Enforcing Full Arc Consistency in Asynchronous Forward Bounding Algorithm

Rachid Adrdor, and Lahcen Koutti

Original scientific article

***Abstract*—The AFB_BJ$^+$_DAC* is the latest variant of asynchronous forward bounding algorithms used to solve Distributed Constraint Optimization Problems (DCOPs). It uses Directional Arc Consistency (DAC*) to remove, from domains of a given DCOP, values that do not belong to its optimal solution. However, in some cases, DAC* does not remove all suboptimal values, which causes more unnecessary research to reach the optimal solution. In this paper, to clear more and more suboptimal values from a DCOP, we use a higher level of DAC* called Full Directional Arc Consistency (FDAC*). This level is based on reapplying AC* several times, which gives the possibility of making more deletions and thus quickly reaching the optimal solution. Experiments on some benchmarks show that the new algorithm, AFB_BJ$^+$_FDAC*, is better in terms of communication load and computation effort.**

***Index Terms*—DCOP, AFB_BJ$^+$_AC*, Soft Arc Consistency, Full Directional Arc Consistency.**

I. Introduction

There are a large number of multi-agent problems that can be modeled as DCOPs such as meetings scheduling [17], sensor networks [7], [18], and so on. In a DCOP, variables, domains, and constraints are distributed among a set of agents. Each agent has full control over a subset of variables and constraints that involve them [11]. A DCOP is solved in a distributed manner via an algorithm allowing the agents to cooperate and coordinate with each other to find a solution with a minimal cost. A solution of a DCOP is a set of value assignments, each representing the value assigned to one of the variables of that DCOP. Algorithms with various search strategies have been suggested to solve DCOPs [9], [10]. Among them, there are Adopt [19], BnB-Adopt [25], BnB-Adopt$^+$ [13], SyncBB [15], AFB [11], [21], AFB_BJ$^+$ [23], AFB_BJ$^+$_AC* [1]–[3], AFB_BJ$^+$_DAC* [4], [5], etc.

In AFB_BJ$^+$_DAC*, to find the optimal solution to a given problem, the agents synchronously exchange a current partial assignment (CPA) containing their assignments. During this process, and to reduce the number of exchanges, each agent uses directional arc consistency (DAC*) to remove any suboptimal values in its domain. The positive behavior of DAC* depends closely on DCOP to be solved in terms of its constraints and costs. This is what sometimes prevents DAC* from behaving better in AFB_BJ$^+$_DAC* algorithm. This often occurs in DCOPs where the constraints are sparse or they are dense but most of their costs are zero. For that, we suggest in this paper to upgrade DAC* to the next higher level, which is Full Directional Arc Consistency (FDAC*). The new algorithm is called AFB_BJ$^+$_FDAC* and allows agents to perform AC* multiple times and thus remove more suboptimal values from their domains.

Manuscript received April 10, 2021; revised December 21, 2021. Date of publication January 18, 2022. Date of current version January 18, 2022.

Authors are with the Department of Computer Science, Faculty of Sciences, Agadir, Morocco (e-mails: rachid.adrdor@edu.uiz.ac.ma, l.koutti@uiz.ac.ma).

Digital Object Identifier (DOI): 10.24138/jcomss-2021-0083

Our experiments on different benchmarks show the superiority of AFB_BJ$^+$_FDAC* algorithm in terms of communication load and computation effort.

This paper is made up of three main sections. Section II presents an overview of DCOPs, soft arc consistency rules, AFB_BJ$^+$_AC* algorithm, and AFB_BJ$^+$_DAC* algorithm. Section III gives a description of AFB_BJ$^+$_FDAC* algorithm. Section IV exposes the experiments carried out on some benchmarks.

II. Background

A. Distributed Constraint Optimization Problem (DCOP)

A DCOP [12] is defined by 4 sets, set of agents $\mathcal{A} = \{A_1, A_2, ..., A_k\}$, set of variables $\mathcal{X} = \{x_1, x_2, ..., x_n\}$, set of domains $\mathcal{D} = \{D_1, D_2, ..., D_n\}$, each D_i is the possible values of x_i in $\mathcal{X}$, and set of soft constraints $\mathcal{C} = \{C_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+\} \cup \{C_i : D_i \rightarrow \mathbb{R}^+\}$. In a DCOP, each agent is fully responsible for a subset of variables and the constraints that involve them.

In this paper, while maintaining the generality, we only consider DCOPs in that each agent is responsible for a single variable and that every two variables, at most, are linked by a constraint (i.e., unary or binary constraint) [20].

We consider these notations : A_j is an agent, where j is its level or rank in the default ordering. (x_j, v_j) is an assignment of A_j, where $v_j \in D_j$ and $x_j \in \mathcal{X}$. C_{ij} is a binary constraint between x_i and x_j. C_{ij}^{ac} is an identical copy of the C_{ij} constraint, used in AC* process. C_j is a unary constraint on x_j. C_ϕ is the global zero-arity constraint that represents a lower bound of any solution of a given DCOP. C_{ϕ_j} is the local zero-arity constraint that represents the contribution value of A_j in C_ϕ (i.e., $C_\phi = \sum_{A_j \in \mathcal{A}} C_{\phi_j}$). UB_j is the cost of the optimal solution reached so far. $[A_1, A_2, \ldots, A_n]$ is the lexicographic ordering of agents (the default ordering). $\Gamma(x_j) = \{\Gamma^- : x_i \in \mathcal{X} \mid C_{ij} \in \mathcal{C}, i < j\} \cup \{\Gamma^+ : x_i \in \mathcal{X} \mid C_{ij} \in \mathcal{C}, i > j\}$ is the set of neighbors of A_j. Γ^- (resp. Γ^+) is the set of neighbors with a higher priority (resp. with a lower priority). $Y = Y^j = [(x_1, v_1), \ldots, (x_j, v_j)]$ is a current partial assignment (CPA). v_j^* is the optimal value of A_j. $lb_k[i][v_j](Y^j)$ are the lower bounds of a lower neighbor A_k obtained for Y^j. GC (resp. GC^*) are the guaranteed costs of Y (resp. in AC*). $DVals$ is a list of n arrays

