Journal of Artificial Intelligence Research 45 (2012) 287-304 Submitted 05/12; published 10/12

Research Note

Removing Redundant Messages in N-ary BnB-ADOPT

Patricia Gutierrez PATRICIA@IIIA.CSIC.ES
Pedro Meseguer PEDRO@IIIA.CSIC.ES
IIIA - CSIC, Universitat Autònoma de Barcelona
08193 Bellaterra, Spain

Abstract

This note considers how to modify BnB-ADOPT, a well-known algorithm for optimally solving distributed constraint optimization problems, with a double aim: (i) to avoid sending most of the redundant messages and (ii) to handle cost functions of any arity. Some of the messages exchanged by BnB-ADOPT turned out to be redundant. Removing most of the redundant messages increases substantially communication efficiency: the number of exchanged messages is –in most cases– at least three times fewer (keeping the other measures almost unchanged), and termination and optimality are maintained. On the other hand, handling n-ary cost functions was addressed in the original work, but the presence of thresholds makes their practical usage more complex. Both issues –removing most of the redundant messages and efficiently handling n-ary cost functions– can be combined, producing the new version BnB-ADOPT$^+$. Experimentally, we show the benefits of this version over the original one.

1. Introduction

Distributed Constraint Optimization Problems (DCOPs) have been used to model many actual world multiagent coordination problems, such as meeting scheduling (Maheswaran, Tambe, Bowring, Pearce, & Varakantham, 2004), sensor network (Jain, Taylor, Tambe, & Yokoo, 2009), traffic control (Junges & Bazzan, 2008), and coalition structure generation (Ueda, Iwasaki, & Yokoo, 2010). DCOPs include a finite number of agents, with the usual assumption that each agent holds one variable with a finite and discrete domain. Variables are related by cost functions that define a cost for every combination of value assignments. Such costs represent preferences or penalty relations. The cost of a particular assignment is the sum of all cost functions evaluated on that assignment. The goal is to find a complete assignment of minimum cost by message passing.

Considering distributed search for DCOP solving, the first proposed complete asynchronous algorithm was ADOPT (Modi, Shen, Tambe, & Yokoo, 2005). Later on, the closely related BnB-ADOPT (Yeoh, Felner, & Koenig, 2010) was presented. BnB-ADOPT changes the nature of the search from ADOPT best-first search to a depth-first branch-and-bound strategy, obtaining better performance. Both algorithms are complete, so they compute the optimum cost and are guaranteed to terminate. ADOPT and BnB-ADOPT have a similar communication strategy, using a similar set of messages (with small differences in their processes). They also share the same data structures and semantics to store and update their internal tables.

In the last years, several complete DCOP algorithms have been proposed. Following the classification that appears in the work of Yeoh et al. (2010), in the search algorithm class we mention (in addition to the previously cited ADOPT and BnB-ADOPT), Synchronous Branch and Bound (SBB) (Hirayama & Yokoo, 1997), No Commitment Branch and Bound (NCBB) (Chechetka & Sycara,

2006) and Asynchronous Forward Bounding (AFB) (Gershman, Meisels, & Zivan, 2009). As distributed inference algorithms we mention Dynamic Programming Optimization (DPOP) (Petcu & Faltings, 2005). Our interest for ADOPT and BnB-ADOPT comes from the fact that they require polynomial memory, they are asynchronous and communication is limited to neighbors. Distributed search algorithms use polynomial memory, while DPOP requires exponential memory in the worst case. Asynchronous algorithms have been shown more suitable to the distributed context, because agents are active all the time and this approach is globally more robust to failures. SBB and NCBB are examples of synchronous algorithms. Limiting communication to neighbors (no agent can communicate with other agent it is not constrained with) is a common requirement for some applications (for instance in sensor networks). There are algorithms, such as AFB, which do not have this restriction and allow agents to broadcast messages.

ADOPT and –to a lesser extent– BnB-ADOPT exchange a large number of messages. Often, this is a major drawback for their practical applicability, despite their good theoretical properties (soundness, optimality, termination). Aiming at decreasing the number of exchanged messages without compromising optimality and termination, this paper contains results for detecting redundant messages in BnB-ADOPT. Using these results we generate a new version which avoids sending most of the redundant messages. Experimentally, we have seen that removing most of the redundant messages significantly decreases communication costs (the new algorithm exchanges at least three times fewer messages) on several widely used DCOP benchmarks. The proposed modifications can also be applied to ADOPT (Gutierrez & Meseguer, 2010a), although we do not detail results here since –according to our experiments– BnB-ADOPT usually achieves better performance.

In DCOPs, it is somehow natural to use binary cost functions, because an agent has a one-to-one relation with each of its neighbors. Since each agent holds a single variable, this naturally brings to binary functions. However, in some cases an agent may have a cost function of higher arity with a subset of agents. The original ADOPT (Modi et al., 2005) proposes a way to deal with n-ary constraints, and BnB-ADOPT takes the exact same strategy to deal with constraints with arity higher than two. However, this strategy may cause inefficiency when applied to BnB-ADOPT. We provide a simple way to correct this inefficiency, which can be easily integrated in BnB-ADOPT.

We have combined both improvements in a new version of the algorithm called BnB-ADOPT$^+$. Experimental results show the benefits of the proposed approach with respect to the original algorithm. A comparison with other state-of-the-art DCOP algorithms is also included.

This paper is structured as follows. First, we provide some basic definitions and a short description of the BnB-ADOPT algorithm in Section 2. We introduce the detection of redundant messages in Section 3. We discuss how to generalize the algorithm to deal efficiently with cost functions of any arity in Section 4. Using these results, we propose a new version of BnB-ADOPT, called BnB-ADOPT$^+$. In Section 5 we report experimental results on the benefits of BnB-ADOPT$^+$ with respect to its previous version. In addition, we present results comparing BnB-ADOPT$^+$ with other DCOP algorithms. Finally, we conclude in Section 6.

2. Background

In this Section we provide some basic DCOP definitions and a short description of the BnB-ADOPT algorithm (Yeoh et al., 2010).

2.1 DCOP

A *Distributed Constraint Optimization Problem* (DCOP) (Modi et al., 2005), is formally defined by $(\mathcal{X}, \mathcal{D}, \mathcal{F}, \mathcal{A}, \phi)$ where:

- $\mathcal{X} = \{x_1, ..., x_n\}$ is a set of n variables.
- $\mathcal{D} = \{D_1, ..., D_n\}$ is a collection of finite domains such that variable x_i takes values in D_i.
- $\mathcal{F}$ is a set of binary cost functions; $f_{ij} \in \mathcal{C}$ involving the set of variables $var(f) = \{x_i, x_j\}$ is a mapping $D_i \times D_j \mapsto \mathbb{N} \cup \{0, \infty\}$ that associates a non-negative cost to each combination of values of variables x_i and x_j. The scope of f is $var(f)$, and its arity is $|var(f)|$.
- $\mathcal{A} = \{1, \dots, p\}$ is a set of p agents.
- $\phi : \mathcal{X} \to \mathcal{A}$, maps each variable to one agent.

