Connecting AFB_BJ\textsuperscript{+} 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

Copyright © 2018 Rachid Adrdor, Redouane Ezzahir and Lahcen Koutti. This article is distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract

The Distributed Constraint Optimization Problem (DCOP) is a major and powerful paradigm for modeling and solving problems in multi-agent coordination. AFB_BJ\textsuperscript{+} is one of the most excellent algorithms for solving DCOPs. Recently, researchers have shown that including soft arc consistency (AC\textsuperscript{*}) in DCOP algorithms causes significant improvements in their performance. In this paper, we introduce AFB_BJ\textsuperscript{+}-AC\textsuperscript{*} algorithm which connects AFB_BJ\textsuperscript{+} with soft arc consistency (AC\textsuperscript{*}). It relies on pruning non-optimal values of an agent domain, using AC\textsuperscript{*}, 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 AC\textsuperscript{*}, AFB_BJ\textsuperscript{+}-AC\textsuperscript{*} improves the basic AFB_BJ\textsuperscript{+}. 
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+-AC* [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 AC*.

In this paper, we introduce AFB_BJ+-AC* algorithm which connects the slightly asynchronous algorithm AFB_BJ+ with soft arc consistency (AC*). It relies on pruning non-optimal values of an agent domain, using AC*, 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-AC 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 \((A, X, D, C)\) \cite{10}, where \(A = \{A_1, A_2, \ldots, A_k\}\) is a set of agents, \(X = \{x_1, x_2, \ldots, x_n\}\) is a set of variables, \(D = \{D_1, D_2, \ldots, D_n\}\) is a set of domains, where each \(D_i\) in \(D\) 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. \(C = \{f : x_j \in X \mid D_j \rightarrow \mathbb{R}^+\}\) 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 \(f \in C\) 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. \(\Gamma^-\)) and those with a lower priority (\(A_k > A_j\)) (resp. \(\Gamma^+\)). A neighbor of \(A_j\) is an agent that shares a constraint with it. Let \(Y = Y^j = [(x_1, v_1), (x_2, v_2), \ldots, (x_j, v_j)]\) 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).

\[
GC(Y) = \sum_{c_{ij} \in C} c_{ij}(v_i, v_j) \mid (x_i, v_i), (x_j, v_j) \in Y
\]

(1)

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

2.2 AFB_BJ algorithm

In AFB_BJ\cite{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.

\[
FC_j(v_j) = \sum_{x_k \in \Gamma^-(x_j)} \min_{v_k \in D_k} \{c_{jk}(v_j, v_k)\}
\]

(2)

Then, \(A_j\) assigns its variable and sends an Ok? message to the next agent and FB? messages to unassigned agents. All these messages contain \(Y^j\) and
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 \(x_i\), \(C_0\) be a zero-arity constraint that represents a lower bound of any full assignment and \(\Gamma\) be the lowest unacceptable cost (i.e.,

---

**Proc 1: ProjectUnary()**

1. \(\beta \leftarrow \min_{v_i \in D_i} \{c_i(v_i)\}; \) \(C_{\phi_i} \leftarrow C_{\phi_i} + \beta;\)
2. foreach \((v_i \in D_i)\) do \(c_i(v_i) \leftarrow c_i(v_i) - \beta;\)

**Proc 2: ProjectBinary\((x_i, x_j)\)**

1. foreach \((v_i \in D_i)\) do \(a \leftarrow \min_{v_j \in D_j} \{c_{ij}(v_i, v_j)\};\)
2. foreach \((v_j \in D_j)\) do \(c_{ij}(v_i, v_j) \leftarrow c_{ij}(v_i, v_j) - \alpha;\)
3. if \((A_i\ is\ current\ agent)\) then \(c_i(v_i) \leftarrow c_i(v_i) + \alpha;\)

**Proc 3: ProcessPruning\((msg)\)**

1. \(DelVals \leftarrow msg.DelVals;\)
2. foreach \((A_k \in \Gamma)\) do foreach \((a \in DelVals[k])\) do \(D_k \leftarrow D_k - a;\)
3. \(\text{ProjectUnary}();\)
4. \(\text{ProjectBinary}(x_i, x_k);\)
5. \(\text{CheckPruning}();\)
6. \(\text{ExtendCPA}();\)

---

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 FB? message, it responds by sending a LB 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 \(Y^h\).

