BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm*

William Yeoh[†] Ariel Felner[‡] Sven Koenig[†]

[†]Computer Science
University of Southern California
Los Angeles, CA 90089-0781, USA
{wyeoh,skoenig}@usc.edu

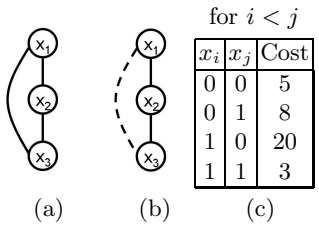
[‡]Information Systems Engineering
Ben-Gurion University of the Negev
Beer-Sheva, 85104, Israel
felner@bgu.ac.il

Abstract. Distributed constraint optimization problems (DCOPs) are a popular way of formulating and solving agent-coordination problems. It is often desirable to solve DCOPs optimally with memory-bounded and asynchronous algorithms. We thus introduce Branch-and-Bound ADOPT (BnB-ADOPT), a memory-bounded asynchronous DCOP algorithm that uses the message passing and communication framework of ADOPT, a well known memory-bounded asynchronous DCOP algorithm, but changes the search strategy of ADOPT from best-first search to depth-first branch-and-bound search. Our experimental results show that BnB-ADOPT is up to one order of magnitude faster than ADOPT on a variety of large DCOPs and faster than NCBB, a memory-bounded synchronous DCOP algorithm, on most of these DCOPs.

1 Introduction

Distributed constraint optimization problems (DCOPs) are a popular way of formulating and solving agent-coordination problems, including scheduling meetings [6], coordinating unmanned aerial vehicles [11] and allocating targets to sensors in sensor networks [1]. A DCOP consists of several agents that need to take on values so that the sum of the resulting constraint costs is minimal. Each agent often has only a fixed amount of memory available. This calls for memory-bounded DCOP algorithms. The agents have to communicate with each other, but communication is often restricted to nearby agents. This calls for DCOP algorithms that restrict communication to agents that share constraints. Agents that act independently without having to wait for other agents are fast. This calls for asynchronous DCOP algorithms. ADOPT is a popular DCOP algorithm that satisfies these three constraints and uses best-first search to solve DCOPs optimally [9]. In this paper, we study whether there are other DCOP algorithms that satisfy the three constraints and solve DCOPs optimally but are faster than ADOPT.

* We thank the reviewers for their comments. This research has been partly supported by an NSF award to Sven Koenig under contract IIS-0350584. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied of the sponsoring organizations, agencies, companies or the U.S. government.



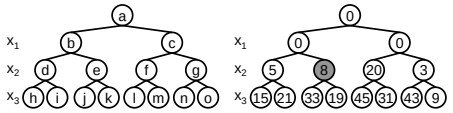
(a) (b)

for $i < j$

x_i	x_j	Cost
0	0	5
0	1	8
1	0	20
1	1	3

(c)

Fig. 1. Example DCOP



(a) (b)

Fig. 2. Search Tree

DCOPs are combinatorial search problems with depth-bounded search trees. It is known that such search problems can often be solved faster with memory-bounded depth-first branch-and-bound search than with memory-bounded best-first search since memory-bounded best-first search needs to repeatedly reconstruct partial solutions that it purged from memory [13]. Consequently, depth-first branch-and-bound search has been extended to distributed constraint satisfaction problems [12, 2] and even to DCOPs. However, the existing depth-first branch-and-bound DCOP algorithms are either synchronous (NCBB) [3] or broadcast messages and thus do not restrict communication to agents that share constraints (AFB) [4]. We therefore introduce Branch-and-Bound ADOPT (BnB-ADOPT), a memory-bounded asynchronous DCOP algorithm that uses the message passing and communication framework of ADOPT to restrict communication to agents that share constraints. Our experimental results show that BnB-ADOPT is up to one order of magnitude faster than ADOPT on a variety of large DCOPs, namely for coloring graphs, scheduling meetings and allocating targets to sensors in sensor networks. It is also faster than NCBB on most of these DCOPs.

2 DCOPs

A DCOP consists of a finite set of agents with their finite domains and a finite set of constraints. Each agent takes on a value from its domain. Each constraint involves two agents and specifies its non-negative constraint cost as a function of the values of these two agents. A solution is an agent-value assignment for all agents, while a partial solution is an agent-value assignment for a subset of agents. The cost of a solution is the sum of the constraint costs of all constraints. Solving the DCOP optimally means to determine the cost of a cost-minimal solution. Each agent needs to decide which value to take on based on its knowledge of the constraints that it is involved in. We make the following assumptions. An agent can neither observe the constraint costs of constraints that it is not involved in nor the values that the other agents take on. However, it can exchange messages with the agents that it is involved in constraints with. Messages from one agent to another agent can be delayed by a finite amount of time but are never lost. The receiving agent receives all messages from the same sending agent in the same order that they were sent in.

3 BnB-ADOPT

A constraint graph is a graph whose vertices are the agents of the DCOP and whose edges are the constraints. A constraint tree is a spanning tree of the

constraint graph with the property that edges in the constraint graph connect a vertex with one of its ancestors or descendants in the constraint tree. An edge of the constraint graph that is not part of the constraint tree is called a back-edge. Figure 1(a) shows the constraint graph of an example DCOP with three agents that can each take on the values zero or one. There is a constraint between each pair of agents. Figure 1(c) shows the constraint costs, which are the same for all three constraints. Figure 1(b) shows one possible constraint tree, where the dotted line is a back-edge. Figure 2 shows a search tree, where levels 1, 2 and 3 of the tree correspond to agents x_1, x_2 and x_3, respectively. Left branches correspond to the agents taking on the value zero, and right branches to the agents taking on the value one. Nodes are labeled with identifiers (left) and the sum of the constraint costs of all constraints that involve only agents with known values (right). These sums correspond to the f-values of an A* search [5], where we assume for simplicity that the heuristics are always zero. ADOPT explores the search tree in best-first order. Thus, it explores nodes for the first time in the order of their f-values, namely in the order a, b, c, g, d, e, and o. In order to be memory-bounded, it maintains only a branch from the root node to the currently explored node and thus needs to repeatedly reconstruct nodes that it purged from memory. For example, it has the branch from a to e in memory when it explores node e but then needs to have the branch from a to o in memory when it explores node o. Thus, it needs to reconstruct the part of this branch from a to g. BnB-ADOPT, on the other hand, explores the search tree in depth-first branch-and-bound order. It explores the children of a node in the order of their f-values and prunes those nodes whose f-values are no smaller than the smallest f-value (called the threshold) of any leaf node that it has already observed. Thus, it explores nodes in the order a, b, d, h, (i), e, (k), (j), c, g, and o, where it prunes the nodes in parentheses. It is memory-bounded without having to repeatedly reconstruct nodes that it purged from memory but explores some nodes that a best-first search does not explore, such as node h.