A cost function (also called soft constraint elsewhere) f evaluated on a particular value tuple t gives the cost one has to pay for taking the values of t on the variables $var(f)$. Completely permitted tuples in f have 0 cost, while completely forbidden tuples have ∞ cost. Intermediate costs are associated to tuples which are neither completely permitted nor completely forbidden. The cost of a complete assignment is the sum of the evaluation of all cost functions on that assignment. A *solution* is a complete assignment with cost below a user threshold. A solution is *optimal* if it has minimum cost. We make the usual simplifying assumption that each agent owns exactly one variable, so agents and variables can be used interchangeably (we connect agents and variables by subindexes, agent i owns variable x_i). We also assume that a cost function f among several variables is known by every agent that owns a variable in $var(f)$ (Yokoo, Durfee, Ishida, & Kuwabara, 1998). Agents communicate through messages which are never lost and, for any pair of agents, messages are delivered in the same order that they were sent.

A *constraint graph* represents a DCOP instance, where nodes in the graph correspond to variables and edges connect pairs of variables appearing in the same cost function. A *depth-first search* (DFS) *pseudo-tree* arrangement has the same nodes and edges as the constraint graph and satisfies that (i) there is a subset of edges, called *tree edges*, that form a rooted tree and (ii) two variables in a cost function appear in the same branch of that tree. The other edges are called *backedges*. Tree edges connect parent-child nodes, while backedges connect a node with its pseudo-parents and its pseudo-children. DFS pseudo-trees can be constructed using distributed algorithms (Petcu, 2007).

2.2 BnB-ADOPT

BnB-ADOPT (Yeoh et al., 2010) is a distributed algorithm that optimally solves DCOPs using a depth-first branch-and-bound search strategy. It is closely related to ADOPT (Modi et al., 2005), maintaining most of its data structures and communication framework. BnB-ADOPT starts constructing a DFS pseudo-tree arrangement of its agents. After this, each agent knows its parent, pseudo-parents, children and pseudo-children.

2.2.1 DATA STRUCTURES

During execution an agent i maintains: its current value d_i; its current context X_i, which is its knowledge about the current value assignment of its ancestors; a timestamp for the current value

d_i and each value assignment of the current context (so value recency can be compared); for every value $d \in D_i$ and context X_i, a lower and upper bound $\boldsymbol{LB}_i(d)$ and $\boldsymbol{UB}_i(d)$; and two bounds $\boldsymbol{LB}_i$ and $\boldsymbol{UB}_i$ that are calculated in the following way:

$$\delta_i(d) = \sum_{(x_j,d_j)\in X_i} f_{ij}(d, d_j)$$

$$\boldsymbol{LB}_i(d) = \delta_i(d) + \sum_{x_c \in children} lb_{i,c}(d)$$
$$\boldsymbol{UB}_i(d) = \delta_i(d) + \sum_{x_c \in children} ub_{i,c}(d)$$

$$\boldsymbol{LB}_i = \min_{d \in D_i}\{\boldsymbol{LB}_i(d)\}$$
$$\boldsymbol{UB}_i = \min_{d \in D_i}\{\boldsymbol{UB}_i(d)\}$$

where $\delta_i(d)$ is the sum of costs of all cost functions between i and its ancestors given that i assigns value d and ancestors assign their respective values in X_i. Tables $lb_{i,c}(d)$ and $ub_{i,c}(d)$ store upper and lower bounds of children c, for all values $d \in D_i$ and current context X_i. $\boldsymbol{LB}_i$ and $\boldsymbol{UB}_i$ are lower and upper bounds of the optimal solution for context X_i. Due to memory limitations, agent i can only store lower and upper bounds for *one* context. Agents may reinitialize their bounds each time there is a context change.

The goal of every agent i is to explore the search space and ultimately chooses the value that minimizes LB_i. An agent in BnB-ADOPT changes its value assignment only when it is able to determine that the optimal solution for that value is provably no better than the best solution found so far for its current context. In other words, when $\boldsymbol{LB}_i(d_i) \geq \boldsymbol{UB}_i$ for current value d_i.

To prune values during search, agents use a threshold value TH, initially ∞. TH_{root} remains ∞ during the entire solving process. In other agents, threshold values are calculated and sent from parent to children in the following way. A threshold th sent from agent i with value d to its child c is calculated as:

$$th = min(TH_i, UB_i) - \delta_i(d) - \sum_{ch \in children, ch \neq c} lb_{i,ch}(d)$$

Thresholds represent an estimated upper bound for the current context. Therefore, agents can prune values using thresholds sent from their parents. That is, agent i changes its value d_i (prunes it) when $\boldsymbol{LB}_i(d_i) \geq \min\{\boldsymbol{TH}_i, \boldsymbol{UB}_i\}$. If $LB_i = UB_i$, agent i has reached the cost of an optimal solution for context X_i.

2.2.2 Communication

Some communication is needed in BnB-ADOPT to calculate the global costs of individual agents assignments and to coordinate search towards the optimal solution. BnB-ADOPT agents use three types of messages: VALUE, COST and TERMINATE, defined as follows:

- VALUE($i; j; val; th$): agent i informs child or pseudo-child j that it takes value val with threshold th;
- COST($k; j; context; lb; ub$): agent k informs parent j that with $context$ its bounds are lb, ub;
- TERMINATE($i; j$): agent i informs child j that i terminates.

As mentioned above, in BnB-ADOPT each value assignment is timestamped (both in VALUE or COST messages). This permits VALUE and COST messages to update the context of the receiver agent, only if their values are more recent.

Upon reception of a VALUE message, value val is copied in the receiver context if its timestamp is more recent, and threshold th is updated if the sender is the parent of the receiver.

Upon reception of a COST message from child c, values in the $context$ of the COST message more recent than in the receiver context are copied in the receiver agent. If the receiver context is compatible with the COST message $context$, then the agent updates its lower and upper bounds $lb_{i,c}(d)$ and $ub_{i,c}(d)$ with the lower and upper bounds in the COST message, respectively. Otherwise, the COST message is discarded. Contexts are *compatible* iff they agree on common agent-value pairs. Every time there is a context change, agents check if their bounds must be reinitialized.

2.2.3 EXECUTION

A BnB-ADOPT agent executes the following loop. First, it reads and processes all incoming messages. After completely processing the message queue, it changes its value if the lower bound of the current value surpasses $min\{TH_i, UB_i\}$, because in that case the value is proven to be suboptimal for the current context. Finally, the agent sends the following messages: a VALUE per child, a VALUE per pseudo-child and a COST to its parent. This process repeats until the root agent r reaches the termination condition $LB_r = UB_r$, which means that it has found the minimum cost. Then it sends a TERMINATE message to each of its children and terminates. Upon reception of a TERMINATE message, agent i sends TERMINATE messages to its children; agent i terminates when $LB_i = UB_i$.

For the rest of the paper, we assume that the reader has some familiarity with BnB-ADOPT (for a more detailed description, see the original source, Yeoh et al., 2010).

3. Redundant Messages

In this Section we present the results on redundant messages considering BnB-ADOPT working on DCOP instances with binary cost functions. In the following i, j and k are agents, all executing BnB-ADOPT. Agent i, holding variable x_i, *takes value* v when the assignment $x_i \leftarrow v$ is made and i informs its children, pseudo-children and parent about its new value assignment. The *state* of i is defined by: (i) its value, (ii) its context, $context_i$, as the set of value assignments of agents located before i in its branch (timestamps are not considered part of the context), and (iii) for each possible value d and each children c, the lower and upper bounds $lb_{i,c}(d)/ub_{i,c}(d)$.