P. 1: ProjectUnary()

```
β ← min_{v_i ∈ D_i} {c_i(v_i)} ;
C_{φ_i} ← C_{φ_i} + β ;
foreach (v_i ∈ D_i) do
    c_i(v_i) ← c_i(v_i) − β ;
```

P. 2: ProjectBinary(x_i, x_j)

```
foreach (v_i ∈ D_i) do
    α ← min_{v_j ∈ D_j} {c_ij(v_i, v_j)} ;
    foreach (v_j ∈ D_j) do
        c_ij(v_i, v_j) ← c_ij(v_i, v_j) − α ;
    if (A_i is the current agent)
        c_i(v_i) ← c_i(v_i) + α ;
```

P. 3: Extend(x_i, x_j, E)

```
foreach (v_i ∈ D_i) do
    foreach (v_j ∈ D_j) do
        c_ij(v_i, v_j) ← c_ij(v_i, v_j) + E[v_i] ;
    if (A_i is the current agent)
        c_i(v_i) ← c_i(v_i) − E[v_i] ;
```

P. 4: AC* ()

```
foreach (A_k ∈ Γ+) do
    ProjectBinary(x_j, x_k) ;
    ProjectBinary(x_k, x_j) ;
foreach (A_k ∈ Γ−) do
    ProjectBinary(x_k, x_j) ;
    ProjectBinary(x_j, x_k) ;
ProjectUnary() ;
```

containing deleted values. Each array, $DVals[j]$, contains two elements, $listVals$ which is the list of values deleted by A_j and $UnvNbrs$ which is a counter of the A_j neighbors that have not yet processed $listVals$. $EVals$ is a list of arrays containing extension values.

The guaranteed cost of Y (1) is the sum of c_{ij} involved in Y.

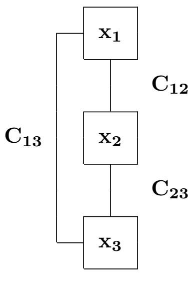
$$GC(Y) = \sum_{C_{ij} \in \mathcal{C}} c_{ij}(v_i, v_j), \qquad (x_i, v_i),\ (x_j, v_j)\ \in Y \tag{1}$$

If a CPA Y comprises a value assignment for each variable of a given DCOP, then it is called a *complete assignment* (i.e., a solution). This solution is said to be *optimal* (2) when the sum of all the constraint costs that it implies is minimal.

$$Y^* = \arg\min_{Y}\{GC(Y) \mid var(Y) = \mathcal{X}\} \tag{2}$$

Fig.1 shows an example of a DCOP in which each agent $A_i \in \mathcal{A} = \{A_1, A_2, A_3\}$ takes control of a single variable $x_i \in \mathcal{X} = \{x_1, x_2, x_3\}$, each being defined on a domain of values $D_i = \{0, 1\}$ in $\mathcal{D} = \{D_1, D_2, D_3\}$. Each pair of variables in $\mathcal{X}$ is connected by a binary constraint $C_{ij} \in \mathcal{C} = \{C_{12}, C_{13}, C_{23}\}$. The costs of the combinations of values of each constraint C_{ij} are indicated in the side tables.

x_1	x_3	c_{13}
0	0	1
0	1	4
1	0	3
1	1	2



x_1	x_2	c_{12}
0	0	7
0	1	5
1	0	2
1	1	1

x_2	x_3	c_{23}
0	0	4
0	1	8
1	0	5
1	1	2

Fig. 1. Example of a distributed constraint optimization problem (DCOP).

B. Soft Arc Consistency Techniques

Soft arc consistency techniques are used when solving a given problem to delete values that are not part of the optimal solution of that problem. To apply these techniques, we use a set of transformations known as *equivalence preserving transformations*. They allow the exchange of costs between the constraints of the problem according to three manners that are a binary projection, a unary projection, and an extension.

The binary projection (P. 2) is an operation that subtracts, for a value v_i of D_i, the smallest cost α of a binary constraint C_{ij} and adds it to the unary constraint C_i.

The unary projection (P. 1) is an operation that subtracts the smallest cost β of a unary constraint C_i and adds it to the zero-arity constraint C_ϕ.

The extension (P. 3) is an operation that subtracts, for a value v_i of D_i, the extension value ($E[v_i]$) of v_i from a unary constraint C_i and adds it to the binary constraint C_{ij}, with $0 < E[v_i] \leq c_i(v_i)$.

All of these transformations are applied to a problem under a set of conditions represented by soft arc consistency levels [16], namely:

Node Consistency (NC)*: a variable x_i is NC* if each value $v_i \in D_i$ satisfies $C_\phi + c_i(v_i) < UB_i$ and there is a value $v_i \in D_i$ with $c_i(v_i) = 0$. A problem is NC* if each variable x_i of this problem is NC*.

Arc Consistency (AC)*: a variable x_i is AC* with respect to its neighbor x_j if x_i is NC* and there is, for each value $v_i \in D_i$, a value $v_j \in D_j$ which satisfies $c_{ij}(v_i, v_j) = 0$. v_j is called a *simple support* of v_i. A problem is AC* if each variable x_i of this problem is AC*.

