# Using Directional Arc Consistency with Asynchronous Forward-Bounding algorithm

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 (DCOPs). It is based on simple arc consistency (AC$^*$) to speed up the process of solving a problem by permanently removing any value that doesn't belong to its optimal solution. In this paper, we use a directional arc consistency (DAC$^*$), the next higher level of AC$^*$, 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]. 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 the constraints that involve them [2]. A solution to a DCOP is a set of value assignments, each representing the value assigned to one of the variables in that DCOP.

To solve DCOPs, algorithms with different search strategies have been suggested in the literature, for example, Adopt[3], BnB-Adopt[4], BnB-Adopt$^+$[5], BnB-Adopt$^+$-AC$^*$[6], SyncBB[7], AFB[2], AFB_BJ$^+$[8], etc. AFB_BJ$^+$-AC$^*$[9, 10, 11] is one of the recent algorithms that uses arc consistency (AC$^*$) to solve DCOPs.

In this paper, instead of using the basic level of arc consistency (AC$^*$), we use directional arc consistency (DAC$^*$). DAC$^*$ allows AFB_BJ$^+$ to generate more deletions and thus quickly reach the optimal solution to 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.

## 2. Background

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

A DCOP [11, 12, 13] is defined by 4 sets, agents $\mathcal{A} = \{A_1, A_2, ..., A_k\}$, variables $\mathcal{X} = \{x_1, x_2, ..., x_n\}$, domains $\mathcal{D} = \{D_1, D_2, ..., D_n\}$, where each $D_i$ contains the possible values for $x_i$, and constraints $\mathcal{C} = \{C_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+\} \cup \{C_i : D_i \rightarrow \mathbb{R}^+\}$. In this article, we consider that each agent of a given DCOP is responsible for a single variable and that at most two variables are related 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 constraint between $x_i$ and $x_j$. $C_j$ is a constraint on $x_j$. $C_\phi$ 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. $[A_1, A_2, \ldots, A_n]$ 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

*OVERLAY 2021: 3rd Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, September 22, 2021, Padova, Italy*

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

**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:** 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. 3:** DAC*()

```
1 foreach (a ∈ D_j) do
2 |  if (c_j(a) + C_φ ≥ UB_j)
3 |  |  D_j ← D_j − a ; DVals[j].add(a) ;
4 foreach (A_k ∈ Γ+) do
5 |  foreach (v_j ∈ D_j) do
6 |  |  E[v_j] ← c_j(v_j) ;
7 |  Extend(x_j, x_k, E) ;
8 |  EVals[jk].put(E) ;
9 |  ProjectBinary(x_k, x_j) ;
```

**Proc. 4:** 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. 5:** 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  EVals ← msg.EVals ;
6  foreach (A_k ∈ Γ−) do
7  |  Extend(x_k, x_j, EVals[kj]) ;
8  |  EVals[kj].clear ;
9  |  ProjectBinary(x_j, x_k) ;
10 |  ProjectUnary() ;
11 C_φ ← max {C_φ, msg.C_φ} + C_{φ_j} ;
12 C_{φ_j} ← 0 ;
13 DAC*() ;
14 ExtendCPA() ;
```

of $A_j$. $Y = Y^j = [(x_1, v_1), \ldots, (x_j, v_j)]$ is a current partial assignment (CPA) or a feasible solution. $lb_k[i][v_j]$ are the lower bounds of a lower neighbor $A_k$ obtained for $Y^j$. $DVals$ is a list of arrays containing values deleted by each agent $A_j$. $EVals$ is a list of arrays containing extension values. $GC$ (resp. $GC^*$) is the guaranteed cost of $Y$ (resp. in AC*).

$$GC(Y) = GC^*(Y) = \sum_{C_{ij}, C_i, C_j \in \mathcal{C}} c_{ij}(v_i, v_j) + c_i(v_i) + c_j(v_j) \mid (x_i, v_i),\ (x_j, v_j)\ \in Y$$

## 2.2. Soft arc consistency

Soft arc consistency techniques are used when solving a problem to delete values that are not part of its optimal solution. They are based on three operations:

The binary projection (Proc. 4) 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. 2) 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 operations 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 its variables are NC*.

**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 any 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. 3, $l.$ 6). Next, we perform the extension operation (Proc. 3, $l.$ 7) by subtracting the extension values from the unary constraints $C_i$ and adding them to the binary ones $C_{ij}$ (Proc. 2). Then, each neighbor $x_j$ performs, successively, a binary projection (Proc. 4), a unary projection (Proc. 1), and finally a deletion of non-NC* values.

## 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

**Proc. 6:** AFB_BJ$^+$-DAC*()

