Asynchronous Forward Bounding Revisited

Mohamed Wahbi, Redouane Ezzahir, Christian Bessière

Asynchronous Forward Bounding Revisited

Mohamed Wahbi[1], Redouane Ezzahir[2], Christian Bessiere[3]
mohamed.wahbi@emn.fr, red.ezzahir@gmail.com,
bessiere@lirmm.fr

[1] TASC (INRIA/CNRS), Mines Nantes, France
[2] ENSA Agadir, University Ibn Zohr, Morroco
[3] University of Montpellier, France

Abstract. The Distributed Constraint Optimization Problem (DCOP) is a powerful framework for modeling and solving applications in multi-agent coordination. Asynchronous Forward Bounding (AFB_BJ) is one of the best algorithms to solve DCOPs. We propose AFB_BJ$^+$, a revisited version of AFB_BJ in which we refine the lower bound computations. We also propose to compute lower bounds for the whole domain of the last assigned agent instead of only doing this for its current assignment. This reduces both the number of messages needed and the time future agents remain idle. In addition, these lower bounds can be used as a value ordering heuristic in AFB_BJ$^+$. The experimental evaluation on standard benchmark problems shows the efficiency of AFB_BJ$^+$ compared to other algorithms for DCOPs.

1 Introduction

Distributed Constraint Optimization Problem (DCOP) is a powerful framework to model a wide range of applications in multi-agent coordination such as distributed scheduling [14], distributed planning [4], distributed resource allocation [17], target tracking in sensor networks [15] distributed vehicle routing [12], etc. A DCOP consists of a group of autonomous agents, where each agent has an independent computing power. Each agent owns a local constraint network. Variables owned by different agents are connected by constraints. These constraints specify a non-negative constraint cost for combinations of values assigned to the variables they connect. In general, constraints or value assignments may be strategic information or private choice that should not be revealed or delegated to other agents. Thus, each agent only has control on its variables and only knows constraints that involve them. DCOP addresses problems in which agents must, *in a distributed manner*, assign values to their variables such that the sum of the constraint costs of all constraints is minimized.

Several complete algorithms for solving DCOPs have been proposed in the last decade. The pioneer complete asynchronous algorithm is Adopt [15]. Later on, the closely related BnB-Adopt [20] was presented. BnB-Adopt changes the nature of the search from Adopt best-first search to a depth-first branch-and-bound strategy, obtaining better performance. Gutierrez and Meseguer show that some of the messages exchanged by Adopt and BnB-Adopt turned out to be redundant [9]. By removing

these redundant messages they obtain more efficient algorithms: Adopt$^+$ and BnB-Adopt$^+$. The algorithms mentioned so far perform assignments concurrently and asynchronously. Thereby, the perception of agents on the variable assignments of other agents is in general inconsistent.

Another category of algorithms for solving DCOPs is that of algorithms performing assignments sequentially and synchronously. The synchronous branch and bound (SyncBB) [10] is the basic systematic search algorithm in this category. In SyncBB, only the agent holding the token is allowed to perform an assignment while the other agents remain idle. Once it assigns its variables, it passes on the token and then remains idle. Thus, SyncBB does not make any use of concurrent computation. No-Commitment Branch and Bound (NCBB) is another synchronous polynomial-space search algorithm for solving DCOPs [5]. To capture independent sub-problems, NCBB arranges agents in constraint tree ordering. NCBB incorporates, in a synchronous search, a concurrent computation of lower bounds in non-intersecting areas of the search space based on the constraint tree structure.

Another attempt to incorporate a concurrent computation in a synchronous search was applied in Asynchronous Forward Bounding (AFB) [6]. AFB can be seen as an improvement of SyncBB where agents extend a partial assignment as long as the lower bound on its cost does not exceed the global upper bound (i.e., the cost of the best solution found so far). In AFB, the lower bounds are computed concurrently by unassigned agents. Thus, each synchronous extension of the current partial assignment is followed by an asynchronous forward bounding phase. Forward bounding propagates the bounds on the cost of the partial assignment by sending to all unassigned agents copies of the extended partial assignment. When the lower bound of all assignments of an agent exceeds the upper bound, it performs a simple backtrack to the previous assigned agent. Later, the AFB has been enhanced by the addition of a backjumping mechanism, resulting in the AFB_BJ algorithm [7]. The authors report that AFB_BJ, especially combined with the minimal local cost value ordering heuristic performs significantly better than other DCOP algorithms.

In this paper, we propose AFB_BJ$^+$, a revisited version of AFB_BJ in which we refine the lower bound computations. We also propose to compute lower bounds for the whole domain of the last assigned agent instead of only doing this for its current assignment. Thus, an unassigned agent computes the lower bound for each value in the domain of the agent requesting it. This reduces both the number of messages needed and the time future agents remain idle. Hence, we take all possible advantage from the asynchronicity of the system. In addition, these lower bounds are used as a value ordering heuristic in AFB_BJ$^+$. Thus, an agent assigns first values with minimal lower bound.

This paper is structured as follows. Section 2 gives the necessary background on DCOP and a short description of the AFB algorithms. We present the AFB_BJ$^+$ algorithm in Section 3. Correctness proofs are given in Section 4. We report experimental results in Section 5. Finally, we conclude in Section 6.

2 Background

2.1 Basic definitions and notations

The *Distributed Constraint Optimization Problem* (DCOP) has been formalized in [15] as a quadruple $(\mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{A}$ is a set of p agents $\{A_1, \ldots, A_p\}$, $\mathcal{X}$ is a set of n variables $\{x_1, \ldots, x_n\}$, where each variable x_j is controlled by one agent in $\mathcal{A}$. $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of n domains, where D_j is the set of possible values to which variable x_j may be assigned. Only the agent controlling a variable can assign a value to it and has knowledge of its domain. $\mathcal{C} = \{c_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+\}$ is a set of binary utility constraints (i.e., soft constraints). Each utility constraint $c_{ij} \in \mathcal{C}$ is defined over the pair of variables $\{x_i, x_j\} \subseteq \mathcal{X}$. We say that x_i and x_j are *neighbors*.