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

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 FB? messages only for the parts of lower bounds that are discarded \([11]\).
The global upper bound. We consider the following soft local consistencies [8]:

Node Consistency (NC*): \((x_i, v_i)\) is NC* if \(C_\phi + c_i(v_i) < T\). 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 \(x_i \in X\) 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 AC* if any variable \(x_i \in X\) 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 w.r.t each \(C_{ij}\) of the problem to \(C_i\). Secondly, by a unary projection (P.1) which projects the smallest cost \(\beta\) of \(C_i\) to \(C_\phi\). Finally, by removing not NC* values.

AC* can be extended to work in a distributed environment with some changes. Firstly, to obtain the global \(C_\phi\), each agent \(A_i\) stores temporarily its contribution value in a local variable \(C_{\phi_i}\) (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 AC* 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 \(C_\phi\) (P.2, l.5) to avoid counting the same cost twice in \(C_\phi\) [5].

### 3 Integrating AC* in AFB_BJ⁺ algorithm

There are different ways to integrate AC* 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⁺-AC* algorithm [5], but it’s not useful with AFB_BJ⁺. For that, we propose another way to integrate AC* with AFB_BJ⁺. It relies on pruning non-optimal values of an agent domain, using AC*, 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 \(C_{ij}^{ac}\) for AFB_BJ⁺ and AC* computations respectively, to avoid the problems generated by waiting for

<table>
<thead>
<tr>
<th>Proc 4: AC* ()</th>
<th>Proc 5: CheckPruning()</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 foreach ((A_k \in \Gamma^+)) do</td>
<td>1 foreach ((a \in D_j) do</td>
</tr>
<tr>
<td>2 [ProjectBinary(x_j, x_k)] ;</td>
<td>2 if ([c_j(a) + C_\phi \geq UB_j])</td>
</tr>
<tr>
<td>3 [ProjectUnary()] ;</td>
<td>3 [D_j \leftarrow D_j - a;] ;</td>
</tr>
<tr>
<td>4 [ProjectBinary(x_k, x_j)] ;</td>
<td>4 foreach ((A_k \in \Gamma)) do</td>
</tr>
<tr>
<td>5 foreach ((A_k \in \Gamma^-)) do</td>
<td>5 [ProjectBinary(x_k, x_j)] ;</td>
</tr>
<tr>
<td>6 [ProjectBinary(x_j, x_k)] ;</td>
<td>6 if ((D_j) is empty)</td>
</tr>
<tr>
<td>7 [ProjectUnary()] ;</td>
<td>7 broadcastMsg : (stp(UB_j)); end (\leftarrow true);</td>
</tr>
</tbody>
</table>
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 $AC^*$ ($GC^*(Y)$) defined by the sum of the costs of all constraints $C_i$ and $C_{ij}^*$ involved in $Y$ (Eq. 7). (3) Three new procedures, $AC^*$ (P.4) is used for updating $C_\phi$, $CheckPruning$ (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, 1.4, 11, 12). (a) $GC^*$ is added to Ok? and FB? messages. (b) $C_\phi$ and DelVals, a list of $n$ arrays containing values deleted by each agent, are added to Ok? and Back messages.

$$GC^*(Y^j) = GC^*(Y^{j-1}) + c_j(v_j) + \sum_{c_{ij} \in C} c_{ij}(v_i, v_j) \mid (x_i, v_i) \in Y^{j-1}$$ (7)

### 3.1 AFB BJ$^+$ - AC$^*$ description

Procedure 6 presents AFB BJ$^+$ - AC$^*$ 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 $\bar{T}$ for $AC^*$ 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 $(x_j, v_j)$. $C_\phi$ is a lower bound of any solution. $C_{\phi j}$ is the contribution value of $A_j$ in $C_\phi$. $GC$ (resp. $GC^*$) stores the guaranteed costs of $Y$ (resp. in $AC^*$). 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, 1.1-2) and begins the AC$^*$ process (P.4) as described in (§2.3). If $A_j$ is the 1st agent (P.6, 1.3), it updates $C_\phi$ by adding to it, $C_{\phi j}$, its contribution value, it resets $C_{\phi j}$ 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, 1.5). It updates $UB_j$ and $v_j^*$ when the received upper bound ($msg.UB$) is smaller than the stored one (P.6, 1.6). Then, If the received CPA ($msg.Y$) is stronger than $Y$ (P.6, 1.7), 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, 1.14).

When receiving an Ok? message, $A_j$ resets mustSendFB to true, to send FB? messages, it updates $GC^*$ (P.6, 1.9) and calls $ProcessPruning$ (P.3).

