International Journal of Computing and Optimization
Vol. 5, 2018, no. 1, 9 - 20
HIKARI Ltd, www.m-hikari.com
https://doi.org/10.12988/ijco.2018.857

Connecting AFB_BJ$^+$ with Soft Arc Consistency

Rachid Adrdor

Department of Computer Science
FSA Agadir, B.P 8106, Agadir, Morocco

Redouane Ezzahir

Department of Computer Science
ENSA Agadir, B.P 1136, Agadir, Morocco

Lahcen Koutti

Department of Computer Science
FSA Agadir, B.P 8106, Agadir, Morocco

Abstract

The Distributed Constraint Optimization Problem (DCOP) is a major and powerful paradigm for modeling and solving problems in multi-agent coordination. AFB_BJ$^+$ is one of the most excellent algorithms for solving DCOPs. Recently, researchers have shown that including soft arc consistency (**A**C*) in DCOP algorithms causes significant improvements in their performance. In this paper, we introduce AFB_BJ$^+$-**A**C* algorithm which connects AFB_BJ$^+$ with soft arc consistency (**A**C*). It relies on pruning non-optimal values of an agent domain, using **A**C*, and propagating them, without adding other types of messages, for generating other deletions that will likewise be propagated. Our experimental analysis on several benchmarks shows that thanks to **A**C*, AFB_BJ$^+$-**A**C* improves the basic AFB_BJ$^+$.

Keywords: DCOP, AFB_BJ$^+$, Soft Arc Consistency

1 Introduction

The Distributed Constraint Optimization Problem (DCOP) is a major and powerful paradigm for modeling and solving multi-agent problems such as meetings scheduling [9], sensor networks [2], etc. A DCOP is composed of a set of autonomous agents, where each agent has control only on its variables and the constraints that connect them [3]. In DCOP, agents try, in a distributed manner, to assign values to their variables such that the sum of the costs of all constraints is minimized.

Several distributed algorithms have been introduced to solve DCOP. The famous asynchronous algorithms are Adopt [10] and BnB-Adopt [13]. BnB-Adopt performs better than Adopt because of using a depth-first search strategy instead of a best-first search in Adopt. However, Gutierrez and Meseguer show that Adopt and BnB-Adopt use some unnecessary messages in the search [6]. For that, they try to remove them, resulting in two more efficient algorithms, Adopt$^+$ and BnB-Adopt$^+$.

The synchronous branch and bound (SyncBB) [7] is a synchronous algorithm where only one agent is allowed to assign its variables while the others, of sequence, remain idle. Once that agent assigns its variables, it gives the right to the next agent and then remains idle.

Asynchronous Forward Bounding (AFB)[3] is an improvement of SyncBB. In AFB, agents try to extend a current partial assignment (CPA) in a way that the lower bound on its cost doesn't exceed the cost of the best solution found so far (i.e., the global upper bound). The lower bounds are computed by sending, simultaneously, copies of CPA to unassigned agents. When all lower bounds of an agent exceed the upper bound, it backtracks to the last assigned agent. The AFB has been enhanced to the AFB_BJ algorithm [3] by using a backjumping mechanism instead of backtracking. Later, the AFB_BJ has been enhanced to the AFB_BJ$^+$ algorithm [11] by changing the value ordering strategy from lowest-cost-first to promising-first and computing lower bounds for the entire domain of the last assigned agent.

BnB-Adopt$^+$-**A**C* [4, 5] is an example of DCOP algorithm that combine *entirely asynchronous* search with soft arc consistency. It relies on propagating the values deleted, unconditionally, using **A**C*.

In this paper, we introduce AFB_BJ$^+$-**A**C* algorithm which connects the *slightly asynchronous* algorithm AFB_BJ$^+$ with soft arc consistency (**A**C*). It relies on pruning non-optimal values of an agent domain, using **A**C*, and propagating them, without adding other types of messages, for generating other deletions that will likewise be propagated.

This paper is constructed as follows. Section 2 gives a background on

DCOP, AFB_BJ$^+$ algorithm and soft arc consistency. We describe the AFB_BJ$^+$-**A**C* algorithm in Section 3. In Section 4, we present the experimental results conducted on several benchmarks. Finally, in Section 5, we conclude.

