Framework for modeling reordering heuristics for asynchronous backtracking

Marius Călin Silaghi
Florida Institute of Technology
Melbourne, USA

Abstract

Dynamic reordering of variables is known to be important for solving constraint satisfaction problems (CSPs). Efforts to apply this principle for improving polynomial space asynchronous backtracking (ABT) started with [1], using a solution based on synchronization points. [17] shows how to asynchronously enable reordering heuristics in ABT and proposes a general protocol called Asynchronous Backtracking with Reordering (ABTR).

In this work we introduce a first framework for modeling heuristics possible with asynchronous backtracking. We also show that ABTR enables heuristics that displace even the agent requesting the reordering, as in the reordering of Dynamic Backtracking. They have not been illustrated in [17]. The most efficient self-reordering heuristic that we introduce and experiment, approx-AWC1*, is inspired from Asynchronous Weak-Commitment [21] and brings small but significant improvements, comparable to the results in [1]. We also report that min-domain dynamic ordering heuristics for ABTR are worse than no reordering and better than max-domain (in experiments that also use maintenance of arc consistency).*

1 Introduction

Distributed Constraint Satisfaction (DisCSP) is a powerful framework for modeling distributed combinatorial problems. A **DisCSP** is defined in [21, 4] as: a set of agents $\mathcal{A} = \{A_1, ..., A_n\}$ where each agent A_i controls exactly one distinct variable x_i, and each agent knows all constraint predicates relevant to its variable. This is an acceptable simplification since the case with more variables in an agent can be easily obtained from it. The case when an agent does not know all constraints relevant to its variables has impact on some details, as mentioned in the following on a case by case basis. Asynchronous Backtracking (ABT) [21, 11] is the first *complete* and *asynchronous* search algorithm for DisCSPs. ABT can be run with polynomial space complexity. It uses a total priority order on variables. In ABT, agents propose assignments for their variables using **ok?** messages. No reply is provided for an accepted assignment. When rejecting an assignment, an explanation is provided using a **nogood** message that lists conflicting assignments having higher priority.

1.1 ABT and the Reordering Problem

The completeness of ABT is ensured with help of *a static order* imposed on agents. ABT is a slow algorithm and [21] introduces a faster algorithm called Asynchronous Weak Commitment (AWC). AWC allows for reordering agents asynchronously by decreasing their position when they are over-constrained. AWC has proved to be more efficient than ABT but its requirement for an exponential space could not be eliminated without losing completeness.

ABT with dynamic re-ordering (ABTR) [17] is an extension of ABT that allows the agents to asynchronously and concurrently propose changes to their order. ABTR was defined as a simple family of algorithms that can be parametrized with any complex dynamic ordering heuristic that respects certain simple guidelines. At its introduction [17], ABTR was exemplified with toy heuristics similar to min-domain, but a heuristic in ABTR is actually defined by a very general framework, that here we reformulate in a more compact and easier to manage formal structure, namely as a tuple ($\mathbb{K}$,$\mathbb{S}$,$\mathbb{M}$,$\mathbb{P}$,$\mathbb{H}$):

- $\mathbb{K}$: a knowledge domain of interest for the heuristic (e.g., current domain sizes, existence of a current domain wipe-out, positions of agents).
- $\mathbb{S}$: a set of integers. Only its intersection with $[0..(n-2)]$ is typically relevant.
- $\mathbb{M}$: a policy dynamically mapping a set of counters, $\{C^r_k | k \in \mathbb{S}\}$, to the n agents as function of $\mathbb{K}$ (e.g., the counter C^r_k goes to the agent on position k).
- $\mathbb{P}$: an ordering policy for each agent that holds a counter C^r_k. It proposes a given ordering of the last $n-k$ agents

as function of $\mathbb{K}$. This reordering (new $\mathbb{K}$), when projected through $\mathbb{M}$, should impact only the mapping of counters $\{C_i^r | i \geq k\}$.

$\mathbb{H}$: a set of rules specifying when an agent that holds a counter C_k^r may use (or be told) new data from $\mathbb{K}$ for proposing a new order.

ABTR's proof guarantees that any heuristic that can be described with this general framework and that respects a small set of additional constraints on its $\mathbb{H}$ leads to a sound, complete and terminating asynchronous distributed algorithm for solving DisCSPs. We say about a heuristic respecting ABTR's guidelines that it is *ABTR compliant*.

1.2 Intuition of our new self-reordering heuristics (aka retroactive heuristics)

A typical use of reordering is for efficiency, but it is not obvious to find *time*-efficient reordering heuristics for DisCSPs, specially since min-domain heuristics alone seem not to work very well. The reordering heuristic of AWC has proved to be cheap and efficient for distributed algorithms but it cannot be directly used in ABTR. Here we introduce a reordering heuristic approximating AWC, called approx-AWC1, which moves the agent discovering a nogood in front of the agent that receives the nogood (but not as much in front as in AWC). Understanding these kind of heuristics requires more discussions and in [17] we skipped illustrating how to model such heuristics with ABTR's framework (so far they appeared in the technical report [19]). Here we show how to model such heuristics with the framework of ABTR and how to prove that they comply with the guidelines of ABTR for correct heuristics. We also show how one can obtain a reordering heuristic with identical behavior as the one in Dynamic Backtracking (DB).