Directional Arc Consistency (DAC)*: a variable x_i is DAC* with respect to its lower neighbor $x_{j(j>i)}$ if x_i is NC* and there is, for each value $v_i \in D_i$, a value $v_j \in D_j$ which satisfies $c_{ij}(v_i, v_j) + c_j(v_j) = 0$. v_j is called a *full support* of v_i. A problem is DAC* if each variable x_i of this problem is DAC* with its lower neighbors $x_{j(j>i)}$.

Full Directional Arc Consistency (FDAC)*: A problem is FDAC* if this problem is AC* and DAC*.

To make any problem AC*, it is necessary to apply, for each variable of this problem, a binary projection (P. 2), then a unary projection (P. 1), and finally a deletion of non-NC* values. These three instructions are repeated each time a value is deleted. In a distributed case, each agent A_i performs AC* locally (P. 4) and shares its contribution value stored in C_{ϕ_i} (P.

P. 5: DAC*()

```
foreach (A_k ∈ Γ+) do
    foreach (v_k ∈ D_k) do
        P[v_k] ← min_{v_j ∈ D_j} {c_jk(v_j, v_k) + c_j(v_j)} ;
    foreach (v_j ∈ D_j) do
        E[v_j] ← max_{v_k ∈ D_k} {P[v_k] − c_jk(v_j, v_k)} ;
    Extend(x_j, x_k, E) ;
    ExtVals[jk].put(E) ;
    ProjectBinary(x_k, x_j) ;
```

P. 6: ProcessPruning(msg)

```
if (msg.type = “Ok”)
    ExtVals ← msg.ExtVals ;
    foreach (A_k ∈ Γ−) do
        Extend(x_k, x_j, ExtVals[kj]) ;
        ExtVals[kj].clear ;
DelVals ← msg.DelVals ;
foreach (A_k ∈ Γ) do
    foreach (a ∈ DelVals[k].listVals) do
        D_k ← D_k − a ;
    /* -----AC*-----                    */
    ProjectBinary(x_j, x_k) ;
    ProjectUnary() ;
    /* -------------                    */
    if (D_k is changed)
        DelVals[k].nbUnvisitedNbrs.decrement(−1) ;
    if (DelVals[k].nbUnvisitedNbrs = 0)
        DelVals[k].listVals.clear ;
C_φ ← max {C_φ, msg.C_φ} + C_φj ; C_φj ← 0 ;
if (C_φ ≥ UB_j)
    broadcastMsg : stp(UB_j) ;
    end ← true ;
CheckPruning() ;
DAC*() ; // here FDAC* is achieved
ExtendCPA() ;
```

P. 7: CheckPruning()

```
foreach (a ∈ D_j) do
    if (c_j(a) + C_φ ≥ UB_j) ∨
      ((A_j = A_1) ∧ (lb(Y ∪ (x_j, a)) ≥ UB_j))
        D_j ← D_j − a ;
        DelVals[j].listVals.add(a) ;
if (D_j is changed)
    DelVals[j].nbUnvisitedNbrs ← A_j.nbNbrs;
    foreach (A_k ∈ Γ−) do
        ProjectBinary(x_k, x_j) ;
if (D_j is empty)
    broadcastMsg : stp(UB_j) ;
    end ← true ;
```

1, line 2) with the other agents in order to calculate the global C_ϕ (i.e., $C_\phi = \sum_{A_i \in \mathcal{A}} C_{\phi_i}$). Each agent A_i keeps locally for each of its constraints C_{ij} an identical copy marked by C_{ij}^{ac} and used in AC* procedure. During AC*, C_{ij}^{ac} constraints are changed. To keep the symmetry of these constraints in the agents, each agent A_i applies, on its copy C_{ij}^{ac}, the same action of its neighbor A_j and vice versa (P. 4, line 3, 5) [14].

In the same way, we can make any problem DAC*. But in this case, we must first extend (P. 3), for each variable, from its unary costs to its binary costs, the minimum cost required to perform again AC* by its lower neighbors (P. 5).

By executing AC* and DAC* successively for each variable, we can make the problem FDAC*.

C. AFB_BJ$^+$_AC* Algorithm

Each agent A_j carries out the AFB_BJ$^+$_AC* [3] [2] according to three phases. First, A_j initializes its data structures and performs AC* to delete suboptimal values from its domain D_j. Second, A_j chooses, for its variable x_j, a value from its previously filtered domain D_j in order to extend the CPA Y^j by its value assignment (x_j, v_j). If A_j has successfully extended the CPA, it sends an **ok?** message to the next agent asking it to continue the extension of CPA Y^j. This message loads the extended CPA Y^j, its guaranteed cost (3), its guaranteed cost of AC* (4), the C_ϕ, and the list $DVals$.

$$GC(Y^j)[j] = GC(Y^{j-1}) + \sum_{(x_i,v_i) \in Y^{j-1} \mid i<j} c_{ij}(v_i, v_j) \tag{3}$$

$$\underset{(x_i,v_i) \in Y^{j-1}}{GC^*(Y^j)} = GC^*(Y^{j-1}) + c_j(v_j) + \sum_{C_{ij}^{ac} \in \mathcal{C}} c_{ij}(v_i, v_j) \tag{4}$$

In case A_j fails to extend the CPA, either because it doesn't find a value that gives a valid CPA, or because all the values in its domain are exhausted, it stops the CPA extension and sends a **back** message, containing the same data structures as an **ok?** message excluding GC and GC^*, to the appropriate agent. If such an agent does not exist or the domain of A_j becomes empty, A_j stops its execution and informs the others via **stp** messages. A CPA Y^j is said to be valid if its lower bound (5) does not exceed the global upper bound UB_j, which represents the cost of the optimal solution achieved so far.

$$LB(Y^j)[i] = GC(Y^j)[i] + \sum_{A_k > A_j} LB_k(Y^j)[i] \tag{5}$$

Third, A_j evaluates the extended CPA by sending **fb?** messages, which hold the same data structures as an **ok?** message excluding C_ϕ and $DVals$, to unassigned agents asking them to evaluate the CPA and send the result of the evaluation. When an agent has completed its evaluation, it sends the result directly to the sender agent via an **lb** message. The evaluation is based on the calculation of appropriate lower bounds for the received CPA Y^i. The lower bound of Y^i (6) is the minimum lower bound over all values of D_j with respect to Y^i.