Depth-first branch-and-bound search makes for a simple centralized DCOP algorithm. There are two obstacles to transforming it into an asynchronous DCOP algorithm that restricts communication to agents that share constraints. First, depth-first branch-and-bound search orders the agents totally (for the example DCOP: x_1, x_2 and x_3) and thus operates on constraint trees that are chains, that is, makes different agents take on values one after the other. BnB-ADOPT can take advantage of concurrency to solve DCOPs faster by operating on subtrees of the constraint trees in parallel, that is, allows different agents to take on values in parallel. Second, agents cannot observe the values that the other agents take on and their knowledge from previous messages might thus be outdated, meaning that the agents need to synchronize. BnB-ADOPT can take advantage of concurrency by not synchronizing the agents tightly. We achieve these two desired properties by using the message passing and communication framework of ADOPT. We give a stand-alone description of BnB-ADOPT in the following that requires no knowledge of ADOPT, with the intuition to create a self-contained and hopefully easy-to-read overview.

3.1 Notation

We use the following notation from ADOPT to describe BnB-ADOPT: V is the finite set of agents of the DCOP. $Dom(a)$ is the domain of agent $a \in V$. $ValInit(a) \in Dom(a)$ is the value that we use as initial value of agent $a \in V$. $C(a) \subseteq V$ is the set of children of agent a in the constraint tree and $CD(a) \subseteq V$ is the set of its descendants (including its children) that it is involved in constraints with. $pa(a) \subseteq V$ is the parent of agent $a \in V$, $A(a) \subseteq V$ is the set of its ancestors (including its parent), $SCA(a) \subseteq A(a)$ is the set of its ancestors (including its parent) that it or one of its descendants is involved in constraints with and $CA(a) \subseteq SCA(a)$ is the set of its ancestors (including its parent) that it is involved in constraints with.

3.2 Key Variables

Consider any agent $a \in V$. Assume that the values of all ancestors $a' \in A(a)$ are given by the set X of agent-value assignments, called the context of agent a. $\delta^a_X(d)$ is the sum of the constraint costs of all constraints that involve both agent a and one of its ancestors, under the assumption that agent a takes on value d and its ancestors take on the values in X. $\gamma^a_X(d)$ is the sum of the constraint costs of all constraints that involve agent a and/or one of its descendants,[1] under the assumption that agent a takes on value d and its ancestors take on the values in X, minimized over the possible values of its descendants.[2] We define $\gamma^a_X := \min_{d' \in Dom(a)}\{\gamma^a_X(d')\}$ for all agents $a \in V$ and all contexts X of agent a. Then, $\gamma^a_X(d) = \delta^a_X(d) + \sum_{a' \in C(a)} \gamma^{a'}_{X \cup (a,d)}$ for all agents $a \in V$, all values $d \in Dom(a)$ and all contexts X of agent a. Solving the DCOP optimally means to determine $\gamma^r_{\{\}}$ for the root agent r in the constraint tree since $\gamma^r_{\{\}}$ is the sum of the constraint costs of all constraints, minimized over the possible values of all agents.

Imagine that every agent $a \in V$ stores and updates several lower and upper bounds, namely $lb^{a,a'}_X(d)$, $LB^a_X(d)$, LB^a_X, $ub^{a,a'}_X(d)$, $UB^a_X(d)$ and UB^a_X for all values $d \in Dom(a)$, all contexts X of agent a, and all children $a' \in C(a)$, maintaining the **"Bound Property"**:

$$\begin{aligned} lb^{a,a'}_X(d) \leq \gamma^{a'}_{X \cup (a,d)} &\leq ub^{a,a'}_X(d) \\ LB^a_X(d) \leq \gamma^a_X(d) &\leq UB^a_X(d) \\ LB^a_X \leq \gamma^a_X &\leq UB^a_X. \end{aligned}$$

[1] Thus, the constraints involve either both agent a and one of its ancestors, both agent a and one of its descendants, both a descendant and an ancestor of agent a, or two descendants of agent a.

[2] In other words, $\gamma^a_X(d)$ is the smallest increase in the sum of the constraint costs of all constraints that involve only agents with known values when one augments the partial agent-value assignment $X \cup (a, d)$ with agent-value assignments for all descendants of agent a.

It initializes $lb_X^{a,a'}(d) := h_X^{a,a'}(d)$ for admissible heuristics $0 \leq h_X^{a,a'}(d) \leq \gamma_{X\cup(a,d)}^{a'}$ and $ub_X^{a,a'}(d) := \infty$ for all values $d \in Dom(a)$, all contexts X of agent a, and all children $a' \in C(a)$. It then uses repeatedly the **"Update Equations"**:

$$lb_X^{a,a'}(d) := \max\{lb_X^{a,a'}(d), LB_{X\cup(a,d)}^{a'}\}$$
$$LB_X^a(d) := \delta_X^a(d) + \sum_{a'\in C(a)} lb_X^{a,a'}(d)$$
$$LB_X^a := \min_{d'\in Dom(a)}\{LB_X^a(d')\}$$
$$ub_X^{a,a'}(d) := \min\{ub_X^{a,a'}(d), UB_{X\cup(a,d)}^{a'}\}$$
$$UB_X^a(d) := \delta_X^a(d) + \sum_{a'\in C(a)} ub_X^{a,a'}(d)$$
$$UB_X^a := \min_{d'\in Dom(a)}\{UB_X^a(d')\}$$

for all values $d \in Dom(a)$, all contexts X of agent a and all children $a' \in C(a)$, which improves the bounds monotonically (that is, decreases the upper bounds and increases the lower bounds) while maintaining the Bound Property.[3] After a finite amount of time, $LB_X^a = UB_X^a$ for all agents $a \in V$, all values $d \in Dom(a)$ and all contexts X of agent a. Then, $LB_{\{\}}^r = \gamma_{\{\}}^r = UB_{\{\}}^r$ and the DCOP is solved optimally.