A message msg sent from i to j is *redundant* if, at some future time t, other messages received by j between msg and t would cause the same effect, so msg could have been avoided. A message msg sent from i to j containing the assignment $x_i \leftarrow v$ with timestamp t updates $context_j[i]$ (that is, the part of $context_j$ containing the value of x_i) which has timestamp t' if and only if $t > t'$.

Lemma 1 *In BnB-ADOPT, if agent i sends two consecutive VALUE messages to agent j with timestamps t_1 and t_2, there is no message with timestamp t for agent i assignment such that $t_1 < t < t_2$.*

Proof. There is no VALUE message with timestamp between t_1 and t_2 for i, since both VALUE messages are consecutive and sent from agent i. COST messages build their contexts from the information in VALUE messages. Since no VALUE contains a timestamp between t_1 and t_2 for i, no COST will contain it for i. □

Theorem 1 *In BnB-ADOPT, if agent i sends to agent j two consecutive VALUE messages with the same val, the second message is redundant.*

Proof. Let V_1 and V_2 be two consecutive VALUE messages sent from agent i to agent j with the same value val and timestamps t_1 and t_2, $t_1 \leq t_2$. If $t_1 = t_2$, V_2 will always be discarded (and therefore V_2 will always be redundant), so we concentrate on the second option, $t_1 < t_2$. Between V_1 and V_2 agent j may receive any messages coming from other agents. When V_1 reaches j, the following cases are possible:

1. V_1 does not update $context_j[i]$ (V_1 is discarded). When V_2 arrives to j it may happen:
   (a) V_2 does not update $context_j[i]$ (V_2 is discarded). Future messages will be processed as if V_2 has not been received, so V_2 is redundant.
   (b) V_2 updates $context_j[i]$ which has timestamp t. There are two options: (i) $t_2 > t > t_1$ and (ii) $t_2 > t = t_1$. Option (i) is impossible according to Lemma 1. Option (ii) is possible, but since $t = t_1$ the value contained in V_2 is already in $context_j[i]$. About future messages, every message accepted with timestamp t_2 in $context_j[i]$ would also be accepted with timestamp t_1 in $context_j[i]$. Since there are no messages with timestamp between t_1 and t_2 for i, we conclude that V_2 is redundant.
2. V_1 updates $context_j[i] \leftarrow val$, timestamp t_1. When V_2 arrives to j it may happen:
   (a) V_2 does not update $context_j[i]$; as in case (1.a).
   (b) V_2 updates $context_j[i]$: since V_1 updated $context_j$ and Lemma 1, the timestamp of $context_j[i]$ must be t_1. V_2 does not change $context_j[i]$ but its timestamp is rewritten with t_2. Since there are no messages with timestamp between t_1 and t_2 (Lemma 1), any future message that could update $context_j$ with t_2 would also update it with t_1. So V_2 is redundant.

We have not considered the threshold contained in VALUE messages because BnB-ADOPT is complete and terminates without the use of thresholds (they are included to increase efficiency). To see this, it is enough to realize that BnB-ADOPT without using thresholds is equivalent to BnB-ADOPT where thresholds were always equal to ∞ (equal to the initial value of threshold in each agent). All results of Section 5 of the work of Yeoh et al. (2010) remain valid if BnB-ADOPT works with ∞ thresholds.[1] Therefore, BnB-ADOPT without thresholds terminates with the cost of an optimal solution. $\Box$

Lemma 2 *In BnB-ADOPT, if agent k sends two consecutive COST messages C_1 and C_2 with the same context, and k has not detected a context change between C_1 and C_2, then there is no message with timestamp for agent i between the ones of C_1 and C_2 for i, such that its context is incompatible with C_1 and C_2.*

Proof. Each time agent i constrained with agent k changes value it sends a VALUE message to *all* its children and pseudo-children. Since there is no context change between C_1 and C_2, no message with timestamp for i between the ones of C_1 and C_2 can contain a context incompatible with C_1 and C_2; otherwise agent k would have necessarily detected a context change. $\Box$

1. To see this, it is enough to realize that thresholds are not used in any proof of Section 5 but in the proof of Lemma 8. However, the proof of this Lemma remains valid replacing threshold by ∞.

Theorem 2 *In BnB-ADOPT, if agent k sends to agent j two consecutive COST messages with the same content (context, lower/upper bound) and k has not detected a context change, the second message is redundant.*

Proof. Let C_1 and C_2 be two consecutive COST messages sent from k to j with the same content, and $context_k$ has not changed between sending them. Any message may arrive to j between C_1 and C_2 (coming from other agents). Upon reception, the more recent values of C_1 (and later of C_2) are copied to $context_j$ (by **PriorityMerge**, see Yeoh et al., 2010). If j detects a context change, tables $lb_{j,c}(d)$ and $ub_{j,c}(d)$ could be reinitialized. Otherwise, $context_j$ is compatible with the COST context and tables $lb_{j,c}(d)$ and $ub_{j,c}(d)$ are updated with the information contained in the COST message.

Copying C_2 more recent values to $context_j$ is not essential. Let us assume that these values are not copied. Since there is no context change between C_1 and C_2, any message with timestamps between those of C_1 and C_2 will necessarily include a context compatible with C_2, according to Lemma 2. Therefore when C_2 arrives, if C_2 contains values with a more recent timestamp they will not be incompatible with $context_j$, so C_2 only updates timestamps.

Now, let us consider some possible $lb_{j,c}(d)$, $ub_{j,c}(d)$ reinitializations. Since C_2 does not cause a context change in j, because it updates timestamps only (Lemma 2), it does not cause a lower/upper bound reinitialization on j. Because of that, our proof concentrates on the update of bounds.

When C_1 arrives, it may happen:

1. C_1 is not compatible with $context_j$, its bounds are discarded. When C_2 arrives it may happen:
   (a) C_2 is not compatible with $context_j$, so its bounds are discarded. Since $C_2 = C_1$ (except for timestamps), actions that should be done when detecting that C_2 is not compatible were already done after detecting that C_1 was not compatible. Thus, C_2 is redundant.
   (b) C_2 is compatible with $context_j$, so its bounds are included in j. Since C_1 was not compatible, there must be at least one agent above j that changed its value, received by j between C_1 and C_2. There are one or several VALUE messages on its/their way towards k or k descendants. Upon reception, one or several COST messages will be generated. The last of them will be sent from k to j with more updated bounds, so C_2 could have been avoided because a more updated COST will arrive to j. Consequently, C_2 is redundant.

2. C_1 is compatible with $context_j$, so its bounds are included. When C_2 arrives to j it may happen:
   (a) C_2 is not compatible with $context_j$, so its bounds are discarded. Bounds provided by C_2 are useless, because are based on outdated information. In the future a more updated COST will reach j (same reasons as previous case 1.b). So C_2 is redundant.
   (b) C_2 is compatible with $context_j$, so its bounds are included. However, this causes no change in j bounds, unless bounds have been reinitialized. In this case there is at least one agent above j that has changed its value, and that information reached j between C_1 and C_2. The situation is the same as in case (1.b). Hence, C_2 is redundant. $\square$

Temporarily, we define BnB-ADOPT$^+$ as our version of BnB-ADOPT with the following changes: (i) the second of two consecutive VALUE messages with the same i, j, val and th is not sent, and

(ii) the second of two consecutive COST messages with the same k, j, $context$, lb and ub when k detects no context change is not sent. These changes do not affect the algorithm's optimality, as proved next.