P. 8: AFB_BJ$^+$_FDAC* ()

$UB_j \leftarrow +\infty$;
$v_j^* \leftarrow empty$;
$Y \leftarrow []$;
$GC[i..j-1] \leftarrow [0,...,0]$;
$\underset{(A_k > A_j) \wedge (v_j \in D_j)}{lb_k[0][v_j]} \leftarrow min_{v_k \in D_k}\{c_{jk}(v_j, v_k)\}$;
$mustSendFB \leftarrow True$;
$C_\phi \leftarrow 0$; $C_{\phi_j} \leftarrow 0$;
$GC^*[i..j-1] \leftarrow [0,...,0]$;
$\forall a \in D_j,\ c_j(a) \leftarrow 0$;
$AC^*()$;
if $(A_j = A_1)$
  $C_\phi \leftarrow C_\phi + C_{\phi_j}$;
  $C_{\phi_j} \leftarrow 0$;
  $CheckPruning()$;
  DAC*() ; // here FDAC* is achieved
  $ExtendCPA()$;
while $(\neg end)$ **do**
  $msg \leftarrow getMsg()$;
  if $(msg.UB < UB_j)$
    $UB_j \leftarrow msg.UB$;
    $v_j^* \leftarrow v_j$;
  if $(msg.Y$ *is stronger than* $Y)$
    $Y \leftarrow msg.Y$;
    $GC \leftarrow msg.GC$;
    clear irrelevant $lb()$;
    reset D_j ;
  switch $(msg.type)$ **do**
    case ***ok?*** **do**
      $mustSendFB \leftarrow True$;
      $GC^* \leftarrow msg.GC^*$;
      $ProcessPruning(msg)$;
    case ***back*** **do**
      $Y \leftarrow Y^{j-1}$;
      $ProcessPruning(msg)$;
    case ***fb?*** **do**
      $GC^* \leftarrow msg.GC^*$;
      foreach $(v_j \in D_j)$ **do**
        $cost \leftarrow C_\phi + GC^*(Y^{j-1}) + c_j(v_j)$;
        if $(cost \geq UB_j)$
          $D_j \leftarrow D_j - v_j$;
      sendMsg : $\underset{\text{to } A_i}{\textbf{lb}}\ (lb_j(Y^i)[],\ msg.Y)$;
    case ***lb*** **do**
      $lb_k(Y^j) \leftarrow msg.lb$;
      if $(lb(Y^j) \geq UB_j)$
        $ExtendCPA()$;
    case ***stp*** **do**
      $end \leftarrow true$;

$$LB_j(Y^i)[h] \underset{(h \leq i < j)}{=} \min_{v_j \in D_j} \Bigg\{ \sum_{\substack{(x_k, v_k) \in Y^h \\ (k \leq h)}} c_{kj}(v_k, v_j) + \sum_{\substack{k=h+1 \\ (h<k<i)}}^{i-1} \min_{v_k \in D_k} \{c_{kj}(v_k, v_j)\} + c_{ij}(v_i, v_j) + \sum_{\substack{x_k \in \Gamma^+(x_j) \\ (k>j)}} \min_{v_k \in D_k} \{c_{jk}(v_j, v_k)\} \Bigg\} \tag{6}$$

P. 9: ExtendCPA()

$v_j \leftarrow argmin_{v'_j \in D_j} \left\{ lb(Y \cup (x_j, v'_j)) \right\}$;
if $(lb(Y \cup (x_j, v_j)) \geq UB_j) \vee (C_\phi + GC^*(Y^{j-1}) + c_j(v_j) \geq UB_j)$
  for $i \leftarrow j-1$ **to** 1 **do**
    if $(lb(Y)[i-1] < UB_j)$
      sendMsg : $\underset{\text{to } A_i}{\textbf{back}}(Y^i,\ UB_j,\ DelVals,\ C_\phi)$;
      return ;
  broadcastMsg : **stp**(UB_j) ;
  $end \leftarrow true$;
else
  $Y \leftarrow \{Y \cup (x_j, v_j)\}$;
  if $(var(Y) = X)$
    $UB_j \leftarrow GC(Y)$;
    $v_j^* \leftarrow v_j$;
    $Y \leftarrow Y^{j-1}$;
    $CheckPruning()$;
    $ExtendCPA()$;
  else
    sendMsg : $\underset{\text{to } A_{j+1}}{\textbf{ok?}}\ (Y, GC, UB_j, DelVals, ExtVals, C_\phi, GC^*)$;
    $ExtVals.clear$;
    if $(mustSendFB)$
      sendMsg : $\underset{\text{to } A_{k>j}}{\textbf{fb?}}\ (Y,\ GC,\ UB_j,\ GC^*)$;
      $mustSendFB \leftarrow false$;

D. AFB_BJ$^+$_DAC* Algorithm

The AFB_BJ$^+$_DAC* [4] algorithm follows the same steps as AFB_BJ$^+$_AC* algorithm except that it performs DAC* instead of AC*. With AC*, we can find for each value of a given agent the corresponding simple support in the domains of its lower and higher neighbors. While with DAC* which is the next level of AC* and the best in reducing the domains of a given DCOP, we can find for each value of a given agent the corresponding full support in the domains of its lower neighbors only (§II-B).

III. The AFB_BJ$^+$_FDAC* Algorithm

In AFB_BJ$^+$_FDAC* algorithm, instead of using AC* and DAC* separately as in previous versions, we use FDAC* which provides the same effect of both together.