3.3 Simplified Version of BnB-ADOPT

In actuality, every agent $a \in V$ stores $lb_X^{a,a'}(d)$, $LB_X^a(d)$, LB_X^a, $ub_X^{a,a'}(d)$, $UB_X^a(d)$ and UB_X^a for all values $d \in Dom(a)$ and all children $a' \in C(a)$ but only *one* context X of agent a at a time because it would not be memory-bounded otherwise. Thus, it can work on only one context at a time. This context is stored in the variable X, which makes it unnecessary to index the other variables with X (although we continue to do so). BnB-ADOPT uses depth-first search as search strategy, which ensures that every agent needs to work on and thus store only one context at a time. We now give a simplistic description of how an agent operates. Consider any agent $a \in V$ with context X and value $d \in Dom(a)$. The agent sends so-called VALUE messages to all children $a' \in C(a)$ with the context $X\cup(a,d)$. The children return so-called COST messages with $LB_{X\cup(a,d)}^{a'}$ and $UB_{X\cup(a,d)}^{a'}$. The agent then uses the Update Equations to improve $lb_X^{a,a'}(d)$, $LB_X^a(d)$, LB_X^a, $ub_X^{a,a'}(d)$, $UB_X^a(d)$, and UB_X^a and sends LB_X^a and UB_X^a to its parent in a COST message. If $LB_X^a(d) < UB_X^a(d)$, then the agent repeats the process. After a finite amount of time, $LB_X^a(d) = UB_X^a(d)$, which means that $LB_X^a(d)$ and $UB_X^a(d)$ cannot be improved further. The agent then takes on the new value $d := \arg\min_{d'\in Dom(a)}\{LB_X^a(d')\}$ and repeats the process until either its context X changes (because the agent receives a VALUE message

[3] Leaf agents in the constraint tree use the same Update Equations. Since they do not have children, the sums over their children evaluate to zero. For example, $LB_X^a(d) = UB_X^a(d) = \delta_X^a(d)$ for all leaf agents $a \in V$, all values $d \in Dom(a)$ and all contexts X of agent a.

from its parent with a different context) or $LB^a_X(d) = UB^a_X(d)$ for all values $d \in Dom(a)$ and thus $LB^a_X = UB^a_X$ after a finite amount of time. Every agent $a \in V$ takes on every value $d \in Dom(a)$ at most once until its context X changes or $LB^a_X = UB^a_X$. BnB-ADOPT thus performs depth-first search.

3.4 Performing Branch-and-Bound

The description so far of how an agent operates is simplistic. We now describe how an agent uses branch-and-bound search to reduce the runtime. Every agent $a \in V$ with value $d \in Dom(a)$ and context X maintains a threshold TH^a_X, initialized to infinity. (The threshold of the root agent r in the constraint tree is always infinity.) The threshold is used for pruning during the depth-first search, resulting in the agent not taking on some values. Agent a needs to improve LB^a_X and UB^a_X monotonically so that they obey the Bound Property and, after a finite amount of time, $LB^a_X = UB^a_X$ or $LB^a_X \geq TH^a_X$ (**"Agent Property"**). Thus, after a finite amount of time, $LB^r_{\{\}} = \gamma^r_{\{\}} = UB^r_{\{\}}$. Then, the DCOP is solved optimally and all agents can terminate, which is achieved by sending TERMINATE messages from parents to children down the constraint tree. It is not difficult for the agents to cache information that allows them to determine a cost-minimal solution.

Every agent $a \in V$ with value $d \in Dom(a)$ and context X operates as described before, with two differences: First, it uses $\min\{TH^a_X, UB^a_X\}$ instead of the larger $UB^a_X(d)$ in the condition that determines whether it should take on a new value d, resulting in the agent not taking on some values. Consequently, if $LB^a_X(d) \geq \min\{TH^a_X, UB^a_X\}$, then it takes on the new value $d := \arg\min_{d' \in Dom(a)}\{LB^a_X(d')\}$. Second, when it sends a VALUE message to its child $a' \in C(a)$, it now includes not only the desired context $X \cup (a,d)$ but also the desired threshold $\min\{TH^a_X, UB^a_X\} - \delta^a_X(d) - \sum_{a'' \in C(a), a'' \neq a'} lb^{a,a''}_X(d)$ for the child. This desired threshold is chosen such that $LB^a_X(d)$ for the agent reaches $\min\{TH^a_X, UB^a_X\}$ and the agent thus takes on a new value when $LB^{a'}_{X \cup (a,d)}$ for the child reaches the desired threshold.

We now explain why the Agent Property holds, by induction on the level of the agent in the constraint tree. It is easy to prove that every leaf agent $a \in V$ has the Agent Property. Now consider any non-leaf agent $a \in V$ with value $d \in Dom(a)$, context X and threshold TH^a_X. Assume that its children $a' \in C(a)$ have the Agent Property. The agent sends VALUE messages to all children $a' \in C(a)$ with the context $X \cup (a,d)$ and the threshold $\min\{TH^a_X, UB^a_X\} - \delta^a_X(d) - \sum_{a'' \in C(a), a'' \neq a'} lb^{a,a''}_X(d)$. The children return COST messages with $LB^{a'}_{X \cup (a,d)}$ and $UB^{a'}_{X \cup (a,d)}$. The agent then uses the Update Equations to improve $lb^{a,a'}_X(d)$, $LB^a_X(d)$, LB^a_X, $ub^{a,a'}_X(d)$, $UB^a_X(d)$ and UB^a_X and sends LB^a_X and UB^a_X to its parent in a COST message. $LB^{a'}_{X \cup (a,d)}$ and $UB^{a'}_{X \cup (a,d)}$ improve monotonically and obey the Bound Property since the children $a' \in C(a)$ have the Agent Property. Consequently, $lb^{a,a'}_X(d)$, $LB^a_X(d)$, LB^a_X, $ub^{a,a'}_X(d)$, $UB^a_X(d)$ and UB^a_X improve monotonically and obey the Bound Property since the Update Equations preserve monotonicity and