Heuristics introduced here, such as the exact reordering heuristic of DB and the approximation of AWC that we denote *approx-AWC1* (aka retroactive), are depicted in Figure 1. The scenario in Figure 1 illustrates the reorderings requested by an agent *Bob* upon receiving a **nogood** message from agent *Eve* when the previous ordering was: Alice, Bob, Carol, Dave, Eve, Fred. With the exact reordering heuristic of Dynamic Backtracking (b) it is Bob, the owner of the culprit variable, that is moved on the position after the owner of the variable discovering the nogood, here Eve. Like in AWC, in *approx-AWC1*, Eve is moved in front of Bob. We show that these two heuristics have very similar models in the framework of heuristics for ABTR, and do not differ in features that are subject to ABTR's constraints for correct complete and terminating heuristics. Therefore the proof that any one of them conforms to the guidelines of ABTR also proves that the other heuristic is ABTR compliant, illustrating the power of the framework of ABTR.

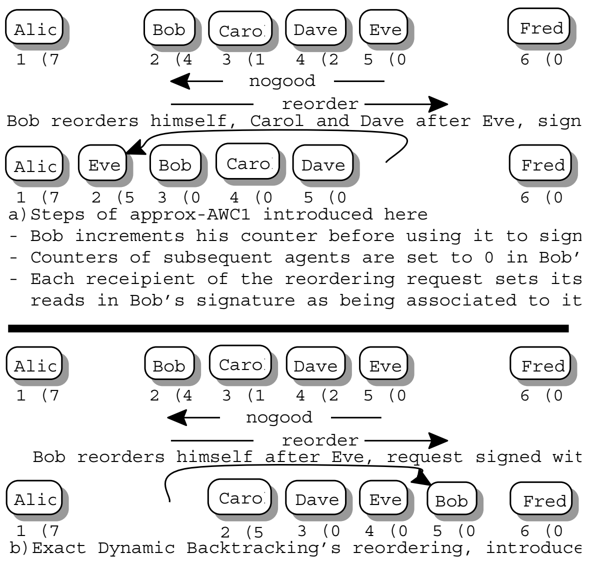
Alic
Bob
Caro
Dave
Eve
Fred
1 (7
2 (4
3 (1
4 (2
5 (0
6 (0
nogood
reorder
Bob reorders himself, Carol and Dave after Eve, sign
Alic
Eve
Bob
Caro
Dave
Fred
1 (7
2 (5
3 (0
4 (0
5 (0
6 (0
a) Steps of approx-AWC1 introduced here
- Bob increments his counter before using it to sign
- Counters of subsequent agents are set to 0 in Bob'
- Each receipient of the reordering request sets its
reads in Bob's signature as being associated to it
Alic
Bob
Caro
Dave
Eve
Fred
1 (7
2 (4
3 (1
4 (2
5 (0
6 (0
nogood
reorder
Bob reorders himself after Eve, request signed wit
Alic
Caro
Dave
Eve
Bob
Fred
1 (7
2 (5
3 (0
4 (0
5 (0
6 (0
b) Exact Dynamic Backtracking's reordering, introduce

Figure 1. Receiving a nogood: a) approx-AWC1 and b) the exact heuristic of Dynamic Backtracking. Below each agent is written its current position and, in parentheses, the value of its current reordering counter.

ABTR's ordering coherence is enabled by having reordering counters distributed among the agents and tagging messages with the vector-clock [13] induced by these counters. We show that counters function smoothly with the new reordering heuristics since, at reordering, the reordering counter of Carol (that will replace Bob) is set to the value of the counter of Bob in Bob's signature of the request for reordering (Figure 1.b). This feature provided by ABTR, besides being compatible with *min-domain*-like heuristics, is shown to enable additional heuristics not yet explored. Namely, one can associate any dynamic variable reordering heuristic with a complex policy $\mathbb{M}$ of dynamic changes to the mapping of reordering counters to agents (enabling complex heuristics).

