

## Corrigendum to "Min-domain retroactive ordering for Asynchronous Backtracking"

Younes Mechqrane, Mohamed Wahbi, Christian Bessière, El Houssine Bouyakhf, Amnon Meisels, Roie Zivan

► **To cite this version:**

Younes Mechqrane, Mohamed Wahbi, Christian Bessière, El Houssine Bouyakhf, Amnon Meisels, et al.. Corrigendum to "Min-domain retroactive ordering for Asynchronous Backtracking". Constraints, Springer Verlag, 2012, 17, pp.348-355. <lirmm-00748174>

**HAL Id: lirmm-00748174**

**<https://hal-lirmm.ccsd.cnrs.fr/lirmm-00748174>**

Submitted on 5 Nov 2012

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Corrigendum to “Min-domain retroactive ordering for Asynchronous Backtracking”

Younes Mechqrane<sup>1</sup> Mohamed Wahbi<sup>2</sup> Christian Bessiere<sup>2</sup>  
El Houssine Bouyakhf<sup>1</sup> Amnon Meisels<sup>3</sup> Roie Zivan<sup>3</sup>

<sup>1</sup> LIMIARF, University Mohammed V Agdal, Rabat, Morocco

<sup>2</sup> LIRMM, University of Montpellier, France

<sup>3</sup> Ben-Gurion University, Beer-Sheva, Israel

## Abstract

The asynchronous backtracking algorithm with dynamic ordering (ABT\_DO), proposed in [ZM06], allows changing the order of agents during distributed asynchronous complete search. In a later study [ZZA09], retroactive heuristics which allowed more flexibility in the selection of new orders were introduced, resulting in the ABT\_DO-Retro algorithm, and a relation between the success of heuristics and the min-domain property was identified. Unfortunately, the description of the time-stamping protocol used to compare orders in ABT\_DO-Retro in [ZZA09] is confusing and may lead to an implementation in which ABT\_DO-Retro may not terminate. In this corrigendum, we demonstrate the possible undesired outcome and give a detailed and formal description of the correct method for comparing time-stamps in ABT\_DO-Retro.

## 1 Introduction

The ABT\_DO algorithm allows the use of dynamic ordering heuristics in asynchronous search algorithms for solving DisCSPs. This algorithm was the main theme of two recent publications in the *Constraints* journal, [ZM06] and [ZZA09]. The algorithm proposed in the first among them ([ZM06]) allows the use of heuristics where agents can propose order changes when they replace the value assignment to their variables. This change can include only agents that are ordered after (with lower priority) the agent that replaced its assignment. In the second ([ZZA09]), more flexible heuristics that allow changing the order of agents that come before the agent that replaces its assignment (retroactive heuristics) are introduced to the algorithm.

The most successful ordering heuristic found in [ZM06] was the *nogood-triggered* heuristic in which an agent that receives a nogood moves the nogood generator to be right after it in the order. The heuristic investigation in [ZZA09] demonstrates the relation between the success of the nogood-triggered heuristic and the min-domain property. This relation was exploited in the retroactive version of this heuristic by moving a nogood generator to the highest position in the order that does not causes values previously removed to be reentered into the variable’s current domain (and increase its size).

Recent attempts to implement the ABT\_DO-Retro algorithm proposed in [ZZA09] have revealed a specific detail of the algorithm that was vaguely described and can lead to an interpretation that affects the correctness of the algorithm. In this corrigendum we address this vague description by describing the undesired outcome and propose an alternative deterministic description that ensures the outcome expected in [ZZA09].