Theorem 3 *BnB-ADOPT$^+$ terminates with the cost of an optimal solution.*

Proof. By Theorem 1 if agent i sends two consecutive VALUE messages with the same val to agent j, the second is redundant. However, if they differ in th, we also send the second VALUE message for efficiency purposes. Observe that sending some redundant messages does not cause any incorrect behavior. By Theorem 2, COST messages not sent by BnB-ADOPT$^+$ are redundant so they can be eliminated. BnB-ADOPT terminates with a the cost of an optimal solution (Yeoh et al., 2010), so BnB-ADOPT$^+$ also terminates with the cost of an optimal solution. $\square$

Experimentally, this version only caused some minor benefits. We realized that we have ignored threshold management. Observe that thresholds are reinitialized to ∞ after each context change (caused by VALUE or COST messages); this causes no special difficulty in the original BnB-ADOPT algorithm because each time an agent processes its message queue, it sends VALUE messages to its children containing their thresholds. Now, if some of these VALUE messages are not sent, children will run the algorithm with an ∞ threshold during some periods. To avoid this, children should have a way to ask for threshold to their parents after each reinitialization. This is done using COST messages, which are sent from children to parents. Thus, we define BnB-ADOPT$^+$ as our BnB-ADOPT version with the following changes:

1. Agent i remembers the last message sent to each of its neighbors.
2. A COST message from j to i includes a boolean $ThReq$, set to true when j threshold was reinitialized.
3. If j has to send a COST message to i that is equal to (ignoring timestamps) the last message sent, the new COST message is sent if and only if j has detected a context change between them.
4. If i has to send a VALUE message to j that is equal to (ignoring timestamps) the last message sent, the new VALUE message is sent if and only if the last COST message that i received from j had $ThReq = true$.

It is immediate to see that these changes do not alter the optimality and termination properties of BnB-ADOPT$^+$: original BnB-ADOPT, that includes redundant messages, terminates with the cost of an optimal solution (Yeoh et al., 2010); sending *some* of those redundant messages the algorithm remains optimal and terminates. The proposed changes to avoid redundant messages can also be applied to the ADOPT algorithm (Gutierrez & Meseguer, 2010a).

4. Dealing with N-ary Cost Functions

We have defined DCOPs using binary cost functions, although DCOP definition can be easily extended to include cost functions of any arity. Similarly, ADOPT and BnB-ADOPT can be extended to deal with cost functions of any arity. The proposed BnB-ADOPT extension is exactly the same as the ADOPT extension to handle n-ary cost functions (Yeoh et al., 2010). That extension, described in the work of Modi et al. (2005), is as follows: [2]

> ...a ternary constraint f_{ijk} ... defined over three variables x_i, x_j, x_k ... Suppose x_i and x_j are ancestors of x_k... With our ternary constraint, both x_i and x_j will send VALUE messages to x_k. x_k then evaluates the ternary constraint and sends COST messages back up the tree as normal ... Thus, we deal with an n-ary constraint by assigning responsibility for its evaluation to the lowest agent involved in the constraint. The only difference between evaluation of an n-ary constraint and a binary one is that the lowest agent must wait to receive all ancestors' VALUE messages before evaluating ...

In other words (replacing "constraint" by "cost function"), in the proposed extension agents must send their VALUE messages to the lowest agent of the DFS pseudo-tree involved in a cost function. In the case of a binary cost function, the lower agent (of the two involved in the cost function) always receives VALUE messages. In the case of n-ary cost functions (involving more than two agents), intermediate agents do not receive VALUE messages from the rest of the agents involved in that function. The lowest agent x_k must receive all VALUE messages before evaluating the cost function. Because of this, it is called the evaluator agent. Upon reception of all these messages, x_k evaluates the function and sends a COST message to its parent, which receives and processes the message as any other COST message. When applying this technique to BnB-ADOPT some issues appear, as explained in the following.

4.1 TERMINATE Messages

In our version of binary BnB-ADOPT, each time an agent changes value it sends VALUE messages to its children and pseudo-children. A *non-root* agent terminates when it reaches the termination condition $LB = UB$ after receiving a TERMINATE message from its parent (for the *root* agent no TERMINATE message is needed). As a side-effect, the last value taken by all agents is the optimal value. This feature is very appreciated in distributed environments, because optimal values are distributed among agents without requiring a central agent in charge of the whole solution.

In n-ary BnB-ADOPT, although the *root* agent computes the minimum cost, a direct implementation may not terminate with an optimal value assigned to every agent. Let us consider a ternary cost function among agents i, j and k (as in Figure 1); i is at the root of the DFS pseudo-tree, and k evaluates the cost function. Agent i may explore its last value, jump back to its best value where it reaches termination condition $LB = UB$, it sends a VALUE message to k and a TERMINATE message to its child j. Upon reception of VALUE message, k will send a COST message to j. This COST message contains the last assignment (optimal value) made by i. However, if j has already processed the TERMINATE message from i, it will have ended without processing that COST message, which means that j may end with an outdated context: [3] this causes j to end with an assigned value which may not be an optimal one, since it does not truly minimize the cost of the global solution. [4]

2. It must also be assured that all agents involved in an n-ary cost function lie on the same branch of the pseudo-tree. This is guaranteed since agents sharing n-ary cost functions form a clique in the constraint graph. When performing a depth first traversal to construct the DFS pseudo-tree, agents of the clique will necessarily lie on the same branch.
3. This would not happen if agent i would have sent VALUE messages to every child j informing of value changes, but this is not the original BnB-ADOPT strategy to deal with n-ary cost functions.
4. Technically speaking, j might terminate with an *incorrect* context.

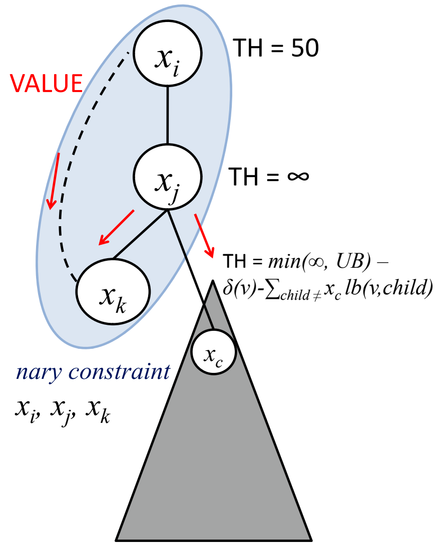

Figure 1: Original BnB-ADOPT dealing with n-ary constrains, use of VALUE messages

A simple way to correct this is to include in TERMINATE messages the last assignment made by the sender agent. In this way, the receiver can update its context and terminate with the value that truly minimizes the lower bound.

