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

William Yeoh
Computer Science
University of Southern California
Los Angeles, CA 90089, USA
wyeoh@usc.edu

Ariel Felner
Information Systems Engineering
Ben-Gurion University
Beer-Sheva, 85104, Israel
felner@bgu.ac.il

Sven Koenig
Computer Science
University of Southern California
Los Angeles, CA 90089, USA
skoenig@usc.edu

ABSTRACT

Distributed constraint optimization (DCOP) problems are a popular way of formulating and solving agent-coordination problems. It is often desirable to solve DCOP problems optimally with memory-bounded and asynchronous algorithms. We 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 DCOP problems and faster than NCBB, a memory-bounded synchronous DCOP algorithm, on most of these DCOP problems.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms

Keywords

Agent cooperation::distributed problem solving

1. INTRODUCTION

Distributed constraint optimization (DCOP) problems are a popular way of formulating and solving agent-coordination problems, including scheduling meetings [8], coordinating unmanned aerial vehicles [14] and allocating targets to sensors [1]. A DCOP problem 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. DCOPs can be solved quickly if the agents can act independently without having to wait for other agents. This calls for asynchronous DCOP algorithms. ADOPT is a popular DCOP algorithm that satisfies these three constraints and uses best-first search to solve DCOP problems optimally [12]. In this paper, we develop another DCOP algorithm that satisfies the three constraints and solves DCOP problems optimally but is faster than ADOPT on many DCOP problems.

DCOP problems 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 [16]. Consequently, depth-first branch-and-bound search has been extended to distributed constraint satisfaction problems [6, 2] and even to DCOP problems. However, the existing depth-first branch-and-bound DCOP algorithms are either synchronous (NCBB [3] and SBB [6]) 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 DCOP problems, namely for coloring graphs, scheduling meetings and allocating targets to sensors. It is also faster than NCBB on most of these DCOP problems.

2. DCOP PROBLEMS

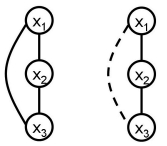
A DCOP problem consists of a finite set of agents with independent computing power and a finite set of constraints. Each agent takes on (= assigns itself) a value from its finite

*This research was done while Ariel Felner spent his sabbatical at the University of Southern California, visiting Sven Koenig. 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.

Cite as: BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm, W. Yeoh, A. Felner, S. Koenig, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 591-598.

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 problem 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 and messages that it can exchange with the other agents. These messages can be delayed by a finite amount of time but are never lost.

3. BnB-ADOPT



(a) (b)

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

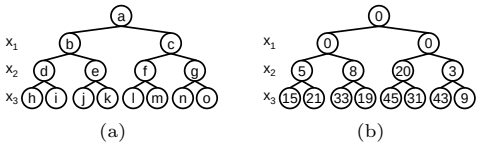
for $i < j$

(c)

Figure 1: Example DCOP Problem

DCOP problems can be represented with constraint graphs, whose vertices are the variables and whose edges are the constraints. Figure 1(a) shows the constraint graph of an example DCOP problem 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 of the example DCOP problem, which are the same for all three constraints. Constraint trees are spanning trees of constraint graphs with the property that edges of the constraint graphs can connect vertices only with their ancestors or descendants in the constraint trees. Sibling subtrees represent disjoint subproblems of the DCOP problem. Figure 1(b) shows one possible constraint tree of the example DCOP problem, where the dotted line is part of the constraint graph but not the constraint tree. This constraint tree is actually a constraint chain and thus there are no disjoint subproblems. The operation of DCOP algorithms on constraint trees can be visualized with search trees. Figure 2 shows a search tree for this constraint tree, where levels 1, 2 and 3 of the search 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. Each non-leaf node thus corresponds to a partial solution of the DCOP problem and each leaf node to a solution. Figure 2(a) shows the identifiers of the nodes that allow us to refer to them easily, and Figure 2(b) shows the sums of the constraint costs of all constraints that involve only agents with known values. These sums correspond to the f-values of an A* search [5] since we assume for simplicity of illustration that all heuristics are zero for our example DCOP problem. For example, node f corresponds to agent x_1 taking on value one, agent x_2 taking on value zero and agent x_3 having an unknown value. Thus, the sum of the constraint costs of all constraints that involve only agents with known values is 20, namely the cost of the constraint that involves agents x_1 and x_2.