For simplicity purposes, we consider a restricted version of DCOP where each agent holds exactly one variable ($p = n$). Thus, we use the terms agent (A_j) and variable (x_j) interchangeably and we identify the agent ID with its variable index (j). Furthermore, all agents store a unique total order $\prec$ on agents. Agents appearing before an agent $A_j \in \mathcal{A}$ in the total order are the higher priority agents and those appearing after A_j are the lower priority agents. The order $\prec$ divides the set $\Gamma(x_j)$ of neighbors of A_j into higher priority neighbors $\Gamma^{-}(x_j)$, and lower priority neighbors $\Gamma^{+}(x_j)$. For sake of clarity, we assume that the total order is the lexicographic ordering $[A_1, A_2, \ldots, A_n]$. In the rest of the paper, we consider a generic agent $A_j \in \mathcal{A}$. Thus, j is the level of agent A_j.

An *assignment* for agent A_j is a tuple (x_j, v_j), where v_j is a value from D_j. A_j maintains a counter t_j and increments it whenever it changes its value. The value of the counter *tags* each generated assignment. When comparing two assignments for the same agent, the most up to date is the one with the greatest tag. A *current partial assignment* (CPA) is an ordered set of assignments, e.g., $Y = [(x_1, v_1), \ldots, (x_j, v_j)]$ s.t. $x_1 \prec \ldots \prec x_j$. The set of all variables assigned in Y is denoted by $var(Y) = \{x_1, \ldots, x_j\}$. A *time-stamp* associated to a CPA Y is an ordered list of counters $[t_1, t_2, \ldots, t_j]$ where t_i is the tag of the variable x_i s.t. $x_i \in var(Y)$ [16,19]. When comparing two CPAs, the *strongest* one is that associated with the lexicographically greater time-stamp. That is, the CPA with greatest value on the first counter on which they differ, if any, otherwise the longest one. Let $Y = [(x_1, v_1), \ldots, (x_i, v_i), \ldots, (x_j, v_j)]$ be a CPA, the subset of Y including all variables down to x_i is denoted by $Y^i = [(x_1, v_1), \ldots, (x_i, v_i)]$.

The guaranteed cost of a CPA Y, denoted by $gc(Y)$, is the sum of all utility constraints c_{ij} s.t. x_i and x_j are assigned in Y (Eq. 1).

$$gc(Y) = \sum_{c_{ij} \in \mathcal{C}} c_{ij}(v_i, v_j) \mid (x_i, v_i), (x_j, v_j) \in Y. \quad (1)$$

A full assignment Y is a CPA that involves all variables of the problem, i.e., $var(Y) = \mathcal{X}$. The goal of a DCOP solver is to distributively find a full assignment Y^* with minimal cost, that is, $Y^* = \arg \min_{Y} \{gc(Y) \mid var(Y) = \mathcal{X}\}$.

In the following, we will present standard AFB algorithms. We will use a nomenclature of messages and data structures different from those used in the original paper [7] in order to be closer to those used in our approach.

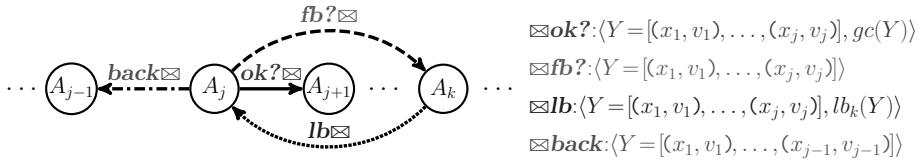


Fig. 1: The messages exchanged by the AFB algorithm.

2.2 Asynchronous Forward Bounding (AFB) algorithm

In Asynchronous Forward Bounding (AFB) [6], agents assign their variables sequentially and unassigned agents asynchronously try to compute lower bounds on the CPA, say Y. Agents perform assignments of their variables only when they hold the current partial assignments Y (i.e., Y is the token). Each extension of the CPA Y, is followed by a Forward Bounding (FB) phase. The FB phase is performed by sending forward copies of Y to all unassigned agents. In the FB phase, it is required from unassigned agents to compute a lower bound on the cost increment caused by an assignment of their variables on Y. Once computed, the lower bounds are sent back to the agent that sent the request, i.e., the last assigned agent in Y. Due to the asynchronous nature of the FB phase, multiple CPAs may be present at a given moment in time. However, the time-stamp mechanism is used by agents to discard obsolete ones.

The lower bounds collected from unassigned agents are used to compute a lower bound on the CPA. When the computed lower bound becomes larger than the current upper bound (i.e., the cost of the best full CPA found so far), the CPA is pruned. Concretely, whenever agent A_j receives a valid lower bound from an unassigned agent, it adds it to that received from other agents and checks if the cumulative lower bound exceeds the upper bound. In such a case, A_j tries to assign an alternative value to its variable. If such value is not available, it needs to backtrack. When agent A_j takes the decision to backtrack, it sends the CPA (Y^{j-1}) backwards to the last agent assigned on it (i.e., A_{j-1}). However, if A_j is the first agent in the ordering, it ends the search process after claiming this to other agents. The AFB algorithm then reports that the optimal solution is the best full CPA found so far.

Fig. 1 shows the messages exchanged by the AFB algorithm.[4] AFB agents exchange the following types of messages:

***ok?*:** a message which contains the CPA Y with its cost $gc(Y)$. When A_j assigns its variable, it sends this message to the next agent in the ordering (A_{j+1}).

***back*:** a message which contains an inconsistent CPA. It is sent back to agent A_{j-1} requiring it to change its assignment.

fb?*:** a message which contains a copy of an ***ok? message. It is sent by A_j to unassigned agents to compute a lower bound on the CPA it carries.

lb*:** a message which contains a lower bound on the current partial assignment. It is sent as response to a ***fb? message.

[4] The names of the message types here are closer to that used in our approach and then different from that used in the original AFB paper [6].