4.2 VALUE Messages

If the ADOPT strategy for n-ary cost functions is applied literally to BnB-ADOPT, there are scenarios which are inefficiently solved. For instance, consider Figure 1, where variables of agents i, j, k share a ternary cost function. Agent i is the root of the DFS pseudo-tree and k is the lowest agent (therefore the evaluator) of the ternary cost function. Suppose x_j is constrained with other variables of the problem, represented here as a gray subtree. Since VALUE messages are sent from i and j to k to inform value changes, but no VALUE messages are sent from i to j, agent j will not have a threshold provided by its parent. Thresholds provided by parents are the lowest upper bounds among all visited contexts (the cost of the best solution found so far), while thresholds computed by agents themselves are upper bounds of the cost for the current context. Thresholds were introduced in BnB-ADOPT to speed up problem resolution and increase pruning opportunities, so not having the tightest threshold on agent j –and on j subtree– is clearly detrimental to the algorithm performance.

A simple way to avoid this issue is to send VALUE messages to all descendants (children and pseudo-children). However, this is more than needed. We can avoid unnecessary messages by only sending VALUE messages to the lowest agent in charge of evaluating the cost function –to generate COST messages with updated LB and UB– and to children –to propagate the TH value down in the DFS pseudo-tree–. Any agent involved in a cost function with i, such that it is neither the evaluator of the cost function nor a child of i does not need to receive VALUE messages from i. This is our proposed extension for BnB-ADOPT to deal with n-ary cost functions. Observe that in the binary case, this proposal collapses into the existing operation for both algorithms. In the rest of the paper we assume this extension is included in our version of BnB-ADOPT, to deal with cost functions of any arity.

4.3 Correctness and Completeness

We define n-ary BnB-ADOPT as in the work of Yeoh et al. (2010) (i.e., each agent sends VALUE messages to the evaluator agent of the cost functions it is involved in) plus each agent sends its VALUE messages to all its children in the pseudo-tree. It is easy to show that this n-ary BnB-ADOPT version is optimal and terminates. First, we prove that if agents send VALUE messages to their children and pseudo-children, this extended n-ary BnB-ADOPT terminates with the cost of an optimal solution. Second, we show that VALUE messages send to pseudo-children are redundant (except if the pseudo-child is the evaluator agent of a cost function that involved the sender). Third, we demonstrate that redundant messages in the binary case remain redundant in the n-ary case. Combining these results we obtain the desired output: a proof that n-ary BnB-ADOPT$^+$ terminates with the cost of an optimal solution.

Theorem 4 *N-ary BnB-ADOPT terminates with the cost of an optimal solution.*

Proof. Imagine an extended version of n-ary BnB-ADOPT where agents send VALUE messages to all their descendants (children and pseudo-children) in the pseudo-tree. This extended version of n-ary BnB-ADOPT is working as in the binary case: each agent sends VALUE messages to all its descendants (children or pseudo-children) and it sends a COST message to its parent. In this case, it is easy to check that all the results of Section 5 in the paper of Yeoh et al. (2010) apply here (it is long but conceptually easy; observe that no result of Section 5 in the paper of Yeoh et al. uses the fact that cost functions are binary). In particular, Yeoh et al. proved that binary BnB-ADOPT terminates with the cost of an optimal solution. Therefore, this extended version of n-ary BnB-ADOPT terminates with the cost of an optimal solution.

Now, we consider VALUE messages sent from agent i to its pseudo-children that are not the evaluators of any cost function involving agent i. We show that these messages are redundant. After receiving a VALUE message from i:

1. Agent j updates its context.
2. If the message comes from its parent, agent j rewrites its own threshold with the message threshold.

We know that i is not the parent of j, so we consider point (1) only. Agent j is not the evaluating agent of a cost function involving i; thus, there is another agent k –in the same branch and below i and j– that is in charge of such evaluation. This agent k will receive –for sure– the VALUE messages coming from its ancestors, and then it will send COST messages up the tree. When these COST messages reach j, they will update its context exactly in the same way as after receiving j a VALUE message from i. Original VALUE messages are redundant because the same effect can be obtained with the COST messages arriving from k. Therefore, we can remove these VALUE messages from our extended version of n-ary BnB-ADOPT, and the algorithm will terminate with the cost of an optimal solution. By definition, this algorithm is n-ary BnB-ADOPT. □

Next we prove that redundant messages in the binary case remain redundant in the n-ary case.

Lemma 3 *VALUE and COST messages found redundant for binary BnB-ADOPT remain redundant for n-ary BnB-ADOPT.*

Proof. To prove this Lemma, it is enough to realize that Theorems 1 and 2 remain valid for n-ary BnB-ADOPT. Observe that in the proofs of Lemma 1, Theorems 1 and 2 do not required the use of binary cost functions. The proof of Lemma 2 can be easily generalized to the n-ary case, simply replacing "to all its children and pseudo-children" by 'to all its children and evaluator pseudo-children". □

We define n-ary BnB-ADOPT$^+$ as n-ary BnB-ADOPT removing redundant VALUE and COST messages, as in Section 3.

Corollary 1 *N-ary BnB-ADOPT$^+$ terminates with the cost of an optimal solution.*

Proof. Combining Theorem 4 and Lemma 3 we prove that n-ary BnB-ADOPT is not sending redundant VALUE or COST messages and terminates with the cost of an optimal solution.

As in Section 3, a child may ask its parent to resend the threshold in a VALUE message if its threshold has been reinitialized. These exceptions do not cause any extra difficulty here. The justification to show that sending these messages does not question optimality and termination in the binary case, remains fully valid for the n-ary case. □

In the following we do not make any difference between the binary and n-ary cases; instead we use a single algorithm, n-ary BnB-ADOPT$^+$, that we simply name BnB-ADOPT$^+$.

5. Experimental Results

We experimentally evaluated the performance of original BnB-ADOPT (binary and n-ary versions) against our proposal for n-ary BnB-ADOPT (Section 4) and against BnB-ADOPT$^+$ (that includes the changes proposed in Sections 3 and 4), using a discrete event simulator. Performance is evaluated in terms of communication cost (total number of messages exchanged) and computation effort (NCCCs, non-concurrent constraint checks, see Meisels, Kaplansky, Razgon, & Zivan, 2002). We also consider the number of cycles as the number of iterations the simulator must perform until the solution is found. On each cycle, agents read incoming messages from their message queue, process them, and send outgoing messages when required. The DFS pseudo-tree is generated in a distributed form, following a most-connected heuristic.

First, we evaluate the impact of removing redundant messages in the binary case, comparing original BnB-ADOPT against BnB-ADOPT$^+$. Second, we evaluate performance in non-binary instances. We compare original BnB-ADOPT with our proposal for n-ary BnB-ADOPT (changes of Section 4) and against BnB-ADOPT$^+$ (n-ary BnB-ADOPT saving redundant messages). Lastly, we compare BnB-ADOPT$^+$ against two other well-known algorithms for DCOP solving: Synchronous Branch and Bound (SBB) (Hirayama & Yokoo, 1997) and Asynchronous Forward Bounding (AFB) (Gershman et al., 2009). SBB is a completely synchronous algorithm whereas AFB performs synchronous value assignments but asynchronously computes the bounds used for pruning. SBB and AFB maintain a total order of variables to perform assignments while BnB-ADOPT$^+$ uses a partial ordering following the DFS pseudo-tree structure. We present this last comparison to provide an overall picture of BnB-ADOPT$^+$ and how its asynchronous nature affects the number of messages exchanged and computation.

