Asynchronous Breadth-First Search
DCOP Algorithm

Redouane EZZAHIR
LIMIARF/FSR, Université Mohammed V agdal
BP 1014, Rue Ibn Battota, Agdal, Rabat, Moroc
ezzahir@lirmm.fr

Christian BESSIERE
LIRMM/CNRS, Université de Montpellier 2
161, Rue Ada, 34392, Cedex 5, France
bessiere@lirmm.fr

Imade BENELALLAM
LIMIARF/FSR, Université Mohammed V agdal
BP 1014, Rue Ibn Battota, Agdal, Rabat, Moroc
imade.benelallam@ieee.org

El Houssine BOUYAKHF
LIMIARF/FSR, Université Mohammed V agdal
BP 1014, Rue Ibn Battota, Agdal, Rabat, Moroc
bouyakhf@fsr.ac.ma

Mustapha BELAISSAOUI
ENCG- Settat, Université Hassan I
m.belaissaoui@encg-settat.ma

Abstract
In MultiAgent systems, Distributed Constraint Optimization Problem (DCOP) has been used as a formalism to model a wide range of agents coordination issues. The Asynchronous forward-bounding with backjumping (AFB-BJ) algorithm was recently proposed as a new way to solve DCOP. The AFB-BJ shows better performance in comparison
to Adopt algorithm. In this paper, we introduce Asynchronous Breadth-First Search DCOP algorithm (ABFS) that improves AFB-BJ. In ABFS we use a new search strategy based on tree structure where the agents are ordered on a breadth first search traversal. Detailed experimental results show that on benchmark problems (random Max-DisCSPs, graph coloring and Meeting Schedule problems) the proposed algorithm (ABFS) obtains more improvements than AFB-BJ algorithm.

**Keywords:** Distributed Constraints Optimization Problems, AFB with backjumping, Multi-agents System

1 Introduction

Distributed Constraint Optimization Problem (DCOP) is a powerful formalism to model a wide range of applications in MultiAgent system such as distributed resource allocation problems, distributed scheduling problems, distributed planning problems, and so forth. Differently from COP, in DCOP collaborative agents must find solutions over a distributed set of constraints. In [8] an asynchronous complete method for distributed constraint optimization (called Adopt) is proposed to find the optimal solution for problems formalized as DCOP. Adopt provides quality/optimality guarantees on system performance and asynchrony on communication among agents.

The Asynchronous forward-bounding (AFB) algorithm was recently proposed as a new way to solve DCOP [4]. The AFB algorithm is extended by adding a backjumping mechanism, resulting in the AFB with backjumping algorithm (AFB-BJ) [3]. The AFB-BJ shows better performance in comparison to Adopt algorithm. In AFB-BJ a static final order on the agents is constructed before starting distributed search. Copies of the Current Partial Assignment CPA are sent forward to all unassigned agents to estimate the lower bounds on the cost of any full assignment extended from each past partial assignment CPA[1..k] (where CPA[1..k] ⊆ CPA is the partial assignment that holds the first k agents assigned in CPA). Forward bounding messages sent to all unassigned agents may cause redundancy of several operations of cost computing and messages sending during search.

In this paper, we introduce an Asynchronous Breadth-First Search DCOP algorithm (ABFS) that improves AFB-BJ. The ABFS algorithm uses a new search strategy based on tree structure, where the agents are ordered on a breadth first search traversal. Experiments where performed on DisChoco plateform [5]. Comparing ABFS and AFB-BJ on random Max-DisCSPs, graph coloring and Meeting Schedule problems show that ABFS performs better than AFB-BJ.

The rest of the paper is organized as follow: Section 2 gives an overview of the DCOP formalism. Section 3 presents our approach based on tree structure.
Section 4 describes the Asynchronous Breadth-First Search DCOP algorithm (ABFS). Finally, in section 5 we present an experimental evaluations of the presented algorithm on random Max-DisCSPs, graph coloring and Meeting Schedule problems.