The computation of lower bounds on AFB is performed as follows. In a preprocessing step, A_j computes the minimal future cost estimation for each possible value in its domain incurred by every lower priority agent A_k (Eq. 2). $fc_j(v_j)$ is a lower bound on the cost of constraints involving the assignment (x_j, v_j) and all its lower priority neighbors. Agents compute these estimations only once and store them.

$$fc_j(v_j) = \sum_{x_k \in \Gamma^+(x_j)} \min_{v_k \in D_k} \{c_{jk}(v_j, v_k)\}. \tag{2}$$

Given a current partial assignment Y^i, and an unassigned agent A_j, the local cost of assigning a value v_j to A_j is the sum of the constraints costs of this value with all assignments in Y^i s.t. $i < j$ (Eq. 3).

$$lc_j(Y^i, v_j) = \sum_{(x_h, v_h) \in Y^i \ s.t. \ h \leq i < j} c_{hj}(v_h, v_j) \tag{3}$$

Summing the local cost of an assignment (x_j, v_j) and the minimal future cost incurred by lower priority neighbors provides a future cost of an eventual extension of Y^i with (x_j, v_j). The lower bound of Y^i on a unassigned agent (A_j) is the minimal future cost over all its values (Eq. 4). Thus, whenever a higher priority agent A_i requires from A_j to compute a lower bound on a CPA Y^i, it responds by sending back the minimal lower bound of Y^i over all values in its domain, i.e., $lb_j(Y^i)$.

$$lb_j(Y^i) = \min_{v_j \in D_j} \{lc_j(Y^i, v_j) + fc_j(v_j)\}. \tag{4}$$

By collecting lower bounds from lower priority agents, agent A_j can compute a lower bound on its CPA Y^j. The lower bounds on a CPA Y^j reported by lower priority agents are accumulated and summed up with the guaranteed cost of Y^j to provide a lower bound on the cost of a complete extension of Y^j (Eq. 5).

$$lb(Y^j) = gc(Y^j) + \sum_{A_k \succ A_j} lb_k(Y^j) \tag{5}$$

If the computed lower bound $lb(Y^j)$ exceeds the current known upper bound (UB_j), A_j needs to change its current value v_j on Y^j by a new value v'_j generating a stronger CPA. Then, search continues with the generated CPA. If A_j has already tested all possible values for its variable, it backtracks, asking the previous agent in the ordering (A_{j-1}) to assign a new value to its variable through a ***back*** message.

2.3 Asynchronous Forward Bounding with CBJ (AFB_BJ)

The Asynchronous Forward Bounding with backjumping (AFB_BJ) was obtained by adding a backjumping mechanism to standard AFB [7]. When the lower bounds of all values exceed the upper bound, instead of backtracking to the most recently assigned variable, AFB_BJ tries to jump to the last assigned agent such that its re-assignment could possibly lead to a solution. To this end, agents in AFB_BJ use some maintained data structures that we introduce in the following. Another feature of AFB_BJ is that the agent that performs an assignment, uses the minimal local cost (Eq. 3) as value ordering heuristic.

When an agent A_j assigns its variable, it sends an ***ok?*** message to the next agent in the ordering. In AFB_BJ, the ***ok?*** message contains the current partial assignment Y^j and an array of guaranteed costs, one for each level. $gc(Y^j)[j]$ is the cost of $Y^j = [(x_1, v_1), \ldots, (x_j, v_j)]$ where $gc(Y^j)[0] = 0$. For each $i \in 1..j-1$, $gc(Y^j)[i]$ equals that received from A_{j-1}, i.e., $gc(Y^{j-1})[i]$.

$$gc(Y^j)[j] = gc(Y^j) = gc(Y^{j-1}) + lc_j(Y^{j-1}, v_j) \tag{6}$$

In order to perform backjumping, AFB_BJ agents compute a lower bound for each level on the CPA. Concretely, instead of computing the lower bound for the whole received CPA Y^i, A_j computes it for each Y^h where $h \leq i < j$. To this end, A_j first computes the local cost for each level h (Eq. 7).

$$lc_j(Y^i, v_j)[h] = lc_j(Y^h, v_j) \tag{7}$$

The lower bound at level h (Eq. 8) is then the minimal lower bound of Y^h over all values in D_j. When a higher agent A_i requests from A_j to compute its lower bound on a CPA Y^i, it answers by sending an array of lower bounds, one for each level h where $h \leq i < j$.

$$lb_j(Y^i)[h] = \min_{v_j \in D_j} \{lc_j(Y^i, v_j)[h] + fc_j(v_j)\}. \tag{8}$$

When A_j successfully assigns a value v_j to its variable, it sends forward copies of the extended CPA, Y^j, to each unassigned agent A_k and awaits for receiving from them the array of lower bounds. The lower bounds denoted by $lb_k(Y^j)$, is an array in which the i^{th} element ($1 \leq i \leq j$) contains a lower bound on the cost of assigning a value to A_k with respect to the assignment on Y^i (Eq. 8). Once A_j receives lower bounds arrays, it computes a lower bound on the cost of any full assignment (Eqs. 9).

$$lb(Y^j)[i] = gc(Y^j)[i] + \sum_{A_k \succ A_j} lb_k(Y^j)[i]. \tag{9}$$

These lower bounds are used by the AFB_BJ to determinate the backjumping level. For more details about the way in which the level of backjumping is calculated we refer the reader to [7].

3 Asynchronous Forward Bounding revisited

AFB_BJ$^+$ is a revisited version of AFB_BJ in which we propose a refinement of the lower bound computations. We also propose to compute lower bounds for the whole domain of the last assigned agent instead of only doing this for its current assignment. Thus, an unassigned agent computes the lower bound for each value in the domain of the agent requesting it. In addition, these lower bounds are used as a value ordering heuristic.

3.1 Lower bound refinement

When an agent A_i successfully assigns a value v_i to its variable, it sends forward copies of the extended CPA, Y^i, to each unassigned agent and awaits for receiving from them

the array of lower bounds. When agent A_j receives this CPA Y^i (through a ***fb?*** message), it computes the lower bound for each level h where $h \leq i < j$ (Eq. 8). When computing the lower bound of level h only assignments on Y^h are considered. We also add the cost of assigning v_i to x_i (i.e., (x_i, v_i)) to this lower bound. Moreover, we also add the minimal cost of constraints with variables (x_m) between x_h and x_i. Thus, instead of being a lower bound on a possible extension of Y^h by a possible assignment of A_j, it will be a lower bound on a possible extension of Y^h by both A_i and A_j and agents between x_h and x_i. Hence, we revise Eq. 8 to get Eq. 10 where the first and the last terms remain as in the original equation.

$$lb_j(Y^i)[h] = \min_{v_j \in D_j} \Big\{ lc_j(Y^i, v_j)[h] + \sum_{m=h+1}^{i-1} \min_{v_m \in D_m} \{c_{mj}(v_m, v_j)\} + c_{ij}(v_i, v_j) + fc_j(v_j) \Big\} \tag{10}$$

At first glance, it seems that this will require more computational effort from unassigned agents, however it is not the case. One can simply compute the array of lower bounds, as is already done in AFB_BJ, and at the end it adds to each level the cost with variable x_i. We obtain the addition of the third term, i.e., $c_{ij}(v_i, v_j)$. To get the quantity to be added by the second term (i.e., $\sum_{m=h+1}^{i-1} \min_{v_m \in D_m} \{c_{mj}(v_m, v_j)\}$), we use the same principle used in Eq. 2. Agents compute for each value the estimations of each level only once and store them.