## 2 Background

The degree of flexibility of the retroactive heuristics mentioned above depends on a parameter  $K$ .  $K$  defines the level of flexibility of the heuristic with respect to the amount of information an agent can store in its memory. Agents that detect a dead end move themselves to a higher priority position in the order. If the length of the nogood created is not larger than  $K$  then the agent can move to any position it desires (even to the highest priority position) and all agents that are included in the nogood are required to add the nogood to their set of constraints and hold it until the algorithm terminates. If the size of the created nogood is larger than  $K$ , the agent that created the nogood can move up to the place that is right after the second last agent in the nogood. Since agents must store nogoods that are smaller than or equal to  $K$ , the space complexity of agents is exponential in  $K$ .

The best retroactive heuristic introduced in [ZZA09] is called ABT\_DO-Retro-MinDom. This heuristic does not require any additional storage (i.e.,  $K = 0$ ). In this heuristic, the agent that generates a nogood is placed in the new order between the last and the second last agents in the generated nogood. However, the generator of the nogood moves to a higher priority position than the backtracking target (the agent the nogood was sent to) only if its domain is smaller than that of the agents it passes on the way up. Otherwise, the generator of the nogood is placed right after the last agent with a smaller domain between the last and the second last agents in the nogood.

In asynchronous backtracking algorithms with dynamic ordering, agents propose new orders asynchronously. Hence, one must enable agents to coherently decide which of two different orders is more up-to-date. To this end, as it has been explained in [ZM06] and recalled in [ZZA09], each agent in ABT\_DO holds a *counter vector* (one counter attached to each position in the order). The counter vector and the indexes of the agents currently in these positions form a time-stamp. Initially, all counters are set to zero and all agents are aware of the initial order. Each agent that proposes a new order increments the counter attached to its position in the current order and sets to zero counters of all lower priority positions (the counters of higher priority positions are not modified). The most up-to-date order is determined by a lexicographic comparison of counter vectors combined with the agent indexes. However, the rules for reordering agents in ABT\_DO imply that the most up-to-date order is always the one for which the first different counter is larger.

Regarding the procedure by which orders are compared in ABT\_DO-Retro, the description given by the authors was vague and was limited to two sentences: *“The most relevant order is determined lexicographically. Ties which could not have been generated in standard ABT\_DO, are broken using the agents indexes”*[quoted from [ZZA09], page 190, Theorem 1].

The natural understanding of this description is that the most up-to-date order is the one associated with the lexicographically greater counter vector, and when the counter vectors are equal the lexicographic order on the indexes of agents breaks the tie by preferring the one with smaller vector of indexes. We will refer to this interpretation as method  $m_1$ . Let us illustrate method  $m_1$  via an example. Consider two orders  $o_1=[A_1, A_3, A_2, A_4, A_5]$  and  $o_2=[A_1, A_2, A_3, A_4, A_5]$  where the counter vector associated with  $o_1$  equals  $[2, 4, 2, 2, 0]$  and the counter vector associated with  $o_2$  equals  $[2, 4, 2, 1, 0]$ .

Since in  $m_1$  the most up-to-date order is determined by comparing lexicographically the counter vectors, in this example  $o_1$  is considered more up-to-date than  $o_2$ . In Section 3 of this corrigendum, we show that method  $m_1$  may lead ABT\_DO-Retro to fall in an infinite loop when  $K = 0$ .

The right way to compare orders is to compare their counter vectors, one position at a time from left to right, until they differ on a position (preferring the order with greater counter) or they are equal on that position but the indexes of the agents in that position differ (preferring the smaller index). We will refer to this method as  $m_2$ . Consider again the two orders  $o_1$  and  $o_2$  and associated counter vectors defined above. The counter at the first position equals 2 on both counter vectors and the index of the first agent in  $o_1$  (i.e.,  $A_1$ ) is the same as in  $o_2$ , the counter at the second position equals 4 on both counter vectors, however the index of the second agent in  $o_2$  (i.e.,  $A_2$ ) is smaller than the index of the second agent in  $o_1$  (i.e.,  $A_3$ ). Hence, in this case  $o_2$  is considered more up-to-date than  $o_1$ . (Note that according to  $m_1$ ,  $o_1$  is more up-to-date than  $o_2$ .) In Section 4 of this corrigendum, we give the proof that method  $m_2$  for comparing orders is correct.