1.3 Basic description of ABTR with motivations

Now we intuitively introduce ABTR by example (details and proof are presented later). For the simple case of min-domain dynamic ordering we take $(\mathbb{K}_0, \mathbb{S}_0, \mathbb{M}_0, \mathbb{P}_0, \mathbb{H}_0)$. $\mathbb{K}_0$ contains domain sizes and current orderings. $\mathbb{S}_0 = [0..n]$ [20]. $\mathbb{M}_0$ states that each agent on position k, holds reordering counter C_k^r. The first agent also holds C_0^r. Each reordering request is tagged with a *signatures vector-clock* [13, 17, 23] listing all values of these counters known to the sender, here $|c_0, ..., c_n|$. c_k is the value of C_k^r known to the sender. Note that the original ABTR would represent signature $|5, 2, 0, 0, 0|$ in Figure 2 as $|0{:}5|1{:}2|$, namely a sequence of pairs *position-k:value-*C_k^r [17, 15, 23]. An agent holding counter C_k^r can request a reordering of the agents on positions $k+1$ through n, and increments C_k^r whenever he proposes such a reordering. An example of a reordering for this case is shown in Figure 2. $\mathbb{P}_0$ specifies that (e.g.), A_1, predicting that his proposal $x_1{=}2$ will reduce the domain of x_3 (described in $\mathbb{K}_0$), reorders A_3 before A_2. Counter values for $C_j^r, j{>}k$, are set to 0 in A_1's signature, since A_1 does not know their newer values. We describe $\mathbb{H}_0$ after introducing some notations.

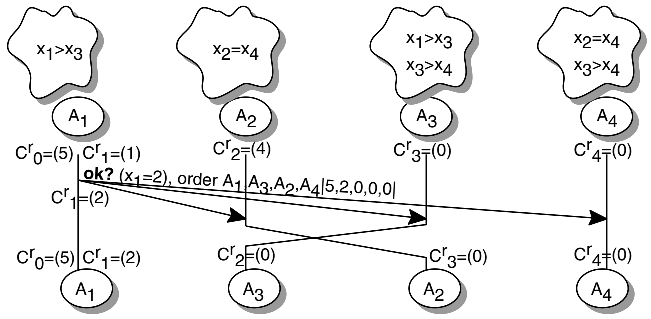

Figure 2. Each new assignment proposal by the agent A_i holding a counter C_k^r can be associated with proposing a new order on agents with position higher than k. Here $\mathbb{S} = \{0, 1, 2, 3, 4\}$.

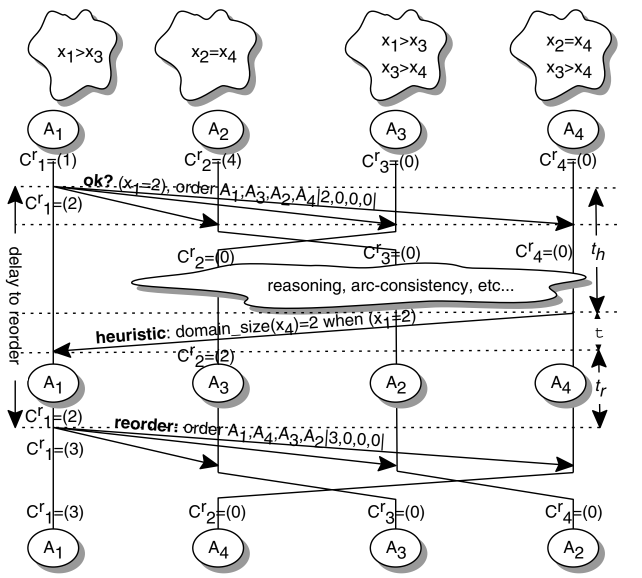

Figure 3. The agent proposing reordering does not have the whole knowledge for efficient decisions. He can be informed with heuristic messages, if it can be proven that the delay t_h needed to generate such messages is finite (shown t_h is since A^1 sends the ok? message to when A^4 replies to the owner of C_1^r, which here is A^1). Message travel time τ and time to compute $\mathbb{P}$, t_r, are assumed finite.

1.3.1 Some notations

An ordering on agents is defined by a permutation of the set of agents. The agent on position k in an ordering o is denoted $A^k(o)$. A synonym of $A^k(o)$ is $\mathbf{A_i^k(o)}$ which also tells that the agent on position k in ordering o is A_i. A similar notation is introduced for variables. $\mathbf{x_i^k(o)}$ tells that the variable of the agent on position k in ordering o is x_i. $x^k(o)$ denotes the variable of the agent on position k. The agent maintaining C_k^r in order o is denoted $\mathbf{R^k(o)}$. Sometimes the ordering o is missing from the previous notations in the context of a procedure of some agent A_j, and then the use of the current ordering known by A_j is implied.

1.3.2 Private constraints or arc consistency

For the case where an agent does not know all constraints on his variable (e.g., assume that in Figure 2, A_1 does not know the constraint between x_1 and x_3), that agent may not be able to predict how his assignments modify the domains of future variables, and min-domain heuristics cannot be implemented with the above protocol. Even when agents know all constraints on their variable, if arc consistency is maintained then agents cannot foresee the effect of their assignments on domains of lower priority agents.