the Bound Property. If $LB^a_X(d) < \min\{TH^a_X, UB^a_X\}$, then the agent repeats the process. After a finite amount of time, $LB^{a'}_{X\cup(a,d)} = UB^{a'}_{X\cup(a,d)}$ or $LB^{a'}_{X\cup(a,d)} \geq \min\{TH^a_X, UB^a_X\} - \delta^a_X(d) - \sum_{a''\in C(a), a''\neq a'} lb^{a,a''}_X(d)$ for all children $a' \in C(a)$ since the children have the Agent Property. If $LB^{a'}_{X\cup(a,d)} = UB^{a'}_{X\cup(a,d)}$ for all children $a' \in C(a)$, then $LB^a_X(d) = \delta^a_X(d) + \sum_{a'\in C(a)} lb^{a,a'}_X(d) \geq \delta^a_X(d) + \sum_{a'\in C(a)} LB^{a'}_{X\cup(a,d)} = \delta^a_X(d) + \sum_{a'\in C(a)} UB^{a'}_{X\cup(a,d)} \geq \delta^a_X(d) + \sum_{a'\in C(a)} ub^{a,a'}_X(d) = UB^a_X(d) \geq UB^a_X \geq \min\{TH^a_X, UB^a_X\}$. Otherwise, $LB^{a'}_{X\cup(a,d)} \geq \min\{TH^a_X, UB^a_X\} - \delta^a_X(d) - \sum_{a''\in C(a), a''\neq a'} lb^{a,a''}_X(d)$ for at least one child $a' \in C(a)$ and then $LB^a_X(d) = \delta^a_X(d) + \sum_{a''\in C(a)} lb^{a,a''}_X(d) = \delta^a_X(d) + \sum_{a''\in C(a), a'\neq a''} lb^{a,a''}_X(d) + lb^{a,a'}_X(d) \geq \delta^a_X(d) + \sum_{a''\in C(a), a'\neq a''} lb^{a,a''}_X(d) + LB^{a'}_{X\cup(a,d)} \geq \delta^a_X(d) + \sum_{a''\in C(a), a'\neq a''} lb^{a,a''}_X(d) + \min\{TH^a_X, UB^a_X\} - \delta^a_X(d) - \sum_{a''\in C(a), a''\neq a'} lb^{a,a''}_X(d) = \min\{TH^a_X, UB^a_X\}$. Thus, after a finite amount of time, $LB^a_X(d) \geq \min\{TH^a_X, UB^a_X\}$. The agent then takes on the new value $d := \arg\min_{d'\in Dom(a)}\{LB^a_X(d')\}$ and repeats the process until either its context X changes (because the agent receives a VALUE message from its parent with a different context) or $LB^a_X(d) \geq \min\{TH^a_X, UB^a_X\}$ for all values $d \in Dom(a)$ and thus $LB^a_X \geq \min\{TH^a_X, UB^a_X\}$ after a finite amount of time. If $TH^a_X < UB^a_X$, then $LB^a_X \geq TH^a_X$. If $TH^a_X \geq UB^a_X$, then $LB^a_X \geq UB^a_X$ and thus $LB^a_X = UB^a_X$. Thus, any non-leaf agent has the Agent Property as well. Every agent $a \in V$ takes on every value $d \in Dom(a)$ at most once until its context X changes or $LB^a_X \geq \min\{TH^a_X, UB^a_X\}$ since UB^a_X decreases monotonically and the condition $LB^a_X(d) \geq \min\{TH^a_X, UB^a_X\}$ thus remains satisfied. BnB-ADOPT thus performs depth-first search. If the parent of the agent sends it a VALUE message with a context different from its context, then the agent changes its context to the context X in the VALUE message, changes its threshold to the threshold in the VALUE message, initializes $lb^{a,a'}_X(d) := h^{a,a'}_X(d)$ and $ub^{a,a'}_X(d) := \infty$ and uses the Update Equations to initialize $LB^a_X(d)$, $UB^a_X(d)$, LB^a_X and UB^a_X for all values $d \in Dom(a)$ and all children $a' \in C(a)$, takes on the new value $d := \arg\min_{d'\in Dom(a)}\{LB^a_X(d')\}$ and repeats the process.

3.5 Increasing Concurrency

The description so far of how an agent operates is still simplistic since BnB-ADOPT does not synchronize agents tightly. BnB-ADOPT uses the following techniques to increase concurrency: First, the agents use reduced contexts, that are subsets of the contexts described so far. Consider an agent $a \in V$ with context X_1 that contains the agent-value assignments for all ancestors $a' \in A(a)$. Then, $\gamma^a_{X_1} = \gamma^a_{X_2}$ where $X_2 \subseteq X_1$ is the subset of agent-value assignments for all ancestors $a' \in SCA(a)$ that it or one of its descendants is involved in constraints with. Thus, in the implemented version of BnB-ADOPT, the agents use these reduced contexts instead. Second, the agents propagate contexts differently than described so far to increase concurrency. So far, agents sent VALUE messages to all children with the desired context of the receiving agent. Every agent that