### 3 ABT\_DO-Retro May Not Terminate

In this section we show that ABT\_DO-Retro may not terminate when using  $m_1$  and when  $K = 0$ . We illustrate this on ABT\_DO-Retro-MinDom as described in [ZZA09] as it is an example of ABT\_DO-Retro where  $K = 0$ . Consider a DisCSP with 5 agents  $\{A_1, A_2, A_3, A_4, A_5\}$  and domains  $D(x_1) = D(x_5) = \{1, 2, 3, 4, 5\}$ ,  $D(x_2) = D(x_3) = D(x_4) = \{6, 7\}$ . We assume that, initially, all agents store the same order  $o_1 = [A_1, A_5, A_4, A_2, A_3]$  with associated counter vector  $s_1 = [0, 0, 0, 0, 0]$ . The constraints are:

- $c_{12} : (x_1, x_2) \notin \{(1, 6), (1, 7)\}$ ;
- $c_{13} : (x_1, x_3) \notin \{(2, 6), (2, 7)\}$ ;
- $c_{14} : (x_1, x_4) \notin \{(1, 6), (1, 7)\}$ ;
- $c_{24} : (x_2, x_4) \notin \{(6, 6), (7, 7)\}$ .
- $c_{35} : (x_3, x_5) \notin \{(7, 5)\}$ .

In the following we give a possible execution of ABT\_DO-Retro-MinDom.

- $t_0$ : All agents assign the first value in their domains to their variables and send **ok?** messages to their neighbors.
- $t_1$ :  $A_4$  receives the first **ok?** ( $x_1 = 1$ ) message sent by  $A_1$  and generates a nogood  $ng_1 : \neg(x_1 = 1)$ . Then, it proposes a new order  $o_2 = [A_4, A_1, A_5, A_2, A_3]$  with  $s_2 = [1, 0, 0, 0, 0]$ . Afterwards, it assigns the value 6 to its variable and sends **ok?** ( $x_4 = 6$ ) message to all its neighbors (including  $A_2$ ).
- $t_2$ :  $A_3$  receives  $o_2 = [A_4, A_1, A_5, A_2, A_3]$  and deletes  $o_1$  since  $o_2$  is more up-to-date;  $A_1$  receives the nogood sent by  $A_4$ , it replaces its assignment to 2 and sends an **ok?** ( $x_1 = 2$ ) message to all its neighbors.
- $t_3$ :  $A_2$  has not yet received  $o_2$  and the new assignment of  $A_1$ .  $A_2$  generates a new nogood  $ng_2 : \neg(x_1 = 1)$  and proposes a new order  $o_3 = [A_2, A_1, A_5, A_4, A_3]$  with  $s_3 = [1, 0, 0, 0, 0]$ ; Afterwards, it assigns the value 6 to its variable and sends **ok?** ( $x_2 = 6$ ) message to all its neighbors (including  $A_4$ ).

$$\begin{aligned}
o_1 &= [A_1, A_5, A_4, A_2, A_3] & s_1 &= [0, 0, 0, 0, 0] \\
o_2 &= [A_4, A_1, A_5, A_2, A_3] & s_2 &= [1, 0, 0, 0, 0] \\
o_3 &= [A_2, A_1, A_5, A_4, A_3] & s_3 &= [1, 0, 0, 0, 0] \\
o_4 &= [A_4, A_3, A_1, A_5, A_2] & s_4 &= [1, 1, 0, 0, 0]
\end{aligned}$$