To enable min-domain heuristics in such cases ABTR allows lower priority agents to inform others about features in $\mathbb{K}$, of interest to the reordering (such as new domain sizes), using **heuristic** messages specified by the $\mathbb{H}$ part of the heuristic. Based on this information, agents can renew the reordering proposed with a previous assignment (see Figure 3). This is shown later to lead to correct and terminating protocols for the heuristics where it can be proven separately that $\mathbb{H}$ complies with the following requirement:

Rule 1 (delay to reorder) *The delay between the moment the last* **ok?** *message was sent by agents $A^k, k{\le}i$, (or from start) and any subsequent reordering request sent by the owner of C_i^r,* (**delay to reorder** *in Figure 3)* **must be** *finite.*

One can use t_h instead of the delay to reorder (see Figure 3). An equivalent formulation is: *For a given* **ok?** *message sent by A^i or predecessors, the owner of C_i^r may propose any finite number of ordering requests. Additionally, any finite number of reorderings may be requested after start.* $\mathbb{H}_0$ for min-domain literally repeats this rule.

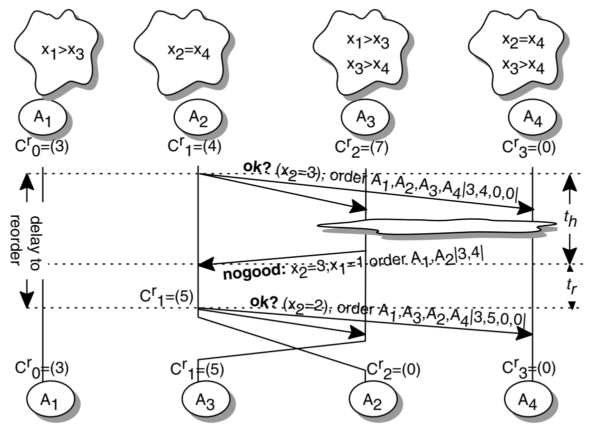

Figure 4. To obtain DB's heuristic, the reordering counter of an agent on a position k will count the reordering events of the sets of agents monitored in previous versions of ABTR by C^r_{k-1}, and initializes it from signatures of received messages. $\mathbb{S} = \{0, 1, 2, 3\}$. The counter C^r_0 tags the reorderings of the first agent. On receiving the nogood, A_2 implicitly and instantaneously delivers to itself a heuristic message. Similar to DB's termination proof we show that the time t_h is finite, implying as for previous ABTR heuristics that this algorithm is correct and terminates.

1.3.3 Modeling DB's and approx-AWC1 heuristics

The first instance of ABTR, implemented for obtaining min-domain heuristics in our system [18, 20, 16], is as follows: *Upon receiving a reordering request 'o' with signatures vector-clock (e.g., $|c_0, ..., c_n|$) stronger than any other request received before, an agent A^j*, $\mathbf{j{>}1}$, *requested by 'o' to move to another position k will reset its new reordering counter C^r_k to* **zero**.

Note that a vector-clock h_1 is stronger than h_2 of h_1 follows h_2 in lexicographic order. However, the corresponding **counters rule** of the ABTR description [17], provided to also support the heuristics described here is:
Upon receiving a reordering request 'o' with signatures vector-clock (e.g., $|c_0, ..., c_n|$) stronger than any other request received before, an agent A^j requested by 'o' to move to another position, k, will initialize its new reordering counter C^r_k **with $\mathbf{c_k}$**.

Note that this changes nothing for heuristics like min-domain, since in them the corresponding values of c_k are zero (because the sender does not have any opportunity to learn about changes to counters in lower priority agents).

The counters rule, if coupled with a policy $\mathbb{M}$ to have each agent A^k hold the counter C^r_{k-1} is shown here to be sufficient for modeling approx-AWC1 and Dynamic Backtracking's reordering heuristic. Namely, holding the counter C^r_{k-1} gives A^k the right to reorder itself (according to the above definition of $\mathbb{P}$), while the counters rule ensure continuity in the value of C^r_{k-1} when the agent on position k and holding C^r_{k-1} is moved because of its own decision (see Figure 4).

1.3.4 Further generalization:

ABTR provides for a scenario not used in min-domain or in mentioned self-reordering heuristics, namely where the reordering counters C^r_k have a dynamic mapping $\mathbb{M}$ to agents, including the possibility that a reordering request can explicitly specify how to move lower priority counters to other agents. The corresponding **general counters rule** of ABTR supporting this case is: *Upon receiving a reordering request 'o' with signatures vector-clock (e.g., $|c_0, ..., c_n|$) stronger than any other request received before, an agent A^j requested by 'o'* **to start maintaining a reordering counter** *C^r_k will initialize his new reordering counter C^r_k with c_k.*

This generalization is proved directly in [17] since its proof applies to all aforementioned specializations. This scenario allows all previous heuristics and new ones. No overhead in message sizes occurs if the new mapping of reordering counters to agents associated with a new ordering 'o' does not need to be explicitly transmitted but can be inferred from 'o' using some predefined convention (as happens in the min-domain ABTR scenario where we know that C^r_k is mapped to the agent on position k).