received a VALUE message changed its context to the one in the VALUE message. By contrast, in the implemented version of BnB-ADOPT, an agent sends VALUE messages to all descendants that it is involved in constraints with (although the thresholds are used only by its children). These VALUE messages contain the value of the sending agent rather than the desired context of the receiving agent. Every receiving agent changes the value of the sending agent in its context to the one in the VALUE message. Thus, the VALUE messages allow the receiving agent to update the values of its ancestors that it is involved in constraints with. In the implemented version of BnB-ADOPT, agents still send COST messages to their parents but the COST messages now include the context of the sending agent. Every receiving agent changes the values of its ancestors that it is not involved in constraints with to the ones in the context of the COST message. Thus, the COST messages allow the receiving agent to update the values of its ancestors that its descendants are involved in constraints with.[4] Thus, the VALUE and COST messages together allow an agent to update the values of all ancestors that it or one of its descendants is involved in constraints with, which make up exactly its context. Third, the agent can no longer assume that its children send bounds in their COST messages for the context desired by it. Thus, in the implemented version of BnB-ADOPT, the agent needs to check whether the context in a COST message is a subset of its context after it has last updated its context as described above. If not, it ignores the COST message since it is irrelevant for improving the bounds for its context. Fourth, if the context of an agent changes, the desired context of some of its children can now remain unchanged since the context of a child of an agent can be a *strict* subset of the context of the agent augmented by the agent-value assignment for the agent itself. Thus, if the context of an agent $a \in V$ changes in the implemented version of BnB-ADOPT, it needs to check for which children $a' \in C(a)$ it needs to initialize $lb_X^{a,a'}(d)$ and $ub_X^{a,a'}(d)$ for all values $d \in Dom(a)$. If the context of the agent changes due to a COST message from its child, it might be able to use the bounds in the COST message to initialize $lb_X^{a,a'}(d')$ and $ub_X^{a,a'}(d')$ for the value $d' \in Dom(a)$ that the agent takes on in the context in the COST message. Then, the agent uses the Update Equations to initialize $LB_X^a(d)$, $UB_X^a(d)$, LB_X^a and UB_X^a for all values $d \in Dom(a)$ and takes on the new value $d := \arg\min_{d' \in Dom(a)} \{LB_X^a(d')\}$.

3.6 Pseudocode

Figure 3 show the BnB-ADOPT pseudocode of every agent. BnB-ADOPT uses the message passing and communication framework of ADOPT. It uses the same VALUE, COST and TERMINATION messages as ADOPT; the same strategy to update the context of an agent based on VALUE messages from its ancestors and COST messages from its children; the same semantics for the lower and upper bounds $lb_X^{a,a'}(d)$, $LB_X^a(d)$, LB_X^a, $ub_X^{a,a'}(d)$, $UB_X^a(d)$ and UB_X^a; and

[4] The agent uses the VALUE messages rather than the COST messages to update the values of its ancestors that it is involved in constraints with since it receives them directly from these ancestors, which propagates information fast.

```
procedure Start ()
{01}  X := ⋃_{a'∈SCA(a)} (a', ValInit (a'));
{02}  TH^a := ∞;
{03}  forall a' ∈ C(a), d ∈ Dom(a)
{04}    InitChild (a', d);
{05}  InitSelf ();
{06}  Backtrack ();
{07}  loop forever
{08}    if (message queue is not empty)
{09}      while (message queue is not empty)
{10}        pop msg off message queue;
{11}        When Received (msg);
{12}      Backtrack ();

procedure InitChild (a', d)
{13}  lb^{a,a'}(d) := h^{a,a'}(d);
{14}  ub^{a,a'}(d) := ∞;

procedure InitSelf ()
{15}  d^a := arg min_{d'∈Dom(a)} {δ^a(d') + Σ_{a'∈C(a)} lb^{a,a'}(d')};
{16}  TH^a := ∞;

procedure Backtrack ()
{17}  forall d ∈ Dom(a)
{18}    LB^a(d) := δ^a(d) + Σ_{a'∈C(a)} lb^{a,a'}(d);
{19}    UB^a(d) := δ^a(d) + Σ_{a'∈C(a)} ub^{a,a'}(d);
{20}  LB^a := min_{d∈Dom(a)} {LB^a(d)};
{21}  UB^a := min_{d∈Dom(a)} {UB^a(d)};
{22}  if (TH^a ≤ LB^a)
{23}    TH^a := ∞;
{24}  if (LB^a(d^a) ≥ min{TH^a, UB^a})
{25}    d^a := arg min_{d'∈Dom(a)} {LB^a(d')};
{26}  if ((a is root and UB^a = LB^a) or termination message received)
{27}    Send (TERMINATE) to each a' ∈ C(a);
{28}    terminate execution;
{29}  Send (VALUE, a, d^a, min{TH^a, UB^a} − δ^a(d^a) − Σ_{a''∈C(a),a''≠a'} lb^{a,a''}(d^a)) to each a' ∈ C(a);
{30}  Send (VALUE, a, d^a, ∞) to each a' ∈ CD(a) \ C(a);
{31}  Send (COST, a, X, LB^a, UB^a) to pa(a) if a is not root;

procedure When Received (VALUE, p, d^p, TH^p)
{32}  if (¬Compatible ((p, d^p), X))
{33}    PriorityMerge ((p, d^p), X);
{34}    forall a' ∈ C(a), d ∈ Dom(a)
{35}      if (p ∈ SCA(a'))
{36}        InitChild (a', d);
{37}    InitSelf ();
{38}  if (p = pa(a))
{39}    TH^a := TH^p;

procedure When Received (COST, c, X^c, LB^c, UB^c)
{40}  X' := X;
{41}  notConnectedContext := ⋃_{(p,d^p)∈X^c | p∉CA(a), p≠a} (p, d^p);
{42}  if (¬Compatible (notConnectedContext, X'))
{43}    PriorityMerge (notConnectedContext, X);
{44}    forall a' ∈ C(a), d ∈ Dom(a)
{45}      if (¬Compatible (⋃_{(a'',d^{a''})∈X' | a''∈SCA(a')} (a'', d^{a''}), X))
{46}        InitChild (a', d);
{47}  if (Compatible (X^c, X))
{48}    lb^{a,c}(d) := max{lb^{a,c}(d), LB^c} for d ∈ Dom(a) | (a, d) ∈ X^c;
{49}    ub^{a,c}(d) := min{ub^{a,c}(d), UB^c} for d ∈ Dom(a) | (a, d) ∈ X^c;
{50}  if (¬Compatible (notConnectedContext, X'))
{51}    InitSelf ();

procedure When Received (TERMINATE)
{52}  record termination message received;
```