When calling $ProcessPruning$ (P.3), $A_j$ updates its DelVals by the received one (P.3, 1.1). 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, 1.2-4). Then, $A_j$ performs again the successive projections to ensure the AC$^*$ (P.3, 1.5-6). Afterwards, it updates its global $C_\phi$ by the received one (P.3, 1.7) and it increases the global $C_\phi$ by adding its contribution value $C_{\phi j}$. If $C_\phi$ exceeds the $UB_j$, $A_j$ stops its execution and
informs the others (P.3, l.8-9). Finally, $A_j$ calls \textit{CheckPruning()} to prune its domain and extends the received CPA by calling \textit{ExtendCPA()} (P.3, l.10).

When calling \textit{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 $C_o$ 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 $C_{ijc}^*$ (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 \textit{ExtendCPA()} (P.7), $A_j$ tries to assign its variable by a value $v_j$ (P.7, l.1.1-2). 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 \textit{CheckPruning()} and then \textit{ExtendCPA()} to continue the search (P.7, l.1.9). Otherwise, $A_j$ sends the extended $Y$ to the next agent (P.7, l.11) and \textit{FB?} messages to unassigned agents (P.7, l.12).

When $A_j$ receives a \textit{FB?} message, it updates $GC^*$ 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 \textit{LB} message (P.6, l.15).

When $A_j$ receives a \textit{LB} 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 \textit{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 \textit{Back} message. Otherwise, $A_j$ stops its execution and informs the others via \textit{Stp}

\begin{verbatim}
Proc 6: AFB_BJ+\textit{-AC*} ( )
\end{verbatim}
Proc 7: ExtendCPA()

1 \( v_j \leftarrow \arg\min_{v_j' \in D_j} \{ \text{lb}(Y \cup (x_j, v_j')) \} \); */ Eq. 6
2 if \((\text{lb}(Y \cup (x_j, v_j)) \geq U_B j) \lor (C_\phi + G C^*(Y^{j-1}) + c_j(v_j) \geq U_B j)\)
3 for \(i \leftarrow j - 1 \) to 1 do
4 if \((\text{lb}(Y \cup [j-1]) < U_B j)) sendMsg : \text{back}(Y^i, U_B j, \text{DelVals}, C_\phi) \rightarrow A_i; \text{return} ;
5 broadcastMsg : \text{stp}(U_B j) ; end \leftarrow \text{true} ;
6 else
7 \( Y \leftarrow \{ Y \cup (x_j, v_j) \} \);
8 if \((\text{var}(Y) = X)) \]
9 \( U_B j \leftarrow GC(Y) ; \ v_j^* \leftarrow v_j ; \ Y \leftarrow Y^{j-1} ; \text{CheckPruning}(); \text{ExtendCPA}();
10 else
11 sendMsg : \text{ok?}(Y, GC, U_B j, \text{DelVals}, C_\phi, GC^*) \rightarrow A_{j+1} ;
12 if \((\text{mustSendFB}) \text{sendMsg : \text{fb?}(Y, GC, U_B j, GC^*)} \rightarrow A_k | A_k > A_j) ;
13 mustSendFB \leftarrow \text{false} ;

messages (P.7, l.5).

3.2 AFB\_BJ\_AC\_ Correctness

Theorem 1. AFB\_BJ\_AC\_ is guaranteed to compute the optimum cost and terminates.

Proof. (Sketch) AFB\_BJ\_AC\_ outperforms AFB\_BJ\_ by performing, based on AC\_, two pruning tests of an agent domain. So to prove that AFB\_BJ\_AC\_ 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\_AC\_ 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 U_B j)(\text{cond. } 8, 9)) by using AC\_ 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.

\[
(C_\phi + c_j(v_j) \geq U_B j) \quad \implies \quad (C_\phi + G C^*(Y^{j-1}) + c_j(v_j) \geq U_B j) \quad (8)
\]
\[
\implies \quad (\text{lb}(Y^{j-1} \cup (x_j, v_j)) \geq U_B j) \quad (9)
\]

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\_AC\_ terminates [11].

4 Experimental Results

In this section we experimentally compare AFB\_BJ\_AC\_ to AFB\_BJ\_ [11], BnB-Adopt\_AC\_ [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
AFB, BJ\textsuperscript{+}-AC\textsuperscript{*} algorithm

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

<table>
<thead>
<tr>
<th>( p_2 )</th>
<th>ncccs ( \times 10^2 )</th>
<th>msg ( \times 10^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>2890</td>
<td>29573</td>
</tr>
<tr>
<td>0.7</td>
<td>8116</td>
<td>892</td>
</tr>
<tr>
<td>0.8</td>
<td>15702</td>
<td>1731</td>
</tr>
<tr>
<td>0.9</td>
<td>29713</td>
<td>3375</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>( p_2 )</th>
<th>ncccs ( \times 10^2 )</th>
<th>msg ( \times 10^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>707</td>
<td>64</td>
</tr>
<tr>
<td>0.7</td>
<td>1228</td>
<td>114</td>
</tr>
<tr>
<td>0.8</td>
<td>2304</td>
<td>208</td>
</tr>
<tr>
<td>0.9</td>
<td>3197</td>
<td>293</td>
</tr>
</tbody>
</table>