2 Related Work

General frameworks for comprehensive description of possible dynamic ordering heuristics for centralized CSP solvers were provided mainly in connection with Dynamic Backtracking [9]. They describe supported heuristics in terms of the set of reorderings allowed at each step. The first complete asynchronous search algorithm for DisCSPs is the Asynchronous Backtracking (ABT) [21]. The approach in [21, 4, 14] considers that each agent maintains only one distinct variable. Other definitions of DisCSPs have considered the case where constraints are distributed among agents [22, 8, 6]. The order on variables in distributed search was so far addressed in [5, 21, 10, 1, 17, 7, 23], showing the interest and strong impact it has on the solving algorithms. [1] is the first to evaluate the impact of reordering in asynchronous backtracking, by introducing synchronization points that separate short asynchronous epochs, and report up to 30% improvement for different ordering heuristics over random ordering.

The first description of ABTR is provided in [20] and focuses on introducing min-domain dynamic ordering in generalizations of ABT that include consistency maintenance and generalized nogoods [2]. A short but complete description of ABTR, with examples based on *min-domain*-like toy heuristics in ABT, is introduced in [17]. The framework for describing heuristics for ABTR is presented there in a less systematic formalism than here. The description in [17] mentions no motivation or example for several important features, such as the ones supporting the self-reordering heuristics introduced here. Finally, these examples, motivations and experimentation appeared in the technical report [19] which served as the basis of the current article.

A number of ulterior publications [7, 23, 24] describe independently developed reordering techniques. These techniques were proposed with their own proof, but are shown in Section 4 to be instances of ABTR for certain ABTR-compliant heuristics.

3 Protocol Details

Let us consider two different orderings, o_1 and o_2, with their signatures h_1 and h_2: $O_1 = \langle o_1, h_1\rangle$, $O_2 = \langle o_2, h_2\rangle$.

Definition 1 (Reorder position) *The* **reorder position** *of* h_1 *and* h_2, $\mathbf{R(h_1,h_2)}$, *is the position of the highest priority agent reordered between the generation of the signatures* h_1 *and* h_2. *If* u *is the lowest element in* $\mathbb{S}$ *where signatures vector-clocks* h_1 *and* h_2 *differ then* $R(h_1, h_2)=u+1$.

New (optional) messages for ABTR with a heuristic $(\mathbb{K}, \mathbb{S}, \mathbb{M}, \mathbb{P}, \mathbb{H})$ are:

- **heuristic** messages for heuristic dependent data $\mathbb{K}$, specified by $\mathbb{H}$ (e.g. announcing changes of a variable's domain size to higher priority agents), and
- **reorder** messages requesting a new ordering o (defined by $\mathbb{P}$ and $\mathbb{M}$) and tagged with signatures vector-clock h, $\langle o, h\rangle$.

An agent R^i announces its requested order, o, by sending **reorder** messages to successor agents $\mathbb{S}_i = \{A^k(o) \mid k > i\}$, and to all agents $R^k(o), k>i$, not in $\mathbb{S}_i$. Each agent A_i stores its *known order* denoted $\mathbf{O^{crt}}(A_i)$. For allowing asynchronous reordering, each **ok?** and **nogood** message receives as additional parameter an order and its signature (see Algorithm 1). The **ok?** messages hold the strongest *known order of* the sender. Each **nogood** message sent from A^j to A^i is tagged with an order consisting of the prefix of i agents in the $O^{crt}(A^j)$ and the prefix of i counters in the signatures vector-clock of $O^{crt}(A^j)$.

When A_i receives a message which contains an order with a signature h that is newer than the signature h^* of $O^{crt}(A_i)$, the assignments known by A_i for the variables x^k, $k \geq R(h, h^*)$, are invalidated.