2 Background

2.1 DCOP

Distributed Constraint Optimization Problem (DCOP) is represented as a tuple $(\mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C})$ [10], where $\mathcal{A} = \{A_1, A_2, ..., A_k\}$ is a set of agents, $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ is a set of variables, $\mathcal{D} = \{D_1, D_2, ..., D_n\}$ is a set of domains, where each D_i in $\mathcal{D}$ is a finite set of possible values for its associated variable x_i. Each agent has the right to assign values only to its variables and to know their domains. $\mathcal{C} = \{f :_{x_j \in \mathcal{X}} D_j \to \mathbb{R}^+\}$ is a set of soft constraints. In this paper, without loss of generality, we only consider a DCOP where each agent controls exactly one variable and each constraint $f \in \mathcal{C}$ involves one or two variables (i.e., unary or binary constraint respectively). We identify the binary (resp. unary) constraint involving x_i and x_j (resp. x_i) by C_{ij} (resp. C_i) and its cost $f(x_i, x_j)$ (resp. $f(x_i)$) by c_{ij} (resp. c_i). Let A_j be the current agent with j is its level and (x_j, v_j) be an assignment of A_j where $v_j \in D_j$. Let $[A_1, A_2, \ldots, A_n]$ be the lexicographic ordering of agents and $\Gamma(x_j)$ be the set of neighbors of A_j. A_j is a borderline between agents (resp. neighbors) with a higher priority ($A_k < A_j$)(resp. Γ^-) and those with a lower priority ($A_k > A_j$) (resp. Γ^+). A neighbor of A_j is an agent that shares a constraint with it. Let $Y = Y^j = [(x_1, v_1), \ldots, (x_j, v_j)]$ be a current partial assignment (CPA). The guaranteed cost of Y, $GC(Y)$, is the sum of all costs c_{ij} s.t. x_i and x_j are assigned in Y (Eq.1).

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

Each Y becomes a full assignment when Y includes all variables of the problem, i.e., $var(Y) = \mathcal{X}$. The purpose of DCOP solvers is to reach a full assignment Y^* with a minimal cost, i.e., $Y^* = \arg\min_{Y}\{GC(Y) \mid var(Y) = \mathcal{X}\}$.

2.2 AFB_BJ$^+$ algorithm

In AFB_BJ$^+$[11], each agent A_j computes the future cost for each value in its domain constrained with every lower priority neighbor A_k (Eq.2). Agents compute these costs only once and store them.

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

Then, A_j assigns its variable and sends an **O**k? message to the next agent and **F**B? messages to unassigned agents. All these messages contain Y^j and

Proc 1: ProjectUnary()

$\beta \leftarrow min_{v_i \in D_i} \{c_i(v_i)\}$; $\mathbf{C}_{\phi_i} \leftarrow \mathbf{C}_{\phi_i} + \beta$;
foreach $(v_i \in D_i)$ **do** $c_i(v_i) \leftarrow c_i(v_i) - \beta$;

Proc 2: ProjectBinary(x_i, x_j)

foreach $(v_i \in D_i)$ **do**
    $\alpha \leftarrow min_{v_j \in D_j} \{c_{ij}(v_i, v_j)\}$;
    foreach $(v_j \in D_j)$ **do**
        $c_{ij}(v_i, v_j) \leftarrow c_{ij}(v_i, v_j) - \alpha$;
    if $(A_i$ *is current agent*$)$
        $c_i(v_i) \leftarrow c_i(v_i) + \alpha$;

Proc 3: ProcessPruning(msg)

$DelVals \leftarrow msg.DelVals$;
/* Γ is set of neigbors of A_j */
foreach $(A_k \in \Gamma)$ **do**
    foreach $(a \in DelVals[k])$ **do**
        $D_k \leftarrow D_k - a$;
    $ProjectBinary(x_j, x_k)$;
    $ProjectUnary()$;
$\mathbf{C}_\phi \leftarrow max\{\mathbf{C}_\phi, msg.\mathbf{C}_\phi\} + \mathbf{C}_{\phi_j}$;
$\mathbf{C}_{\phi_j} \leftarrow 0$;
if $(\mathbf{C}_\phi \geq UB_j)$
    broadcastMsg : **stp**(UB_j) ; $end \leftarrow true$;
$CheckPruning()$; $ExtendCPA()$;

an array of guaranteed costs (Eq.3), one for each level.

$$GC(Y^j)[j] = GC(Y^j) = GC(Y^{j-1}) + LC_j(Y^j, v_j)[j-1] \tag{3}$$

where $LC_j(Y^j, v_j)[h]$ (Eq.4) is the local cost, at level h, of assigning a value v_j to A_j w.r.t Y^h.

$$LC_j(Y^j, v_j)[h] = LC_j(Y^h, v_j) = \sum_{(x_k, v_k) \in Y^h} c_{kj}(v_k, v_j) \; s.t. \; k \leq h < j \tag{4}$$