Experiments are performed on three different benchmarks: random DCOPs (binary and ternary cases), meeting scheduling and sensor networks (both are binary, obtained from a public DCOP repository, Yin, 2008). Random DCOPs are characterized by $\langle n, d, p_1 \rangle$, where n is the number of variables, d is the domain size and p_1 is the network connectivity. Random generation assures connected problems, so all agents of the problem belong to the same constraint graph and the same DFS pseudo-tree. Binary instances contain $p_1 * n(n-1)/2$ binary cost functions, while ternary instances contain $p_1 * n(n-1)(n-2)/6$ ternary cost functions. Costs are selected randomly from the set $\{$0,..., 100$\}$. In meeting scheduling, variables represent meetings, domains represent time slots assigned for meetings, and there are cost functions between meetings that share participants (Maheswaran et al., 2004). In sensor networks, variables represent areas that need to be observed, domains represent time slots, and there are cost functions between adjacent areas (Maheswaran et al., 2004).

Results of the first experiment comparing BnB-ADOPT and BnB-ADOPT$^+$ appear in Table 1 and Table 2. Table 1 shows results on binary random problems averaged over 50 instances. Table 1(a) shows results varying network connectivity with $\langle n = 10, d = 10, p_1 = 0.2...0.8 \rangle$. Table 1(b) shows results varying domain size with $\langle n = 10, d = 6...12, p_1 = 0.5 \rangle$. Table 1(c) shows results varying the number of variables with $\langle n = 6...12, d = 10, p_1 = 0.5 \rangle$. Table 2 (a) shows meeting scheduling instances in 4 cases with different hierarchical scenarios: case A (8 variables), B (10 variables), C (12 variables) and D (12 variables). Table 2 (b) shows sensor network instances in 4 cases with different topologies: cases A (16 variables), B (16 variables), C (10 variables) and D (16 variables). In these two last benchmarks, results are averaged over 30 instances.

Experiments with binary random DCOPs show that our algorithm BnB-ADOPT$^+$ obtains important communication savings with respect to original BnB-ADOPT. The number of messages is reduced between 3 and 6 times when connectivity and domain size increases, also showing a consistent reduction, by a factor between 4 and 5, when increasing the number of variables. For meeting scheduling instances, messages are reduced by a factor between 3 and 9, and for sensor networks, by a factor between 5 and 8. The standard deviation of messages also decreases in all problems considered.

Regarding NCCCs, the mean is also moderately reduced in all instances (around 10%). In the binary random benchmark, their standard deviation is also slightly reduced. In meeting scheduling and sensor networks, their standard deviation increases. However, when looking at every problem separately, the number of NCCCs of BnB-ADOPT$^+$ is always smaller in every instance. The number of cycles remain practically unchanged. These results clearly indicate that, in the binary case, removing redundant messages is very beneficial to enhance communication, achieving also some moderated gains in computation.

In addition, we took a particular random instance $\langle n = 10, d = 10, p_1 = 0.5 \rangle$, and solved it repeatedly using original BnB-ADOPT and BnB-ADOPT$^+$, varying the order in which agents are activated in the simulator (using the same DFS pseudo-tree in all executions). Results were quite similar across executions. Regarding saved messages, BnB-ADOPT always required between 4.3 and 4.4 times more messages than BnB-ADOPT$^+$ (considering individual executions). These results show that the activation order of agents in the simulator has no impact in the message reduction achieved by BnB-ADOPT$^+$.

Results of the second experiment appear in Table 3, which contains results of ternary random instances with $\langle n = 8, d = 5, p_1 = 0.4...0.8 \rangle$ averaged over 50 instances. First row contains results of original BnB-ADOPT (including the modification of Section 4.1). Second row contains results for our n-ary BnB-ADOPT proposal (Section 4), where thresholds are propagated to all children.

(a) $< n = 10, d = 10, p_1 >$

p_1	#Messages	#NCCC	#Cycles
	1068 (274)	904 (23)	**62 (15)**
0.2	**416 (74)**	**881 (23)**	**62 (15)**
	39,158 (36,578)	68,882 (62,180)	**1,751 (1,625)**
0.3	**11,774 (10,105)**	**62,031 (53,085)**	1,753 (1,629)
	270,379 (432,782)	504,373 (796,625)	**10,313 (16,478)**
0.4	**69,277 (92,291)**	**475,534 (776,820)**	10,317 (16,483)
	2,273,768 (2,149,369)	4,311,524 (3,923,577)	**73,715 (69,676)**
0.5	**493,137 (422,360)**	**4,112,299 (3,760,583)**	73,792 (69,808)
	11,439,563 (10,231,971)	23,759,356 (22,468,476)	**331,947 (299,259)**
0.6	**2,205,848 (1,802,655)**	**22,783,209 (21,040,893)**	332,841 (300,784)
	60,221,283 (34,121,853)	134,868,051 (90,469,274)	**1,526,394 (862,540)**
0.7	**8,930,713 (5,092,602)**	**129,143,706 (89,328,458)**	1,527,960 (865,974)
	161,327,710 (94,398,879)	360,857,244 (212,464,295)	**3,752,164 (2,210,488)**
0.8	**22,972,676 (13,464,530)**	**353,180,585 (209,726,371)**	3,755,118 (2,213,631)

(b) $< n = 10, d, p_1 = 0.5 >$

d	#Messages	#NCCC	#Cycles
	618,005 (573,704)	701,352 (642,821)	**20,305 (18,869)**
6	**119,841 (104,980)**	**657,276 (593,830)**	20,342 (18,924)
	1,362,586 (951,900)	2,090,231 (1,470,631)	**44,507 (31,209)**
8	**288,422 (201,488)**	**1,986,430 (1,398,420)**	44,562 (31,238)
	2,711,719 (2,929,759)	5,092,387 (5,376,943)	**88,224 (96,033)**
10	**597,325 (633,879)**	**4,842,265 (5,133,731)**	88,329 (96,195)
	4,871,563 (9,725,100)	10,969,641 (20,549,608)	**157,856 (314,908)**
12	**1,015,541 (1,706,302)**	**10,342,414 (18,679,208)**	157,994 (315,137)

(c) $< n, d = 10, p_1 = 0.5 >$

n	#Messages	#NCCC	#Cycles
	4,388 (3,272)	13,077 (13,214)	**350 (259)**
6	**1,514 (1,020)**	**12,221 (12,439)**	**350 (259)**
	72,783 (54,772)	173,038 (126,743)	**3,576 (2,679)**
8	**20,326 (12,971)**	**159,698 (113,801)**	3,581 (2,689)
	2,603,727 (3,358,285)	5,289,823 (6,844,174)	**84,706 (112,469)**
10	**547,079 (656,709)**	**5,005,774 (6,576,047)**	84,816 (112,774)
	111,436,193 (133,362,317)	187,178,211(237,619,542)	**2,633,456 (3,148,339)**
12	**20,169,771 (23,877,564)**	**179,110,208 (228,862,664)**	2,636,675 (3,152,543)

Table 1: Results (mean and standard deviation between parenthesis) of random binary benchmarks when varying network connectivity, domain size and number of variables: BnB-ADOPT (first row), BnB-ADOPT$^+$ (second row).

Third row contains BnB-ADOPT$^+$ results, which enhances this last version by removing redundant messages.