Due to the delay-to-reorder rule, **heuristic** messages may be sent to R^k only in response to a new assignment for $x^j, j{\leq}k$, or at start (the maximum possible delay between the generation of such an assignment and sending the **heuristic** message is denoted $\mathbf{t_h}$, see Figure 3). An agent R^k generating or receiving an assignment for some variable $x^j, j{\leq}k$, or a nogood, may implicitly deliver to itself a **heuristic** message with knowledge from $\mathbb{K}$. This convention unifies the framework with the cases where no explicit **heuristic** messages are needed. We can say that the reception of an assignment (and the start), are "enabling events" for sending a corresponding **heuristic** message.

```
when received (ok?, $\langle x_j, d_j\rangle$, $\langle o, h\rangle$) do
    getOrder($\langle o, h\rangle$); if(old) return;
    add($x_j$,$d_j$) to agent_view; clean nogoods;
    check_agent_view; send optional heuristic messages;
end do.
when received (nogood, $A_j$, $\neg N$, $\langle o, h\rangle$) do
    getOrder($\langle o, h\rangle$); if I am not enforcing $\neg N$ return;
    if (($\langle x_i, d\rangle$c$\in N$ and I have better nogood for $x_i$=$d$ or
    (if $\neg N$ contains invalid assignments)) then
        discard $\neg N$;
    else
        when unconnected $\langle x_k, d_k\rangle$ is contained in $\neg N$
            send add-link to $A_k$;
            add $\langle x_k, d_k\rangle$ to agent_view; clean nogoods;
        put $\neg N$ in nogood-list for $x_i$=$d$;
        add other new assignments to agent_view;
        clean nogoods;
    old_value $\leftarrow$ current_value; check_agent_view;
    when old_value = current_value and
        if $A_j$ has lower priority than $A_i$
1.1     send (ok?,$\langle x_i$, current_value$\rangle$,$O^{crt}$) to $A_j$;
    send heuristic messages according to used heuristic;
end do.
when received heuristic(level, data) do
    if not holding $C^r_{level}$ then return;
    $C^r_{level}$++; use data to generate new order $O'$;
    send (reorder,$O'$) to $R^j, j > level$; getOrder($O'$);
end do.
function getOrder($\langle o, h\rangle$) $\rightarrow$ bool
    when not newer $h$ than $O^{crt}$ then return true;
    $I \leftarrow$ reorder position for $h$ and the signature of $O^{crt}$;
    invalidate assignments for $x^j, j \geq I$; $\langle o, h\rangle \rightarrow O^{crt}$;
1.2 send (ok?,$\langle x_i$, some value$\rangle$ ,$O^{crt}$) to all lower prior-
    ity agents in outgoing links;
    return true;
end.
```

Algorithm 1: Procedures of A_i for receiving messages in ABTR. **check_agent_view** and **backtrack** are the corresponding procedures of ABT [21] where sent messages are tagged with the current order.

Theorem 1 (ABTR [17]) *ABTR is correct, complete and terminates for any heuristics where the local computation times t_h of A^i reacting with* **heuristic** *messages to an* **ok?** *message can separately be proven finite after $A_j, j<i$, reach quiescence. (Messages travel times τ is assumed finite, i.e. no message loss, and the time t_r of answering with* **reorder** *messages to a* **heuristic** *message is also assumed finite.)*

For proof see [17].

Remark 1 *Note that the Theorem 1 does not prove that approx-AWC1, Dynamic Backtracking's heuristic (or some other named heuristic) would lead to a correct and terminating ABTR. It only proves that any heuristic and dynamic mapping policy respecting the rules mentioned so far leads to a correct and terminating protocol with ABTR. As we do next, one has to prove separately for each proposed heuristic the fact that it respects the conditions set so far (in particular the finiteness of local computation time t_h from receiving an assignment or from start, to the moment of sending corresponding* **heuristic** *messages).*

Let us not formally define approx-AWC1 as $(\mathbb{K}_1, \mathbb{S}_1, \mathbb{M}_1, \mathbb{P}_1, \mathbb{H}_1)$, and the heuristic of Dynamic Backtracking as $(\mathbb{K}_1, \mathbb{S}_1, \mathbb{M}_1, \mathbb{P}_{db}, \mathbb{H}_1)$:

$\mathbb{S}_1$ $[0..(n-1)]$.

$\mathbb{M}_1$ maps each counter C^r_k to the agent on position $k-1$.

$\mathbb{K}_1$ contains information about current orderings and about wipe-outs for domains of variables, namely events where an agent did not find for its variable any value consistent with higher priority assignments.

$\mathbb{P}_1$ specifies that the owner A_i of the k^{th} counter, $k>0$, should reorder the first agent whose domain wipe-out was due to A_i immediately, before itself.

$\mathbb{P}_{db}$ specifies that the owner A_i of the k^{th} counter, $k>0$, should reorder itself immediately after the first agent whose domain wipe-out is due to A_i's current assignment.

$\mathbb{H}_1$ specifies that information about $\mathbb{K}$ is never transmitted via explicit **heuristic** messages but may be learned from incoming valid nogoods and orderings.

As mentioned in Introduction, the two heuristics differ only in $\mathbb{P}$. Next we prove that the exact heuristic of DB as well as approx-AWC1 are ABTR compliant (Rule 1). The obtained instances are called ABTR-db respectively ABTR-wc.

Theorem 2 *Valid nogoods can be received by agent A^i in ABTR-db/ABTR-wc only in finite time after the agents $A^j, j<i$, have reached quiescence (i.e., terminate).*

Proof. Each valid nogood received by A^i eliminates a value. Each value eliminated by such a nogood after all predecessors $A^j, j<i$, reach quiescence, will never be available again since the nogoods eliminating them use fixed assignments of quiescent agents and cannot be invalidated. There are maximum d such values. All agents learn a new assignments within time $n\tau$ from its proposal. Therefore, all valid nogoods are received in finite time. □

Corollary 1 *There exists a finite t_h such that any valid is received by R^{k-1} (i.e., A^k) in a time bounded by t_h after the quiescence of the agents $A^l, l<k$. Q.E.D.*

4 Generality of new framework

Reordering by the first agent (restarts): Experiments with dynamic ordering in ABT are described in [7], focusing on detailed analysis of the case where only the first agent reorders all other agents. While that technique was developed independently, it can be easily shown that it is an instance of ABTR, e.g. for a heuristic $(\mathbb{K}_r, \{1\}, \mathbb{M}_r, \mathbb{P}_r, \mathbb{H}_r)$:

$\mathbb{M}_r$ maps counter C^r_1 to the agent on the first position.