Fig. 3. Pseudocode of BnB-ADOPT

the same Update Equations to update the lower and upper bounds. However, BnB-ADOPT uses a different semantics for the threshold than ADOPT since it uses the threshold for pruning while ADOPT uses it to reconstruct previously explored partial solutions. Initially, BnB-ADOPT calls Start() for every agent $a \in V$. The code is identical for every agent except that the variable a is a "self" variable that points to the agent itself. The pseudocode uses the predicate Compatible$(X, X') = \neg\exists_{a \in V, d, d' \in Dom(a)}(d \neq d' \wedge (a, d) \in X \wedge (a, d') \in X')$ that determines whether two contexts are compatible by checking that they do not make the same agent take on different values. The pseudocode uses the procedure PriorityMerge(X, X') that executes $X' := X \cup \{(a, d) \in X' \mid \neg\exists_{d' \in Dom(a)}(a, d') \in X\}$ that sets the second context to the union of two contexts but removes all agent-value assignments from the second context that are not compatible with the first context. When an agent a receives a VALUE message from one of its ancestors then the "When Received" handler for VALUE messages gets called with p being the sending ancestor, d^p being the value of the sending ancestor, and TH^p being the desired threshold for agent a if the sending ancestor is its parent (and infinity otherwise). When agent a receives a COST message from one of its children then the "When Received" handler for COST messages gets called with c being the sending child, X^c being the context of the sending child, and LB^c and UB^c being the lower bound $LB^c_{X^c}$ and upper bound $UB^c_{X^c}$, respectively, of the sending child. Finally, when agent a receives a TERMINATE message then the "When Received" handler for TERMINATE messages gets called without any arguments. Leaf agents and the root agent in the constraint tree use the same handlers. Leaf agents do not have children and thus do not send VALUE messages. The root agent does not have a parent and thus does not send COST messages. The pseudocode is still work in progress. It experimentally determines optimal DCOP solutions but we have not yet proved its termination and correctness.

3.7 Execution Trace

Table 1 shows a trace of the execution of BnB-ADOPT for our example DCOP under the following assumptions. First, the initial values $ValInit(a)$ are zero for all agents $a \in V$. Second, the heuristics are always zero to prevent BnB-ADOPT from solving the DCOP immediately. Finally, we partition time into cycles (time slices) [9]. Every agent processes all messages during a cycle that it received in the previous cycle. Thus, when an agent sends a message to some other agent, the other agent receives the message only at the beginning of the next cycle. Table 1 shows the context, value, threshold, and lower and upper bounds for agent x_1 (first nine rows), agent x_2 (second nine rows) and agent x_3 (last nine rows) in that order at the end of each cycle for the eight cycles until BnB-ADOPT terminates.

4 Experimental Evaluation

We now compare BnB-ADOPT to two other memory-bounded DCOP algorithms that also restrict communication to agents that share constraints, namely

Cycle	1	2	3	4	5	6	7	8
X^{x_1}								
d^{x_1}	0	0	0	0	1	1	1	1
TH^{x_1}	∞	∞	∞	∞	∞	∞	∞	∞
$LB^{x_1}(0)$	0	5	8	8	15	15	15	15
$LB^{x_1}(1)$	0	0	0	0	0	0	3	9
LB^{x_1}	0	0	0	0	0	0	3	9
$UB^{x_1}(0)$	∞	∞	15	15	15	15	15	15
$UB^{x_1}(1)$	∞	∞	∞	∞	∞	∞	∞	9
UB^{x_1}	∞	∞	15	15	15	15	15	9
X^{x_2}	$(x_1,0)$	$(x_1,0)$	$(x_1,0)$	$(x_1,0)$	$(x_1,0)$	$(x_1,1)$	$(x_1,1)$	$(x_1,1)$
d^{x_2}	0	1	1	1	1	1	1	1
TH^{x_2}	∞	∞	∞	15	15	15	15	15
$LB^{x_2}(0)$	5	15	15	15	15	20	20	20
$LB^{x_2}(1)$	8	8	8	19	19	3	9	9
LB^{x_2}	5	8	8	15	15	3	9	9
$UB^{x_2}(0)$	∞	15	15	15	15	∞	∞	∞
$UB^{x_2}(1)$	∞	∞	∞	19	19	∞	9	9
UB^{x_2}	∞	15	15	15	15	∞	9	9
X^{x_3}	$(x_1,0)$	$(x_1,0)$	$(x_1,0)$	$(x_1,0)$	$(x_1,0)$	$(x_1,1)$	$(x_1,1)$	$(x_1,1)$
	$(x_2,0)$	$(x_2,0)$	$(x_2,1)$	$(x_2,1)$	$(x_2,1)$	$(x_2,1)$	$(x_2,1)$	$(x_2,1)$
d^{x_3}	0	0	1	1	1	1	1	1
TH^{x_3}	∞	∞	∞	∞	∞	∞	∞	∞
$LB^{x_3}(0)$	10	10	25	25	25	40	40	40
$LB^{x_3}(1)$	16	16	11	11	11	6	6	6
LB^{x_3}	10	10	11	11	11	6	6	6
$UB^{x_3}(0)$	10	10	25	25	25	40	40	40
$UB^{x_3}(1)$	16	16	11	11	11	6	6	6
UB^{x_3}	10	10	11	11	11	6	6	6

Table 1. Example Trace of BnB-ADOPT

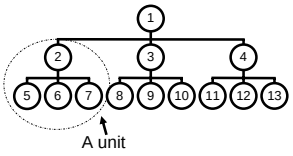


Fig. 4. Example: Scheduling Meetings

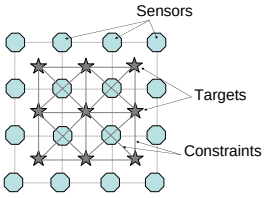


Fig. 5. Example: Allocating Targets

ADOPT and NCBB. NCBB is a memory-bounded synchronous branch-and-bound DCOP algorithm with the unusual feature that an agent can take on a different value for each one of its children. We compare BnB-ADOPT, ADOPT and NCBB on a variety of DCOPs, namely for scheduling meetings, allocating targets to sensors in sensor networks and coloring graphs, using DP2 [1] as admissible heuristics. We use the two most popular evaluation proxies for the runtime of a distributed system, namely the number of cycles [9], as introduced above, and the number of concurrent constraint checks (CCC) [8]. CCCs measure a weighted sum of processing and communication time. Every agent $a \in V$ maintains a counter CCC^a, which is initialized to zero. It increases its counter by one every time it performs a constraint check to account for the time it takes to perform the constraint check. It assigns $CCC^a := \max(CCC^a, CCC^{a'} + c)$ every time it receives a message from agent $a' \in V$ to account for the time it takes to wait for agent a' to send the message ($CCC^{a'}$) and the transmission time of the message (c). We use communication cost $c = 0$ to simulate fast communication and $c = 1000$ to simulate slow communication. The CCC then is the largest counter value of any agent.