The refinement of the lower bounds computation allows agents to get more accurate lower bounds on their assignments. Thus, the accumulated lower bound at each level is increased. This mechanism allows earlier detection of CPAs with lower bound larger than the upper bound. In addition, by doing this, the ***back*** message will be sent as high as possible in the agent ordering, thus saving unnecessary search effort.

3.2 Lower bounds for the whole domain

In AFB_BJ, the forward bounding phase is very expensive in term of communication load. FB requires for each value in D_j, $2 \times (n - j)$ messages (a ***fb?*** and a ***lb*** message for each lower agent). Thus, FB needs, for each CPA Y^{j-1}, $2 \times |D_j| \times (n-j)$ messages.

In AFB_BJ$^+$, we propose to compute lower bounds for the whole domain of the last assigned agent instead of only computing this for its current assignment. When an agent receives a ***fb?*** message it answers by sending back a two-dimensional array, an array for each value in the domain of the receiver agent. Hence, the forward bounding phase will need, for each CPA Y^{j-1}, only $2(n - j)$ messages, 2 messages for each lower agent. When agent A_j receives a ***fb?*** from agent A_i, instead of computing $lb_j(Y^i)[h]$ only for the current value of x_i, A_j computes it for each value v_i in D_i, $lb_j(Y^{i-1})[h][v_i]$ using Eq. 11.[5]

$$\forall h \in 1..i\text{-}1,\ \forall v_i \in D_i,\ lb_j(Y^{i-1})[h][v_i] = lb_j(Y^{i-1} \cup (x_i, v_i))[h] \tag{11}$$

[5] If x_i and x_j are not neighbors, a simple array is sufficient since the lower bound is the same for all values in D_i. Moreover, x_j does not known D_i.

3.3 Avoiding redundancy

Another feature of the AFB_BJ$^+$ algorithm is that agents retain and maintain the received lower bounds to avoid redundant messages. When an agent A_j receives a ***fb?*** message from agent A_i, instead of clearing all information it stores (namely the collected lower bounds and the computed ones with their local costs), it clears only irrelevant information w.r.t the received CPA Y^i. Concretely, A_j compares the time-stamp of the received CPA with its CPA. If its CPA is stronger than the received one, the message is discarded. Otherwise, A_j gets the index $h \leq i$ of the first counter on which they differ. All local costs and lower bounds on the current partial assignment Y^{h-1} remain valid. Thus, agent A_j will not re-compute lower bounds for this part.

The same thing is done for ***ok?*** messages. Whenever agent A_j receives a CPA Y^{j-1}, it updates all stored information by only removing parts that are not compatible with Y^{j-1}. When A_j succeeds in assigning its variable, it sends forward copy of the extended CPA Y^j in ***fb?*** messages to its lower agents. However, some of these messages are redundant. To avoid this, each agent A_j stores, for each lower priority agent A_k, the agent which is the closest to A_j in the neighbors of A_k higher than A_j. As long as the assignment of such agent or agents higher than him were not updated, there is no need to send ***fb?*** message to A_k. Thus, redundant messages and computations are saved. Moreover, the agent assigning its variable has more accurate lower bounds for all values in its domain. As long as the new complete array of lower bounds has not yet been received, the remaining valid part can be used as a lower bound estimation for each value in the current domain using Eq. 12. h is the lowest valid level for lower bound received from A_k.

$$lb(Y^j) = gc(Y^j) + \sum_{k \succ j} lb_k(Y^{j-1})[h][v_j] \;\; s.t. \;\; (x_j, v_j) \in Y^j \tag{12}$$

3.4 Promising value ordering heuristic

Unlike AFB_BJ that uses minimal local cost as value ordering heuristic, an AFB_BJ$^+$ agent uses a different strategy for reordering values in its current domain. All computations performed so far by unassigned agents to calculate lower bounds are used to reorder values in the current domain. Thus, when receiving an ***ok?*** message, A_j computes the lower bounds for all values in its domain using Eq. 12. A_j chooses to assign first values with minimal lower bound. Then, instead of considering only costs with past variable, both costs with past variables and estimations of costs on future variables are considered. We mimic an informed memory-bounded version of A^*, instead of simulating an uninformed memory-bounded version of A^*.

3.5 AFB_BJ$^+$ description

Fig. 2 presents the pseudo-code of AFB_BJ$^+$ executed by every agent A_j. Agent A_j maintains a variable UB_j that stores the current upper-bound (the cost of the best solution found so far) initialized to $+\infty$, v_j^* that stores the value of A_j on the solution, Y that stores the strongest received CPA, GC an array of size $j-1$ that stores the guaranteed costs where $GC[i] = gc(Y^i)$, and $lb_k(Y)[]$ that stores the lower bounds received

procedure AFB-BJ$^+$ ()

01. $UB_j \leftarrow +\infty$; $v_j^* \leftarrow empty$; $Y \leftarrow \{\}$; $GC[1..j-1] \leftarrow [0, \ldots, 0]$;
02. $mustSendFB \leftarrow True$;
03. **foreach (** $A_k \succ A_j$ **) do**
04. **foreach (** $v_j \succ D_j$ **) do** $lb_k(Y)[0][v_j] \leftarrow \min_{v_k \in D_k} \{c_{jk}(v_j, v_k)\}$;
05. **if (** $A_j = A_1$ **) then** ExtendCPA() ;
06. **while (** $\neg end$ **) do**
07. $msg \leftarrow$ getMsg() ;
08. **if (** $msg.UB < UB_j$ **) then** $UB_j \leftarrow msg.UB$; $v_j^* \leftarrow v_j$;
09. **if (** $msg.Y$ is stronger than Y **) then**
10. $Y \leftarrow msg.Y$; $GC \leftarrow msg.GC$;
11. clear irrelevant lower bounds ;
12. **switch (** $msg.type$ **) do**
13. ***ok?*** : $mustSendFB \leftarrow true$; ExtendCPA() ;
14. ***back*** : $Y \leftarrow Y^{j-1}$; ExtendCPA() ;
15. ***fb?*** : sendMsg: ***lb***$\langle lb_j(Y^i)[],\ msg.Y\rangle$ **to** A_i ; /* A_i is msg sender */
16. ***lb*** : ProcessLB(msg) ;
17. ***stp*** : $end \leftarrow true$;

procedure ExtendCPA()