FDAC* as mentioned in section II-B is executed by executing AC* and DAC* successively. This allows getting for each domain value of each variable simple support in the domains of its higher neighbors (Γ^-) and full support in the domains of its lower neighbors (Γ^+). With FDAC*, we can continuously exchange costs between agents, from unary constraints to binary ones and vice versa. This allows the unary costs of each agent and the global zero-arity constraint (C_ϕ) to be continuously updated. So, with these updates, we can significantly reduce the agent domain. In short, FDAC* is a technique that allows agents to choose more precisely the best values for their variables by removing more and more invalid values in their domains.

The skeleton of AFB_BJ$^+$_FDAC* algorithm is different from those of AFB_BJ$^+$_AC* and AFB_BJ$^+$_AC* in two things:

The first one is the DAC* procedure (P. 5), which is responsible for finding, for each value of an agent, its full support in the domains of its lower neighbors. In DAC* procedure, only a part of the unary costs of a given agent is transferred to its lower neighbors as extension values, not the total of those costs as in AFB_BJ$^+$_DAC* algorithm. This is so that the AC* condition that this agent must keep with its higher neighbors is not violated (P. 5, line 2-5) [16].

The second is the condition (P. 7, line 2) that allows the first agent to permanently delete the values having a global lower bound exceeding the global upper bound and the values that it has already evaluated. This condition remains correct only for the first agent according to the static order of agents. This is because the first agent does not have a previous agent, which allows it to permanently delete any value that proved to be inconsistent.

A. *Description of AFB_BJ$^+$_FDAC**

The AFB_BJ$^+$_FDAC* (P. 8) is performed by each agent A_j as follows :

A_j starts with the initialization step (P. 8, line 1-10) in which it performs the AC* (P. 4). If A_j is the 1^{st} agent (P. 8, line 11), it filters its domain by calling $CheckPruning()$ (P. 7), then performs DAC*() (P. 5) after AC* to ensure the achievement of the FDAC*, and finally calls $ExtendCPA()$ to generate a CPA Y.

Next, A_j starts processing the messages (P. 8, line 17). First, it updates UB_j and v_j^* (P. 8, line 21). Then, A_j updates Y and GC and erases all unrelated lower bounds if the received CPA $(msg.Y)$ is fresh compared to the local one (Y) (P. 8, line 22). Thereafter, A_j restores all temporarily deleted values (P. 8, line 40).

When receiving an **ok?** message (P. 8, line 28), A_j authorizes the sending of **fb?** messages and calls $ProcessPruning()$ (P. 6).

When calling $ProcessPruning()$ (P. 6), A_j deals initially, for **ok?** messages only, with extensions of its higher neighbors (P. 6, line 1-5). Afterward, it updates its $DVals$, then its neighbors' domains separately in order to keep the same domains as these agents (P. 6, line 6-9). After that, it performs once more the AC* (P. 6, line 10-11). Next, A_j decrements the unvisited neighbors of A_k, $DVals[k].UnvNbrs$, and then checks whether it is the last visited neighbor of this agent A_k in order to reset its list of deleted values $DVals[k].listVals$ (P. 6, line 12-15). Then, A_j updates its global C_ϕ (P. 6, line 16). If C_ϕ exceeds the UB_j, A_j turns off its execution and notifies the others (P. 6, line 17-19). Finally, A_j calls $CheckPruning()$ to prune its domain, DAC*() (P. 5) to achieve FDAC*, and $ExtendCPA()$ to extend the received CPA (P. 6, line 20-22).

When calling DAC*() (P. 5), A_j performs the proper extensions from C_j to each C_{ij} (P. 5, line 6-7). To do that, A_j calculates, for each value v_j of D_j, its extension value (P. 5, line 4-5) based on the prior computation of the values of the later projections on its lower neighbors (P. 5, line 2-3) [16]. Once completed, A_j performs a binary projection to keep the symmetry of C_{ij}^{ac} constraints (P. 5, line 8). It should be noted that the direction taken into account by each agent A_j for the extension of its costs is towards its lower neighbors $(\Gamma^+(x_j))$.

When calling $CheckPruning()$ (P. 7), A_j deletes any value from its domain for which the sum of the C_ϕ with the unary cost of this value exceeds UB_j. If A_j is the first agent, it also deletes any value whose global lower bound exceeds UB_j and any value has already been evaluated (P. 7, line 2-3). With each new deletion, A_j initializes the number of its neighbors not yet visited (P. 7, line 5-6). Then, it performs a binary projection to keep the symmetry of C_{ij}^{ac} constraints (P. 7, line 8). If A_j domain becomes empty, A_j turns off its execution and notifies the others (P. 7, line 9-11).

TABLE I
TOTAL OF MESSAGES $(msgs)$ SENT AND NON-CONCURRENT CONSTRAINT CHECKS $(ncccs)$ FOR SOFT GRAPH COLORING, $p_1 = 0.4$

	$ncccs$					$msgs$				
n	**6**	**8**	**10**	**12**	**14**	**6**	**8**	**10**	**12**	**14**
AFB_BJ$^+$	920	3,015	10,975	29,681	163,841	97	437	1,970	6,142	35,715
AFB_BJ$^+$_AC*	518	1,588	7,231	22,454	134,629	69	320	1,758	5,601	33,841
AFB_BJ$^+$_DAC*	450	951	5,218	15,909	127,919	57	234	980	3,631	21,160
AFB_BJ$^+$_FDAC*	**389**	**733**	**4,112**	**13,501**	**122,811**	**41**	**191**	**723**	**2,101**	**17,020**
BnB-Adopt$^+$_DP2	544	4,018	129,967	171,125	32,402,765	136	1,262	34,772	356,984	7,215,399