$\mathbb{K}_r$ is empty since no knowledge is needed for reordering.

$\mathbb{P}_r$ specifies that the first agent, holding counter C^r_1, requests at certain time intervals a random dynamic ordering on the rest of the agents. It stops requesting reordering after a timeout t_h.

$\mathbb{H}_r$ specifies that agents do not need to announce any knowledge to the first agent.

This heuristic is in a straightforward observance to "the letter" of Rule 1. Experimental improvements of approximately 5% were mentioned for this heuristic in [7].

ABT_DO: The ABT_DO algorithm developed and introduced in [23] is shown now to be an instance of ABTR, for an ABTR compliant heuristic that we will denote *approx-AWC2*. For the scenario introduced in Figure 1, with the *approx-AWC2* heuristic used in ABT_DO [23], it is the agents/variables after Bob, namely Carol and Dave, that are moved beyond Eve. This is different from our approx-AWC1 where, like in AWC, Bob is also moved beyond Eve. A model of approx-AWC2 in ABTR's framework is $(\mathbb{K}_1, \mathbb{S}_2, \mathbb{M}_1, \mathbb{P}_2, \mathbb{H}_1)$, where $\mathbb{K}1$, $\mathbb{M}_1$, and $\mathbb{H}_1$ are as for approx-AWC1, and:

$\mathbb{S}_2$ $\mathbb{S}_2 = [1..n]$.

$\mathbb{P}_2$ specifies that the owner A_i of the k^{th} counter, $k>0$, should reorder the last agent whose domain wipe-out was due to A_i immediately after itself.

The proof that approx-AW2, i.e., $(\mathbb{K}_1, \mathbb{S}_2, \mathbb{M}_1, \mathbb{P}_2, \mathbb{H}_1)$, is ABTR compliant is the same as the one for approx-AWC1 and for the exact heuristic of Dynamic Backtracking, since only $\mathbb{S}$ and $\mathbb{P}$ components differ and they were not involved in approx-AWC's correctness proof. While the differences between approx-AWC1 and approx-AWC2 seem small, the authors of ABT_DO describe differences of orders of magnitude in efficiency. Another recent development by the authors of approx-AWC2 consists of the suggestion that mixing approx-AWC1 with min-domain also results in large improvements.

The ABTR model for the min-domain heuristics in [23] (that cannot reorder the first agent) differs from the min-domain heuristic $(\mathbb{K}_0, \mathbb{S}_0, \mathbb{M}_0, \mathbb{P}_0, \mathbb{H}_0)$ we proposed in [20] (see Introduction), only in the usage of the counter C_0^r (which is absent in ABT_DO). Their other experiments with min-domain heuristics correlate with our previous results.

5 Experimentation

We have run tests on random problems with 20 agents with a really distributed implementation of ABT and of three versions of ABTR-wc. ABTR-wc-a reuses the current order on agents with lower priority. ABTR-wc-b is like ABTR-wc-a, but exchanges more information on the current ordering of future agents (with each message). The agents were placed on distinct computers of our lab. We have generated problems of variable tightness for a density of 27% where each variable has 3 values.

We chose our metric as close as possible to the most used metric in multi-party computations and asynchronous distributed CSPs solvers, namely the *number of rounds* [3, 14, 21]. However, this metric cannot be directly computed as such in a real asynchronous implementation (which has nothing like rounds), and therefore we chose the most similar measure that we could find and that, if the system and the measurement would be run on a simulator, would be expected to return the same value as the number-of-rounds. Therefore, the chosen metric is the *length of the longest causal chain of messages*, effectively measured as the longest chain of sequential messages using the algorithm in [12]. On a simulator this measure would be equal to the number of rounds. Theoretically, this measure is the only important cost when agents are placed remotely on Internet and the local problems are not hard. Each point in Figure 5 is averaged over 100 random problems and shows the average length of the longest causal chain of messages required to solve the problem. The experiments show that ABTR-wc-b performed clearly better in average than ABT and performed better than other versions of ABTR-wc. For under-constrained problems (tightness under 15%) where solutions are found without resorting to many **no-good** messages, few reorderings are proposed by ABTR-wc

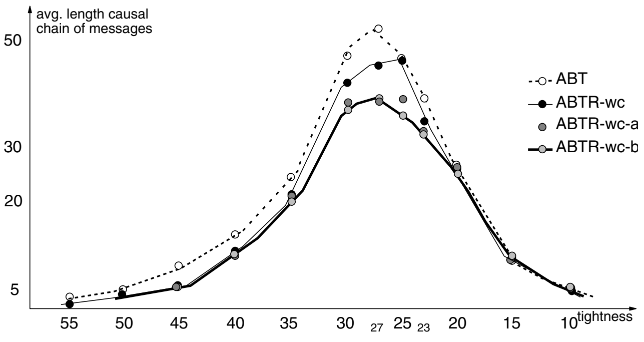

Figure 5. Experiments

and therefore the new algorithms perform quite similarly to ABT.