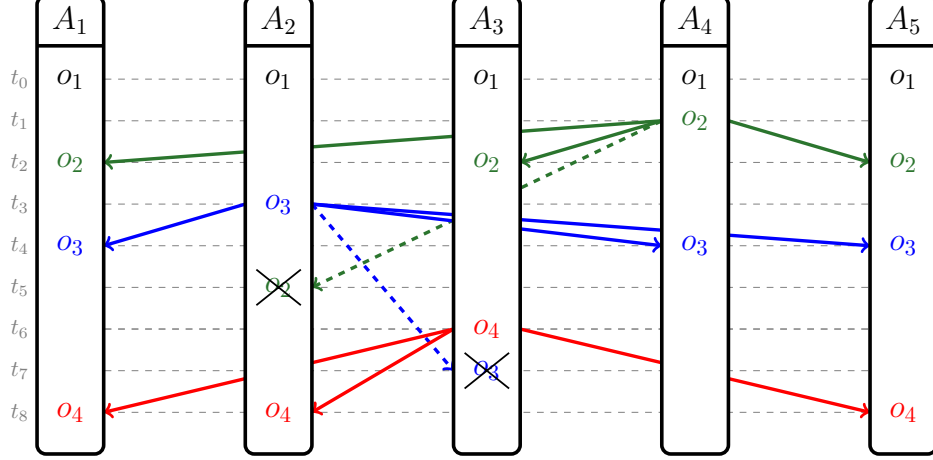


Figure 1: The schema of exchanging **order** messages by ABT\_DO-Retro

- $t_4$ :  $A_4$  receives the new assignment of  $A_2$  (i.e.,  $x_2 = 6$ ) and  $o_3 = [A_2, A_1, A_5, A_4, A_3]$ . Afterwards, it discards  $o_2$  since  $o_3$  is more up-to-date; Then,  $A_4$  tries to satisfy  $c_{24}$  because  $A_2$  has a higher priority according to  $o_3$ . Hence,  $A_4$  replaces its current assignment (i.e.,  $x_4 = 6$ ) by  $x_4 = 7$  and sends an **ok?** ( $x_4 = 7$ ) message to all its neighbors (including  $A_2$ ).
- $t_5$ : When receiving  $o_2$ ,  $A_2$  discards it because its current order is more up-to-date;
- $t_6$ : After receiving the new assignment of  $A_1$  (i.e.,  $x_1 = 2$ ) and before receiving  $o_3 = [A_2, A_1, A_5, A_4, A_3]$ ,  $A_3$  generates a nogood  $ng_3 : \neg(x_1 = 2)$  and proposes a new order  $o_4 = [A_4, A_3, A_1, A_5, A_2]$  with  $s_4 = [1, 1, 0, 0, 0]$ ; The order  $o_4$  is more up-to-date according to  $m_1$  than  $o_3$ . Since in ABT\_DO, an agent sends the new order only to lower priority agents,  $A_3$  will not send  $o_4$  to  $A_4$  because it is a higher priority agent.
- $t_7$ :  $A_3$  receives  $o_3$  and then discards it because it is obsolete;
- $t_8$ :  $A_2$  receives  $o_4$  but it has not yet received the new assignment of  $A_4$ . Then, it tries to satisfy  $c_{24}$  because  $A_4$  has a higher priority according to its current order  $o_4$ . Hence,  $A_2$  replaces its current assignment (i.e.,  $x_2 = 6$ ) by  $x_2 = 7$  and sends an **ok?** ( $x_2 = 7$ ) message to all its neighbors (including  $A_4$ ).
- $t_9$ :  $A_2$  receives the **ok?** ( $x_4 = 7$ ) message sent by  $A_4$  in  $t_4$  and changes its current value (i.e.,  $x_2 = 7$ ) by  $x_2 = 6$ . Then,  $A_2$  sends an **ok?** ( $x_2 = 6$ ) message to all its neighbors (including  $A_4$ ). At the same time,  $A_4$  receives **ok?** ( $x_2 = 7$ ) sent by  $A_2$  in  $t_8$ .  $A_4$  changes its current value (i.e.,  $x_4 = 7$ ) by  $x_4 = 6$ . Then,  $A_4$  sends an **ok?** ( $x_4 = 6$ ) message to all its neighbors (including  $A_2$ ).