ADOPT [12] is a best-first search algorithm. Best-first



(a) (b)

Figure 2: Search Tree

search expands nodes in the search tree for the first time in order of increasing f-values until it finds a solution. For our example DCOP problem, best-first search expands nodes in the search tree for the first time in the order a, b, c, g, d, e, and o. ADOPT is basically a distributed version of RFBS [7]. In order to be memory-bounded, it maintains only a branch from the root node to the currently expanded 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 expands node e but then needs to have the branch from a to o in memory when it expands node o. Thus, it needs to reconstruct the part of this branch from a to g. Depth-first branch-and-bound search, on the other hand, expands the children of a node in order of increasing f-values and prunes those nodes whose f-values are no smaller than the smallest f-value of any leaf node that it has already observed. It backtracks once all children of a node have been expanded or pruned. For our example DCOP problem, depth-first branch-and-bound search expands nodes in the search tree 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 expands some nodes that a best-first search does not expand, such as node h. Centralized depth-first branch-and-bound search algorithms often run faster than centralized best-first search algorithms. They can be used to solve DCOP problems but typically order the agents completely. Distributed depth-first branch-and-bound search algorithms might be able to solve DCOP problems faster by operating on disjoint subproblems concurrently, as demonstrated by AOBB [10]. One can convert centralized depth-first branch-and-bound search algorithms relatively easily into synchronous distributed DCOP algorithms. Asynchronous distributed DCOP algorithms might be able to solve DCOP problems faster by not having to synchronize the agents tightly.

We therefore develop BnB-ADOPT, a novel asynchronous distributed DCOP algorithm that performs depth-first branch-and-bound search, by using the existing architecture and communication framework of ADOPT, resulting in an asynchronous distributed version of AOBB [10]. However, we do not describe BnB-ADOPT as an extension of ADOPT since this requires the readers to have an in-depth understanding of ADOPT. Instead, we give a stand-alone description of BnB-ADOPT that requires no knowledge of ADOPT, with the intention to create a self-contained and hopefully easy-to-read overview. In the following, we introduce the notation that we need for describing BnB-ADOPT and describe some of its key variables, including how they are updated. We describe a simplified depth-first search version of BnB-ADOPT, which we then enhance by performing branch-and-bound and increasing concurrency. We show the pseudocode of BnB-ADOPT, outline its correctness proof and finally describe our experimental results.

3.1 Notation

We use the following notation from ADOPT to describe BnB-ADOPT: V is the finite set of agents of the DCOP problem. $Dom(a)$ 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 partial solution X^a (a set of agent-value assignments), called the context of agent a. $\delta^a_{X^a}(d)$ 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 their respective values in X^a. $\gamma^a_{X^a}(d)$ is the sum of the constraint costs of all constraints that involve agent a and/or one of its descendants,[1] minimized over the possible values of its descendants, under the assumption that agent a takes on value d and its ancestors take on the values in X^a.[2] We use the relationships

$$\begin{aligned} \gamma^a_{X^a} &:= \min_{d' \in Dom(a)} \{\gamma^a_{X^a}(d')\} \\ \gamma^a_{X^a}(d) &:= \delta^a_{X^a}(d) + \sum_{a' \in C(a)} \gamma^{a'}_{X^a \cup (a,d)} \end{aligned}$$

for all agents $a \in V$, all values $d \in Dom(a)$ and all contexts X^a of agent a. Solving the DCOP problem optimally means to determine $\gamma^r_{X^r}$ for the root agent r in the constraint tree since $\gamma^r_{X^r}$ is the sum of the constraint costs of all constraints, minimized over the possible values of all agents.[3]

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

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

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

$$\begin{aligned} lb^{a,a'}_{X^a}(d) &:= \max\{lb^{a,a'}_{X^a}(d), LB^{a'}_{X^a \cup (a,d)}\} \\ LB^a_{X^a}(d) &:= \delta^a_{X^a}(d) + \sum_{a' \in C(a)} lb^{a,a'}_{X^a}(d) \\ LB^a_{X^a} &:= \min_{d' \in Dom(a)} \{LB^a_{X^a}(d')\} \\ ub^{a,a'}_{X^a}(d) &:= \min\{ub^{a,a'}_{X^a}(d), UB^{a'}_{X^a \cup (a,d)}\} \\ UB^a_{X^a}(d) &:= \delta^a_{X^a}(d) + \sum_{a' \in C(a)} ub^{a,a'}_{X^a}(d) \\ UB^a_{X^a} &:= \min_{d' \in Dom(a)} \{UB^a_{X^a}(d')\} \end{aligned}$$

