is received (or start).

Besides $C_k^{\text{order}}$ and $C^{ord}$, the other structures that have to be maintained by $R^k$, as well as the content of **heuristic** messages depend on the reordering heuristic. The space complexity for $A^k$ remains the same as with ABT.

### *5.2 ABT with Asynchronous Reordering (ABTR)*

In fact, we have introduced the physical agents $R^i$ in the previous subsection only in order to simplify the description of the algorithm. Any of the agents $A_i$ or other entity can be delegated to act for any $R^j$. When proposing a new order, $R^i$ can also simultaneously delegate the identity of $R^{i+1}, ..., R^{n-2}$ to other entities[f], $P_k$, by attaching a sequence $R^0{\rightarrow}P_{k_i}, ..., R^{n-2}{\rightarrow}P_{k_j}$ to the ordering. At a certain moment, due to message delays, there can be several entities believing that they are delegated to act for $R^i$ based on the ordering they know. However, any other agent can coherently discriminate among

[b]The agents absent from the ordering in a nogood are typically not needed by $A^i$. $A^i$ receives them when it receives the corresponding **reorder** message.

[c]Alternative rule: $A^i$ can keep valid the assignments of new variables $x^k$, $i \geq k \geq I^r$ but broadcasts $x^i$ again.

[d]e.g. first the agents forming a coalition with $R^i$.

[e]This can aim to improve the efficiency of the search. Since ABT performs forward checking, it may be possible to design useful heuristics.

[f]In [11] we explain how $R^i$ can redelegate itself.

**when received** *(***ok?***,*$(x_j, d_j$ *,$c_{x_j}$, $\langle o,h\rangle$ *)) **do**
  *$\mathbf{if}(\neg \mathbf{getOrder}(\langle o,h\rangle)$ or old $c_{x_j}$) return*; //ABTR;
  add($x_j$,$d_j$*,$c_{x_j}$*) to *agent_view*; **check_agent_view**;
**end do.**
**when received** *(***nogood**,$A_j$,nogood*,$\langle o,h\rangle$ *) **do**
  *$\mathbf{if}(\neg \mathbf{getOrder}(\langle o,h\rangle))$ return*; //ABTR;
  *discard *nogood* if it contains invalid assignments **else***; //ABTR;
    **when** ($x_k$,$d_k$,$c_k$), where $x_k$ is not connected, is contained in *nogood*
      send **add-link** to $A_k$; add ($x_k$,$d_k$,$c_k$) to *agent_view*;
    add *nogood* to *nogood-list*; add other new assignments to *agent_view*;
  *old_value* ← *current_value*; **check_agent_view**;
  **when** *old_value* = *current_value*
    send (**ok?**,($x_j$,*current_value*,$c_{x_i}$),known_order($A_i$)) 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* ← d; $c_{x_i}$++; $O$ ← known_order($A_i$);
      send (**ok?**,($x_i$,d,$c_{x_i}$),$O$) to lower priority agents in *outgoing links*;
    **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 that there is no solution, and terminate this algorithm;
  **for every** $V \in$ *nogoods*;
    select ($x_j$,$d_j$*,$c_{x_j}$*) where $x_j$ has the lowest priority in $V$;
    send (**nogood**,$x_i$,$V$,$O_j$) to $A_j$; remove ($x_j$,$d_j$*,$c_{x_j}$*) from *agent_view*;
  **check_agent_view**;
**end do.**
**function getOrder***($\langle o,h\rangle$) → bool //ABTR*
  **when** $h$ is invalidated by the history of $C^{ord}$ **then** return false;
  **when** not newer $h$ than $C^{ord}$ **then** add $\langle o,h\rangle$ to $C^{ord}$; return true;
  $I$ ← *reorder position* for $h$ and the history of $C^{ord}$;
  invalidate assignments for $x^j, j \geq I$ (alternative$^d$); add $\langle o,h\rangle$ to $C^{ord}$;
**end.**

Algorithm 1: *Procedures for Receiving Messages in ABT and ABTR.*

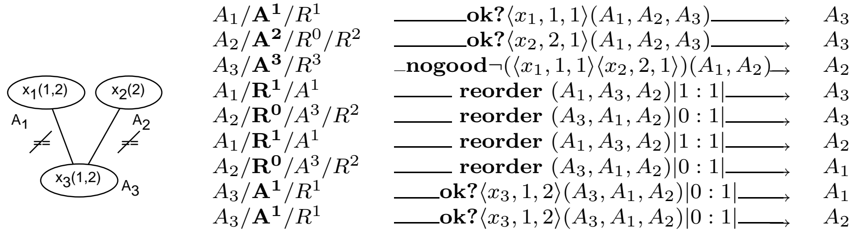


Figure 2. Simplified example for ABTR with random reordering. $R^i$ delegations are done implicitely by adopting the convention "$A^i$ is delegated to act for $R^i$". Left column: $A_i/A^j/R^{i_1}/R^{i_2}$... shows the roles played by $A_i$ when the message is sent. In bold is shown the capacity in which the agent $A_i$ sends the message. The addlink message in not shown.

messages from simultaneous $R^i$s using the histories that $R^i$s generate. The $R^i$ themselves coherently agree when the corresponding orders are received. The delegation of $R^i, i > 0$ from a physical entity to another poses no problem of information transfer since the counter $C_i^{\text{order}}$ of $R^i$ is reset on this event.