Experiments with ternary random DCOPs show that assuring the propagation of threshold values to all children produces clear performance benefits (Table 3, second row). Agents send some extra messages to children containing threshold values, which are not sent in the original version, but these extra messages contribute to a better pruning. As a global effect, less communication is required in the overall search, and significant reductions are obtained in all metrics (messages, NCCCs and cycles). Maintaining this positive effect, we remove redundant messages using BnB-ADOPT$^+$ (Table 3, third row). Removing redundant messages causes savings up to one order of magnitude in the number of messages exchanged. This result is very positive because execution time is often dominated by communication time. Observe that the number of cycles have very little variation between the second and third row. Also, there are some savings in NCCCs, although they are not very significant. From these results we conclude that, in the n-ary case, our proposal for n-ary BnB-ADOPT yields clear benefits in both communication and computation, and that removing redundant messages substantially reduces communication.

Finally, we present a third experiment comparing BnB-ADOPT with SBB and AFB in Figure 2. These experiments were performed on binary random (top), meeting scheduling (middle) and sensor network (bottom) instances. SBB variables were statically ordered using the width heuristic described by Hirayama and Yokoo (1997), AFB variables were ordered following this same heuristic, and BnB-ADOPT$^+$ variables were partially ordered using a most connected heuristic when constructing the pseudo-tree. We restrict the solving time to one hour; in the case of SBB and AFB

(a) Meeting Scheduling

	#Messages	#NCCC	#Cycles
	178,899 (3,638)	446,670 (2,786)	**8,202 (302)**
A	**18,117 (627)**	**413,507 (13,446)**	8,203 (302)
	65,556 (912)	125,331 (1,963)	**2,663 (43)**
B	**15,373 (426)**	**120,900 (1,969)**	2,665 (43)
	62,707 (741)	80,369 (54)	**2,353 (34)**
C	**11,343 (347)**	**74,518 (407)**	2,355 (35)
	41,282 (862)	60,424 (460)	**1,545 (48)**
D	**13,354 (455)**	**49,878 (1,692)**	1,547 (48)

(b) Sensor Network

	#Messages	#NCCC	#Cycles
	9,369 (99)	7,241 (52)	313 (3)
A	**1,103 (73)**	**450 (232)**	**307 (4)**
	12,917 (116)	11,054 (135)	**414 (4)**
B	**1,569 (77)**	**592 (879)**	409(4)
	6,429 (59)	8,786 (52)	**340 (5)**
C	**1,177 (51)**	**1,495 (2,490)**	**340 (6)**
	15,560 (145)	12,641 (57)	**477 (2)**
D	**2,155 (81)**	**2,137 (3,552)**	**477 (2)**

Table 2: Results (mean and standard deviation between parenthesis) of meeting scheduling and sensor network instances: BnB-ADOPT (first row), BnB-ADOPT$^+$ (second row).

p_1	#Messages	#NCCC	#Cycles
	257,238 (294,137)	1,522,580 (1,863,696)	10,880 (12,295)
	147,379 (139,646)	829,594 (852,754)	**5,793 (5,357)**
0.2	**28,614 (23,822)**	**764,516 (783,253)**	5,797 (5,360)
	648,217 (413,580)	4,029,045 (2,807,389)	24,026 (15,085)
	401,306 (239,271)	2,414,946 (1,641,836)	**13,938 (8,271)**
0.3	**63,778 (33,025)**	**2,237,786 (1,520,761)**	13,943 (8,270)
	1,642,247 (975,131)	12,585,339 (8,483,693)	55,156 (32,553)
	1,210,143 (523,825)	9,194,309 (4,938,582)	**39,373 (17,188)**
0.4	**164,901 (68,876)**	**8,744,465 (4,813,577)**	39,381 (17,197)
	2,321,729 (1,106,146)	19,424,669 (10,248,817)	74,279 (36,150)
	1,771,256 (678,893)	14,775,187 (6,678,667)	**55,101 (21,280)**
0.5	**225,836 (69,630)**	**14,337,952 (6,372,442)**	55,103 (21,280)
	3,666,514 (1,316,782)	35,743,718 (15,729,105)	115,523 (43,001)
	2,973,239 (1,406,400)	29,252,469 (16,146,978)	**92,020 (45,010)**
0.6	**311,524 (93,062)**	**27,614,749 (14,316,979)**	**92,020 (45,010)**
	4,013,891 (897,046)	41,469,157 (11,804,005)	124,408 (29,353)
	3,537,027 (1,119,242)	36,966,889 (13,604,359)	**108,858 (36,028)**
0.7	**348,199 (73,620)**	**35,314,718 (12,331,316)**	**108,858 (36,028)**
	4,892,733 (788,897)	55,151,742 (11,209,964)	150,077 (25,114)
	4,616,032 (1,135,830)	52,221,369 (14,948,424)	**140,472 (35,672)**
0.8	**399,662 (70,572)**	**49,189,230 (13,197,765)**	**140,472 (35,672)**

Table 3: Results (mean and standard deviation between parenthesis) on random ternary DCOPs: original BnB-ADOPT (first row), our proposal for n-ary BnB-ADOPT (second row), and BnB-ADOPT$^+$ (third row).