for all values $d \in Dom(a)$, all contexts X^a of agent a and all children $a' \in C(a)$, which improve the bounds monotonically (that is, decrease the upper bounds and increase the lower bounds) while maintaining the Bound Property.[4] After a finite amount of time, $LB^a_{X^a} = UB^a_{X^a}$ for all agents $a \in V$, all values $d \in Dom(a)$ and all contexts X^a of agent a. Then, $LB^r_{X^r} = \gamma^r_{X^r} = UB^r_{X^r}$, which means that the DCOP problem is solved optimally. Termination is achieved by sending TERMINATE messages from parents to children down the constraint tree.

3.3 Simplified Version of BnB-ADOPT

In actuality, every agent $a \in V$ stores $lb^{a,a'}_{X^a}(d)$, $LB^a_{X^a}(d)$, $LB^a_{X^a}$, $ub^{a,a'}_{X^a}(d)$, $UB^a_{X^a}(d)$ and $UB^a_{X^a}$ for all values $d \in Dom(a)$ and all children $a' \in C(a)$ but only *one* context X^a 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^a, which makes it unnecessary to index the other variables with X^a (although we continue to do so). BnB-ADOPT uses depth-first search as its 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^a and value $d^a \in Dom(a)$. The agent sends so-called VALUE messages to all children $a' \in C(a)$ with the context $X^a \cup (a, d^a)$. The children return so-called COST messages with $LB^{a'}_{X^a \cup (a,d^a)}$ and $UB^{a'}_{X^a \cup (a,d^a)}$. The agent then uses the Update Equations to improve $lb^{a,a'}_{X^a}(d^a)$, $LB^a_{X^a}(d^a)$, $LB^a_{X^a}$, $ub^{a,a'}_{X^a}(d^a)$, $UB^a_{X^a}(d^a)$, and $UB^a_{X^a}$ and sends $LB^a_{X^a}$ and $UB^a_{X^a}$ to its parent in a COST message. If $LB^a_{X^a}(d^a) < UB^a_{X^a}(d^a)$, then the agent repeats the process. After a finite amount of time, $LB^a_{X^a}(d^a) = UB^a_{X^a}(d^a)$, which means that $LB^a_{X^a}(d^a)$ and $UB^a_{X^a}(d^a)$ cannot be improved further. The agent then takes on the new value $d^a := \arg\min_{d \in Dom(a)}\{LB^a_{X^a}(d)\}$ and repeats the process until either its context X^a changes (because the agent receives a VALUE message from its parent with a different context) or $LB^a_{X^a}(d) = UB^a_{X^a}(d)$ for all values $d \in Dom(a)$ and thus $LB^a_{X^a} = UB^a_{X^a}$ after a finite amount of time. Every agent $a \in V$ takes on every value $d \in Dom(a)$ at most

[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, $\gamma^a_{X^a}(d)$ 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 solution $X^a \cup (a, d)$ with agent-value assignments for all descendants of agent a.

[3] X^r is always $\{\}$.

[4] 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, $LB^a_{X^a}(d) = UB^a_{X^a}(d) = \delta^a_{X^a}(d)$ for all leaf agents $a \in V$, all values $d \in Dom(a)$ and all contexts X^a of agent a.

once until its context X^a changes or $LB^a_{X^a} = UB^a_{X^a}$. 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. (A more detailed explanation is given in [15].) Every agent $a \in V$ maintains a threshold $TH^a_{X^a}$, initialized to infinity (Line 17).[5] The threshold is used for pruning during the depth-first search, resulting in the agent not taking on some values. Every agent $a \in V$ with value $d^a \in Dom(a)$ and context X^a operates as described before until its context changes, with two differences: First, it uses $\min\{TH^a_{X^a}, UB^a_{X^a}\}$ instead of the larger $UB^a_{X^a}(d^a)$ in the condition that determines whether it should take on a new value, resulting in the agent not taking on some values. Consequently, if $LB^a_{X^a}(d^a) \geq \min\{TH^a_{X^a}, UB^a_{X^a}\}$, then it takes on the new value $d^a := \arg\min_{d \in Dom(a)}\{LB^a_{X^a}(d)\}$ (Lines 23, 24). Second, when agent $a \in V$ sends a VALUE message to its child $a' \in C(a)$, it now includes not only the desired context $X^a \cup (a, d^a)$ but also the desired threshold $\min\{TH^a_{X^a}, UB^a_{X^a}\} - \delta^a_{X^a}(d^a) - \sum_{a'' \in C(a), a'' \neq a'} lb^{a,a''}_{X^a}(d^a)$ for the child (Line 29). This desired threshold is chosen such that $LB^a_{X^a}(d^a)$ for the agent reaches $\min\{TH^a_{X^a}, UB^a_{X^a}\}$ and the agent thus takes on a new value when $LB^{a'}_{X^a \cup (a,d^a)}$ for the child reaches the desired threshold.