2 Distributed Constraint Optimization Problem (DCOP)

In Multiagent systems, Distributed Constraint Optimization Problem (DCOP) has been used as a formalism to model a wide range of agents coordination issues and has generated significant interest from researchers [8, 7]. Formally, a DCOP is formalized by Modi et al. [7] as a tuple \((A, X, D, C, \phi)\), were \(A\) is a set of agents \(\{A_1, A_2, ... , A_k\}\), \(X\) is a set of variables \(\{x_1, x_2, ..., x_n\}\), each assigned to an agent, where the values of the variables are taken from finite, discrete domains \(D_1, D_2, ..., D_n\), respectively \((D = \{D_1, D_2, ..., D_n\})\). Only the agent who is assigned a variable has control of its value and knowledge of its domain. \(C = \{\phi_{ij} : D_i \times D_j \rightarrow N\}, \) with \(i, j = 1...n, i \neq j\) is a set of constraints, represented by a cost function \(\phi_{ij}\) for each pair of variables \(x_i\) and \(x_j\).

The goal is to choose values for variables such that an objective function \(\phi\) is minimized or maximized. For clarity, we consider an objective function described as addition over costs, where cost is represented as a natural number. However, the techniques described in this paper can be applied to any associative, commutative, monotonic aggregation operator defined over a totally ordered set of valuations. This class of optimization functions is described formally by Schiex, Fargier and Verfaillie as Valued CSPs [15]. The cost functions in DCOP are the analogue of constraints from DisCSP (for convenience in this paper, we sometimes refer to cost functions as constraints). They take values of variables as input and, instead of returning satisfied or unsatisfied, they return a valuation.

The objective is to find an assignment \(A\) of values to variables such that the total cost, denoted \(\phi\), is minimized and every variable has a value. \(\phi\) is defined as

\[
\phi(A) = \sum_{x_i, x_j \in X} \phi_{ij}(d_i, d_j), \quad \text{where} \quad x_i = d_i, x_j = d_j \in A
\]

We will consider that the delay in delivering a message is finite \([10], [7]\). Furthermore, we assume a static final order on the agents, known to all agents participating in the search process \([10]\).
3 Exploiting tree structure to improve AFB-BJ

3.1 Breadth-First Search Tree

In the AFB-BJ algorithm, a single most up-to-date current partial assignment CPA is passed among the agents [4] [3]. Agents assign their variables only when they hold the up-to-date Current Partial Assignment (CPA). The CPA_MSG is a message that is passed between agents, and carries the partial assignment that agents attempt to extend into a complete and optimal solution by assigning their variables on it. The CPA also carries the accumulated cost of constraints between all assignments it contains, as well as a unique time-stamp. Only one agent performs an assignment on the CPA at any given time. Copies of the CPA are sent forward (on FB_CPA messages) and are concurrently processed by multiple agents. Each unassigned agent computes a vector of the estimated lower bounds on the cost of assigning a value to its variable with respect to each past CPA (i.e. CPA[1..k]), and sends this vector back to the agent which performed the assignment (on FB_ESTIMATE messages) [3].

More formally the estimated cost is computed as follow:
Denote by \( \text{cost}((i, v), (j, u)) \) the cost of assigning both \( A_i = v \) and \( A_j = u \). If \( v \in D_i \) then we denote by \( h_j(v) = \min(\text{cost}((i, v), (j, u)) \text{ s.t. } u \in D_j) \). We define \( h(v) \), to be the sum of \( h_j(v) \) over all \( j \succ i \). Intuitively, \( h(v) \) is a lower bound on the cost of constraints involving the assignment \( A_i = v \) and all agents \( A_j \) who follow \( A_i \) in the agent ordering. Notice that this bound can be computed once per agent, since it is independent of the assignments of higher priority agents.

When receiving a FB_CPA message, the agent computes the cost incremented to the CPA cost from assigning the value \( v \) to its variable. To this cost the agent adds \( h(v) \) since this is the minimal cost which will be added from conflicts between this assignment and assignments of lower priority agents (who are unassigned on the FB_CPA). This sum is denoted by \( \delta(v) \). The estimate returned by this agent (on a FB_ESTIMATE message) is the minimal \( \delta(v) \), s.t. \( v \in D_i \), since this is a lower bound of the cost which will be added to the CPA cost once this agent extends the CPA with an assignment to its variable.

However, when there is no constraints between the agent computing the returned estimate and agents assigned in the CPA, the minimal of \( \delta(v) \) is equal to \( \min\{h(v) \text{ s.t. } v \in D_i\} \) who is independent of the assignments of higher priority agents. So, it can be computed once per agent in a preprocessing step (in the rest of the paper, we call this value by \( h_{min} \) bound). Consequently, in the AFB-BJ algorithm some costs computing and messages processing are
Figure 1: A problem graph (a) and a BFS tree (b).