18. $v_j \leftarrow \arg \min_{v'_j \in D_j} \{lb(Y \cup (x_j, v'_j))\}$; /* Eq. 12 */
19. **if (** $lb(Y \cup (x_i, v_i)) \geq UB_j$ **) then** Backtrack() ;
20. **else**
21. $Y \leftarrow \{Y \cup (x_j, v_j)\}$; $t_j \leftarrow t_j + 1$;
22. **if (** $var(Y) = \mathcal{X}$ **) then**
23. $UB_j \leftarrow gc(Y)$; /* $A_j = A_n$ */
24. $v_j^* \leftarrow v_j$;
25. $Y \leftarrow Y^{j-1}$;
26. ExtendCPA() ;
27. **else**
28. sendMsg: ***ok?***$\langle Y,\ GC,\ UB_j\rangle$ **to** A_{j+1} ;
29. **if (** $mustSendFB$ **) then**
30. **foreach (** $A_k \succ A_j$ **) do** sendMsg: ***fb?***$\langle Y,\ GC,\ UB_j\rangle$ **to** A_k ;
31. $mustSendFB \leftarrow false$;

procedure Backtrack()

32. **for (** $i \leftarrow j$-1 **dowTo** 1 **) do**
33. **if (** $lb(Y)[i-1] < UB_j$ **) then**
34. sendMsg: ***back***$\langle Y^i,\ UB_j\rangle$ **to** A_i ; **return**;
35. broadcastMsg: ***stp***$\langle UB_j\rangle$;
36. $end \leftarrow true$;

procedure ProcessLB(msg)

37. $lb_k(Y^j) \leftarrow msg.lb$; /* A_k is the sender of msg */
38. **if (** $lb(Y^j) \geq UB_j$ **) then** ExtendCPA() ;

Fig. 2: The AFB_BJ$^+$ algorithm running on agent A_j.

from a lower agent A_k. Since $lb_k(Y)[0][v_j]$ depends only on the assignments of x_j and x_k, it is initialized to $\min_{v_k \in D_k} \{c_{jk}(v_j, v_k)\}$. Thus, it is a valid lower bound for all CPAs that contains (x_j, v_j). Eq. 2 is obtained by summing $lb_k(Y)[0][v_j]$ for each lower agent A_k.

AFB_BJ$^+$ starts by initializing the local data structures of A_j (lines 1-4). A_j then enters in the waiting and processing message loop (line 6). Each received message holds a CPA $msg.Y$ and its corresponding guaranteed costs $msg.GC$. Due to the asynchronous nature of the algorithm, some messages may be obsolete. A_j uses the time-stamping mechanism to discard those messages (line 9). If the received CPA ($msg.Y$) is stronger than Y, A_j updates Y and GC by the received ones (line 10). Then, A_j clears all irrelevant lower bounds computed or received so far (line 11). Agent A_j attaches to each message it sends its UB_j. The upper bound UB_j and v_j^* are updated when a received message carries a new upper bound smaller than the stored one (line 8).

Upon receiving an ***ok?*** message, A_j marks that it must send ***fb?*** messages by setting $mustSendFB$ to $true$. Next, it attempts to extend the received CPA by calling procedure `ExtendCPA()` (line 13).

When calling `ExtendCPA()`, A_j tries to find a value with the minimum lower bound (Eq. 12) without exceeding UB_j (lines 18-19). If such value does not exist, A_j backtracks (`Backtrack()` call, line 19). Otherwise, A_j extends the CPA by adding its new assignment and increments its counter t_j. If the resulting CPA includes assignments of all agents (line 22), a solution is found and then the upper bound is updated. Instead of broadcasting the new solution and its associated upper-bound, A_j calls `ExtendCPA()` to continue the search (line 26). Since UB_j is always attached to the exchanged messages, other agents will be informed of this new upper bound when continuing the search. At the end of the search, the best assignment of A_j is stored in v_j^*. If A_j is not the last agent on the ordering, it sends the extended CPA to the next agent (line 28). Afterwards, A_j sends ***fb?*** messages to all lower priority agents (lines 30-31).[6]

When A_j receives a ***fb?*** message, it computes for each value from the domain of the sender a lower bound on the cost increment caused by adding an assignment to its variable using Eq. 10. These lower bounds are sent back to the agent who sent the ***fb?*** message through a ***lb*** message.

When A_j receives a valid ***lb*** message, it saves the attached lower bounds (line 37). It checks if this new information causes the current partial assignment to exceed the upper-bound. In such a case, A_j calls `ExtendCPA()` in order to change its assignment (line 38).

Agent A_j calls procedure `Backtrack()` whenever the lower bounds of all its values exceed the upper-bound. When this occurs, A_j computes to which agent the CPA Y should be sent to (the backtracking target). A_j goes over all candidates, from $j-1$ down to 1, looking for the first agent it finds that its reassignment could lead to a full assignment with a cost lower than UB_j. This agent is the latest assigned agent A_i where $lb(Y)[i-1] < UB_j$ (line 33). If such an agent exists, A_j sends him a ***back*** message (line 34). Otherwise, A_j reports this to other agents through ***stp*** messages (line 35) and terminates its execution.

[6] In our implementation the ***fb?*** messages are sent under certain conditions to avoid redundancy, see Section 3.3.

4 Correctness proofs

Lemma 1. *AFB_BJ$^+$ is guaranteed to terminate.*

Proof. (Sketch) The proof is close to the one given in [19]. It can easily be obtained by induction on the agent ordering that there will be a finite number of new generated CPAs (at most d^n, where n is the number of variables and d is the maximum domain size), and that agents can never fall into an infinite loop for a given CPA. □

To prove that AFB_BJ$^+$ is correct, we need to prove that the correctness inherent to AFB_BJ is not violated by the lower bound refinements and the non-broadcasting of solution messages.

(Sketch) Assuming the correctness of AFB_BJ, the lower bounds without refinement terms are consistent. It is thus enough to prove that the costs included in the refinement terms are not redundant. All constraints considered in the calculation of the second and third terms of Eq. 10 have not been included in the first and fourth terms. Moreover, these constraints are not included in the lower bounds computed by other lower priority agents. Therefore, costs added by refinement terms are not redundant and then Eq. 12 is a lower bound on Y^j.