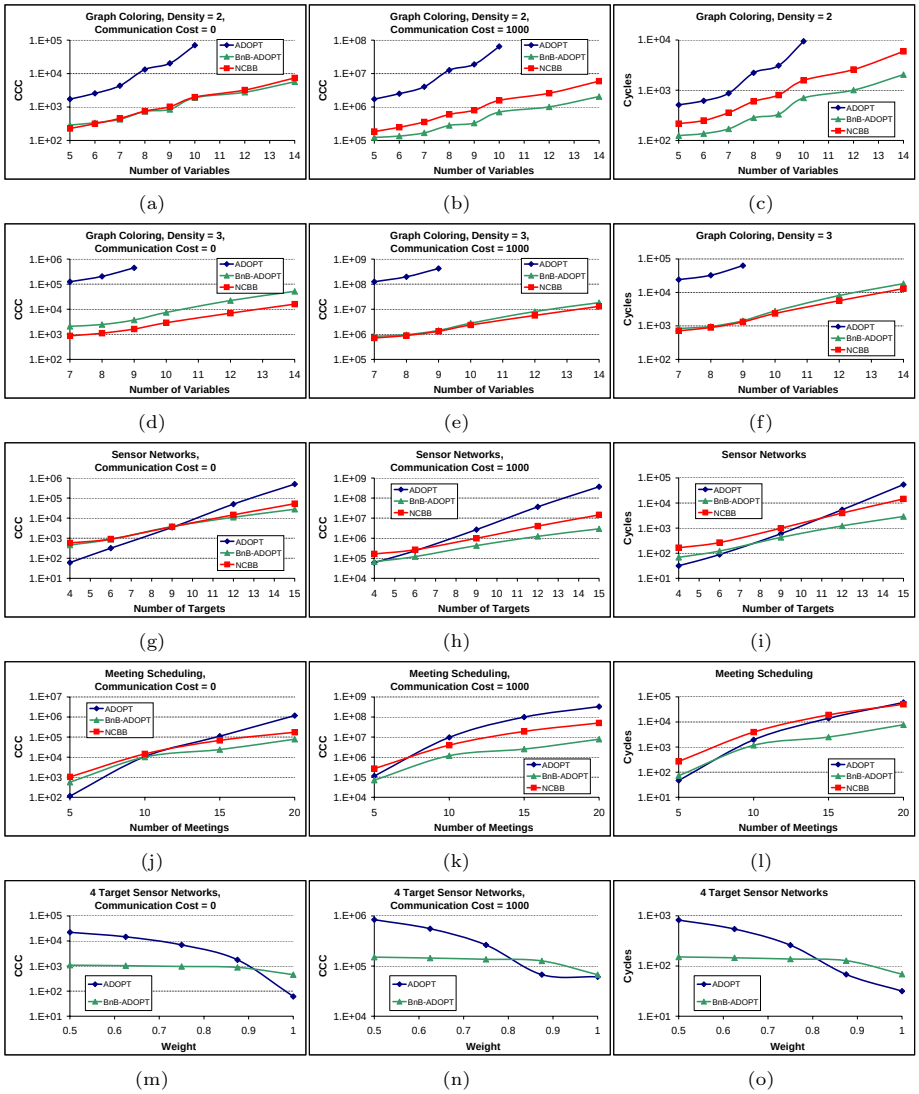


Fig. 6. Experimental Results

4.1 Domain: Coloring Graphs

"Coloring graphs" involves coloring the vertices of a graph, taking restrictions between the colors of adjacent vertices into account. The agents are the vertices, their domains are the colors, and the constraints are between adjacent vertices. We vary the number of vertices from 5 to 15 and the density, defined as the ratio between the number of constraints and the number of agents, from 2 (sparse graphs) to 3 (dense graphs). Each agent always has three possible values. All costs are randomly generated from 0 to 10,000. We average the experimental results over 50 DCOP instances with randomly generated constraints.

4.2 Domain: Scheduling Meetings

"Scheduling meetings" involves scheduling meetings between the employees of a company, taking restrictions in their availability as well as their priorities into account. The agents are the meetings, their domains are the time slots when they can be held, and the constraints are between meetings that share participants [6]. Figure 4 shows a hierarchical organization with four units of a supervisor and their three subordinates, such as supervisor 2 with their three subordinates 5, 6 and 7. In each unit, we assume five possible meetings: one of the entire unit (2, 5, 6, 7), two parent-child meetings (2, 5 and 2, 7), and two sibling-sibling meetings (5, 6 and 6, 7). We vary the number of meetings from 5 (1 unit) to 20 (4 units) . We always use 8 time slots. The cost of assigning a time slot to a meeting that has at least one participant who has another meeting during the same time slot is infinity (to be precise: 1,000,000) since the same person cannot attend more than one meeting at a time. The cost of a non-scheduled meeting is 100. All other costs are randomly generated from 0 to 100. We average the experimental results over 50 DCOP instances.

