# Asynchronous Forward-Bounding algorithm with Directional Arc Consistency

Rachid Adrdor[1], Lahcen Koutti[1]

[1]Ibn Zohr University, Faculty of Sciences, Department of Computer Science, Agadir, Morocco

**Abstract**

The AFB_BJ$^+$-AC$^*$ algorithm is one of the latest algorithms used to solve Distributed Constraint Optimization Problems known as DCOPs. It is based on soft arc consistency techniques (AC$^*$) to speed up the process of solving a problem by permanently removing any value that doesn't belong to the optimal solution. In fact, these techniques have greatly contributed to improving the performance of the AFB_BJ$^+$ algorithm in solving DCOPs, but there are some exceptions in which they have no effect due to the limited number of deletions made. For that, we use in this paper a higher consistency level, which is a directional arc consistency (DAC$^*$). This level makes it possible to erase more values and thus to quickly reach the optimal solution of a problem. Experiments on some benchmarks show that the new algorithm, AFB_BJ$^+$-DAC$^*$, is better in terms of communication load and computation effort.

**Keywords**

DCOP, AFB_BJ$^+$, AC$^*$, Directional Arc Consistency

## 1. Introduction

A large number of multi-agent problems can be modeled as DCOPs such as meetings scheduling [1], sensor networks [2], 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 [3]. 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 to a DCOP is a set of value assignments, each representing the value assigned to one of the variables in that DCOP. Algorithms with various search strategies have been suggested to solve DCOPs, for example, Adopt[4], BnB-Adopt[5], BnB-Adopt$^+$[6], SyncBB[7], AFB[3], AFB_BJ$^+$[8], AFB_BJ$^+$-AC$^*$[9, 10, 11], etc.

In AFB_BJ$^+$-AC$^*$, agents synchronously develop a current partial assignment (CPA) in order to find the optimal solution to the problem to be solved. During this process, and in order to reduce the number of retries, each agent uses arc consistency (AC$^*$) to remove any suboptimal values in its domain. But sometimes, the number of deletions generated by AC$^*$ is insufficient, which negatively affects the performance of the algorithm.

In this paper, instead of using the basic level of arc consistency (AC$^*$), we use directional arc consistency (DAC$^*$), which is the next higher level of AC$^*$. DAC$^*$ allows AFB_BJ$^+$ to generate

*ASPOCP 2021: Workshop on Answer Set Programming and Other Computing Paradigms 2021 co-located with ICLP 2021 Porto, Portugal, September 21, 2021*

rachid.adrdor@edu.uiz.ac.ma (R. Adrdor); l.koutti@uiz.ac.ma (L. Koutti)

more deletions and thus quickly reach the optimal solution of a problem. The new algorithm is called AFB_BJ$^+$-DAC$^*$. It uses DAC$^*$ to filter agent domains by performing a set of cost extensions from an agent to its neighbors, then executing AC$^*$. Our experiments on different benchmarks show the superiority of AFB_BJ$^+$-DAC$^*$ algorithm in terms of communication load and computation effort.

This paper comprises four sections. Section 2 gives an overview of DCOPs, soft arc consistency rules, and AFB_BJ$^+$-AC$^*$ algorithm. Section 3 gives a description of AFB_BJ$^+$-DAC$^*$ algorithm. Section 4 talks about experiments fulfilled on some benchmarks. The last section gives the conclusion.

## 2. Background

### 2.1. Distributed Constraint Optimization Problem (DCOP)

A DCOP [12, 13, 11] 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 finite domains $\mathcal{D} = \{D_1, D_2, ..., D_n\}$, where each $D_i$ in $\mathcal{D}$ contains the possible values for its associated variable $x_i$ 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, each DCOP is characterized in that each agent is responsible for a single variable and that two variables, at most, are linked by a constraint (i.e., unary or binary constraint) [14].