When an A_j receives a **F**B? message, it responds by sending a **L**B message containing arrays of lower bounds. The lower bound at level h (Eq.5) is the minimal lower bound over all values in D_j w.r.t Y^h.

$$LB_j(Y^i)[h] = \min_{v_j \in D_j} \left\{ LC_j(Y^i, v_j)[h] + \sum_{m=h+1}^{i-1} \min_{v_m \in D_m} \{c_{mj}(v_m, v_j)\} + c_{ij}(v_i, v_j) + FC_j(v_j) \right\} \tag{5}$$

Once an A_j receives arrays of lower bounds, it calculates a lower bound on the cost of any full assignment (Eq.6).

$$LB(Y^j)[i] = GC(Y^j)[i] + \sum_{A_k > A_j} LB_k(Y^j)[i] \tag{6}$$

These lower bounds are used by agents to determine the backtracking level [3]. In AFB_BJ$^+$, agents maintain the valid lower bounds with respect to the most recent time-stamps and use them in the promising value ordering heuristic. They also avoid the redundant messages by sending **F**B? messages only for the parts of lower bounds that are discarded [11].

2.3 Soft arc consistency

Soft arc consistency operations are used to transform a constraint optimization problem (COP) into an equivalent one by shifting costs between constraints. Let (x_i, v_i) be an assignment, C_{ij} be a binary constraint between x_i and x_j, C_i be a unary constraint on x_i, $\mathbf{C}_\phi$ be a zero-arity constraint that represents a lower bound of any full assignment and $\top$ be the lowest unacceptable cost (i.e.,

Proc 4: AC* ()

```
1 foreach (A_k ∈ Γ+) do
2     ProjectBinary(x_j, x_k);
3     ProjectUnary();
4     ProjectBinary(x_k, x_j);
5 foreach (A_k ∈ Γ−) do
6     ProjectBinary(x_k, x_j);
7     ProjectBinary(x_j, x_k);
8     ProjectUnary();
```

Proc 5: CheckPruning()

```
1 foreach (a ∈ D_j) do
2     if (c_j(a) + C_φ ≥ UB_j)
3         D_j ← D_j − a;
4 foreach (A_k ∈ Γ) do
5     ProjectBinary(x_k, x_j);
6 if (D_j is empty)
7     broadcastMsg : stp(UB_j); end ← true;
```

the global upper bound). We consider the following soft local consistencies [8]:

Node Consistency (NC*): (x_i, v_i) is NC* if $\mathbf{C}_\phi + c_i(v_i) < \top$. A variable x_i is **NC*** if any value $v_i \in D_i$ is **NC*** and there is a value $v_i \in D_i$ which satisfies $c_i(v_i) = 0$. A problem is **NC*** if any variable $x_i \in \mathcal{X}$ is **NC*** .

Arc Consistency (AC*): (x_i, v_i) is AC w.r.t C_{ij} if there is a value $v_j \in D_j$ which satisfies $c_{ij}(v_i, v_j) = 0$, v_j called a support of v_i. A variable x_i is AC w.r.t C_{ij} if any value $v_i \in D_i$ has a support in D_j. A problem is **A**C* if any variable $x_i \in \mathcal{X}$ of this problem is **NC*** and AC.

AC* can be enforced as follows. Firstly, by a binary projection (P.2) which projects, for each value v_i of D_i, the smallest cost α w.r.t each C_{ij} of the problem to C_i. Secondly, by a unary projection (P.1) which projects the smallest cost β of C_i to $\mathbf{C}_\phi$. Finally, by removing not **NC*** values.

AC* can be extended to work in a distributed environment with some changes. Firstly, to obtain the global $\mathbf{C}_\phi$, each agent A_i stores temporarily its contribution value in a local variable $\mathbf{C}_{\phi_i}$ (P.1, l.1). Secondly, each agent, A_i or A_j, of a pair of agents that shares a constraint C_{ij}, maintains an identical copy of this C_{ij}. To maintain the same copy of C_{ij} in each agent, during **A**C* transformations, A_i must simulate the action of A_j on its C_{ij} and vice versa (P.4, l.4, 6) (P.5, l.5), but only one of them projects its unary costs on $\mathbf{C}_\phi$ (P.2, l.5) to avoid counting the same cost twice in $\mathbf{C}_\phi$ [5].