Redundant. For example, let assume that we have a problem with \( n \) agents where the first agent \( A_1 \) is connected only with the second. When agent \( A_1 \) perform the CPA it sends \((n - 1)\) FB\_CPA messages for each agent \( A_j \) s.t \( j > 1 \), in consequence \( A_1 \) will receive \((n - 2)\) FB\_ESTIMATE messages (from all unassigned agents \( A_j \) s.t \( j \succ i \)) with an estimated bound equal to \( h_{\min_j} = \min_{v \in D_j}(h_j(v)) \) who is independent to the \( A_1 \)'s assignment. Thus, at the end of search we found that \( A_1 \) has exchanged \((2 \times d \times (n - 2))\) redundant messages (where \( d \) is the domain size of \( A_1 \)).

Using tree structure on agents graph may eliminate the redundancy of some operations in AFB-BJ algorithm. An agent can only receives FB\_CPA message from its linked ancestors, and FB\_ESTIMATE messages from some unassigned agents belonging to the subtree rooted by it, and from its predecessor. On the other hand, in order to increase the asynchronism degree, we use a static final order on the agents according to a breadth first search (BFS) traversal (Fig 1, \( A_1, A_2, A_3, ..., A_{13} \)). The resulting algorithm is called Asynchronous Breadth-First Search DCOP Algorithm (ABFS).

In Constraint Reasoning the Depth-First Search Tree (DFS) is used to arrange variables or agents in a graph on tree structure, where nodes associated with the variables/agents and the edge represents constraints. DFS trees have been investigated as a mean to boost search \[11], [6], [14], and as a base of the well known solvers of the DCOP \[1], [2], [7]. But in our work we are used breadth first search Tree (BFS). Figure 1 shows an example of BFS tree in which we distinguish between tree edges, shown as solid lines (e.g. \( 1 - 3 \)), and back edges, shown as dashed lines (e.g. \( 1 - 12, 2 - 11 \)).
Definition 3.1 (BFS concepts). Given a node $A_i$, we define:

- The level $L_i$ of a node $A_i$ is defined simply in terms of distance from the root of the tree to the node $A_i$.
- The parent $P_i$ of a node $A_i$: the single node on a higher level of the BFS tree that is connected to the node $A_i$ directly through a tree edge (e.g. $P_2 = A_1$).
- The children $\{C^k_i\}$ of a node $A_i$: the set of nodes lower in the BFS tree that connected to the node $A_i$ directly through tree edges (e.g. $\text{Children}(A_2) = \{A_5, A_6\}$).
- The lower-neighbors $LN_i$ of a node $A_i$: the set of nodes descendants connected to $A_i$ through tree or back edges (e.g. $LN_3 = \{A_7, A_8, A_{13}\}$).
- The pseudo-neighbors $PN_i$ of a node $A_i$ is the set of nodes consisted of the union of its lower-neighbors and pseudo-neighbors of its parent ($PN_{P_i}$) (e.g. $PN_3 = \{A_7, A_8, A_{12}, A_{13}\}$).
- We denote by $ST_i$ the subtree rooted at the node $A_i$.

3.2 The Preprocessing Methods

As in Adopt algorithm [8], Asynchronous Breadth-First Search DCOP algorithm (ABFS) performs a preprocessing step. The preprocessing step is realized in two phases. The first phase is to transform the constraint graph into a constraint BFS tree with the property that constraints exist only between a vertex and its ancestors and/or descendants, and to compute the set of pseudo-neighbors $PN_i$ for each agent $A_i$. The second phase is performed in order to provide the agents with $h_{min}$ bounds. This phase will allow agents to eliminate the redundancies observed in AFB-BJ.

Algorithm 1: Preprocessing Method performed by an agent $A_i$ to compute its pseudo-neighbors.

```plaintext
1 # Transform the agents graph to a DFS tree with the property that constraints
2 # exist only between a vertex and its ancestors and/or descendants.
3 # $PN_i[k]$ denote the set of Pseudo-Neighbors of $A_i$ belong to the sub tree
4 # rooted at its child $C^k_i$
5 procedure Init_PN()
6 foreach child $C^k_i$ do
7     $PN_i[k] ← LN_i \cap ST_{C^k_i}$
8     if $i == 1$ then
9         send($PN_{MSG}$, $PN_i[k]$) to $C^k_i$
10 when receive($PN_{MSG}$, $PN$) do
11     foreach child $C^k_i$ do
12         $PN_i[k] ← PN_i[k] \cup (PN \cap ST_{C^k_i})$
13         send($PN_{MSG}$, $PN_i[k]$) to $C^k_i$
```
ABFS DCOP algorithm