We consider these notations: $A_j$ is an agent, where $j$ is its level. $(x_j, v_j)$ 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 $x_j$, and $c_{ij}$ is its binary cost. $C_{ij}^{ac}$ is an identical copy of the $C_{ij}$ constraint, used in the AC$^*$ process. $C_j$ is a unary constraint on $x_j$ and $c_j$ is its unary cost. $C_\phi$ is a zero-arity constraint that represents a lower bound of any problem solution. $C_{\phi_j}$ is the contribution value of $A_j$ in $C_\phi$. $UB_j$ is the cost of the optimal solution reached so far and it is also the lowest unacceptable cost used for AC$^*$ process. $[A_1, A_2, \ldots, A_n]$ is the lexicographic ordering of agents (the default ordering), $\Gamma(x_j) = \{\Gamma^- : x_i \in \mathcal{X} \mid C_{ij} \subseteq D_i \times D_j,\ i < j\} \cup \{\Gamma^+ : x_i \in \mathcal{X} \mid C_{ij} \subseteq D_i \times D_j,\ i > j\}$ is the set of neighbors of $A_j$. $\Gamma^-$ (resp. $\Gamma^+$) is a 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 $A_j$. $LB_k[i][v_j]$ 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 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 $Y$ is the sum of $c_{ij}$ involved in $Y$ (1).

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

A CPA $Y$ is said to be a complete assignment (i.e., a solution) when it comprises a value assignment for each variable of a DCOP. Solving a DCOP is to find a solution such that the sum of the

**Proc. 1:** ProjectUnary()

```
1  β ← min_{v_i ∈ D_i} {c_i(v_i)} ;
2  C_{φ_i} ← C_{φ_i} + β ;
3  foreach (v_i ∈ D_i) do
4  |  c_i(v_i) ← c_i(v_i) − β ;
```

**Proc. 2:** ProjectBinary($x_i, x_j$)

```
1  foreach (v_i ∈ D_i) do
2  |  α ← min_{v_j ∈ D_j} {c_ij(v_i, v_j)} ;
3  |  foreach (v_j ∈ D_j) do
4  |  |  c_ij(v_i, v_j) ← c_ij(v_i, v_j) − α ;
5  |  if (A_i is the current agent)
6  |  |  c_i(v_i) ← c_i(v_i) + α ;
```

**Proc. 3:** Extend($x_i, x_j, E$)

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

**Proc. 4:** CheckPruning()

```
1  foreach (a ∈ D_j) do
2  |  if (c_j(a) + C_φ ≥ UB_j)
3  |  |  D_j ← D_j − {a} ;
4  |  |  DVals[j].listVals.add(a) ;
5  if (D_j is changed)
6  |  DVals[j].UnvNbrs ← A_j.Nbrs;
7  if (D_j is empty)
8  |  broadcastMsg : stp(UB_j) ;
9  |  end ← true ;
```

**Proc. 5:** DAC*()

```
1  foreach (A_k ∈ Γ+) do
2  |  foreach (v_j ∈ D_j) do
3  |  |  E[v_j] ← c_j(v_j) ;
4  |  Extend(x_j, x_k, E) ;
5  |  EVals[jk].put(E) ;
6  |  ProjectBinary(x_k, x_j) ;
```

**Proc. 6:** ProcessPruning(msg)

```
1  DVals ← msg.DVals ;
2  foreach (A_k ∈ Γ) do
3  |  foreach (a ∈ DVals[k]) do
4  |  |  D_k ← D_k − {a} ;
5  |  if (D_k is changed)
6  |  |  DVals[k].UnvNbrs.decr(−1) ;
7  |  if (DVals[k].UnvNbrs = 0)
8  |  |  DVals[k].listVals.clear ;
9  if (msg.type = ok?)
10 |  EVals ← msg.EVals ;
11 |  foreach (A_k ∈ Γ−) do
12 |  |  Extend(x_k, x_j, EVals[kj]) ;
13 |  |  EVals[kj].clear ;
14 |  |  ProjectBinary(x_j, x_k) ;
15 |  |  ProjectUnary() ;
16 C_φ ← max {C_φ, msg.C_φ} + C_{φ_j} ;
17 C_{φ_j} ← 0 ;
18 if (C_φ ≥ UB_j)
19 |  broadcastMsg : stp(UB_j) ;
20 |  end ← true ;
21 CheckPruning() ;
22 DAC*() ;
23 ExtendCPA() ;
```

costs of constraints involved in this solution is minimal, i.e., $Y^* = \arg\min_Y \{GC(Y) \mid var(Y) = \mathcal{X}\}$.