3 Integrating AC* in AFB_BJ$^+$ algorithm

There are different ways to integrate **A**C* with a distributed search algorithm, but the successful way is the one that causes an improvement in its performance in terms of communication load and computation effort. One of those ways is already used in BnB-Adopt$^+$-**A**C* algorithm [5], but it's not useful with AFB_BJ$^+$. For that, we propose another way to integrate **A**C* with AFB_BJ$^+$. It relies on pruning non-optimal values of an agent domain, using **A**C*, and propagating them, without adding other types of messages, for generating other deletions that will likewise be propagated. To perform this proposal, we consider (1) two identical copies of constraints, C_{ij} and C^{ac}_{ij} for AFB_BJ$^+$ and **A**C* computations respectively, to avoid the problems generated by waiting for messages either to ensure the same copies of C_{ij} or to ensure the same domains in case of deletion. (2) The guaranteed cost of Y in **A**C* (**G**$C^*(Y)$) defined by the sum of the costs of all constraints C_i and C_{ij}^{ac} involved in Y (Eq.7). (3) Three new procedures, **A**C*() (P.4) is used for updating $\mathbf{C}_\phi$, $CheckPruning()$ (P.5) is used for pruning an agent domain and $ProcessPruning()$ (P.3) is used for processing the deletions of other agents. (4) A new content of some AFB_BJ$^+$ messages (P.7, l.4, 11, 12). (a) **G**C^* is added to **O**k? and **F**B? messages. (b) $\mathbf{C}_\phi$ and $DelVals$, a list of n arrays containing values deleted by each agent, are added to **O**k? and **B**ack messages.

$$\mathbf{G}C^*(Y^j) = \mathbf{G}C^*(Y^{j-1}) + c_j(v_j) + \sum_{C_{ij}^{ac} \in \mathcal{C}} c_{ij}(v_i, v_j) \mid (x_i, v_i) \in Y^{j-1} \quad (7)$$

3.1 AFB_BJ$^+$-AC* description

Procedure 6 presents AFB_BJ$^+$-**A**C* algorithm running by each agent A_j. A_j uses a set of local structures to store its data. UB_j is the cost of the best solution found so far and it is the inadmissible cost $\top$ for **A**C* process. v_j^* is the optimal value of A_j. Y is the CPA. $lb_k[i][v_j]$ stores the lower bounds, on the cost of any full CPA Y that contains (x_j, v_j). $\mathbf{C}_\phi$ is a lower bound of any solution. $\mathbf{C}_{\phi_j}$ is the contribution value of A_j in $\mathbf{C}_\phi$. GC (resp. **G**C^*) stores the guaranteed costs of Y (resp. in **A**C*). $DelVals$ is a list of n arrays containing values deleted by each agent. $c_j(v_j)$ is the unary cost of v_j value.

A_j starts by initializing its local structures (P.6, l.1-2) and begins the **A**C* process (P.4) as described in (§2.3). If A_j is the 1^{st} agent (P.6, l.3), it updates $\mathbf{C}_\phi$ by adding to it, $\mathbf{C}_{\phi_j}$, its contribution value, it resets $\mathbf{C}_{\phi_j}$ to zero, it calls $CheckPruning()$ (P.5) to prune its domain and finally, it performs $ExtendCPA()$ to generate a CPA and to begin the AFB_BJ$^+$ process. Next, A_j enters in the messages processing loop (P.6, l.5). It updates UB_j and v_j^* when the received upper bound ($msg.UB$) is smaller than the stored one (P.6, l.6). Then, If the received CPA ($msg.Y$) is stronger than Y (P.6, l.7), it updates Y and GC and clears all irrelevant lower bounds. The strongest CPA is the one that has the greater time-stamp. Thereafter, A_j resets its domain D_j by restoring all the values that are temporarily deleted in (P.6, l.14).

When receiving an **O**k? message, A_j resets $mustSendFB$ to $true$, to send **F**B? messages, it updates **G**C^* (P.6, l.9) and calls $ProcessPruning()$ (P.3).

When calling $ProcessPruning()$ (P.3), A_j updates its $DelVals$ by the received one (P.3, l.1). Next, it updates domains of its neighbors one by one, removing all values deleted by each neighbor in order to keep the same domains in those agents (P.3, l.2-4). Then, A_j performs again the successive projections to ensure the **A**C* (P.3, l.5-6). Afterwards, it updates its global $\mathbf{C}_\phi$ by the received one (P.3, l.7) and it increases the global $\mathbf{C}_\phi$ by adding its contribution value $\mathbf{C}_{\phi_j}$. If $\mathbf{C}_\phi$ exceeds the UB_j, A_j stops its execution and

Proc 6: AFB_BJ$^+$-**AC*** ()