We assume that the constraint graph is transformed into BFS tree with the property that constraints exist only between a vertex and its ancestors and/or descendants. Algorithm 1 presents the first phase of the preprocessing step used for computing the set of pseudo-neighbors \( (PN_i) \) of an agent \( A_i \). We write \( PN_i = \cup_{C^k_i} (PN_i[k]) \) where \( PN_i[k] \) is a subset of \( PN_i \) belonging to the subtree rooted by the child \( C^k_i \). The PN_MSG message is used to send from parent to its children a subset \( PN \) of their pseudo-neighbors.

The agent starts by calling Init_PN() procedure to compute for each child \( C^k_i \) a subset of \( PN_i[k] \). This subset is equal to the intersection between the lower neighbors \( LN_i \) and the sub tree rooted at the child \( C^k_i \) (Algorithm 1, lines 5-6). If the agent \( A_i \) is the first agent in the ordering (i.e \( i = 1 \), Algorithm 1, line 7), then it has a complete set of pseudo-neighbors. The agent \( A_1 \) sends on a PN_MSG message for each child \( C^k_1 \) the projection of its \( PN_1 \) on sub-tree rooted at this child \( C^k_1 \) (Algorithm 1, line 8).

When a PN_MSG message is received, for each child \( C^k_i \), the agent conjoint the projection of the carried set \( PN \) on the sub tree rooted at this child \( ST_{C^k_i} \) (Algorithm 1, line 11), next, it sends to the same child the result on a PN_MSG message (Algorithm 1, line 12). This phase is terminated when all leaves calculate their complete set of pseudo-neighbors.

The second phase in the preprocessing step is presented in the following subsection.

3.3 From AFB-BJ to AFBS algorithm

In the AFB-BJ algorithm [3], when an agent \( A_i \) receives a \( CPA = CPA[1..i-1] \) and successfully assigns its variable with a value \( v \), it sends forward copies of \( CPA \) to all unassigned agents and awaits for receiving from them the estimate vectors of the lower bounds. Once it receives an estimate vector, the agent computes a lower bound on the cost of any full assignment extended from \( CPA[1..k] \) (\( FALB[k](v) \)) who is written as follow

\[
FALB(v)[k] = LC(v)[k] + PC[k] + \sum_{j>i} FC_j(v)[k] \tag{1}
\]

Where \( PC \) denotes the Past-Costs, it is a vector of size \( n + 1 \), in which the \( k^{th} \) element \( (0 \leq k \leq n) \) is equal to the cost of \( CPA[1..k] \), \( LC(v) \) denote the Local-Costs, it is a vector of size \( n + 1 \), in which \( k^{th} \) element \( (0 \leq k \leq n) \) is

\[
LC(v)[k] = \sum_{(A_j,v_j)\in CPA \ s.t \ j \leq k} \text{cost}((A_i = v), (A_j = u))
\]

and \( FC_j(v) \) denote the Future-Costs, it is a vector of size \( n + 1 \), in which the \( k^{th} \) element \( (0 \leq k \leq n) \) contains a lower bound on the cost of assigning a value to \( A_j \) with respect to the partial assignment \( CPA[1..k] \).
Both $LC(v)[k]$ and $PC[k]$ are computed locally. Only the $FC_j(v)[k]$’s calculation that needs cooperation of the unassigned agents, and thus poses problem. Let use the vectorial notation $FC_j(v) = [FC_j(v)[1], ..., FC_j(v)[k]]$, we have $FC(v) = \sum_{j>i} FC_j(v)$, thus the equation (1) can be written:

$$FALB(v) = LC(v) + PC + FC(v)$$

Where

$$FC(v) = \sum_{j>i} FC_j(v) = \sum_{j \in ST_i} FC_j(v) + \sum_{j \notin ST_i \& (j>i)} FC_j(v) \quad (2)$$

**Property 1** Let assume that the agents graph is transformed into a BFS tree where the agents are ordered according to the BFS traversal. If an agent $A_i$ has received some FB$_\leq$ESTIMATE messages:

$$\sum_{j \in ST_i} FC_j(v) = \sum_{j \in PN_i} FC_j(v) + \sum_{j \notin PN_i \& j \in ST_i} H_{\leq \min_j} \quad (3)$$

Where $H_{\leq \min_j}$ is a vector with elements are all equal to $h_{\leq \min_j}$ the lower bound on the cost of constraints involving the assignment $A_j = v$ and all agents $A_k$ such that $k > j$.