$t_{10}$ :  $A_2$  receives the **ok?** ( $x_4 = 6$ ) message sent by  $A_4$  in  $t_9$  and changes its current value (i.e.,  $x_2 = 6$ ) by  $x_2 = 7$ . Then,  $A_2$  sends an **ok?** ( $x_2 = 7$ ) message to all its neighbors (including  $A_4$ ). At the same moment,  $A_4$  receives **ok?** ( $x_2 = 6$ ) sent by  $A_2$  in  $t_9$ .  $A_4$  changes its current value (i.e.,  $x_4 = 6$ ) by  $x_4 = 7$ . Then,  $A_4$  sends an **ok?** ( $x_4 = 7$ ) message to all its neighbors (including  $A_2$ ).

$t_{11}$ : We come back to the situation we were facing at time  $t_9$ , and therefore ABT\_DO-Retro-MinDom may fall in an infinite loop when using method  $m_1$ .

## 4 The Right Way to Compare Orders

Let us formally define the second method,  $m_2$ , for comparing orders in which we compare the indexes of agents as soon as the counters in a position are equal on both counter vectors associated with the orders being compared. Given any order  $o$ , we denote by  $o(i)$  the index of the agent located in the  $i$ th position in  $o$  and by  $s(i)$  the counter in the  $i$ th position in the counter vector  $s$ . An order  $o_1$  with counter vector  $s_1$  is more up-to-date than an order  $o_2$  with counter vector  $s_2$  if and only if there exists a position  $i$ ,  $1 \leq i \leq n$ , such that for all  $1 \leq j < i$ ,  $s_1(j) = s_2(j)$  and  $o_1(j) = o_2(j)$ , and  $s_1(i) > s_2(i)$  or  $s_1(i) = s_2(i)$  and  $o_1(i) < o_2(i)$ .

In our correctness proof for the use of  $m_2$  in ABT\_DO-Retro we use the following notations: When an agent proposes a new order where the position of the highest priority agent has been changed, the new order will be denoted by a capital  $O$ . The initial order known by all agents is denoted by  $O_0$ . Each agent stores a current order with an associated counter vector. Each counter vector consists of  $n$  counters  $ct_1, \dots, ct_n$ . We denote by  $ct_1(o)$  the value of the first counter of the counter vector associated with  $o$ . In a similar way, we denote by  $ct_1(A_i)$  the value of the first counter in the counter vector stored by the agent  $A_i$ . We define  $ct_{max}$  to be equal to  $\max(ct_1(A_i) \mid i \in 1..n)$ . The value of  $ct_{max}$  evolves during the search so that it always corresponds to the value of the largest counter among all the first counters stored by agents.

Let  $K$  be the parameter defining the degree of flexibility of the retroactive heuristics (see Section 1). Next we show that the ABT\_DO-Retro algorithm is correct when using  $m_2$  and with  $K = 0$ . The proof that the algorithm is correct when  $K \neq 0$  can be found in [ZZA09].

To prove the correctness of ABT\_DO-Retro we use induction on the number of agents. For a single agent the order is static therefore the correctness of standard ABT implies the correctness of ABT\_DO-Retro. Assume ABT\_DO-Retro is correct for every DisCSP with  $n - 1$  agents. We show in the following that ABT\_DO-Retro is correct for every DisCSP with  $n$  agents. To this end we first prove the following lemmas.