The context of the agent changes when its parent sends it a VALUE message with a context X^a different from its context. The agent then changes its context to X^a, changes its threshold to the threshold in the VALUE message, initializes $lb^{a,a'}_{X^a}(d) := h^{a,a'}_{X^a}(d)$ and $ub^{a,a'}_{X^a}(d) := \infty$ and uses the Update Equations to initialize $LB^a_{X^a}(d)$, $UB^a_{X^a}(d)$, $LB^a_{X^a}$ and $UB^a_{X^a}$ for all values $d \in Dom(a)$ and all children $a' \in C(a)$, takes on the new value $d^a := \arg\min_{d \in Dom(a)}\{LB^a_{X^a}(d)\}$ 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.

Second, the agents propagate contexts differently than described so far. 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) (Lines 29,30). 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 if it is more recent (Line 33). Agents still send COST messages to their parents but the COST messages now include the context of the sending agent (Line 31). Every receiving agent then changes the values of the agents in its context to the ones in the COST message if they are more recent (Line 42). The VALUE and COST messages together allow agents to update the values of all ancestors that they or their descendants are involved in constraints with, which make up exactly their context. To implement this scheme, the agents need to determine which values are more recent: the ones contained in VALUE or COST messages or the ones in their context. To this end, all agents maintains their own counters, called ID, and increment them whenever they change their values (Lines 16, 25). All contexts now contain agent-value-ID assignments. The VALUE messages contain the value of the sending agent and its ID (Lines 29, 30). The receiving agent of VALUE or COST messages changes the values of those agents in its context to the ones in the messages whose values in the messages have larger IDs than their values in its context.

Third, agents can no longer assume that their children send bounds in their COST messages for the desired contexts. Thus, in the implemented version of BnB-ADOPT, agents check whether the contexts in COST messages are the desired contexts (Line 47). If not, they ignore the COST messages since they are irrelevant for improving the bounds for their contexts.

Fourth, if the contexts of agents change, the desired contexts of some of their 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 $lb^{a,a'}_{X^a}(d)$ and $ub^{a,a'}_{X^a}(d)$ for all values $d \in Dom(a)$ (Lines 44, 45, 46). 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 $lb^{a,a'}_{X^a}(d)$ and $ub^{a,a'}_{X^a}(d)$ for the value $d \in Dom(a)$ that the agent takes on in the context in the COST message (Lines 47, 48, 49). Then, the agent uses the Update Equations to initialize $LB^a_{X^a}(d)$, $UB^a_{X^a}(d)$, $LB^a_{X^a}$ and $UB^a_{X^a}$ for all values $d \in Dom(a)$ and takes on the new value $d^a := \arg\min_{d \in Dom(a)}\{LB^a_{X^a}(d)\}$.

3.6 Pseudocode

Figure 3 shows the BnB-ADOPT pseudocode of every agent. The pseudocode uses the predicate $\text{Compatible}(X, X') = \neg\exists_{(a,d,ID) \in X, (a',d',ID') \in X'}(a = a' \wedge d \neq d')$ that determines whether two contexts are compatible by checking that they do not make the same agent take on different values. It also uses the procedure $\text{PriorityMerge}(X, X')$ that executes $X' := \{(a', d', ID') \in X' \mid \neg\exists_{(a,d,ID) \in X}(a = a')\} \cup \{(a', d', ID') \in X' \mid \exists_{(a,d,ID) \in X}(a = a' \wedge ID \leq ID')\} \cup \{(a, d, ID) \in X \mid \exists_{(a',d',ID') \in X'}(a = a' \wedge ID > ID')\}$ and thus replaces the agent-value-ID assignments in the second context with more recent ones for the same agents from the first context.