**Proof:** The agents belong to $ST_i$ can be divided into two disjoints subset: the pseudo-neighbors of $A_i$: $PN_i$ and the remains $\neg PN_i = \{ST_i - PN_i\}$. From the definitions of pseudo-neighbors and future cost [3], it is clear that each agent $A_j \in \neg PN_i$ is not connected with any agent in the CPA[1..i]. Thus, the estimate vector sent by $A_j$ has the form $H_{\leq \min_j} = [h_{\leq \min_j}, h_{\leq \min_j}, ..., h_{\leq \min_j}]$ where $h_{\leq \min_j}$ is the minimal of the lower bound on the cost of constraints involving the assignment $A_j = v$ and all agents $A_k$ such that $k > j$. Therefore $\sum_{j \in ST_i} FC_j(v) = \sum_{j \in PN_i} FC_j(v) + \sum_{j \notin PN_i \& j \in ST_i} H_{\leq \min_j}$.

**Remark 1:** If $C^k_i$ is a child of $A_i$ then:

$$\sum_{j \in ST_{C^k_i}} FC_j(v) = \sum_{j \in PN_i \& j \in ST_{C^k_i}} FC_j(v) + \sum_{j \notin PN_i \& j \in ST_{C^k_i}} H_{\leq \min_j} \quad (4)$$

**Property 2** Let assume that the agents graph is transformed into a BFS tree where the agents are ordered according to the BFS traversal. Let $L_1_i = \{l_1_k\}$ and $L_2_i = \{l_2_k\}$ two disjoints subset of agents having the same level of an node/agent $A_i$. Where $L_1_i$ are already assigned in CPA and $L_2_i$ not assigned in CPA, we have:
Proof: Denote by $\neg ST_{i+} = \{A_j/(j \notin ST_i) \land (j \succ i)\}$ the set of agents who succeed $A_i$ and not in subtree rooted at $A_i$. The agents belong to $\neg ST_{i+}$ can be divided in two disjoints subset: $\cup_{k \in L1, i} ST_{l1k}$ the set of agents in all subtree rooted at all agents in $L1$, and $(\cup_{k \in L2, ST_{l2k}})$ the set of all subtree rooted at all agents in $L2$. This second subset is equal to the set $(\cup_{k \in L2, ST_{l2k}} \land j \in ST_{l2k})$, thus the equation (4) can be written:

$$
\sum_{j \in ST_{i. t. (j > i)}} FC_j(v) = \sum_{k \in L1} [\sum_{j \in PN_{l1k}} FC_j(v)] + \sum_{k \in L2} [\sum_{j \in PN_{l2k} \land j \in ST_{l2k}} FC_j(v)]
$$

Using Property 1 and Remark 1, the prove of the Property 2 is obtained.

Let $L1_i = \{l1k\}$ and $L2_i = \{l2k\}$ two disjoints subset of agents having the same level of an node/agent $A_i$. Where $L1_i$ are already assigned in CPA and $L2_i$ not assigned in CPA, the lower bound on the cost of any full assignment extended from CPA[1..k] computed by an agent $A_i$ can be written:

$$
FALB(v)[k] = LC(v)[k] + PC[k] + FC1[k] + FC2[k] + Hmin[k] \tag{5}
$$

Where $FC1[k] = \sum_{j \in PN_i} FC_j(v)[k]$

$FC2[k] = \sum_{k \in L1} [\sum_{j \in PN_{l1k}} FC_j(v)] + \sum_{k \in L2} [\sum_{j \in PN_{l2k} \land j \in ST_{l2k}} FC_j(v)]$

and

$$
Hmin[k] = \sum_{j \notin PN_i \land j \in ST_i} H\_min_j + \sum_{k \in L1} [\sum_{j \notin PN_{l1k} \land j \in ST_{l1k}} H\_min_j] + \sum_{k \in L2} [\sum_{j \notin PN_{l2k} \land j \in ST_{l2k}} H\_min_j]
$$

Remark 2: The full assignment lower bound $FALB(v)[k]$ is composed of tree parts: the first part is computed locally ( $LC(v)[k] + PC[k]$), the second part is calculated during search ($FC1[k] + FC2[k]$), and a third part $Hmin[k]$ can be computed once per agent in a preprocessing method.