Lemma 2. *By the end of AFB_BJ$^+$, each agent stores in its UB_j the cost of the optimal solution Y^* and in v_j^* its value on Y^*.*

Proof. (Sketch) In agent A_n, $lb(Y^n)$ equals $gc(Y^n)$ because it does not have lower priority agents (Eq. 12). A_n updates its UB_n and v_n^* only when it generates a full CPA (Y^n) with $gc(Y^n)$ smaller than its current upper bound (lines 23-24). Thus, UB_n only decreases. Let σ be the smallest generated UB_n, i.e., σ is the cost of the latest generated full CPA Y^n. In AFB_BJ$^+$, (i) each agent A_j attaches to each message it sends its UB_j. Agent A_j only updates its UB_j and its v_j^* when the upper bound carried in a received message is smaller than the stored one (line 8, Fig. 2). (ii) All agents will receive at least one message after the generation of σ (at least they will receive ***stp*** messages before they stop their execution). (iii) Messages are only sent after receiving and processing other messages. A_n attaches σ to each message it sends. Hence, all messages that follow the generation of σ will contain it. Because σ is the smallest generated upper bound and following (i), (ii) and (iii), when the search is ended, UB_j of each agent A_j equals σ and its v_j^* equals that assigned to x_j in Y^n. Now, one needs to prove that σ is the cost of the optimal solution Y^* (i.e., σ equals $gc(Y^*)$ and Y^n equals Y^*). To prove that σ equals $gc(Y^*)$, it is enough to demonstrate that during search no CPA that can lead to a solution of lower cost than σ is discarded. In AFB_BJ$^+$, the CPAs are discarded only in three places (line 19, procedure `ExtendCPA()`, line 38, procedure `ProcessLB()`, and line 33, procedure `Backtrack()`). In all cases above, we are ensured that the lower bound of the discarded CPAs exceeds UB_j. Thus, they cannot lead to a solution with a cost smaller than UB_j. Now, since $\sigma \leq UB_j$ when discarding those CPAs, we are ensured that they have a cost larger than σ. Thus, σ is the cost of the optimal solution Y^* and then Y^n equals Y^*. Therefore, when AFB_BJ$^+$ terminates, v_j^* is the assignment of x_j on Y^*. Then, the lemma is proved. This also completes the correctness proof of the AFB_BJ$^+$ algorithm. □

Corollary 1. *AFB_BJ$^+$ is sound, complete, and terminates.*

5 Experiments

In this section we experimentally compare AFB_BJ$^+$ to AFB_BJ [7], BnB-Adopt$^+$ [8], and BnB-Adopt-DP2$^+$ (BnB-Adopt$^+$ combined with DP2 value ordering heuristic [1]). Algorithms are evaluated on four commonly used benchmarks: binary random Max-DisCSPs, binary random DCOPs, meeting scheduling and sensor networks. All experiments were performed on the DisChoco 2.0 platform[7] [18], in which agents are simulated by Java threads that communicate only through message passing. We evaluate the performance of the algorithms by communication load and computation effort. Communication load is measured by the total number of exchanged messages among agents during algorithm execution ($\#msg$) [13]. Computation effort is measured by the number of non-concurrent constraint checks ($\#ncccs$) [22]. $\#ncccs$ is the metric used in distributed constraint solving to simulate the computation time.

We simulate two scenarios of communication: fast communication (where message delay is null), and slow communication with uniform random message delay, where the delay costs between 0 and 100 $\#ncccs$ for each message. On slow communication, the trends are similar to those observed for fast communication, so the results are not reported here.

5.1 The benchmark settings

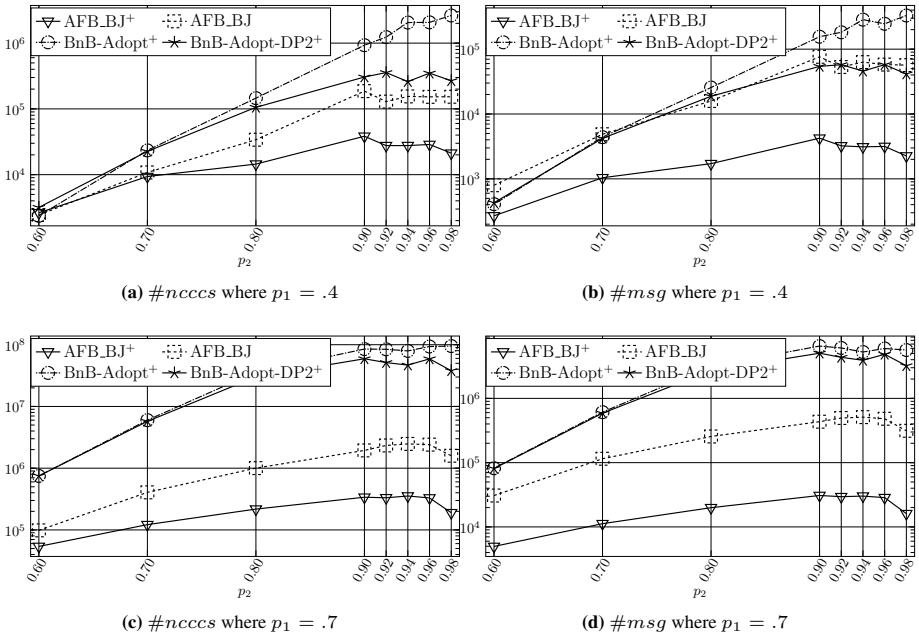
Uniform binary random Max-DisCSPs are characterized by $\langle n,\, d,\, p_1,\, p_2\rangle$, where n is the number of agents/variables, d is the number of values per variable, p_1 is the network connectivity defined as the ratio of existing binary constraints, and p_2 is the constraint tightness defined as the ratio of forbidden value pairs (with a cost of 1). We solved instances of two classes of constraint graphs: sparse graphs $\langle 10, 10, .4, p_2\rangle$ and dense graphs $\langle 10, 10, .7, p_2\rangle$. We varied the tightness from 0.6 to 0.9 by steps of 0.1 and from 0.9 to 0.98 by steps of 0.02. For each pair of fixed density and tightness (p_1, p_2) we report average over 50 instances.

Binary random DCOPs are characterized by $\langle n,\, d,\, p_1\rangle$, where n, d and p_1 are as in Max-DisCSPs [8]. For each value combination a cost is selected randomly from the set $\{0, \ldots, 100\}$. For each $p_1 = 0.4, \ldots, 0.8$, we have generated 50 instances in the class $\langle n = 10,\, d = 10,\, p_1\rangle$.