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. All agents use the same code. Leaf agents in the constraint tree do not have children and thus do not send VALUE or TERMINATE messages. The root agent does not have a parent and thus does not send COST messages. When an agent a receives a VALUE message from one of its ancestors then the

[5]The threshold of the root agent r in the constraint tree is always infinity.

```
procedure Start ()
{01}  X^a := {(a', ValInit (a'), 0) | a' ∈ SCA(a)};
{02}  ID^a := 0;
{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}  ID^a := ID^a + 1;
{17}  TH^a := ∞;

procedure Backtrack ()
{18}  forall d ∈ Dom(a)
{19}    LB^a(d) := δ^a(d) + Σ_{a'∈C(a)} lb^{a,a'}(d);
{20}    UB^a(d) := δ^a(d) + Σ_{a'∈C(a)} ub^{a,a'}(d);
{21}  LB^a := min_{d∈Dom(a)} {LB^a(d)};
{22}  UB^a := min_{d∈Dom(a)} {UB^a(d)};
{23}  if (LB^a(d^a) ≥ min{TH^a, UB^a})
{24}    d^a := arg min_{d∈Dom(a)} {LB^a(d)} (pick the previous d^a if
            possible);
{25}    ID^a := ID^a + 1;
{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, ID^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, ID^a, ∞) to each a' ∈ CD(a) \ C(a);
{31}  Send (COST, a, X^a, LB^a, UB^a) to pa(a) if a is not root;

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

procedure When Received (COST, c, X^c, LB^c, UB^c)
{41}  X' := X^a;
{42}  PriorityMerge (X^c, X^a);
{43}  if (!Compatible (X', X^a))
{44}    forall a' ∈ C(a), d ∈ Dom(a)
{45}      if (!Compatible ({(a'', d'', ID'') ∈ X' | a'' ∈ SCA(a')},X^a))
{46}        InitChild (a',d);
{47}  if (Compatible (X^c, X^a))
{48}    lb^{a,c}(d) := max{lb^{a,c}(d), LB^c} for the unique (a', d, ID) ∈ X^c
                with a' = a;
{49}    ub^{a,c}(d) := min{ub^{a,c}(d), UB^c} for the unique (a', d, ID) ∈ X^c
                with a' = a;
{50}  if (!Compatible (X', X^a))
{51}    InitSelf ();

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

Figure 3: Pseudocode of BnB-ADOPT

"When Received" handler for VALUE messages gets called with p being the sending ancestor, d^p being the value of the sending ancestor, ID^p being the ID 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. An execution trace of the pseudocode is given in [15].

Overall, BnB-ADOPT uses the message passing and communication framework of ADOPT. It uses the same VALUE, COST and TERMINATE 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 $lb^{a,a'}_{X^a}(d)$, $LB^a_{X^a}(d)$, $LB^a_{X^a}$, $ub^{a,a'}_{X^a}(d)$, $UB^a_{X^a}(d)$ and $UB^a_{X^a}$; and 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 partial solutions that it purged from memory. Thus, it uses a different threshold initialization (Line 17), threshold propagation (Line 29), threshold update (Line 40) and termination condition (Line 26). Also, it maintains IDs that indicate the recency of agent-value assignments and contexts that contain agent-value-ID assignments.

3.7 Correctness and Completeness Proofs

DEFINITION 1. *Contexts are correct iff all of the IDs in the agent-value-ID assignments of the contexts correspond to the current IDs of the agents, which implies that all values in the agent-value-ID assignments also correspond to the values that the agents currently take on (= their current values).*

LEMMA 1. *For an agent $a \in V$ with the property that both the current context X^a of itself and the current contexts of its ancestors $a' \in SCA(a)$ are correct and no longer change, $LB^a_{X^a}(d)$ and $LB^a_{X^a}$ are monotonically non-decreasing, $UB^a_{X^a}(d)$ and $UB^a_{X^a}$ are monotonically non-increasing, $LB^a_{X^a}(d) \leq UB^a_{X^a}(d)$ and $LB^a_{X^a} \leq \gamma^a_{X^a} \leq UB^a_{X^a}$ for all values $d \in Dom(a)$.*