Remark 3: In a BFS tree graph, each agent $A_k$ in a subtree rooted by an agent in the same level of an agent $A_i$ is not connected with $A_i$. Therefore, the estimate vector to compute for agent $A_i$ by $A_k$ should have the same values of the first $i - 1$ elements of the estimate vector computed for the agent $A_{i-1}$, and the $i^{th}$ element is equal to element $i - 1^{th}$. Thus, the vector $FC2$ of the equation...
Figure 2: The FB_ESTIMATE messages sent mechanism in ABFS

(4) can be computed by agents previously receiving an FB_ESTIMATE message.

**Proposition** (From AFB-BJ to AFBS algorithm):
Let assume that the agents graph is transformed into a BFS tree where the agents are ordered according to the BFS traversal.

- Instead of sending FB_CPA messages to all unassigned agents, an ABFS agent sends the FB_CPA messages only to its pseudo-neighbors.

- The elements of the vector FC2 of the equation 5 should be received from $A_{i-1}$ (agent that precedes $A_i$) carried in a FB_ESTIMATE message.

The agents send FB_CPA messages only to their pseudo-neighbors. Thus, using the response of this message (FB_ESTIMATE messages) an agent $A_i$ can compute the elements of the vector FC1. Whereas the elements of the vector FC2 are received from $A_{i-1}$ carried in a FB_ESTIMATE message. The figure 2 illustrates an example of FB_ESTIMATE messages sending mechanism in ABFS algorithm. The agent $A_1$ (resp. $A_3$) receives a FB_ESTIMATE message from its pseudo-neighbor $A_{10}$ (resp. $A_{12}$). After having used this message (adding the carried vector to FC1) the agent $A_1$ (resp. $A_3$) increases the size of the received vector (duplicates the last element in this vector) and transmits the resulting vector to its successor $A_2$ (reps. $A_4$) through a FB_ESTIMATE message labelled by its name ($A_1$ (resp. $A_3$)). The agent $A_2$ (resp. $A_4$) uses the message received to calculate FC2 and performs the same operations as its predecessor realized (increases the vector size and transmit it) without changing the message label. The FB_ESTIMATE message labelled by $A_1$ (resp. $A_3$) will be communicated via an agent to its successor. Before that the agent proceeds to use an arrived FB_ESTIMATE message it checks if the message label is equal to the name of its parent. If it is the case the message is ignored (agent $A_4$ (resp. $A_7$)).
Remark 4: The value of $H_{\text{min}}[k]$ is independent of $k$ because it is computed by a set of unassigned agents without using the CPA. Therefore it can be computed in the second phase of the preprocessing step. We note this value by $H_{\text{min}}$.

Algorithm 2: The second phase of the preprocessing step performed by an agent $A_i$ in ABFS algorithm

```plaintext
1 $h_{\text{min}} \leftarrow \text{Compute}_h(h_{\text{min}})$
2 $LH \leftarrow <A_i, h_{\text{min}}>$
3 if $I$ m a leave then
4     Send($LH_{\text{MSG}}, LH$) to my parent
5 end
6 when receive($LH_{\text{MSG}}, lh$) do
7     add $lh$ to $LH$
8     if I have received all $LH_{\text{MSG}}$ messages waited from my children then
9         Send($LH_{\text{MSG}}, LH$) to my parent
10        if $i == 1$ then
11            Broadcast($CLH_{\text{MSG}}, LH$)
12        end
13    end
14 end
15 when receive($CLH_{\text{MSG}}, lh$) do
16     Use the expression of equation (5) to calculate $H_{\text{min}}$
17 end
18 procedure Compute$_h$(h$_{\text{min}}$
19     $h_{\text{min}} \leftarrow \infty$
20     foreach $v \in D$ do
21         $h(v) \leftarrow 0$
22         foreach $A_j$ s.t $j > i$ do
23             $h_j(v) = \min$ (cost$(i, v), (j, u))$ s.t.$ $u \in D_j$
24             $h(v) = h(v) + h_j(v)$
25         end
26     $h_{\text{min}} \leftarrow \min(h_{\text{min}}, h(v))$
27     end
28     return $h_{\text{min}}$
```