1 $UB_j \leftarrow +\infty$; $v_j^* \leftarrow empty$; $Y \leftarrow []$; $GC[i..j-1] \leftarrow [0, ..., 0]$;
$\quad lb_k[0][v_j]_{(A_k > A_j) \wedge (v_j \in D_j)} \leftarrow min_{v_k \in D_k} \{c_{jk}(v_j, v_k)\}$

2 $mustSendFB \leftarrow True$; $\mathbf{C}_\phi \leftarrow 0$; $\mathbf{C}_{\phi_j} \leftarrow 0$; $GC^*[i..j-1] \leftarrow [0, ..., 0]$; $\forall a \in D_j,\ c_j(a) \leftarrow 0$; $AC^*()$;

3 **if** $(A_j = A_1)$ $\mathbf{C}_\phi \leftarrow \mathbf{C}_\phi + \mathbf{C}_{\phi_j}$; $\mathbf{C}_{\phi_j} \leftarrow 0$; $CheckPruning()$; $ExtendCPA()$;

4 **while** $(\neg end)$ **do**

5 $\quad msg \leftarrow getMsg()$;

6 $\quad$ **if** $(msg.UB < UB_j)$ $UB_j \leftarrow msg.UB$; $v_j^* \leftarrow v_j$;

7 $\quad$ **if** $(msg.Y$ *is stronger than* $Y)$ $Y \leftarrow msg.Y$; $GC \leftarrow msg.GC$; clear irrelevant $lb()$; reset D_j ;

8 $\quad$ **switch** $(msg.type)$ **do**

9 $\quad\quad$ **case** ***ok?*** $mustSendFB \leftarrow True$; $\mathbf{G}C^* \leftarrow msg.\mathbf{G}C^*$; $ProcessPruning(msg)$;

10 $\quad\quad$ **case** ***back*** $Y \leftarrow Y^{j-1}$; $ProcessPruning(msg)$;

11 $\quad\quad$ **case** ***fb?***

12 $\quad\quad\quad \mathbf{G}C^* \leftarrow msg.\mathbf{G}C^*$;

13 $\quad\quad\quad$ **foreach** $(v_j \in D_j)$ **do**

14 $\quad\quad\quad\quad$ **if** $(\boldsymbol{C}_\phi + \boldsymbol{GC}^*(Y^{j-1}) + c_j(v_j) \geq UB_j)$ $D_j \leftarrow D_j - v_j$;

15 $\quad\quad\quad$ sendMsg : $\mathbf{lb}(lb_j(Y^i)[], msg.Y)$ **to** A_i ; /* A_i is msg sender */

16 $\quad\quad$ **case** ***lb***

17 $\quad\quad\quad lb_k(Y^j) \leftarrow msg.lb$; **if** $(lb(Y^j) \geq UB_j)$ $ExtendCPA()$;

18 $\quad\quad$ **case** ***stp*** $end \leftarrow true$;

informs the others (P.3, l.8-9). Finally, A_j calls $CheckPruning()$ to prune its domain and extends the received CPA by calling $ExtendCPA()$ (P.3, l.10).

When calling $CheckPruning()$ (P.5), A_j checks if there is any value deletion in its domain D_j. The deletion condition is satisfied if there is a value in D_j having a unary cost plus $\mathbf{C}_\phi$ exceeds the UB_j (P.5, l.2-3). If such values exist, A_j removes them and then performs a binary projection on its neighbors to keep the same copy of C_{ij}^{ac} (P.5, l.5). If A_j domain becomes empty, it stops its execution and informs the others (P.5, l.6-7).

When calling $ExtendCPA()$ (P.7), A_j tries to assign its variable by a value v_j (P.7, l.1-2). If such value doesn't exist, A_j goes back to the previous agents (P.7, l.4). Otherwise, A_j extends Y by adding (x_j, v_j) assignment. If Y becomes a full assignment (P.7, l.8), a solution is found and then the UB_j is updated, which imposes to call $CheckPruning()$ and then $ExtendCPA()$ to continue the search (P.7, l.9). Otherwise, A_j sends the extended Y to the next agent (P.7, l.11) and **F**B? messages to unassigned agents (P.7, l.12).

When A_j receives a **F**B? message, it updates $\mathbf{G}C^*$ and it prunes its domain D_j w.r.t the received Y (P.6, l.14). Next, it computes the suitable lower bounds using Eq.5 and sends them to the sender via **L**B message (P.6, l.15).