DEFINITION 2. *The potential of an agent $a \in V$ is $\sum_{d\in Dom(a)}\{UB^a_{X^a}(d) - LB^a_{X^a}(d)\}$ for its current context X^a.*

THEOREM 1. *For an agent $a \in V$ with the property that both the current context X^a of itself and the current contexts of its ancestors $a' \in SCA(a)$ are correct and no longer change, its potential is monotonically non-increasing and decreases by more than a positive constant every time it changes its value.*

Proof: The potential of agent a is monotonically non-increasing since $LB^a_{X^a}(d)$ is monotonically non-decreasing and $UB^a_{X^a}(d)$ is monotonically non-increasing for all values $d \in Dom(a)$ according to Lemma 1. Agent a changes from its current value d^a to a new value only if

$\min_{d \in Dom(a)}\{LB^a_{X^a}(d)\} < LB^a_{X^a}(d^a)$. Thus, $LB^a_{X^a}(d^a)$ must have strictly increased from the point in time when the agent changed to d^a to the point in time when it changes from d^a to a new value since the $LB^a_{X^a}(d)$ are monotonically non-decreasing for all values $d \in Dom(a)$ according Lemma 1. Thus, its potential decreases by more than a positive constant since the $LB^a_{X^a}(d)$ are monotonically non-increasing and the $UB^a_{X^a}(d)$ are monotonically non-decreasing. The positive constant is the smallest possible increase of $LB^a_{X^a}(d^a)$, which is bounded from below by the greatest common divisor of all constraint costs and heuristics. (If all constraint costs and heuristics are integers, it is at least one.) ■

THEOREM 2. *No agent can change its value an infinite number of times.*

Proof by contradiction: Choose an agent $a \in V$ that changes its value an infinite number of times but all of whose ancestors $a' \in SCA(a)$ change their values only a finite number of times. Then, there exists a point in time when they do not change their values any longer. Furthermore, there exists a (later) point in time when both the current context of agent a and the current contexts of its ancestors $a' \in SCA(a)$ are correct and no longer change since all messages are delivered with finite delay. Every time agent a changes its value, its potential decreases by more than a positive constant towards minus infinity according to Theorem 1. On the other hand, its potential cannot become negative since $LB^a_{X^a}(d) \leq UB^a_{X^a}(d)$ for all values $d \in Dom(a)$ according to Lemma 1, which is a contradiction. Thus, no agent can change its value an infinite number of times. ■

THEOREM 3. *$UB^a_{X^a} = LB^a_{X^a}$ eventually holds for all agents $a \in V$ and their current contexts X^a.*

Proof: Every agent changes its value only a finite number of times according to Theorem 2. Then, there exists a point in time when all agents do not change their values any longer. Furthermore, there exists a (later) point in time when the current contexts of all agents are correct and no longer change since all messages are delivered with finite delay.