Algorithm 2 presents the second phase of the preprocessing step. In this phase we assume that all agents known the constraints graph. The agents exchange two types of messages: $LH_{\text{MSG}}$ message from children to parents, it contains a list of pair $<A_i, h_{\text{min}}>$, and $CLH_{\text{MSG}}$ message that carries the complete list of all minimal cost $h_{\text{min}}$ of all agents. The agents (starting from the
leaves) send LH_MSG messages to their parents. The basic process is as follow: the leaves start by computing and sending LH_MSG messages to their parents (Algorithm 2, lines 1-5). Subsequently, all nodes $A_i$ do:
- receive and join all messages from their children (Algorithm 2, line 8).
- join also the minimal of its lower bound on the cost of constraints involving the assignment $A_i = v$ and all agents $A_j$ such that $j > i$ (i.e. $\min(h(v)))$ (Algorithm 2, line 2).
- sends result to parent as a new LH_MSG message (Algorithm 2, line 9).
- when the root of tree has received all LH_MSG messages waited from its children, it broadcast for all agents a CLH_MSG message who is carried all minimal cost $h_{\min}$ of all agents (Algorithm 2, line 11).
- when an agent receives the CLH_MSG message, it calculates the value of $H_{\min}$ using the expression of equation (5) (Algorithm 2, lines 15-17).

4 Asynchronous Breadth-First Search DCOP Algorithm

Asynchronous Breadth-First Search (ABFS) algorithm is obtained by changing the search strategy of AFB-BJ from performing on an arbitrary final static order to a tree structure with breadth first search (BFS) traversal ordering. Algorithm 3 presents ABFS algorithm running by each agent $A_i$ in the system. For simplicity, we choose to omit the pseudo-code detailing the calculation of LC, PC and FALB. For the same reason, we omit the timestamping mechanism [4] which is used to determine which messages are relevant and which one are obsolete.

After the two phases of preprocessing step presented in algorithms 1, 2. The agents start the Algorithm 3 to solve the distributed problem.

All agents call init() procedure and then perform incoming messages until termination. At first, each agent updates $B$ to be the cost of the best full assignment found so far, and since no such assignment was found it is set to infinity. The $FC1$ and $FC2$ estimate vectors are setting to zeros (Algorithm 3, line 3-4). Only the first agent ($A_1$) creates an empty CPA and then begins the search process by calling assign_CPA (Algorithm 3, lines 6-7), in order to find a value assignment for its variable.

When the agent $A_i$ receives $FB\_MSG$ and $NEW\_SOLUTION$ messages, it performs the similar procedures as AFB-BJ algorithm ([3]). The backtrack procedure of ABFS is also identical to that of AFB-BJ.
Algorithm 3: ABFS DCOP algorithm running by an agent $A_i$
When the agent $A_i$ receives the CPA_MSG message it performs the similar procedure as AFB-BJ algorithm [3] except that only future costs $FC_1$ associated with $A_i$’s pseudo-neighbors are re-initialized (Algorithm 3, lines 15-17). Also, the sum $FC_2 + H_{min}$ is added to the local cost $LC(v)$ in order to ameliorate the value ordering heuristic used in AFB-BJ (Algorithm 3, line 18).

In assign_CPA procedure, the assigned value must be such that the sum of the cost of the CPA, $\delta(v)$, and $H_{min}$ doesn’t exceed the upper bound $B$ (Algorithm 3, lines 50-51). The addition of $H_{min}$ enables us to improve moreover the algorithm performance. If no such value is found, then the assignment of some higher priority agent must be altered, so backtrack is called (Algorithm 3, lines 53). When a full assignment is found which is better than the best full assignment known so far, it is broadcasted to all agents (Algorithm 3, lines 56-57). After succeeding to assign a value, the CPA is sent forward to the next unassigned agent (Algorithm 3, line 61). Concurrently, forward bounding requests (i.e. $FB_{CPA}$ messages) are sent to all pseudo-neighbors of $A_i$ (Algorithm 3, lines 62-63).

In backtrack procedure, the agent calls backtrackTo() to compute to which agent the CPA should be sent, and backtracks the search process (by sending the CPA) back to that agent. If the backtrack is launched from the first agent, the terminate broadcast ends the search process in all agents (Algorithm 3, line 78). The function backtrackTo() goes over all candidates, from $i - 1$ down to 1, looking for the first agent who has a chance to reach a full assignment with a cost lower than $B$. If such agent $j$ exist, before returning $j$ the agent re-initializes all future costs $FC_2$ such that $A_S$ succeeds $A_j$ ($s \geq j$) (Algorithm 3, lines 75-77).