When A_j receives a **L**B message, it saves the attached lower bounds (P.6, l.17) and checks if the new lower bound, on the cost of Y, exceeds the UB_j. In such a case, A_j calls $ExtendCPA()$ to change its assignment. A_j performs a backjumping, to the appropriate agent, whenever the lower bounds of all its values exceed the UB_j (P.7, l.3-4). If such an agent exists, A_j sends it a **B**ack message. Otherwise, A_j stops its execution and informs the others via **S**tp

Proc 7: ExtendCPA()

```
1  v_j ← argmin_{v'_j ∈ D_j} { lb(Y ∪ (x_j, v'_j)) } ; /* Eq.6 */
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)[j − 1] < UB_j) sendMsg : back(Y^i, UB_j, DelVals, C_φ) to A_i; return ;
5      broadcastMsg : stp(UB_j) ;  end ← true ;
6  else
7      Y ← {Y ∪ (x_j, v_j)} ;
8      if (var(Y) = X)
9          UB_j ← GC(Y) ;  v*_j ← v_j ;  Y ← Y^{j-1} ; CheckPruning() ;  ExtendCPA() ;
10     else
11         sendMsg : ok?(Y, GC, UB_j, DelVals, C_φ, GC*) to A_{j+1} ;
12         if (mustSendFB) sendMsg : fb?(Y, GC, UB_j, GC*) to A_k | A_k > A_j ;
           mustSendFB ← false ;
```

messages (P.7, l.5).

3.2 AFB_BJ$^+$-AC* Correctness

Theorem 1. *AFB_BJ$^+$-**A**C* is guaranteed to compute the optimum cost and terminates.*

Proof. (Sketch) AFB_BJ$^+$-**A**C* outperforms AFB_BJ$^+$by performing, based on **A**C*, two pruning tests of an agent domain. So to prove that AFB_BJ$^+$-**A**C* is correct, it is sufficient to prove that these tests don't violate the correctness of AFB_BJ$^+$[11]. In other words, it must prove that AFB_BJ$^+$-**A**C* doesn't remove an optimal value when performing these tests (P.7, l.2), (P.6, l.14) and (P.5, l.2). We can easily prove that both tests lead to the self-evident test $(LB_j \geq UB_j)$(cond. 8, 9) by using **A**C* transformations (§2.3). The correctness of these transformations is already proven in [8]. So, the deleted values, based on these both tests, are non-optimal values.

$$(\mathbf{C}_\phi + c_j(v_j) \geq UB_j) \implies (\mathbf{C}_\phi + \mathbf{G}C^*(Y^{j-1}) + c_j(v_j) \geq UB_j) \quad (8)$$
$$\implies (lb(Y^{j-1} \cup (x_j, v_j)) \geq UB_j) \quad (9)$$

By observation that the number of new generated CPAs is a finite number and the evaluation of each CPA can never lead to an infinite loop, it can be deduced simply that AFB_BJ$^+$-**A**C* terminates [11]. □

4 Experimental Results

In this section we experimentally compare AFB_BJ$^+$-**A**C* to AFB_BJ$^+$ [11], BnB-Adopt$^+$-**A**C* [5] and BnB-Adopt$^+$-DP2 [1] using the simulator DisChoco 2.0 [12]. Four benchmarks are used in experiments: binary random Max-DisCSPs, binary random DCOPs, meetings scheduling and sensors network.

Binary random Max-DisCSPs [11]: which are defined by (n, d, p_1, p_2), with n is the number of variables/agents, d is the number of values in each

Tab. 1. Average of msg sent and ncccs performed on Max-DisCSPs, where $p_1 = 0.4$

	ncccs				*msg*			
p_2	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.9
AFB_BJ$^+$	2890	8116	15702	29573	292	892	1731	3375
AFB_BJ$^+$-**A**C*	**2700**	**7416**	**14635**	**27713**	**284**	**873**	**1673**	**3176**
BnB-Adopt$^+$-**A**C*	3872	17860	117168	705734	586	3582	22642	114082
BnB-Adopt$^+$-DP2	4368	17769	85270	202398	616	3434	16528	38474

Tab. 2. Average of msg sent and ncccs performed on Max-DisCSPs, where $p_1 = 0.7$

	ncccs $\times 10^2$				*msg* $\times 10^2$			
p_2	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.9
AFB_BJ$^+$	707	1228	2304	3197	64	114	208	293
AFB_BJ$^+$-**A**C*	**687**	**1215**	**2272**	**3000**	**62**	**113**	**204**	**273**
BnB-Adopt$^+$-**A**C*	9918	65595	314164	730672	1134	6862	30127	59581
BnB-Adopt$^+$-DP2	9555	62191	262828	503293	1083	6461	25271	44228

domain of a variable, p_1 is the probability of the connection of two variables by a constraint and p_2 is the probability of the conflict between two constrained variables. We have evaluated two classes of instances, ($n = 10$, $d = 10$, $p_1 = 0.4$, p_2) and ($n = 10$, $d = 10$, $p_1 = 0.7$, p_2). For p_2, its value varies between 0.6 and 0.9 by steps of 0.1. For each pair (p_1, p_2), we have generated an average of 50 instances.