For simplicity, in the example in Figure 2 we describe the case where the activity of $R^i$ is always performed by the agent believing itself to be $A^i$. $R^i$ can send a **reorder** message within time $t_r$ after an assignment is made by $A^i$ since a **heuristic** message is implicitely transmitted from $A^i$ to $R^i$. We also consider that $A_2$ is delegated to act as $R^0$. $R^0$ and $R^1$ propose one random ordering each, asynchronously. The receivers discriminate based on histories that the order from $R^0$ is the newest. The known assignments and nogood are discarded. In the end, the *known order* for $A_3$ is $(A_3, A_1, A_2)|0:1|$.

By **quiescence** of a group of agents we mean that none of them will receive or generate any valid nogoods, new valid assignments, **reorder** messages or addlink messages.

**Property 2** *In finite time $t^i$ either a solution or failure is detected, or all the agents $A^j, 0<j\leq i$ reach quiescence in a state where they are not refused an assignment satisfying the constraints that they enforce and their* agent_view.

**Proof.** Let all agents $A^k, k<i$, reach quiescence before time $t^{i-1}$. Let $\tau$ be the maximum time needed to deliver a message.
$\exists t_p^i < t^{i-1}$ after which no **ok?** is sent from $A^k$, $k<i$. Therefore, no heuristic message towards any $R^u$, $u<i$, is sent after $t_h^i = t_p^i + \tau + t_h$. Then, each $R^u$ becomes fixed, receives no message, and announces its last order before a time $t_r^i = t_h^i + \tau + t_r$. After $t_r^i + \tau$ the identity of $A^i$ is fixed as $A_l$. $A_l^i$ receives the last new assignment or order at time $t_o^i < t_r^i + \tau$.
Since the domains are finite, after $t_o^i$, $A_l^i$ can propose only a finite number of

different assignments satisfying its view. Once any assignment is sent at time $t_a^i > t_o^i$, it will be abandoned when the first valid nogood is received (if one is received in finite time). All the nogoods received after $t_a^i + n\tau$ are valid since all the agents learn the last instantiations of the agents $A^k, k < i$ before $t_a^i + n\tau - \tau$. Therefore the number of possible incoming invalid nogoods for an assignment of $A^i$ is finite.

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

2.If all proposals that $A^i$ can make after $t_o^i$ are refused or if it cannot find any proposal, $A^i$ has to send 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 (has sent once an **add-link** to $A^j$), and it will be announced by $A^j$ of the modification by an **ok?** messages. This contradicts the assumption that the last **ok?** message was received by $A^i$ at time $t_o^i$ and the induction step is proved.

From here, the induction step is proven since it was proven for all alternatives. In conclusion, after $t_o^i$, within finite time, the agent $A^i$ either finds a solution and quiescence or an empty nogood signals failure.

$R^0$ is always fixed (or after $t_r$ in the version in [11]) and the property is true for the empty set. The property is therefore proven by induction on $i$ □

**Theorem 1** *ABTR 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:** This is a consequence of the Property 2 for $i = n$.

**Correctness:** All assignments 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 according to the Property 2, all the agents agree with their predecessors and the set of their assignments is a solution. □

## 6 Conclusions

Reordering is a major issue in constraint satisfaction. All previous complete polynomial space asynchronous search algorithms for DisCSPs require a static order of the variables. We have presented an algorithm that allows for asynchronous reordering in ABT. This is the first asynchronous complete algorithm with polynomial space requirements that has the ability to concurrently and asynchronously reorder variables during search. Here we describe a random reordering heuristic that can be useful for special purposes (coalitions, special strategies). However, this algorithm offers a flexible mechanism (general purpose heuristic messages) that allows for implementing most other heuristics that can be believed useful for general or specific applications. Alternative implementations, alternatives to using histories, how to save effort across reordering and efficient heuristics are described in [11].

## References


1. M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed CSP: Formalization and algorithms. *IEEE TKDE, 10(5):673–685, 98.*
2. *M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous consistency maintenance with reordering. TR #01/360, EPFL, March 2001.*
3. *M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In* ICMAS'98*, pages 372–379, 1998.*
4. *M. Yokoo.* Distributed Constraint Satisfaction. Springer, 2001.
5. Y. Zhang and A. K. Mackworth. Parallel and distributed algorithms for finite CSPs. In *Proc. of Symp. on PDP*, pages 394–397, 91.
6. G. Solotorevsky, E. Gudes, and A. Meisels. Distributed CSPs - a model and application. http://www.cs.bgu.ac.il/˜am/papers.html, Oct 97.
7. M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *Proc. of AAAI2000*, pages 917–922, 2000.
8. P. Meseguer and M. A. Jiménez. Distributed forward checking. In *Proc. of DCS*. CP'00, 2000.
9. M. Yokoo. Asynchronous weak-commitment search for solving large-scale distributed CSPs. In *ICMAS*, pages 467–318, 95.
10. Y. Hamadi and C. Bessière. Backtracking in distributed constraint networks. In *ECAI'98*, pages 219–223, 98.
11. M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Hybridizing ABT and AWC into a polynomial space, complete protocol with reordering. TR #01/364, EPFL, Mai 2001.