Experiments using min-domain dynamic ordering with maintenance of arc consistency [20]) (20 agents, 15 variables, 8 values, density 20%, tightness 68%, 500 instances) reveal that *no-reordering* is 30% better than dynamic min-domain ordering, while min-domain is 3 times better than max-domain. This correlates with the prediction in [1], and is confirmed in [23].

Large numbers of new heuristics were enabled here. We no longer have access to a sufficiently large network of computers, but other researchers completed recently new extensive experimentation [7, 23, 24] with our heuristics as well as with new heuristics enabled by ABTR, such as approx-AWC2, and reduced versions of min-domain, proving the importance of this contribution. The implementation used for the experiments reported here is available for free download on our web-site [16] and can be used for further research by whoever owns a sufficiently large network.

6 Conclusions

Reordering is a major issue in constraint satisfaction and is the main strength of Asynchronous Weak-Commitment (AWC). Here we introduce a general and systematic formalism for describing heuristics that are correct with ABTR. We also describe and experiment, *approx-AWC1* (inspired from AWC), the first self-reordering heuristic for polynomial space and complete asynchronous backtracking, namely where an agent decreases its own priority (such heuristics are renamed *retroactive* in [24]). These define the first experimentally efficient dynamic reordering heuristics for ABTR. Our contribution is inspiring others, and independent works show that extensions of approx-AWC1 (e.g., with min-domain [24]) as well as slightly modified versions (approx-AWC2 [23]), lead to even more impressing improvements.

We show that new algorithms are instances of ABTR and show that ABTR remains more general than all new specifications, e.g. by supporting min-domain that reorders the first agent. Additional types of reordering heuristics leading to complete solvers are explained with our framework, and the task of exploring them for useful heuristics define many opportunities and directions for future work. Our really distributed implementation is available for download [16] and further research to whoever owns corresponding resources.

References

[1] A. Armstrong and E. F. Durfee. Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *Proceedings of 15th IJCAI*, 1997.

[2] F. Bacchus and P. Van Beek. On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of the 15th National Conference on Artificial Intelligence*, Madison, Wisconsin, July 1998.

[3] M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computating. In *STOC*, pages 1–10, 1988.

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

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

[6] J. Denzinger. Distributed knowledge based search. IJCAI tutorial notes (MA2), 2001.

[7] C. Fernàndez, R. Béjar, B. Krishnamachari, and C. Gomes. Communication and computation in distributed CSP algorithms. In *CP*, pages 664–679, 2002.

[8] E. C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI'91*, pages 227–233, 1991.

[9] M. L. Ginsberg. Dynamic backtracking. *Journal of AI Research*, 1, 1993.

[10] Y. Hamadi and C. Bessière. Backtracking in distributed constraint networks. In *ECAI'98*, pages 219–223, 1998.

[11] W. Havens. Nogood caching for multiagent backtrack search. In *AAAI Constraints and Agents Workshop*, 1997.

[12] Leslie Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[13] F. Mattern. Algorithms for distributed termination detection. *Distributed Computing*, 2:161–175, 1987.

[14] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *AIJ*, 161, 2005.

[15] Viet Nguyen, Djamila Sam-Haroud, and Boi Faltings. Dynamic distributed backjumping. In *DCR Workshop*, pages 51–65, 2004.

[16] M.-C. Silaghi. DisCSP algorithms on the Mely platform. `www.cs.fit.edu/~msilaghi/discsps/sJavap/`, 2004.

[17] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. ABT with asynchronous reordering. In *IAT*, 2001.

[18] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous consistency maintenance with reordering. Technical Report #01/360, EPFL, March 2001.

[19] 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.

[20] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Polynomial space and complete multiply asynchronous search with abstractions. In *IJCAI-01 DCR Workshop*, pages 17–32, Seattle, August 2001.

[21] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE TKDE*, 10(5):673–685, 1998.

[22] 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.

[23] R. Zivan and A. Meisels. Dynamic ordering for asynchronous backtracking on discsps. In *CP*, pages 161–172, 2005.

[24] R. Zivan, M. Zazone, and A. Meisels. Retroactive ordering heuristics for asynchronous backtracking on discsps. AAMAS-DCR, 2006.