Binary random DCOPs [11]: are defined by (n, d, p_1), which is the same triple (n, d, p_1) of Max-DisCSPs. We have evaluated one class of instances, ($n = 10$, $d = 10$, $p_1 = 0.4$ *to* 0.8). For each constraint, the cost of each value combination is selected from the set $\{0, ..., 100\}$. For each p_1, we have generated, randomly, an average of 50 instances.

Meetings scheduling [9]: are defined by the number of participants, each one has a personal private schedule, and the number of meetings, each one occurs in a particular place. Each variable/agent represents a meeting. Each meeting occurs in a specified time slot which represents the possible values for each meeting/variable. The constraints connect meetings that share participants. We have evaluated 4 cases, each one with a different number of Meetings/Participants [11].

Sensors network [2]: are defined by the number of sensors, and the number of mobiles. One sensor can track one mobile at most and each 3 sensors must track one mobile. Each variable/agent represents a mobile. Each domain of a variable/mobile contains all of the possible combinations of 3 sensors that track it. The constraints connect adjacent mobiles. We have evaluated 4 cases, each one with a different number of Sensors/Mobiles [11].

We compare algorithms efficacy by two measures, the average of messages sent by all agents (*msg*) to evaluate the communication load and the average of non-concurrent constraint checks (*ncccs*) to evaluate the computation effort.

Tables 1 and 2 show the results of experiments on Max-DisCSPs in the sparse case ($p_1 = 0.4$) and the dense one ($p_1 = 0.7$) respectively. Table 3 rep-

Tab. 3. Average of msg sent and ncccs performed on binary random DCOPs

	ncccs × 10^3					*msg* × 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$^+$	33	82	197	251	340	4	9	18	23	30
AFB_BJ$^+$-**A**C*	**31**	**80**	**189**	**246**	**333**	**3**	**8**	**17**	**22**	**29**
BnB-Adopt$^+$-**A**C*	618	2974	18507	47831	58861	113	468	2240	4767	4346
BnB-Adopt$^+$-DP2	190	1050	6531	17253	24881	36	174	835	1814	2081

Tab. 4. Average of msg sent and ncccs performed on Meetings Scheduling

	ncccs				*msg*			
case	A	B	C	D	A	B	C	D
AFB_BJ$^+$	5194	5104	2815	2753	383	992	567	659
AFB_BJ$^+$-**A**C*	**2294**	**2172**	**1134**	**1111**	**290**	**773**	**341**	**373**
BnB-Adopt$^+$-**A**C*	99687	45329	31749	14051	6503	5935	5063	4592
BnB-Adopt$^+$-DP2	6261	4517	2094	1667	700	794	510	530

resents the results of experiments on binary random DCOPs. The comparison between AFB_BJ$^+$-**A**C* and AFB_BJ$^+$shows that their results are close in the sparse graph and become better whenever the graph is dense. Concerning BnB-Adopt$^+$ algorithms, their results show a weak performance compared to AFB_BJ$^+$ algorithms. Table 4 displays the results of experiments on meetings scheduling. AFB_BJ$^+$-**A**C* reduces, in general, the number of *msg* and *ncccs* to almost half compared to AFB_BJ$^+$ and it outperforms the performance of both BnB-Adopt$^+$ algorithms. Table 5 presents the results of experiments on sensors network. AFB_BJ$^+$-**A**C* improves, in general, the performance of AFB_BJ$^+$ by a rate close to half. Comparing AFB_BJ$^+$-**A**C* to BnB-Adopt$^+$-**A**C* , the results show that AFB_BJ$^+$-**A**C* is better, in general, except some classes where BnB-Adopt$^+$-**A**C* needs few messages compared to AFB_BJ$^+$-**A**C*. Otherwise, the results show that the BnB-Adopt$^+$-DP2 outperforms all other algorithms.