## 2.2. Soft arc consistency

Soft arc consistency techniques are used when solving a problem to delete values that are not part of the optimal solution of this problem. To apply these techniques to a problem, a set of transformations known as equivalence preserving transformations are used. 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 (Proc. 2) is an operation which subtracts, for a value $v_i$ of $D_i$, the smallest cost $\alpha$ of a binary constraint $C_{ij}$ and adds it to the unary constraint $C_i$. The unary projection (Proc. 1) is an operation which subtracts the smallest cost $\beta$ of a unary constraint $C_i$ and adds it to the zero-arity constraint $C_\phi$. The extension (Proc. 3) is an operation which 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 [15], 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 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 neighbors $x_{j(j>i)}$.

To make a given problem DAC$^*$, we first compute, for each variable $x_i$ with respect to its neighbors of lower priority $x_{j(j>i)}$, the extension values appropriate to the values of its domain $D_i$ (Proc. 5, $l.$ 3). Next, we perform the extension operation (Proc. 5, $l.$ 4) by subtracting the extension values from the unary constraints $C_i$ and adding them to the binary ones $C_{ij}$ (Proc. 3). Then, each neighbor $x_j$ performs, successively, a binary projection (Proc. 2), a unary projection (Proc. 1), and finally a deletion of non-NC$^*$ values. These last three instructions ensure the fulfillment of arc consistency (AC$^*$).

In a distributed case, each agent $A_i$ performs DAC$^*$ locally and shares the value of its zero-arity constraint $C_{\phi_i}$ with the other agents in order to calculate the global $C_\phi$ (i.e., $C_\phi = \sum_{A_i \in \mathcal{A}} C_{\phi_i}$)(Proc. 6, $l.$ 16). Each agent $A_i$ keeps locally for each of its constraints $C_{ij}$ an identical copy marked by $C_{ij}^{ac}$ and used in DAC$^*$ procedure. During DAC$^*$, $C_{ij}^{ac}$ 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 (Proc. 5, $l.$ 6) [16].

## 2.3. AFB_BJ$^+$-AC$^*$ algorithm

Each agent $A_j$ carries out the AFB_BJ$^+$-AC$^*$[9] according to three phases. First, $A_j$ initializes its data structures and performs the AC$^*$ in which it deletes permanently all 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 (2), its guaranteed cost in AC$^*$ (3), the $C_\phi$, and the list $DVals$.

$$GC(Y^j)[j] = GC(Y^{j-1}) + \sum_{(x_i,v_i)\in Y^{j-1} \mid i<j} c_{ij}(v_i, v_j) \tag{2}$$

$$\underset{(x_i,v_i)\in Y^{j-1}}{GC^*(Y^j)} = GC^*(Y^{j-1}) + c_j(v_j) + \sum_{c_{ij}^{ac}\in\mathcal{C}} c_{ij}(v_i, v_j) \tag{3}$$

Otherwise, that is to say, the agent $A_j$ 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 doesn't 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 (4) doesn't exceed the global upper bound $UB_j$, which represents the cost of the optimal solution achieved so far.

$$LB(Y^j)[i] = GC(Y^j)[i] + \sum_{A_k > A_j} LB_k(Y^j)[i] \tag{4}$$

Third, $A_j$ evaluates the extended CPA by sending **fb?** messages, which hold the same data structures excluding $C_\phi$ 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 $Y^i$ (5) is the minimal lower bound over all values of $D_j$ with respect to $Y^i$.

$$\begin{aligned} LB_j(Y^i)[h] = \min_{\substack{v_j \in D_j \\ (h \leq i < j)}} \Bigg\{ & \sum_{\substack{(x_k, v_k) \in Y^h \\ (k \leq h)}} c_{kj}(v_k, v_j) + \sum_{\substack{k=h+1 \\ (h<k<i)}}^{i-1} \min_{v_k \in D_k} \{c_{kj}(v_k, v_j)\} + \\ & c_{ij}(v_i, v_j) + \sum_{\substack{x_k \in \Gamma^+(x_j) \\ (k>j)}} \min_{v_k \in D_k} \{c_{jk}(v_j, v_k)\} \Bigg\} \end{aligned} \tag{5}$$

# 3. The AFB_BJ$^+$-DAC$^*$ algorithm

The AFB_BJ$^+$-DAC$^*$ algorithm uses a higher consistency level, which is a directional arc consistency (DAC$^*$). It improves the ability of AFB_BJ$^+$-AC$^*$ algorithm to generate more deletions. It is based on executing a set of cost extensions from unary constraints to binary ones, then on executing of AC$^*$. DAC$^*$() (Proc. 5) is the procedure responsible for calculating the extension values (i.e., costs to be transferred) and $Extend()$ (Proc. 3) is the one that performs the extension of costs from the unary constraints towards the binary ones (§2.2). All the extension values used by an agent are stored in a list, $EVals$, and routed to its lower neighbors via an **ok?** message in order to keep the symmetry of $C_{ij}^{ac}$ constraints in each agent and its neighbors. The list of extension values, $EVals$, is processed in the procedure $ProcessPruning()$ (Proc. 6, $l.$ 9-13) in which DAC$^*$() is also performed (Proc. 6, $l.$ 22).