Assume that, at this point in time, agent $a \in V$ is a leaf agent (induction basis). Then, $LB^a_{X^a}(d) = \delta^a(d) = UB^a_{X^a}(d)$ for its current context X^a and all values $d \in Dom(a)$. Thus, $LB^a_{X^a} = UB^a_{X^a}$. Now assume that agent $a \in V$ is not a leaf agent but all of its children $a' \in C(a)$ satisfy $LB^{a'}_{X^{a'}} = UB^{a'}_{X^{a'}}$ for their current contexts $X^{a'}$ (induction step). Thus, eventually $LB^a_{X^a}(d^a) = UB^a_{X^a}(d^a)$ for its current value d^a and its current context X^a. Since agent a does not change its current value d^a at this point in time, it must be that $LB^a_{X^a}(d^a) < \min\{TH^a_{X^a}, UB^a_{X^a}\}$ or $LB^a_{X^a}(d^a) = \min_{d \in Dom(a)}\{LB^a_{X^a}(d)\}$. The first disjunct implies that $\min\{TH^a_{X^a}, UB^a_{X^a})\} \leq UB^a_{X^a} \leq UB^a_{X^a}(d^a) = LB^a_{X^a}(d^a) < \min\{TH^a_{X^a}, UB^a_{X^a}\}$, which is a contradiction. The second disjunct implies that $UB^a_{X^a} \leq UB^a_{X^a}(d^a) = LB^a_{X^a}(d^a) = \min_{d \in Dom(a)}\{LB^a_{X^a}(d)\} = LB^a_{X^a}$ and thus $LB^a_{X^a} = UB^a_{X^a}$ since $LB^a_{X^a} \leq UB^a_{X^a}$ according to Lemma 1. ■

THEOREM 4. *BnB-ADOPT terminates with the cost of a cost-minimal solution.*

Proof: Eventually, $UB^a_{X^a} = LB^a_{X^a}$ for all agents $a \in V$ and their current contexts, X^a. In particular, $UB^r_{X^r} = LB^r_{X^r}$ for the root agent r and BnB-ADOPT terminates. BnB-ADOPT terminates with the cost of a cost-minimal solution since $LB^r_{X^r} \leq \gamma^r_{X^r} \leq UB^r_{X^r}$ according to Lemma 1 and thus $LB^r_{X^r} = \gamma^r_{X^r} = UB^r_{X^r}$. ■

4. EXPERIMENTAL EVALUATION

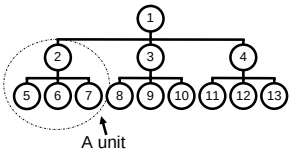


Figure 4: Example: Scheduling Meetings

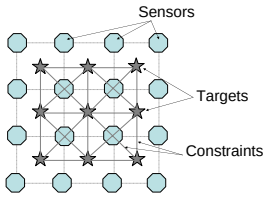


Figure 5: Example: Allocating Targets

We now compare BnB-ADOPT to two other memory-bounded DCOP algorithms that also restrict communication to agents that share constraints, namely 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 DCOP problems, namely for scheduling meetings, allocating targets to sensors and coloring graphs, using DP2 [1] as admissible heuristics. We use the number of non-concurrent constraint checks (NCCCs) [11] as our evaluation metric. NCCCs are a weighted sum of processing and communication time. Every agent $a \in V$ maintains a counter $NCCC^a$, which is initialized to zero. It assigns $NCCC^a := NCCC^a + 1$ every time it performs a constraint check to account for the time it takes to perform the constraint check. It assigns $NCCC^a := \max\{NCCC^a, NCCC^{a'} + c\}$ 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 ($NCCC^{a'}$) and the transmission time of the message (c). We use $c = 0$ to simulate fast communication and $c = 1000$ to simulate slow communication. The number of NCCCs then is the largest counter value of any agent.

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

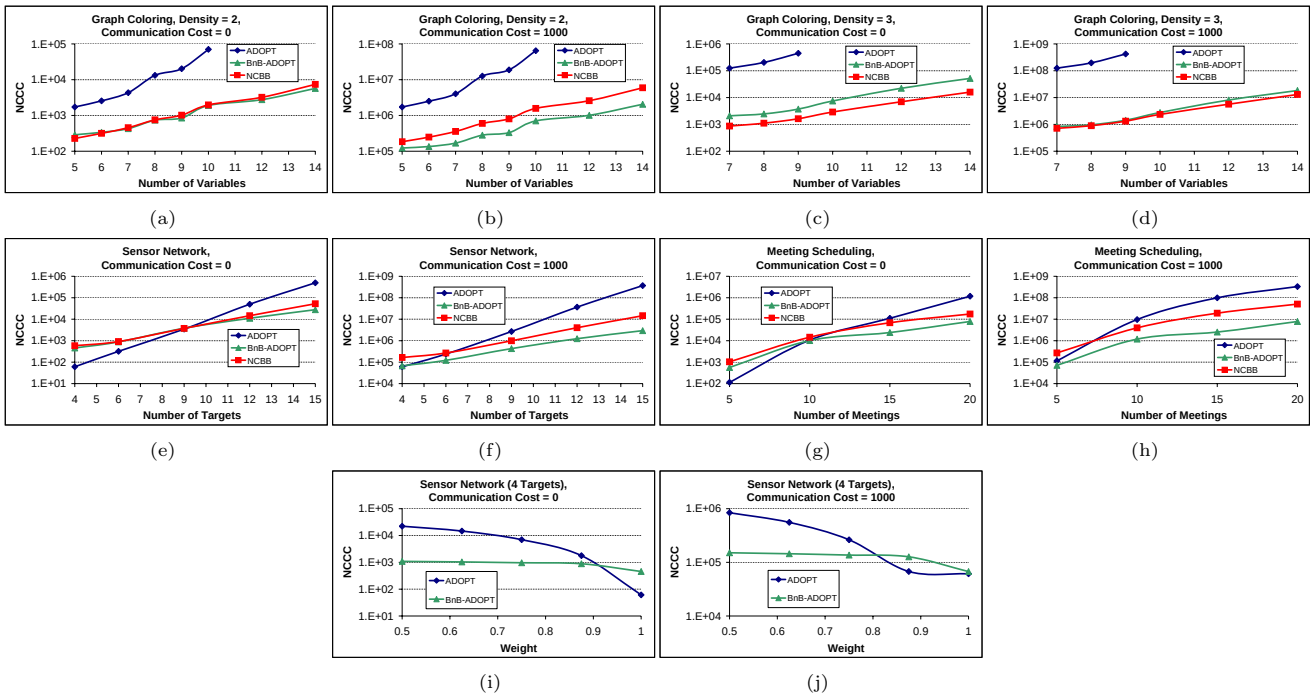


Figure 6: Experimental Results

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 problem 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 [8]. 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 problem instances.

4.3 Domain: Allocating Targets to Sensors

"Allocating targets to sensors" 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 [8]. 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 problem instances.

4.4 Experimental Results

Figure 6(a-h) shows our experimental results for the three domains. The figure shows that BnB-ADOPT is faster than NCBB, and NCBB is faster than ADOPT - although the DCOP problems need to be sufficiently large for this statement to be true in some cases. The following exceptions exist. NCBB is faster than BnB-ADOPT, and BnB-ADOPT is faster than ADOPT for coloring dense graphs with fast communication. BnB-ADOPT and NCBB are equally fast and faster than ADOPT for coloring dense graphs with slow communication and for coloring sparse graphs and allocating targets to sensors with fast communication. 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 purged from memory, especially if the heuristics are poorly informed. BnB-ADOPT uses depth-first branch-and-bound search and thus does not