The meeting scheduling consists of a set of agents, each having a personal private calendar and a set of meetings each taking place in a specified location. The meeting scheduling is encoded as follows. Variables/agents represent meetings. Each meeting/variable has as domain the time slots possible for it. There are constraints between meetings that share participants. We present here 4 cases each with different hierarchical scenarios [21].

The sensor network problem consists of a set of sensors that track a set of mobiles. Each mobile must be tracked by 3 sensors. Each sensor can track at most one mobile. The sensor network problems are encoded as follows. Variables/agents represent mobiles. The possible values of a variable/mobile are all combinations of three sensors that are able to track it. There are constraints between adjacent mobiles. Details are given in [1,11,2]. We present here 4 cases with different topology scenarios [21].

[7] http://dischoco.sourceforge.net/



(a) $\#ncccs$ where $p_1 = .4$ **(b)** $\#msg$ where $p_1 = .4$

(c) $\#ncccs$ where $p_1 = .7$ **(d)** $\#msg$ where $p_1 = .7$

Fig. 3: Total number of messages sent and $\#ncccs$ performed on Max-DisCSP problems in logarithmic scale.

Table 1: Total number of messages sent and $\#ncccs$ performed on binary random DCOPs where costs are randomly selected from 0 to 100.

	$\#ncccs \times \mathbf{10^3}$					$\#msg \times \mathbf{10^3}$				
p_1	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**
AFB_BJ$^+$	**31**	**77**	**148**	**299**	**554**	**3**	**7**	**14**	**27**	**48**
AFB_BJ	122	308	654	1,601	3,442	54	111	186	379	658
BnB-Adopt$^+$	617	3,193	15,436	61,938	98,684	102	419	1,552	5,289	6,312
BnB-Adopt-DP2$^+$	180	958	5,266	25,869	75,414	34	151	636	2,549	5,864

5.2 Results & Discussion

The results on instances of the first set of experiments (Max-DisCSPs) are illustrated in Fig. 3. In terms of computational effort (Figs. 3a and 3c), AFB_BJ$^+$ improves the AFB algorithms and performs faster than both BnB-Adopt$^+$ algorithms. The factor of improvements is 5 for sparse graphs and 7 for dense graphs. Concerning communication load (Figs. 3b and 3d), AFB_BJ$^+$ requires few messages compared to others algorithms. AFB_BJ$^+$ improves AFB_BJ by a factor of 20 (resp. 15) in sparse (resp. dense) instances. In dense instances, AFB_BJ$^+$ outperforms BnB-Adopt-DP2$^+$ by a large scale. BnB-Adopt$^+$ and BnB-Adopt-DP2$^+$ are the less efficient algorithms for solving Max-DisCSPs, and their performance dramatically deteriorates on dense Max-

Table 2: Total number of messages sent and $\#ncccs$ performed on Meeting Scheduling.

	$\#ncccs$				$\#msg$			
$cases$	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
AFB_BJ$^+$	**4,987**	6,536	2,789	2,206	**373**	871	536	582
AFB_BJ	30,332	101,206	15,841	32,364	7,944	32,262	9,441	17,443
BnB-Adopt$^+$	272,490	63,352	51,134	30,030	15,507	10,472	8,717	8,278
BnB-Adopt-DP2$^+$	5,371	**4,224**	**2,165**	**1,647**	636	**749**	**511**	**485**

Table 3: Total number of messages sent and $\#ncccs$ performed on Sensor Network.

	$\#ncccs$				$\#msg$			
$cases$	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
AFB_BJ$^+$	5,599	6,182	2,395	4,869	2,043	1,999	325	1,430
AFB_BJ	167,862	190,423	12,084	33,988	127,544	145,421	7,853	33,280
BnB-Adopt$^+$	4,052	6,337	6,561	8,982	876	1,215	1,198	2,072
BnB-Adopt-DP2$^+$	**992**	**1,046**	**982**	**1,278**	**195**	**238**	**176**	**323**

DisCSP problems. The DP2 heuristic improves the performance of BnB-Adopt$^+$. This improvement is clearer in the sparse problems than in dense ones.

For binary random DCOPs, the results are presented in Table 1. Both versions of BnB-Adopt$^+$ dramatically deteriorate compared to algorithms performing assignments sequentially. Again, the DP2 heuristic improves the performance of BnB-Adopt$^+$. AFB_BJ$^+$ improves the speed-up of AFB_BJ by a factor of 6 in dense instances. Regarding the $\#msg$, the factor of improvement is 13.

Table 2 presents the results on meeting scheduling problems. Comparing AFB_BJ$^+$ to AFB_BJ, the obtained results show that AFB_BJ$^+$ reduces the number of $\#ncccs$ by a factor of 10 and the number of required messages by a factor of 50 in all classes. AFB_BJ$^+$ outperforms BnB-Adopt$^+$ by a large factor on both considered measures. However, BnB-Adopt-DP2$^+$ benefits from its preprocessing step and performs faster than AFB_BJ$^+$.

For sensor networks, the results are presented in Table 3. Again, AFB_BJ$^+$ improves the performance of AFB_BJ by a large scale. Compared to AFB_BJ, AFB_BJ$^+$ reduces the $\#ncccs$ by a factor of 15 and the number of messages by a factor of 50. BnB-Adopt$^+$ performs almost the same $\#ncccs$ and the same number of messages as AFB_BJ$^+$. BnB-Adopt-DP2$^+$ outperforms all other algorithms since it needs very few messages and $\#ncccs$ to resolves sensor networks instances.

Looking at all results together, we come to the straightforward conclusion that AFB_BJ$^+$ performs very well compared to its forward bounding counterparts. The reason for that amounts mainly to refined lower bounds and their use as value ordering heuristic. This guides the search first to promising assignments. The large gap in communication load can be explained by the fact that when an AFB_BJ$^+$ agent has the token to assign, it sends at-most one request to each lower agent, whereas other AFB algorithms need one message for each lower agent for each possible assignment. In addition, AFB_BJ$^+$ stores and maintains valid lower bounds to avoid redundant messages.

Both versions of BnB-Adopt$^+$ perform very poorly when solving Max-DisCSPs and random DCOPs. One possible reason is that in both algorithms, agents have a strongly asynchronous assignments policy. However, for structured problems, BnB-Adopt-DP2$^+$ has performance close to AFB_BJ$^+$. On some highly structured problems (sensor networks), it performs well. When we checked these instances, we found them very sparse with very few constraints. The constraint tree structure used in BnB-Adopt$^+$ combined with the very informed DP2 heuristic allows agents, in such very sparse instances, to initialize their lower bounds of values to a cost close to that of the solution.