**Lemma 1** *Given enough time, if the value of  $ct_{max}$  does not change, the highest priority agent in all orders stored by all agents will be the same.*

**Proof.** Assume the system reaches a state  $\sigma$  where the value of  $ct_{max}$  no longer increases. Let  $h$  be the value of  $ct_{max}$ . Let  $O_1$  be the order that, when generated, caused the system to enter state  $\sigma$ . Inevitably, we have  $ct_1(O_1) = h$ . Assume that  $O_1 \neq O_0$  and let  $A_i$  be the agent that generated  $O_1$ . The agent  $A_i$  is necessarily the highest priority agent in the new order  $O_1$  because, the only possibility for the generator of a new order to change the position of the highest priority agent is to put itself in the first position in the new order. Thus,  $O_1$  is sent by  $A_i$  to all other agents because  $A_i$  must send  $O_1$  to all agents that have a lower priority than itself. So after a finite time all agents will be aware

of  $O_1$ . This is also true if  $O_1 = O_0$ . Now, by assumption the value of  $ct_{max}$  no longer increases. As a result, the only way for another agent to generate an order  $O'$  such that the highest priority agents in  $O_1$  and  $O'$  are different (i.e.,  $O'(1) \neq O_1(1)$ ) is to put itself in first position in  $O'$  and to do that *before* it has received  $O_1$  (otherwise  $O'$  would increase  $ct_{max}$ ). Therefore, the time passed from the moment the system entered state  $\sigma$  until a new order  $O'$  was generated is finite. Let  $O_2$  be the most up-to-date such order and let  $A_j$  be the agent that generated  $O_2$ . That is,  $A_j$  is the agent with smallest index among those who generated such an order  $O'$ . The agent  $A_j$  will send  $O_2$  to all other agents and  $O_2$  will be accepted by all other agents after a finite amount of time. Once an agent has accepted  $O_2$ , all orders that may be generated by this agent do not reorder the highest priority agent otherwise  $ct_{max}$  would increase.  $\square$

**Lemma 2** *If the algorithm is correct for  $n - 1$  agents then it terminates for  $n$  agents.*

**Proof.** If during the search  $ct_{max}$  continues to increase, this means that some of the agents continue to send new orders in which they put themselves in first position. Hence, the nogoods they generate when proposing the new orders are necessarily unary (i.e., they have an empty left-hand side) because in ABT.DO-Retro, when the parameter  $K$  is zero the nogood sender cannot put itself in a higher priority position than the second last in the nogood. Suppose  $ng_0 = \neg(x_i = v_i)$  is one of these nogoods, sent by an agent  $A_j$ . After a finite amount of time, agent  $A_i$ , the owner of  $x_i$ , will receive  $ng_0$ . Three cases can occur. First case,  $A_i$  still has value  $v_i$  in its domain. So the value  $v_i$  is pruned once and for all from  $D(x_i)$  thanks to  $ng_0$ . Second case,  $A_i$  has already received a nogood equivalent to  $ng_0$  from another agent. Here,  $v_i$  no longer belongs to  $D(x_i)$ . When changing its value,  $A_i$  has sent an **ok?** message with its new value  $v'_i$ . If  $A_i$  and  $A_j$  were neighbors, this **ok?** message has been sent to  $A_j$ . If  $A_i$  and  $A_j$  were not neighbors when  $A_i$  changed its value to  $v'_i$ , this **ok?** message was sent by  $A_i$  to  $A_j$  after  $A_j$  requested to add a link between them at the moment it generated  $ng_0$ . Thanks to the assumption that messages are always delivered in a finite amount of time, we know that  $A_j$  will receive the **ok?** message containing  $v'_i$  a finite amount of time after it sent  $ng_0$ . Thus,  $A_j$  will not be able to send forever nogoods about a value  $v_i$  pruned from  $D(x_i)$ . Third case,  $A_i$  already stores a nogood with a non empty left-hand side discarding  $v_i$ . Notice that although  $A_j$  moves to the highest priority position,  $A_i$  may be of lower priority, i.e., there can be agents with higher priority than  $A_i$  according to the current order that are not included in  $ng_0$ . Thanks to the standard *highest possible lowest variable involved* [HY00, BMBM05] heuristic for selecting nogoods in ABT algorithms, we are guaranteed that the nogood with empty left-hand side  $ng_0$  will replace the other existing nogood and  $v_i$  will be permanently pruned from  $D(x_i)$ . Thus, in all three cases, every time  $ct_{max}$  increases, we know that an agent has moved to the first position in the order, and a value was definitively pruned a finite amount of time before or after. There is a bounded number of values in the network. Thus,  $ct_{max}$  cannot increase forever. Now, if  $ct_{max}$  stops increasing, then after a finite amount of time the highest priority agent in all orders stored by all agents will be the same (Lemma 1). Since the algorithm is correct for  $n - 1$  agents, after each assignment of the highest priority agent, the rest of the agents will either reach an idle state,<sup>1</sup> generate an empty nogood indicating that there is no solution, or generate a unary nogood, which is sent to the highest priority agent. Since the number of values in the system is finite, the third option, which is the only one that does not imply immediate termination, cannot occur forever.  $\square$