TABLE II
TOTAL OF MESSAGES $(msgs)$ SENT AND NON-CONCURRENT CONSTRAINT CHECKS $(ncccs)$ FOR SOFT GRAPH COLORING, $p_1 = 0.7$

	$ncccs$					$msgs$				
n	**6**	**8**	**10**	**12**	**14**	**6**	**8**	**10**	**12**	**14**
AFB_BJ$^+$	3,885	12,555	81,753	554,524	3,107,810	363	1,370	10,810	79,748	470,466
AFB_BJ$^+$_AC*	2,292	8,654	63,544	467,559	2,677,628	298	1,249	10,055	75,373	448,832
AFB_BJ$^+$_DAC*	1,421	5,019	59,214	349,706	1,651,628	189	867	7,283	65,893	332,850
AFB_BJ$^+$_FDAC*	**912**	**3,105**	**57,905**	**280,711**	**1,011,208**	**101**	**751**	**5,117**	**60,295**	**295,702**
BnB-Adopt$^+$_DP2	11,199	267,981	10,247,929	98,450,964	100,000,042	2,153	49,883	1,606,033	12,940,164	11,736,067

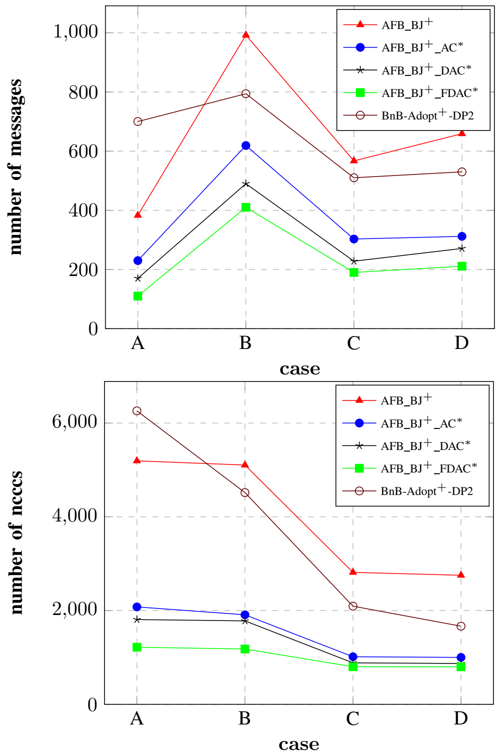


Fig. 2. Total of messages $(msgs)$ sent and non-concurrent constraint checks $(ncccs)$ for meetings scheduling

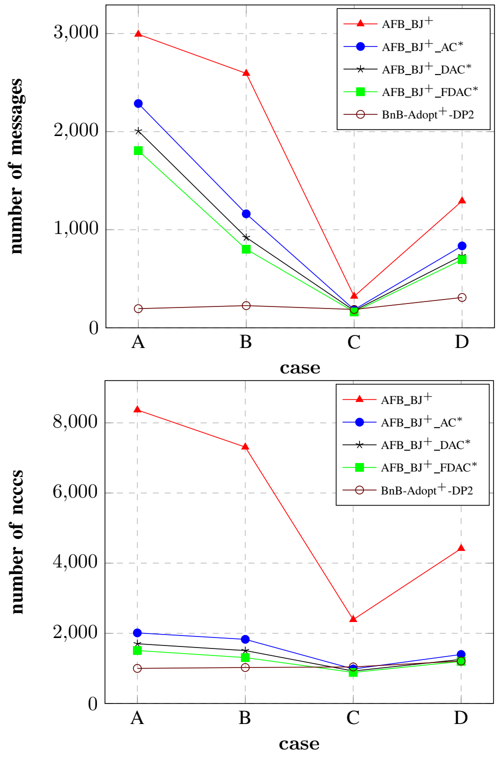


Fig. 3. Total of messages ($msgs$) sent and non-concurrent constraint checks ($ncccs$) for sensors network

When calling $ExtendCPA()$ (P. 9), A_j looks for a value v_j for its variable x_j (P. 9, line 1). If no value exists, A_j returns to the priority agents by sending a **back** message to the contradictory agent (P. 9, line 2-5). If no agent exists, A_j turns off its execution and notifies the others via **stp** messages (P. 9, line 6-7). Otherwise, A_j extends Y by adding its assignment (P. 9, line 9). If A_j is the last agent (P. 9, line 10) then a new solution is obtained and the UB_j is updated, which obliges A_j to call $CheckPruning()$ to filter again its domain and then $ExtendCPA()$ to proceed the search (P. 9, line 11-15). Otherwise, A_j sends an **ok?** message loaded with the extended Y to the next agent (P. 9, line 17) and **fb?** messages to unassigned agents (P. 9, line 20).

When A_j receives an **fb?** message, it filters its domain D_j with respect to the received Y (P. 8, line 36-40), calculates the appropriate lower bounds (6), and immediately sends them to the sender via **lb** message (P. 8, line 41).

When A_j receives an **lb** message, it stores the lower bounds received (P. 8, line 43) and performs $ExtendCPA()$ to modify its assignment if the lower bound calculated, based on the cost of Y (5), exceeds the UB_j.