## 3.1. Description of AFB_BJ$^+$-DAC$^*$

The AFB_BJ$^+$-DAC$^*$ (Proc. 7) is performed by each agent $A_j$ as follows :

$A_j$ starts with the initialization step (Proc. 7, $l.$ 1-3). If $A_j$ is the $1^{st}$ agent (Proc. 7, $l.$ 4), it filters its domain by calling $CheckPruning()$ (Proc. 4), then performs the appropriate extensions through DAC$^*$() (Proc. 5), and finally calls $ExtendCPA()$ to generate a CPA $Y$.

Next, $A_j$ starts processing the messages (Proc. 7, $l.$ 9). First, it updates $UB_j$ and $v_j^*$ (Proc. 7, $l.$ 12). 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 ($Y$) (Proc. 7, $l.$ 13). Thereafter, $A_j$ restores all temporarily deleted values (Proc. 7, $l.$ 28).

**Proc. 7:** AFB_BJ$^+$-DAC$^*$()

```
1  Init. of data structures
2  foreach (A_k ∈ Γ^+) do
3  |  LB_k[0][v_j] ← min_{v_k ∈ D_k} {c_jk(v_j, v_k)} ;
   |                 (v_j ∈ D_j)
4  if (A_j = A_1)
5  |  C_φ ← C_φ + C_φj ; C_φj ← 0;
6  |  CheckPruning() ;
7  |  DAC*() ;
8  |  ExtendCPA() ;
9  while (¬end) do
10 |  msg ← getMsg() ;
11 |  if (msg.UB < UB_j)
12 |  |  UB_j ← msg.UB ; v*_j ← v_j ;
13 |  if (msg.Y is stronger than Y)
14 |  |  Y ← msg.Y ; GC ← msg.GC ;
15 |  |  clear irrelevant LB_k[][] ; reset D_j ;
16 |  if (msg.type = ok?)
17 |  |  mustSendFB ← True ;
18 |  |  GC* ← msg.GC* ;
19 |  |  ProcessPruning(msg) ;
20 |  if (msg.type = back)
21 |  |  Y ← Y^{j-1} ;
22 |  |  ProcessPruning(msg) ;
23 |  if (msg.type = fb?)
24 |  |  GC* ← msg.GC* ;
25 |  |  foreach (v_j ∈ D_j) do
26 |  |  |  cost ← C_φ + GC*(Y^{j-1}) + c_j(v_j) ;
27 |  |  |  if (cost ≥ UB_j)
28 |  |  |  |  D_j ← D_j − v_j ;
29 |  |  sendMsg : lb to A_i (LB_j(Y^i)[], msg.Y) ;
30 |  if (msg.type = stp)
31 |  |  end ← true ;
32
33 if (msg.type = lb)
34 |  LB_k(Y^j) ← msg.LB ;
35 |  if (LB(Y^j) ≥ UB_j)
36 |  |  ExtendCPA() ;
```

**Proc. 8:** ExtendCPA()

```
1  v_j ← argmin_{v'_j ∈ D_j} { LB(Y ∪ (x_j, v'_j)) } ;
2  if (LB(Y ∪ (x_j, v_j)) ≥ UB_j) ∨
     (C_φ + GC*(Y^{j-1}) + c_j(v_j) ≥ UB_j)
3  |  for i ← j − 1 to 1 do
4  |  |  if (LB(Y)[i − 1] < UB_j)
5  |  |  |  sendMsg : back to A_i (Y^i, UB_j, DVals, C_φ) ;
   |  |  |    return ;
6  |  broadcastMsg : stp(UB_j) ;
7  |  end ← true ;
8  else
9  |  Y ← {Y ∪ (x_j, v_j)} ;
10 |  if (var(Y) = X)
11 |  |  UB_j ← GC(Y) ; v*_j ← v_j ;
12 |  |  Y ← Y^{j-1} ;
13 |  |  CheckPruning() ;
14 |  |  ExtendCPA() ;
15 |  else
16 |  |  sendMsg : ok? to A_{j+1} (Y, GC, UB_j, DVals,
   |  |    EVals, C_φ, GC*) ;
17 |  |  EVals.clear ;
18 |  |  if (mustSendFB)
19 |  |  |  sendMsg : fb? to A_k, k>j (Y, GC, UB_j, GC*) ;
20 |  |  |  mustSendFB ← false ;
```

When receiving an **ok?** message (Proc. 7, $l.$ 16), $A_j$ authorizes the sending of **fb?** messages and calls $ProcessPruning()$ (Proc. 6).