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, \((n = 10, d = 10, p_1 = 0.4 \text{ to } 0.8)\). For each constraint, the cost of each value combination is selected from the set \([0, \ldots, 100]\). 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 (\( \text{msg} \)) to evaluate the communication load and the average of non-concurrent constraint checks (\( \text{ncccs} \)) 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-
represents the results of experiments on binary random DCOPs. The comparison between AFB_BJ+\text{-AC}* 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+\text{-AC}* reduces, in general, the number of \textit{msg} and \textit{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+\text{-AC}* improves, in general, the performance of AFB_BJ+ by a rate close to half. Comparing AFB_BJ+\text{-AC}* to BnB-Adopt+\text{-AC*}, the results show that AFB_BJ+\text{-AC}* is better, in general, except some classes where BnB-Adopt+\text{-AC*} needs few messages compared to AFB_BJ+\text{-AC*}. Otherwise, the results show that the BnB-Adopt+\text{-DP2} outperforms all other algorithms.

Looking at all results above, we can deduce that AFB_BJ+\text{-AC}* performs better than AFB_BJ+. This is due to the \textit{AC*} techniques that allow agents to determine and then remove non-optimal values in their domains. However, \textit{AC*} techniques don’t give significant results in all instances, as in the sparse graphs for example. BnB-Adopt+\text{-AC*} and BnB-Adopt+\text{-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 \textit{msg} and \textit{ncccs}. However, for meetings scheduling, the performances of BnB-Adopt+\text{-DP2} and AFB_BJ+\text{-AC*} are close and for sensors network, BnB-Adopt+\text{-DP2} outperforms AFB_BJ+\text{-AC*}. 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+\text{-DP2}, using the DP2 heuristic, select values for their variables close to the solution values.

### Table 3. Average of \textit{msg} sent and \textit{ncccs} performed on binary random DCOPs

<table>
<thead>
<tr>
<th>$p$</th>
<th>\textit{ncccs} $\times 10^3$</th>
<th>\textit{msg} $\times 10^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>33</td>
<td>8</td>
</tr>
<tr>
<td>0.5</td>
<td>82</td>
<td>38</td>
</tr>
<tr>
<td>0.6</td>
<td>197</td>
<td>151</td>
</tr>
<tr>
<td>0.7</td>
<td>251</td>
<td>240</td>
</tr>
<tr>
<td>0.8</td>
<td>340</td>
<td>585</td>
</tr>
</tbody>
</table>

### Table 4. Average of \textit{msg} sent and \textit{ncccs} performed on Meetings Scheduling

<table>
<thead>
<tr>
<th>\textit{case}</th>
<th>\textit{ncccs}</th>
<th>\textit{msg}</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFB_BJ*</td>
<td>5194</td>
<td>104</td>
</tr>
<tr>
<td>AFB_BJ+\text{-AC*}</td>
<td>2294</td>
<td>2172</td>
</tr>
<tr>
<td>BnB-Adopt+\text{-AC*}</td>
<td>9968</td>
<td>4532</td>
</tr>
<tr>
<td>BnB-Adopt+\text{-DP2}</td>
<td>6261</td>
<td>4517</td>
</tr>
</tbody>
</table>
Tab. 5. Average of msg sent and ncccs performed on Sensors Network

<table>
<thead>
<tr>
<th>case</th>
<th>ncccs</th>
<th>msg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>AFB_BJ+</td>
<td>8367</td>
<td>7309</td>
</tr>
<tr>
<td>AFB_BJ+-AC*</td>
<td>2763</td>
<td>2432</td>
</tr>
<tr>
<td>BnB-Adopt+-AC*</td>
<td>2881</td>
<td>5281</td>
</tr>
<tr>
<td>BnB-Adopt+-DP2</td>
<td>1000</td>
<td>1025</td>
</tr>
</tbody>
</table>

5 Conclusion

In this paper, we have introduced AFB_BJ+-AC* algorithm which connects AFB_BJ+ with soft arc consistency (AC*). It relies on pruning non-optimal values of an agent domain, using AC*, 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+-AC* 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 AC* (DAC*) and full DAC* (FDAC*).

References


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