B. Correctness of AFB_BJ$^+$_FDAC*

Theorem 1. *AFB_BJ$^+$_FDAC* is guaranteed to calculate the optimum and terminates.*

Proof. The AFB_BJ$^+$_FDAC* algorithm overrides its previous versions by performing both AC* and DAC*, which is essentially just a set of cost extensions performed between an agent and its neighbors after performing AC*. These extensions have already been proved which are correct in [16] [8], and they are executed by the AFB_BJ$^+$_FDAC* without any cost redundancy (P. 3, line 4), (P. 5, line 8), and (P. 6, line 1-5). □

IV. Experimental Results

In this section, we experimentally compare AFB_BJ$^+$_FDAC* algorithm with its previous versions, AFB_BJ$^+$, AFB_BJ$^+$_AC*, and AFB_BJ$^+$_DAC*, and with BnB-Adopt$^+$_DP2 algorithm [6], which is its famous competitor. Three benchmarks are used in these experiments: soft graph coloring, meetings scheduling, and sensors network. All experiments were performed on DisChoco 2.0 platform [22], in which agents are simulated by Java threads that communicate only through message passing.

Soft graph coloring [25]: are defined by $(n,\ c,\ p_1)$, which are respectively the number of nodes (i.e., variables), the number of possible colors of each node, and the constraint density. The constraints are applied to adjacent nodes. We evaluated two classes of instances $(n = 6..14, c = 8, p_1 = 0.4)$ and $(n = 6..14, c = 8, p_1 = 0.7)$. For the constraint costs, they were randomly selected from the set $\{0, \ldots, 100\}$. For each p_1, we randomly generated an average of 30 instances.

Meetings scheduling [23]: are defined by $(m,\ p,\ ts)$, which are respectively the number of meetings (i.e., variables), the number of participants, and the number of time slots for each meeting. Each participant has a private schedule of meetings and each meeting takes place at a particular location and at a fixed time slot. The constraints are applied to meetings that share participants. We have evaluated 4 cases A, B, C, and D, which are different in terms of meetings/participants [17].

Sensors network [7]: are defined by $(t,\ s,\ d)$, which are respectively the number of targets (i.e., variables), the number of sensors, and the number of possible combinations of 3 sensors reserved for tracking each target. A sensor can only track one target at most and each combination of 3 sensors must track a target. The constraints are applied to adjacent targets. We have evaluated 4 cases A, B, C, and D, which are different in terms of targets/sensors [17].

To compare the algorithms, we use two metrics which are the total of messages exchanged ($msgs$) that represents the communication load and the total of non-concurrent constraint checks ($ncccs$) that represents the computation effort.

In tables I and II, we display respectively the results of experiments carried out on coloring problems of sparse $(p_1 = 0.4)$ and dense $(p_1 = 0.7)$ graphs. The comparison of these results shows an improvement of AFB_BJ$^+$_FDAC* algorithm reaching 4,000 messages (resp. 5,000 checks) in the sparse case, and reaching 30,000 messages (resp. 600,000 checks) in the dense case. As for BnB-Adopt$^+$_DP2 algorithm, it remains largely delayed compared to the other algorithms.

Regarding meetings scheduling problems (Fig. 2), the results show a clear improvement of AFB_BJ$^+$_FDAC* compared to others, whether for $msgs$ or for $ncccs$. But with regard to sensors network problems (Fig. 3), BnB-Adopt$^+$_DP2 algorithm retains the pioneering role, despite the superiority of AFB_BJ$^+$_FDAC* algorithm to its previous versions.

By analyzing the results, we can conclude that the AFB_BJ$^+$_FDAC* is better than its prior versions, because of the existence of Full Directional Arc Consistency (FDAC*) that allows agents to reapply AC* multiple times and thus remove more suboptimal values. Regarding the superiority of BnB-Adopt$^+$_DP2 over AFB_BJ$^+$_FDAC* in sensors network problems, this is mainly due to the arrangement of the pseudo-tree used by this algorithm that corresponds to the structure of these problems, as well as the existence of DP2 heuristic that facilitates the proper choice of values.

V. Conclusion

In this paper, we have introduced the AFB_BJ$^+$_FDAC* algorithm. It relies on Full Directional Arc Consistency (FDAC*) to further reduce the agent domains of a given DCOP and thus quickly reach its optimal solution. FDAC* makes it possible to perform more cost extensions from each agent to its neighbors. This allows reapplying over and over again the AC*, which increases the number of deletions carried out by each agent and thus accelerates the process of solving a problem. Experiments on some benchmarks show that the AFB_BJ$^+$_FDAC* algorithm behaves better than its previous versions. As future work, we propose to generalize the use of soft arc consistency in its different levels with DCOP algorithms.

References

[1] Rachid Adrdor, Redouane Ezzahir, and Lahcen Koutti. Connecting AFB_BJ$^+$ with soft arc consistency. *International Journal of Computing and Optimization*, 5 no. 1:9–20, 2018. [Online]. Available: https://doi.org/10.12988/ijco.2018.857.

[2] Rachid Adrdor, Redouane Ezzahir, and Lahcen Koutti. Consistance d'arc souple appliquée aux problèmes dcop. *Journées d'Intelligence Artificielle Fondamentale (JIAF)*, page 63, 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02951644/document.

[3] Rachid Adrdor and Lahcen Koutti. Enhancing AFB_BJ$^+$_AC* algorithm. In *2019 International Conference of Computer Science and Renewable Energies (ICCSRE)*, pages 1–7. IEEE, July 2019. [Online]. Available: https://doi.org/10.1109/ICCSRE.2019.8807711.

[4] Rachid Adrdor and Lahcen Koutti. Asynchronous forward-bounding algorithm with directional arc consistency. *CEUR Workshop Proceedings (CEUR-WS.org)*, Vol-2970, 2021. [Online]. Available: http://ceur-ws.org/Vol-2970.

[5] Rachid Adrdor and Lahcen Koutti. Using directional arc consistency with asynchronous forward-bounding algorithm. *CEUR Workshop Proceedings (CEUR-WS.org)*, Vol-2987, 2021. [Online]. Available: http://ceur-ws.org/Vol-2987.

[6] Syed Ali, Sven Koenig, and Milind Tambe. Preprocessing techniques for accelerating the dcop algorithm adopt. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1041–1048. ACM, 2005. [Online]. Available: https://dl.acm.org/citation.cfm?id=1082631.