When calling $ProcessPruning()$ (Proc. 6), $A_j$ deals initially, for **ok?** messages only, with extensions of its higher neighbors (Proc. 6, $l.$ 9-13). Afterward, it updates its $DVals$, then its neighbors' domains separately in order to keep the same domains as these agents (Proc. 6, $l.$ 1-4). After that, it performs the two projections fulfilling the condition of AC$^*$ (Proc. 6, $l.$ 14-15). 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 $DVals[k].listVals$ (Proc. 6, $l.$ 5-8). Then, $A_j$ updates its global $C_\phi$ (Proc. 6, $l.$ 16). If $C_\phi$ exceeds the $UB_j$, $A_j$ turns off its execution and notifies the others (Proc. 6, $l.$ 18-20). Finally, $A_j$ calls $CheckPruning()$ to prune its domain, DAC$^*$() (Proc. 5) to make the proper extensions, and $ExtendCPA()$ to extend the received CPA (Proc. 6, $l.$ 21-23).

When calling DAC$^*$() (Proc. 5), $A_j$ performs the proper extensions from $C_j$ to each $C_{ij}$ (Proc. 5, $l.$ 4-5). To do that, $A_j$ calculates, for each value $v_j$ of $D_j$, its extension value (Proc. 5, $l.$ 2-3) based on the unary cost of this value ($0 < E[v_i] \leq c_i(v_i)$). Once completed, $A_j$ performs a binary projection to keep the symmetry of $C_{ij}^{ac}$ constraints (Proc. 5, $l.$ 6). 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 $CheckPruning()$ (Proc. 4), $A_j$ deletes any value from its domain for which the sum of the $C_\phi$ with the unary cost of this value exceeds $UB_j$ (Proc. 4, $l.$ 2-3). With each new deletion, $A_j$ initializes the number of its neighbors not yet visited (Proc. 4, $l.$ 5-6). If $A_j$ domain becomes empty, $A_j$ turns off its execution and notifies the others (Proc. 4, $l.$ 7-9).

When calling $ExtendCPA()$ (Proc. 8), $A_j$ looks for a value $v_j$ for its variable $x_j$ (Proc. 8, $l.$ 1). If no value exists, $A_j$ returns to the priority agents by sending a **back** message to the contradictory agent (Proc. 8, $l.$ 2-5). If no agent exists, $A_j$ turns off its execution and notifies the others via **stp** messages (Proc. 8, $l.$ 6-7). Otherwise, $A_j$ extends $Y$ by adding its assignment (Proc. 8, $l.$ 9). If $A_j$ is the last agent (Proc. 8, $l.$ 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 (Proc. 8, $l.$ 11-14). Otherwise, $A_j$ sends an **ok?** message loaded with the extended $Y$ to the next agent (Proc. 8, $l.$ 16) and **fb?** messages to unassigned agents (Proc. 8, $l.$ 19).

When $A_j$ receives an **fb?** message, it filters its domain $D_j$ with respect to the received $Y$ (Proc. 7, $l.$ 24-28), calculates the appropriate lower bounds (5), and immediately sends them to the sender via **lb** message (Proc. 7, $l.$ 29).

When $A_j$ receives an **lb** message, it stores the lower bounds received (Proc. 7, $l.$ 34) and performs $ExtendCPA()$ to modify its assignment if the lower bound calculated, based on the cost of $Y$ (4), exceeds the $UB_j$.