DCOP Algorithm	Search Strategy	Synchronization	Communication	Topology
SBB [6]	DFBnB	synchronous	point-to-point with neighbors	constraint chain
ADOPT [12]	best-first	asynchronous	point-to-point with neighbors	constraint tree
NCBB [3]	DFBnB	synchronous	point-to-point with neighbors	constraint tree
AFB [4]	DFBnB	asynchronous	broadcast	constraint chain
BnB-ADOPT	DFBnB	asynchronous	point-to-point with neighbors	constraint tree

Table 1: Properties of DCOP Algorithms

exploit the heuristics quite as well but does not have to repeatedly reconstruct partial solutions that it purged from memory. Thus, ADOPT benefits from well informed heuristics. This intuition explains why ADOPT can be faster than BnB-ADOPT for small DCOP problems. We confirm our intuition with an additional experiment on small DCOP problems for allocating four targets to sensors, 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 heuristics are often poorly informed for difficult DCOP problems, such as problems with large numbers of agents, large numbers of values, large numbers of constraints and large variations in constraint costs.

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 1 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 DCOP problems (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 DCOP problems (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. It is also future work to compare BnB-ADOPT to additional DCOP algorithms. We have not compared BnB-ADOPT to DPOP [13] in this paper 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 for allocating targets to sensors. We have not compared BnB-ADOPT to OptAPO [9] in this paper since OptAPO is partially centralized, which can make its application infeasible in domains with privacy concerns, such as for scheduling meetings. We have not compared BnB-ADOPT to SBB [6] in this paper since it has already been shown that ADOPT outperforms SBB [12]. These comparisons as well as the comparison of BnB-ADOPT to AFB [4] are topics of future work.

6. 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 B. 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] K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. In *Principles and Practice of Constraint Programming*, pages 222–236, 1997.

[7] R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.

[8] 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.

[9] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of AAMAS*, pages 438–445, 2004.

[10] R. Marinescu and R. Dechter. AND/OR branch-and-bound for graphical models. In *Proceedings of IJCAI*, pages 224–229, 2005.

[11] 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.

[12] 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.

[13] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of IJCAI*, pages 1413–1420, 2005.

[14] 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.

[15] W. Yeoh, A. Felner, and S. Koenig. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. In *Proceedings of the Distributed Constraint Reasoning Workshop*, 2007.

[16] W. Zhang and R. Korf. Performance of linear-space search algorithms. *Artificial Intelligence*, 79(2):241–292, 1995.