**Lemma 3** *If the algorithm is correct for  $n - 1$  agents then it is sound for  $n$  agents.*

<sup>1</sup>As proved in Lemma 3, this indicates that a solution was found.

**Proof.** Let  $o$  be the most up-to-date order generated before reaching the state of quiescence and let  $O$  be the most-up-to-date order generated such that  $ct_1(O) = ct_1(o)$  (and such that  $O$  has changed the position of the first agent –assuming  $O \neq O_0$ ). Given the rules for reordering agents, the agent that generated  $O$  has necessarily put himself first because it has modified  $ct_1$  and thus also the position of the highest agent. So it has sent  $O$  to all other agents. When reaching the state of quiescence, we know that no order  $O_2$  with  $O_2(1) \neq O(1)$  has been generated because this would break the assumption that  $O$  is the most-up-to-date order where the position of the first agent has been changed. Hence, at the state of quiescence, every agent  $A_i$  stores an order  $o_i$  such that  $o_i(1) = O(1)$ . (This is also true if  $O = O_0$ .) Let us consider the DisCSP  $P$  composed of the  $n - 1$  lower priority agents according to  $O$ . Since the algorithm is correct for  $n - 1$  agents, the state of quiescence means that a solution was found for  $P$ . Also, since all agents in  $P$  are aware that  $O(1)$  is the agent with the highest priority, the state of quiescence also implies that all constraints that involve  $O(1)$  have been successfully tested by agents in  $P$ , otherwise at least one agent in  $P$  would try to change its value and send an **ok?** or nogood message. Therefore, the state of quiescence implies that a solution was found.  $\square$

**Lemma 4** *The algorithm is complete*

**Proof.** All nogoods are generated by logical inferences from existing constraints. Thus, an empty nogood cannot be inferred if a solution exists.  $\square$

Following Lemmas 2, 3 and 4 we obtain the correctness of the main theorem in this corrigendum.

**Theorem 1** *The ABT\_DO-Retro algorithm with  $K = 0$  is correct when using the  $m_2$  method for selecting the most up-to-date order.*

## References

- [BMBM05] C. Bessiere, A. Maestre, I. Brito, and P. Meseguer. Asynchronous backtracking without adding links: a new member in the ABT family. *Artificial Intelligence*, 161:1-2:7–24, 2005.
- [HY00] K. Hirayama and M. Yokoo. The effect of nogood learning in distributed constraint satisfaction. In *Proceedings of the The 20th International Conference on Distributed Computing Systems*, ICDCS'00, pages 169–177, Washington, DC, USA, 2000. IEEE Computer Society.
- [ZM06] R. Zivan and A. Meisels. Dynamic ordering for asynchronous backtracking on DisCSPs. *Constraints*, 11(2-3):179–197, 2006.
- [ZZA09] R. Zivan, M. Zazone, and A. Meisels. Min-domain retroactive ordering for asynchronous backtracking. *Constraints*, 14(2):177–198, 2009.