### 3.2. Correctness of AFB_BJ$^+$-DAC$^*$

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

*Proof.* The AFB_BJ$^+$-DAC$^*$ algorithm outperforms AFB_BJ$^+$-AC$^*$ [9] by executing a set of cost extensions. These extensions have already been proved which are correct in [15, 17], and they are executed by the AFB_BJ$^+$-DAC$^*$ without any cost redundancy (Proc. 3, $l.$ 4), (Proc. 5, $l.$ 6), and (Proc. 6, $l.$ 9-13). □

## 4. Experimental Results

In this section, we experimentally compare the AFB_BJ$^+$-DAC$^*$ algorithm with its older versions [8, 9] and with the BnB-Adopt$^+$-DP2 algorithm [18], which is its famous competitor. Two benchmarks are used in these experiments: meetings scheduling and sensors network.

**Meetings scheduling** [1]: are defined by $(m,\ p,\ ts)$, which are respectively the number of meetings/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 the same cases presented in detail in [1]. These cases are A, B, C, and D, each representing a different scenario in terms of the number of meetings and the number of participants who each have a different hierarchical level.

**Sensors network** [2]: are defined by $(t,\ s,\ d)$, which are respectively the number of targets/variables, the number of sensors, and the number of possible combinations of 3 sensors

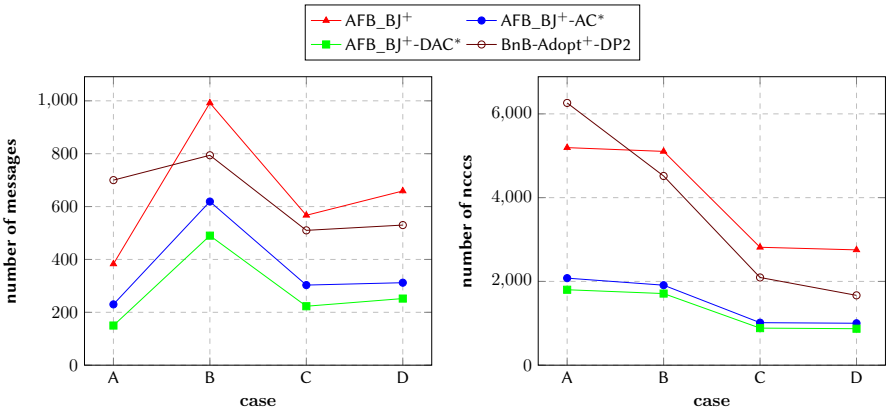


**Figure 1:** Total of messages ($msgs$) sent and non-concurrent constraint checks ($ncccs$) for meetings scheduling

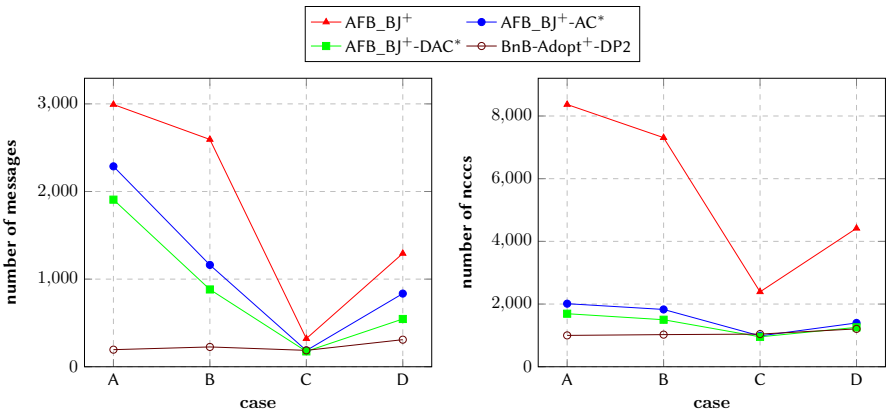


**Figure 2:** Total of messages ($msgs$) sent and non-concurrent constraint checks ($ncccs$) for sensors network

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 the same cases presented in detail in [1]. These cases are A, B, C, and D, each representing a different scenario in terms of the number of targets and the number of sensors which are arranged in different topologies.

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.

Regarding meetings scheduling problems (Fig. 1), the results show a clear improvement of the AFB_BJ$^+$-DAC$^*$ compared to others, whether for $msgs$ or for $ncccs$. But with regard to sensors network problems (Fig. 2), the BnB-Adopt$^+$-DP2 retains the pioneering role, despite the superiority of the AFB_BJ$^+$-DAC$^*$ algorithm to its older versions.