Looking at all results above, we can deduce that AFB_BJ$^+$-**A**C* performs better than AFB_BJ$^+$. This is due to the **A**C* techniques that allow agents to determine and then remove non-optimal values in their domains. However, **A**C* techniques don't give significant results in all instances, as in the sparse graphs for example. BnB-Adopt$^+$-**A**C* and BnB-Adopt$^+$-DP2 perform badly when solving Max-DisCSP and Random DCOPs. The main reason, for that, is the asynchronous nature of both algorithms which imposes them to use a large number of *msg* and *ncccs*. However, for meetings scheduling, the performances of BnB-Adopt$^+$-DP2 and AFB_BJ$^+$-**A**C* are close and for sensors network, BnB-Adopt$^+$-DP2 outperforms AFB_BJ$^+$-**A**C*. One possible reason, behind it, is the density of constraints network. The sensors network instances, for example, appear very sparse with a smaller number of constraints. For this reason, the agents in BnB-Adopt$^+$-DP2 , using the DP2 heuristic, select values for their variables close to the solution values.

Tab. 5. Average of msg sent and ncccs performed on Sensors Network

	ncccs				*msg*			
case	A	B	C	D	A	B	C	D
AFB_BJ$^+$	8367	7309	2390	4417	2993	2594	321	1293
AFB_BJ$^+$-$\mathbf{A}$C*	2763	2432	1044	1768	2659	1950	217	960
BnB-Adopt$^+$-$\mathbf{A}$C*	2881	5281	2861	6111	827	1220	742	1775
BnB-Adopt$^+$-DP2	**1000**	**1025**	**1042**	**1208**	**195**	**226**	**187**	**309**

5 Conclusion

In this paper, we have introduced AFB_BJ$^+$-$\mathbf{A}$C* algorithm which connects AFB_BJ$^+$ with soft arc consistency ($\mathbf{A}$C*). It relies on pruning non-optimal values of an agent domain, using $\mathbf{A}$C*, and propagating them, without adding other types of messages, for generating other deletions that will likewise be propagated. The performed experiments on several benchmarks show that AFB_BJ$^+$-$\mathbf{A}$C* improves the basic AFB_BJ$^+$ in terms of communication load and computation effort. This work can be extended to exploit other types of soft arc consistency such as directional $\mathbf{A}$C* (DAC*) and full DAC* (FDAC*).

References

[1] S. Ali, S. Koenig and 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), 1041–1048. https://doi.org/10.1145/1082473.1082631

[2] R. Béjar, C. Domshlak, C. Fernández, C. Gomes, B. Krishnamachari, B. Selman and M. Valls, Sensor networks and distributed csp: communication, computation and complexity, *Artificial Intelligence*, **161** (2005), no. (1-2), 117–147. https://doi.org/10.1016/j.artint.2004.09.002

[3] A. Gershman, A. Meisels and R. Zivan, Asynchronous forward bounding for distributed cops, *J. Artif. Intell. Res. (JAIR)*, **34** (2009), 61–88.

[4] P. Gutierrez and P. Meseguer, Connecting bnb-adopt with soft arc consistency: initial results, In *Recent Advances in Constraints*, Springer, 2009, 19–37. https://doi.org/10.1007/978-3-642-19486-3_2

[5] P. Gutierrez and P. Meseguer, Improving bnb-adopt+-ac, In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, (2012), 273–280.

[6] P. Gutierrez and P. Meseguer, Removing redundant messages in n-ary bnb-adopt, *J. Artif. Intell. Res. (JAIR)*, **45** (2012), 287–304.

[7] K. Hirayama and M. Yokoo, Distributed partial constraint satisfaction problem, In *Principles and Practice of Constraint Programming-CP97*, Springer, 1997, 222–236. https://doi.org/10.1007/bfb0017442

[8] J. Larrosa and T. Schiex, In the quest of the best form of local consistency for weighted csp, In *IJCAI*, **3** (2003), 239–244.

[9] R.T. Maheswaran, M. Tambe, E. Bowring, J.P. Pearce and 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*, (2004), 310–317.

[10] P.J. Modi, W.M. Shen, M. Tambe and M. Yokoo, Adopt: Asynchronous distributed constraint optimization with quality guarantees, *Artificial Intelligence*, **161** (2005), no. 1-2, 149–180. https://doi.org/10.1016/j.artint.2004.09.003

[11] M. Wahbi, R. Ezzahir and C. Bessiere, Asynchronous forward bounding revisited, In *Principles and Practice of Constraint Programming*, Springer, 2013, 708–723. https://doi.org/10.1007/978-3-642-40627-0_52

[12] M. Wahbi, R. Ezzahir, C. Bessiere and E.H. Bouyakhf, DisChoco 2: A Platform for Distributed Constraint Reasoning, In *Proceedings of DCR'11*, (2011), 112–121.

[13] W. Yeoh, A. Felner and S. Koenig, Bnb-adopt: An asynchronous branch-and-bound dcop algorithm, *J. Artif. Intell. Res. (JAIR)*, **38** (2010), 85–133.

Received: May 26, 2018; Published: June 26, 2018