When an agent $A_i$ receives a estimate vector (when received $FB_{ESTIMATE}$), it saves the carried estimate vector if it is coming from one of its pseudo-
neighbors (Algorithm 3, lines 31-32). Next, the agent increases the size the received vector by one and sends it in a new FB_ESTIMATE message after labelling it with its name (Algorithm 3, line 33-34). If the received message is coming from the agent predecessor \((j = i - 1)\). The agent saves the received estimate vector if the label of received message \((A_S)\) is not equal to the name of \(A_i\)’s parent \((P_i)\) (Algorithm 3, lines 27-28). Next, the agent increases the size the received vector by one and sends it in a new FB_ESTIMATE message with the same received label \((A_S)\) (Algorithm 3, lines 29-30).

5 Experimental Evaluation

We considered three different domains for our experiments. The first was a random Max-DisCSPs in which all constraint costs (weights) are equal to one [7]. The second was a distributed graph coloring problem with 3 colors in which cost of one is incurred if two neighbors choose the same color, and no cost is incurred otherwise [7], [13]. The third domain simulates a scenario of Real-world problem involving the Distributed Meeting Schedule problem (DMS). In a DMS, a group of persons wish to attend several meetings. The attendees try to optimize their calendars according to personal preferences maintaining the privacy of their information. Each meeting is subject to many constraints. We have used the same scenarios used in [9] to test the Adopt algorithm in real-world problems, mapping them as DCOPs using Distribute Multi-Event Scheduling (DiMES), following the private events as variables approach (PEAV) [9].

All experiments were performed on DisChoco plateform [5] in which agents are simulated by threads which communicate only through message passing. We have used a uniform distribution of message delay. The measures of per-
Figure 5: (a) The communication load. (b) Number of no concurrent constraints checks performed by ABFS and ABF-BJ. Both measures are computed for a DMS/PEAV scenario.

Performance used to evaluate presented algorithms are run-time, in the form of the number of no concurrent constraints checks (NCCCs), and communication load, in the form of the total number of messages sent [10]. Each measure presents an average on 100 random problem instances. A random binary Max-DisCSPs generator is characterized by (\(\#n, \#d, p_1, p_2\)) where \(\#n\) is the number of agents/variables, \(\#d\) the number of values in each variable domain, \(p_1\) the probability of a constraint among any pair of variables and \(p_2\) the tightness values (probability for the occurrence of a violation (a non zero cost) among two assignments of values to a constrained pair of variables). Figure 3-(a) (resp. 3-(b)) presents the average of total number of messages sent for a random Max-DisCSP with \(p_1 = 0.4\) (resp. the number of no concurrent constraints checks (NCCCs)) performed by ABFS and AFB-BJ algorithms. We show that communication load of ABFS algorithm is smaller than the AFB-BJ’s one, and the number of NCCCs performed by ABFS algorithm is bit smaller than the AFB-BJ.

The figure 4-(a) (resp. 4-(b)) presents the number of messages sent (resp. the number of no concurrent constraints checks (NCCCs)) performed by ABFS and AFB-BJ algorithms according to the number of agents on the distributed graph coloring with 3 colors. The figures illustrate that ABFS outperform AFB-BJ and the difference between performances of ABFS and AFB-BJ increases when the number of agents increases.

The evaluation of presented algorithm on Distributed Meeting Schedule problem (DMS) has performed on random DMS following the PEAV approach. We have tested 6 DMS problems classes. Each DMS problem class is represented by the pair \((m, p) = (meeting, participant)\) in which there is \(p\) agents with multiple variables \((m\) variables to the maximum). There are 8 elements in domain, each of them represents a possible meeting’s start time. Figures
5-(a) and 5-(b) present the performances of both presented algorithms on a DMS/PEAV scenario. In our implementation we have used the compilation formulation where the local solutions are considered as a complex values, this justify the small number of messages sent in the system. We show that ABFS outperforms AFB-BJ in term of communication load and NCCCs.

6 Conclusion

In this paper we have presented Asynchronous Breadth-First Search DCOP algorithm (ABFS) that improves Asynchronous forward-bounding with backjumping (AFB-BJ). The ABFS uses a new search strategy based on tree structure agents graph where the agents are ordered on a breadth first search traversal, and adds a preprocessing methods allow it to eliminate some redundancy observed in AFB-BJ. Experiments evaluation show that on benchmark problems (random Max-DisCSPs, graph coloring and Meeting Schedule problems) the proposed algorithm (ABFS) obtains more improvements than AFB-BJ algorithm.

References


Received: February 6, 2008