By analyzing the results, we can conclude that the AFB_BJ$^+$-DAC$^*$ is better than its earlier versions, because of the existence of directional arc consistency (DAC$^*$) which allows agents to

remove more suboptimal values. This is due to a set of cost extensions applied to the problem. Regarding the superiority of the BnB-Adopt$^+$-DP2 over the AFB_BJ$^+$-DAC$^*$ 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 facilitates the proper choice of values.

## 5. Conclusion

In this paper, we have introduced the AFB_BJ$^+$-DAC$^*$ algorithm. It is an algorithm that relies on DAC$^*$ to generate more deletions and thus quickly reach the optimal solution of a problem. DAC$^*$ mainly relies on performing a set of cost extensions in one direction from an agent to its lower priority neighbors in order to perform AC$^*$ multiple times, which increases the number of deletions made by each agent and thereby speed up the process of solving a problem. Experiments on some benchmarks show that the AFB_BJ$^+$-DAC$^*$ algorithm behaves better than its older versions. As future work, we propose to exploit the change in the size of the agent domains in variable ordering heuristics.

## References


[1] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, P. 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, IEEE Computer Society, 2004, pp. 310–317.

[2] R. Béjar, C. Domshlak, C. Fernández, C. Gomes, B. Krishnamachari, B. Selman, M. Valls, Sensor networks and distributed csp: communication, computation and complexity, Artificial Intelligence 161 (2005) 117–147.

[3] A. Gershman, A. Meisels, R. Zivan, Asynchronous forward bounding for distributed cops, Journal of Artificial Intelligence Research 34 (2009) 61–88.

[4] P. J. Modi, W.-M. Shen, M. Tambe, M. Yokoo, Adopt: Asynchronous distributed constraint optimization with quality guarantees, Artificial Intelligence 161 (2005) 149–180.

[5] W. Yeoh, A. Felner, S. Koenig, Bnb-adopt: An asynchronous branch-and-bound dcop algorithm, Journal of Artificial Intelligence Research 38 (2010) 85–133.

[6] P. Gutierrez, P. Meseguer, Saving messages in adopt-based algorithms, in: Proc. 12th DCR workshop in AAMAS-10, Citeseer, 2010, pp. 53–64.

[7] K. Hirayama, M. Yokoo, Distributed partial constraint satisfaction problem, in: International Conference on Principles and Practice of Constraint Programming, Springer, 1997, pp. 222–236.

[8] M. Wahbi, R. Ezzahir, C. Bessiere, Asynchronous forward bounding revisited, in: International Conference on Principles and Practice of Constraint Programming, Springer, 2013, pp. 708–723.

[9] R. Adrdor, R. Ezzahir, L. Koutti, Connecting afb_bj$^+$ with soft arc consistency, International Journal of Computing and Optimization 5 no. 1 (2018) 9–20.

[10] R. Adrdor, L. Koutti, Enhancing AFB_BJ$^{+}$AC$^{*}$ algorithm, in: 2019 International Conference of Computer Science and Renewable Energies (ICCSRE), IEEE, 2019, pp. 1–7.
[11] R. Adrdor, R. Ezzahir, L. Koutti, Consistance d'arc souple appliquée aux problèmes dcop, Journées d'Intelligence Artificielle Fondamentale (JIAF) (2020) 63.
[12] T. Grinshpoun, T. Tassa, V. Levit, R. Zivan, Privacy preserving region optimal algorithms for symmetric and asymmetric dcops, Artificial Intelligence 266 (2019) 27–50.
[13] F. Fioretto, E. Pontelli, W. Yeoh, Distributed constraint optimization problems and applications: A survey, Journal of Artificial Intelligence Research 61 (2018) 623–698.
[14] D. T. Nguyen, W. Yeoh, H. C. Lau, R. Zivan, Distributed gibbs: A linear-space sampling-based dcop algorithm, Journal of Artificial Intelligence Research 64 (2019) 705–748.
[15] J. Larrosa, T. Schiex, In the quest of the best form of local consistency for weighted csp, in: IJCAI, volume 3, 2003, pp. 239–244.
[16] P. Gutierrez, P. Meseguer, Improving bnb-adopt+-ac, in: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1, International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 273–280.
[17] M. C. Cooper, S. De Givry, M. Sánchez, T. Schiex, M. Zytnicki, T. Werner, Soft arc consistency revisited, Artificial Intelligence 174 (2010) 449–478.
[18] S. Ali, S. Koenig, M. Tambe, Preprocessing techniques for accelerating the dcop algorithm adopt, in: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, ACM, 2005, pp. 1041–1048.