Our experiments show that AFB_BJ$^+$ needs less messages than other algorithms. However, AFB_BJ$^+$ messages can be longer than those sent by other algorithms. The largest messages in AFB_BJ$^+$ (***lb*** messages) are in $O(nd)$. To see the practical impact of these larger messages, we computed the total number of *bytes* exchanged by all algorithms.[8] AFB_BJ$^+$ is improved by BnB-Adopt-DP2$^+$ by a factor up to 2 on meeting scheduling and 47 on sensor networks. Except for these two cases, AFB_BJ$^+$ improves other algorithms in all benchmarks by a factor up to 94 (instead of factor 73 for $\#msg$) for AFB_BJ, 144 (instead of 236 for $\#msg$) for BnB-Adopt$^+$, and 80 (instead of 203 for $\#msg$) for BnB-Adopt-DP2$^+$.

In all our experiments, the longest message sent by AFB_BJ$^+$ was of size 366 bytes. The minimum datagram size that we are guaranteed to send without fragmentation of a message (in one physical message) is 568 bytes for IPv4 and 1,272 bytes for IPv6 when using either TCP or UDP [3]. Thus, counting the number of exchanged messages is equivalent to counting the number of physical messages.

6 Conclusion

We have proposed AFB_BJ$^+$, a revisited version of the AFB_BJ algorithm in which we refine the computations of lower bounds by future agents. In AFB_BJ$^+$, the lower bounds are computed for the whole domain of the last assigned agent providing him with a very informed value ordering heuristic. Our experiments show that AFB_BJ$^+$ improves the current state of the art in terms of runtime and number of exchanged messages on different distributed problems. The present work is a step forward in order to address real world applications in multi-agent coordination. Several directions need to be explored in the AFB family. A promising direction is that of variable ordering heuristics. Another direction will be to try to maintain consistencies stronger than forward bounding.

References

1. Ali, S., Koenig, S., Tambe, M.: Preprocessing techniques for accelerating the dcop algorithm adopt. In: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems. pp. 1041–1048. AAMAS'05, ACM, New York, NY, USA (2005)
2. Béjar, R., Domshlak, C., Fernández, C., Gomes, C., Krishnamachari, B., Selman, B., Valls, M.: Sensor networks and distributed csp: communication, computation and complexity. Artif. Intel. 161, 117–147 (2005)

[8] In our implementation we do not perform any message compression.

3. Bessiere, C., Bouyakhf, E.H., Mechqrane, Y., Wahbi, M.: Agile Asynchronous Backtracking for Distributed Constraint Satisfaction Problems. In: Proceedings of the IEEE 23rd International Conference on Tools with Artificial Intelligence. pp. 777–784. ICTAI'11, Boca Raton, Florida, USA (November 2011)
4. Bonnet-Torrés, O., Tessier, C.: Multiply-constrained dcop for distributed planning and scheduling. In: AAAI Spring Symposium: Distributed Plan and Schedule Management. pp. 17–24 (2006)
5. Chechetka, A., Sycara, K.: No-Commitment Branch and Bound Search for Distributed Constraint Optimization. In: Proceedings of AAMAS'06. pp. 1427–1429 (2006)
6. Gershman, A., Meisels, A., Zivan, R.: Asynchronous Forward-Bounding for Distributed Constraints Optimization. In: Proceedings of ECAI'06. pp. 103–107 (2006)
7. Gershman, A., Meisels, A., Zivan, R.: Asynchronous Forward Bounding for Distributed COPs. JAIR 34, 61–88 (2009)
8. Gutierrez, P., Meseguer, P.: Saving redundant messages in bnb-adopt. In: AAAI'10 (2010)
9. Gutierrez, P., Meseguer, P.: Removing redundant messages in n-ary bnb-adopt. J. Artif. Intell. Res. (JAIR) 45, 287–304 (2012)
10. Hirayama, K., Yokoo, M.: Distributed partial constraint satisfaction problem. In: Principles and Practice of Constraint Programming. pp. 222–236 (1997)
11. Jung, H., Tambe, M., Kulkarni, S.: Argumentation as Distributed Constraint Satisfaction: Applications and Results. In: Proceedings of AGENTS'01. pp. 324–331 (2001)
12. Léauté, T., Faltings, B.: Coordinating Logistics Operations with Privacy Guarantees. In: Proceedings of the IJCAI'11. pp. 2482–2487 (2011)
13. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Series (1997)
14. Maheswaran, R.T., Tambe, M., Bowring, E., Pearce, J.P., Varakantham, P.: Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In: Proceedings of AAMAS'04 (2004)
15. Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M.: ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. Artif. Intel. 161, 149–180 (2005)
16. Nguyen, V., Sam-Haroud, D., Faltings, B.: Dynamic Distributed BackJumping. In: Faltings, B., Petcu, A., Fages, F., Rossi, F. (eds.) Recent Advances in Constraints, Lecture Notes in Computer Science, vol. 3419, pp. 71–85. Springer Berlin Heidelberg (2005)
17. Petcu, A., Faltings, B.: A Value Ordering Heuristic for Distributed Resource Allocation. In: Proceedings of Joint Annual Workshop of ERCIM/CoLogNet on CSCLP'04. pp. 86–97 (2004)
18. Wahbi, M., Ezzahir, R., Bessiere, C., Bouyakhf, E.H.: DisChoco 2: A Platform for Distributed Constraint Reasoning. In: Proceedings of the IJCAI'11 workshop on Distributed Constraint Reasoning. pp. 112–121. DCR'11, Barcelona, Catalonia, Spain (2011), `http://dischoco.sourceforge.net/`
19. Wahbi, M., Ezzahir, R., Bessiere, C., Bouyakhf, E.H.: Nogood-Based Asynchronous Forward-Checking Algorithms. Constraints 18(3), 404–433 (2013)
20. Yeoh, W., Felner, A., Koenig, S.: BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm. J. Artif. Intell. Res. (JAIR) 38, 85–133 (2010)
21. Yin, Z.: USC dcop repository (2008), `http://teamcore.usc.edu/dcop`
22. Zivan, R., Meisels, A.: Message delay and DisCSP search algorithms. Annals of Mathematics and Artificial Intelligence 46(4), 415–439 (2006)