[7] Ramón Béjar, Carmel Domshlak, Cèsar Fernández, Carla Gomes, Bhaskar Krishnamachari, Bart Selman, and Magda Valls. Sensor networks and distributed csp: communication, computation and complexity. *Artificial Intelligence*, 161(1-2):117–147, 2005. [Online]. Available: https://doi.org/10.1016/j.artint.2004.09.002.

[8] Martin C Cooper, Simon De Givry, Martı Sánchez, Thomas Schiex, Matthias Zytnicki, and Tomáš Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174(7-8):449–478, 2010. [Online]. Available: https://doi.org/10.1016/j.artint.2010.02.001.

[9] Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. Distributed constraint optimization problems and applications: A survey. *arXiv e-prints*, pages arXiv–1602, 2016. [Online]. Available: https://arxiv.org/pdf/1602.06347.

[10] Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, 61:623–698, 2018. [Online].Available: https://doi.org/10.1613/jair.5565.

[11] Amir Gershman, Amnon Meisels, and Roie Zivan. Asynchronous forward bounding for distributed cops. *Journal of Artificial Intelligence Research*, 34:61–88, 2009. [Online]. Available: https://doi.org/10.1613/jair.2591.

[12] Tal Grinshpoun, Tamir Tassa, Vadim Levit, and Roie Zivan. Privacy preserving region optimal algorithms for symmetric and asymmetric dcops. *Artificial Intelligence*, 266:27–50, 2019. [Online]. Available: https://doi.org/10.1016/j.artint.2018.08.002.

[13] Patricia Gutierrez and Pedro Meseguer. Saving messages in adopt-based algorithms. In *Proc. 12th DCR workshop in AAMAS-10*, pages 53–64. Citeseer, 2010. [Online]. Available: http://www.iiia.csic.es/files/pdfs/redun-bnb-adopt-final.pdf.

[14] Patricia Gutierrez and Pedro Meseguer. Improving bnb-adopt+-ac. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 273–280. International Foundation for Autonomous Agents and Multiagent Systems, 2012. [Online]. Available: https://dl.acm.org/citation.cfm?id=2343615.

[15] Katsutoshi Hirayama and Makoto Yokoo. Distributed partial constraint satisfaction problem. In *International Conference on Principles and Practice of Constraint Programming*, pages 222–236. Springer, 1997. [Online]. Available: https://doi.org/10.1007/BFb0017442.

[16] Javier Larrosa and Thomas Schiex. In the quest of the best form of local consistency for weighted csp. In *IJCAI*, volume 3, pages 239–244, 2003. [Online]. Available: https://dl.acm.org/citation.cfm?id=1630694.

[17] Rajiv T Maheswaran, Milind Tambe, Emma Bowring, Jonathan P Pearce, and Pradeep Varakantham. Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 310–317. IEEE Computer Society, 2004. [Online]. Available: https://ink.library.smu.edu.sg/sis_research/935/.

[18] Toshihiro Matsui and Hiroshi Matsuo. A constraint based formalisation for distributed cooperative sensor resource allocation. *International Journal of Intelligent Information and Database Systems*, 4(4):307–321, 2010. [Online]. Available: https://doi.org/10.1504/IJIIDS.2010.035578.

[19] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005. [Online]. Available: https://doi.org/10.1016/j.artint.2004.09.003.

[20] Duc Thien Nguyen, William Yeoh, Hoong Chuin Lau, and Roie Zivan. Distributed gibbs: A linear-space sampling-based dcop algorithm. *Journal of Artificial Intelligence Research*, 64:705–748, 2019. [Online]. Available: https://doi.org/10.1613/jair.1.11400.

[21] Alexandra Olteanu, Thomas Léauté, and Boi Faltings. Asynchronous forward bounding (afb): Implementation and performance experiments. Technical report, 2011. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.385.4571.

[22] Mohamed Wahbi, Redouane Ezzahir, Christian Bessiere, and El-Houssine Bouyakhf. Dischoco 2: A platform for distributed constraint reasoning. *Proceedings of DCR*, 11:112–121, 2011. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.208.4895.

[23] Mohamed Wahbi, Redouane Ezzahir, and Christian Bessiere. Asynchronous forward bounding revisited. In *International Conference on Principles and Practice of Constraint Programming*, pages 708–723. Springer, 2013. [Online]. Available: https://doi.org/10.1007/978-3-642-40627-0_52.

[24] Mohamed Wahbi, Younes Mechqrane, Christian Bessiere, and Kenneth N Brown. A general framework for reordering agents asynchronously in distributed csp. In *International Conference on Principles and Practice of Constraint Programming*, pages 463–479. Springer, 2015. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/3102787.3102821.

[25] William Yeoh, Ariel Felner, and Sven Koenig. Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. *Journal of Artificial Intelligence Research*, 38:85–133, 2010. [Online]. Available: https://doi.org/10.1613/jair.2849.

Rachid Adrdor is currently a member of the Laboratory of Computer Information Systems and Vision of the Faculty of Science, Ibn Zohr University, Agadir, Morocco. He received the Ph.D. degree in Computer Science at the Faculty of Science of the same university. His research interests include Applied Artificial Intelligence, in particular, Distributed Constraint Optimization Problems (DCOPs) and Soft Arc Consistency.

Lahcen Koutti is currently a Professor at Department of Computer Science and a member of the Laboratory of Computer Information Systems and Vision of the Faculty of Science, Ibn Zohr University, Agadir, Morocco. He received the Ph.D. degree in Computational Physics from Paul Verlaine University, France and the Habilitation degree from Ibn Zohr University, Morocco. His research interests include Artificial Intelligence and Computer Vision.