```
1  Init. of data structures
2  if (A_j = A_1)
3  |  C_φ ← C_φ + C_φ_j; C_φ_j ← 0;
4  |  DAC*();
5  |  ExtendCPA();
6  while (¬end) do
7  |  msg ← getMsg();
8  |  if (msg.UB < UB_j)
9  |  |  UB_j ← msg.UB;
10 |  if (msg.Y is stronger than Y)
11 |  |  Y ← msg.Y; GC ← msg.GC;
12 |  if (msg.type = ok?)
13 |  |  mustSendFB ← True;
   |  |    GC* ← msg.GC*;
14 |  |  ProcessPruning(msg);
15 |  if (msg.type = back)
16 |  |  Y ← Y^{j-1}; ProcessPruning(msg);
17 |  if (msg.type = fb?)
18 |  |  sendMsg : lb to A_i (lb_j(Y^i)[], msg.Y);
19 |  if (msg.type = lb)
20 |  |  lb_k(Y^j) ← msg.lb;
21 |  |  if (lb(Y^j) ≥ UB_j) ExtendCPA();
```

**Proc. 7:** ExtendCPA()

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

extended the CPA, it sends an **ok?** message to the next agent asking it to continue the extension of CPA $Y^j$. 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 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 doesn't exceed the global upper bound, which represents the cost of the optimal solution achieved so far. Third, $A_j$ evaluates the extended CPA by sending **fb?** messages 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 $Y^i$ is the minimal lower bound over all values of $D_j$ with respect to $Y^i$.

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

The AFB_BJ$^+$-DAC* algorithm (Proc. 6) is the improved version of the AFB_BJ$^+$-AC* algorithm, which uses DAC* to further reduce the domains of a given DCOP. AFB_BJ$^+$-DAC* follows the same steps as the AFB_BJ$^+$-AC* (§2.3), except that it performs DAC* instead of AC* before each extension of CPA (Proc. 7). DAC* is based on executing a set of cost extensions from unary constraints to binary ones, then on executing of AC*. DAC*() (Proc. 3) is the procedure responsible for calculating the extension costs (i.e., costs to be transferred) and $Extend()$ (Proc. 2) is the one that performs the extension of costs from the unary constraints towards the binary ones (§2.2). All the extension costs 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. 5, $l.$ 7-8).

**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, 16], and they are executed by the AFB_BJ$^+$-DAC* without any cost redundancy (Proc. 2, $l.$ 4), (Proc. 3, $l.$ 9), and (Proc. 5, $l.$ 7-8). □

## 4. Experimental Results

We experimentally compare AFB_BJ$^+$-DAC* with its older versions [8, 9] and with the BnB-Adopt$^+$-DP2 algorithm [17], which is its famous competitor. Two benchmarks are used in these experiments:

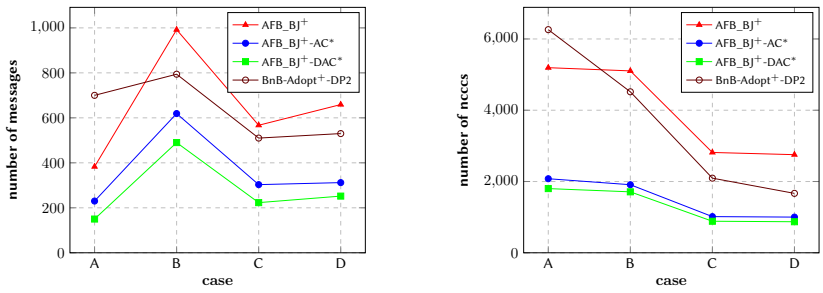


**Figure 1:** Total of $msgs$ sent and $ncccs$ for meetings scheduling

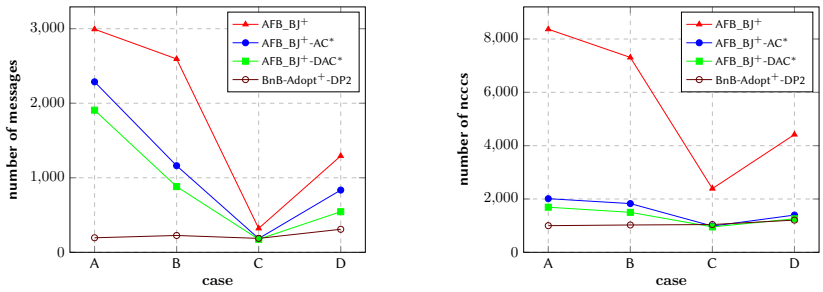


**Figure 2:** Total of $msgs$ sent and $ncccs$ for sensors network

**Meetings scheduling** [1]: are defined by the number of meetings/variables, the number of participants, and the number of time slots for each meeting. We have evaluated 4 cases A, B, C, and D, which are different in terms of meetings/participants [1].

**Sensors network** [18]: are defined by the number of targets/variables, the number of sensors, and the number of possible combinations of 3 sensors reserved for tracking each target. We have evaluated 4 cases A, B, C, and D, which are different in terms of targets/sensors [1].

To compare the algorithms, we use two metrics, the total of messages exchanged ($msgs$) for the communication load and the total of non-concurrent constraint checks ($ncccs$) for 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* 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 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 based on DAC* to increase the number of deletions made by each agent on its domain. 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 and thus generate more deletions of non-optimal values. 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] A. Gershman, A. Meisels, R. Zivan, Asynchronous forward bounding for distributed cops, Journal of Artificial Intelligence Research 34 (2009) 61–88.

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

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

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

[6] 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.

[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] M. C. Cooper, S. De Givry, M. Sánchez, T. Schiex, M. Zytnicki, T. Werner, Soft arc consistency revisited, Artificial Intelligence 174 (2010) 449–478.

[17] 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.

[18] 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.