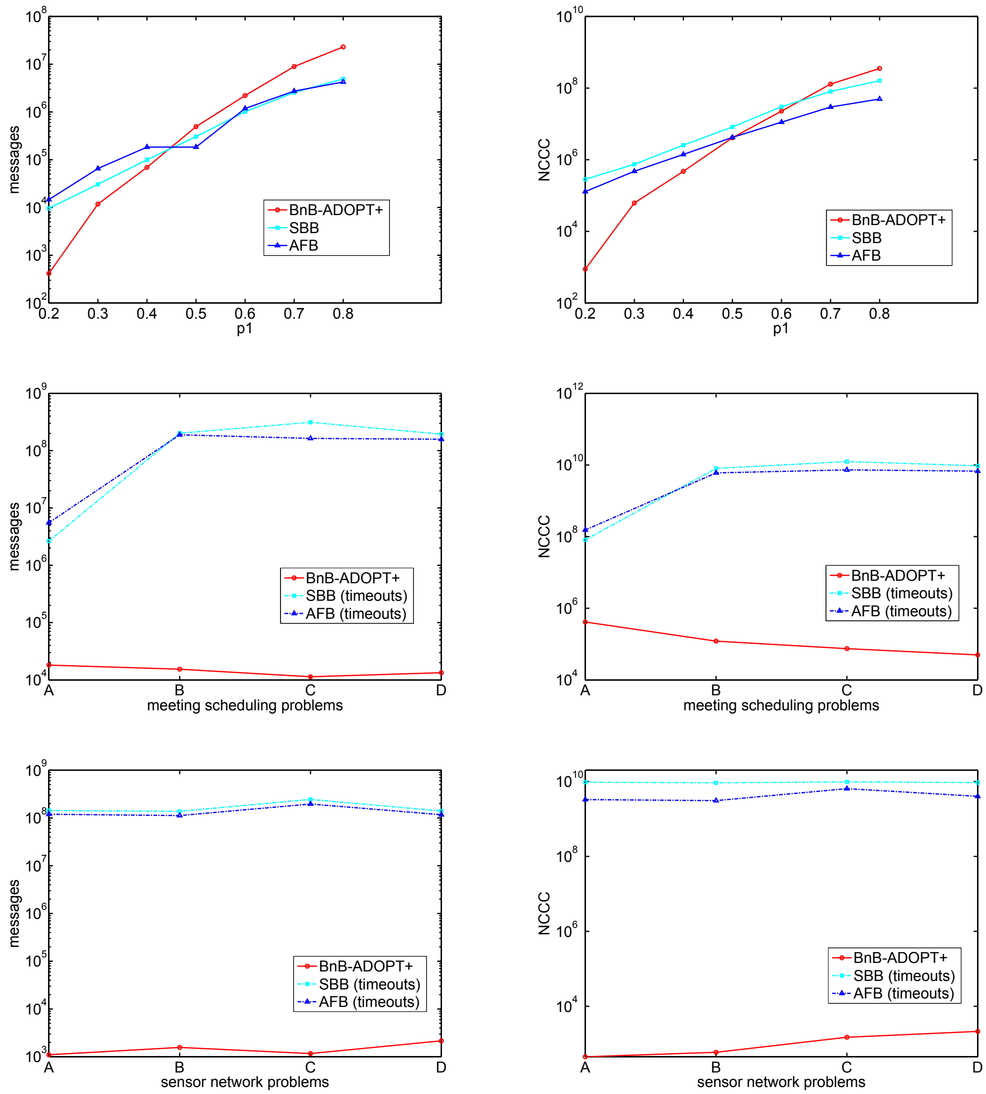

Figure 2: Comparison of algorithms BnB-ADOPT$^+$, SBB, AFB. Top: binary random instances. Middle: meeting scheduling. Bottom: sensor networks.

if a problem instance could not be solved in this amount of time, we present the amount of messages exchanged and NCCCs performed until timeout.

From these results we can observe that for random instances BnB-ADOPT$^+$ is significantly more efficient in low connected problems, however in tightly connected problems it requires more messages and computational effort than SBB and AFB. We explain this behavior given that BnB-ADOPT$^+$ is an asynchronous algorithm, designed to benefit from a pseudo-tree structure, where non-connected agents lying on different branches of the pseudo-tree can explore the search space in parallel. When connectivity increases, the width of the pseudo-tree decreases (in a fully connected problem the pseudo-tree has a single branch where all agents are totally ordered). This makes BnB-ADOPT$^+$ asynchronous potential to decrease. At the same time, a higher number reinitializations (bounds and context) are performed, since agents have more links to ancestors, which reduces pruning effectiveness.

For the meeting scheduling and sensor network instances, we can see that BnB-ADOPT$^+$ is several orders of magnitude more efficient than SBB and AFB. The structured nature of these problems (with a different topology for every case A, B, C or D (Maheswaran et al., 2004) suitable to build balanced pseudo-trees) allows BnB-ADOPT$^+$ to benefit from asynchronous search. In addition, we observed that in these instances the variability of costs is smaller than in random problems: most costs are quite similar, while some others are clearly larger. In this cases, an upper bound close to the optimum cost is reached early in the execution. However to satisfy the pruning condition, lower bounds contributions from almost all cost functions are needed. We observed that SBB and –to a lesser extent– AFB have to go deep in the search tree to obtain such contributions (pruning is usually done in the last agents of the ordering) and finally they are subject to thrashing. On the other hand, BnB-ADOPT$^+$ computes local bounds on every agent since all agents are assigned at every moment of the execution, and specialized upper bounds are propagated to every node of the pseudo-tree. This allows BnB-ADOPT$^+$ to perform pruning and, as a consequence, it reduces the search space faster than AFB and SBB. We have confirmed this fact empirically by testing on instances with small variability costs (even on synthetic instances where all tuples have the same cost).

In summary, we can see that the proposed BnB-ADOPT$^+$ is clearly more efficient than original BnB-ADOPT. In binary instances, BnB-ADOPT$^+$ processes a third (or less) of the total number of messages required by BnB-ADOPT (in some instances, messages are reduced by a factor of 8 or 9), and still reaches an optimal solution in almost the same number of cycles. In ternary instances, savings reach up to one order of magnitude in communication for almost all cases. Regarding the comparison with SBB and AFB, the new BnB-ADOPT$^+$ outperforms both of them (in number of messages and NCCCs) in low connected random instances, while the contrary occurs in highly connected ones. Regarding meeting scheduling and sensor network instances, BnB-ADOPT$^+$ outperforms SBB and AFB by a large margin. These results indicate the value of BnB-ADOPT$^+$ for optimal DCOP solving.

6. Conclusion

We have presented two contributions to increase the performance of BnB-ADOPT, a reference algorithm to optimally solve distributed constrained optimization problems. First, we presented theoretical results to detect redundant messages. Second, we described some difficulties when dealing with n-ary cost functions (the original algorithm was presented in detail for the binary case). Combining these two contributions, we generated a new version of BnB-ADOPT. This new version, called BnB-ADOPT$^+$, obtains substantial savings with respect to the original algorithm, when tested on commonly used benchmarks by the DCOP community.

Acknowledgments

This paper is an extension of a previous publication (Gutierrez & Meseguer, 2010b), with the generalization to n-ary cost functions, deeper explanations, more detailed proofs, and more experiments. Authors sincerely thank reviewers for their useful criticisms.

References

Chechetka, A., & Sycara, K. (2006). No-commitment branch and bound search for distributed constraint optimization. *Proc. AAMAS-06*, 1427–1429.

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

Gutierrez, P., & Meseguer, P. (2010a). Saving messages in adopt-based algorithms. *Proc. 12th DCR workshop in AAMAS-10*, 53–64.

Gutierrez, P., & Meseguer, P. (2010b). Saving messages in BnB-ADOPT. *Proc. AAAI-10*, 1259–1260.

Hirayama, K., & Yokoo, M. (1997). Distributed partial constraint satisfaction problem. *Proc. CP-97*, 222–236.

Jain, M., Taylor, M., Tambe, M., & Yokoo, M. (2009). DCOPs meet the realworld: Exploring unknown reward matrices with applications to mobile sensor networks. *Proc. IJCAI-09*, 181–186.

Junges, R., & Bazzan, A. L. C. (2008). Evaluating the performance of DCOP algorithms in a real world dynamic problem. *Proc. AAMAS-08*, 599–606.

Maheswaran, R., Tambe, M., Bowring, E., Pearce, J., & Varakantham, P. (2004). Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. *Proc. AAMAS-04*, 310–317.

Meisels, A., Kaplansky, E., Razgon, I., & Zivan, R. (2002). Comparing performance of distributed constraints processing algorithms. *Proc. 3rd DCR workshop in AAMAS-02*, 86–93.

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

Petcu, A. (2007). *A class of algorithms for Distributed Constraint Optimization*. Ph.D. thesis.

Petcu, A., & Faltings, B. (2005). A scalable method for multiagent constraint optimization. *Proc. IJCAI-05*, 266–271.

Ueda, S., Iwasaki, A., & Yokoo, M. (2010). Coalition structure generation based on distributed constraint optimization. *Proc. AAAI-10*, 197–203.

Yeoh, W., Felner, A., & Koenig, S. (2010). BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research*, *38*, 85–133.

Yin, Z. (2008). USC DCOP repository..

Yokoo, M., Durfee, E., Ishida, T., & Kuwabara, K. (1998). The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. Know. and Data Engin.*, *10*, 673–685.