4.3 Domain: Allocating Targets to Sensors in Sensor Networks

"Allocating targets to sensors in sensor networks" involves assigning targets to sensors in a sensor network, taking restrictions in the availability of the sensors, restrictions in the number of sensors that need to track each target, and priorities of the targets into account. The agents are the targets, their domains are the time slots when they can be tracked, and the constraints are between adjacent targets [6]. Figure 5 shows a sensor network where the targets are located on a grid and each target is surrounded by 4 sensors, all of which are needed to track the target. We vary the number of targets from 4 to 15. The cost of assigning a time slot to a target that is also assigned to an adjacent target is infinity (to be precise: 1,000,000) since the same sensor cannot track both targets during the same time slot. The cost of targets that are not tracked during any time slot is 100. All other costs are randomly generated from 0 to 100. We average the experimental results over 50 DCOP instances.

4.4 Experimental Results

Figure 6(a-l) shows our experimental results, one row for each domain. The left column shows CCCs for fast communication, the center column shows CCCs for slow communication and the right column shows cycles, which also assume slow communication. We therefore expect the results for the center and right columns to be similar. The figure shows that BnB-ADOPT is faster than NCBB, and NCBB is faster than ADOPT - although the DCOPs need to be sufficiently large for this statement to be true in some cases. The following exceptions exist. BnB-ADOPT is faster than ADOPT and NCBB, which are equally fast, in terms of cycles for scheduling meetings. NCBB is faster than BnB-ADOPT, and BnB-ADOPT is faster than ADOPT in terms of CCCs with fast communication for coloring dense graphs. Finally, BnB-ADOPT and NCBB are equally fast

DCOP Algorithm	Search Strategy	Synchronization	Communication
ADOPT [9]	best-first	asynchronous	point-to-point with neighbors
NCBB [3]	DFBnB	synchronous	point-to-point with neighbors
AFB [4]	DFBnB	asynchronous	broadcast
BnB-ADOPT	DFBnB	asynchronous	point-to-point with neighbors

Fig. 7. Properties of DCOP Algorithms

and faster than ADOPT in terms of both CCCs with slow communication and cycles for coloring dense graphs, and in terms of CCCs with fast communication for coloring sparse graphs and allocating targets to sensors in sensor networks. Thus, BnB-ADOPT is at least as fast as both ADOPT and NCBB in all cases but one. BnB-ADOPT and ADOPT differ only in their search strategy. ADOPT uses memory-bounded best-first search and thus exploits the heuristics well but needs to repeatedly reconstruct partial solutions that it needed to purge from memory, especially if the heuristics are poorly informed. BnB-ADOPT uses depth-first branch-and-bound search and thus does not exploit the heuristics quite as well but does not have to repeatedly reconstruct partial solutions that it needed to purge from memory. Thus, ADOPT benefits from well informed heuristics. This intuition explains why ADOPT can be faster than BnB-ADOPT for small DCOPs, as the figure shows for allocating targets to sensors in sensor networks and scheduling meetings. We confirm our intuition with an additional experiment on small DCOPs for allocating four targets to sensors in sensor networks, where we vary the quality of the heuristics. We use $h' = w \times h$ for $0.5 \leq w \leq 1$, where h are the heuristics used before. Indeed, ADOPT can be faster than BnB-ADOPT for large values of w, that is, well informed heuristics. The runtime of ADOPT depends much more on the informedness of the heuristics than the runtime of BnB-ADOPT since ADOPT relies on the heuristics more than BnB-ADOPT. BnB-ADOPT tends to be faster than ADOPT for small values of w, that is, poorly informed heuristics. Thus, BnB-ADOPT has great potential as a DCOP algorithm since the heuristics tend to be the more poorly informed the larger the DCOPs are.

5 Conclusions

We introduced BnB-ADOPT, a memory-bounded asynchronous DCOP algorithm that uses the message passing and communication framework of ADOPT but changes the search strategy from best-first search to depth-first branch-and-bound search (DFBnB). Figure 7 shows how the properties of BnB-ADOPT compare to those of other memory-bounded DCOP algorithms. Our experimental results showed that BnB-ADOPT was up to one order of magnitude faster than ADOPT on a variety of large DCOPs (since ADOPT uses memory-bounded best-first search and thus needs to repeatedly reconstruct partial solutions that it purged from memory) and faster than NCBB on most of these DCOPs (since NCBB is synchronous and agents are thus often idle while waiting for activation messages from other agents). It is future work to improve BnB-ADOPT and ADOPT further, for example, to reduce the number of messages sent.

We have not compared BnB-ADOPT to DPOP [10] since DPOP is not memory-bounded, which can make its application infeasible in domains where each agent has only a limited amount of memory available, such as allocating targets to sensors in sensor networks. We have not compared BnB-ADOPT to OptAPO [7] since OptAPO is partially centralized, which can make its application infeasible in domains with privacy concerns, such as scheduling meetings. These comparisons as well as the comparison of BnB-ADOPT to AFB [4] are the topic of future work.

References

1. S. Ali, S. Koenig, and M. Tambe. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *Proceedings of AAMAS*, pages 1041–1048, 2005.
2. C. Bessiere, A. Maestre, and P. Messeguer. Distributed dynamic backtracking. In *Proceedings of the Distributed Constraint Reasoning Workshop*, pages 9–16, 2001.
3. A. Chechetka and K. Sycara. No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of AAMAS*, pages 1427–1429, 2006.
4. A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward-bounding for distributed constraints optimization. In *Proceedings of ECAI*, pages 103–107, 2006.
5. P. Hart, N. Nilsson, and N. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC4(2):100–107, 1968.
6. R. Maheswaran, M. Tambe, E. Bowring, J. Pearce, and P. Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. In *Proceedings of AAMAS*, pages 310–317, 2004.
7. R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of AAMAS*, pages 438–445, 2004.
8. A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. Comparing performance of distributed constraints processing algorithms. In *Proceedings of the Distributed Constraint Reasoning Workshop*, pages 86–93, 2002.
9. P. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005.
10. A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of IJCAI*, pages 1413–1420, 2005.
11. N. Schurr, S. Okamoto, R. Maheswaran, P. Scerri, and M. Tambe. Evolution of a teamwork model. In R. Sun, editor, *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*, pages 307–327. Cambridge University Press, 2005.
12. M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.
13. W. Zhang and R. Korf. Performance of linear-space search algorithms. *Artificial Intelligence*, 79(2):241–292, 1995.