# Incremental DCOP Search Algorithms for Solving Dynamic DCOPs*

# (Extended Abstract)

William Yeoh
Computer Science Department
University of Massachusetts
Amherst, MA 01003
wyeoh@cs.umass.edu

Pradeep Varakantham
School of Information Systems
Singapore Management University
Singapore 178902
pradeepv@smu.edu.sg

Xiaoxun Sun, Sven Koenig
Computer Science Department
University of Southern California
Los Angeles, CA 90089
{xiaoxuns,skoenig}@usc.edu

## ABSTRACT

Distributed constraint optimization problems (DCOPs) are well-suited for modeling multi-agent coordination problems. However, most research has focused on developing algorithms for solving static DCOPs. In this paper, we model dynamic DCOPs as sequences of (static) DCOPs with changes from one DCOP to the next one in the sequence. We introduce the ReuseBounds procedure, which can be used by any-space ADOPT and any-space BnB-ADOPT to find cost-minimal solutions for all DCOPs in the sequence faster than by solving each DCOP individually. This procedure allows those agents that are guaranteed to remain unaffected by a change to reuse their lower and upper bounds from the previous DCOP when solving the next one in the sequence. Our experimental results show that the speedup gained from this procedure increases with the amount of memory the agents have available.

## Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed AI

## General Terms

Algorithms; Experimentation

## Keywords

ADOPT; BnB-ADOPT; DCOP; Dynamic DCOP

## 1. INTRODUCTION

Distributed constraint optimization problems (DCOPs) are problems where agents need to coordinate their value assignments to minimize the sum of the resulting constraint costs. DCOPs are well-suited for modeling multi-agent coordination problems where the interactions are primarily between subsets of agents. Most research has focused on developing algorithms for solving static DCOPs, that is, DCOPs that do not change over time. In this paper, we model *dynamic DCOPs* as sequences of (static) DCOPs with changes from one DCOP to the next one in the sequence. The objective is to determine cost-minimal solutions for all DCOPs in the sequence, which could be done with existing DCOP algorithms by solving each DCOP individually. Such a brute force approach can be sped up because it repeatedly solves DCOP subproblems that remain unaffected by the changes. We therefore introduce the ReuseBounds procedure, which allows any-space ADOPT and any-space BnB-ADOPT to reuse information gained from solving the previous DCOP when solving the next one in the sequence.

## 2. BACKGROUND

**DCOPs:** A DCOP is a tuple $\langle A, D, F \rangle$. $A = \{a_i\}_0^n$ is the finite set of agents. $D = \{d_i\}_0^n$ is the set of finite domains, where domain $d_i$ is the set of possible values for agent $a_i$. $F = \{f_i\}_0^m$ is the set of binary constraints, where each constraint $f_i : d_{i_1} \times d_{i_2} \rightarrow \mathbb{R}^+ \cup \infty$ specifies its non-negative constraint cost as a function of the values of two different agents $a_{i_1}$ and $a_{i_2}$ that share the constraint. A solution is an agent-value assignment for all agents. Its cost is the sum of the constraint costs of all constraints. Solving a DCOP optimally means finding a cost-minimal solution. DCOPs are commonly visualized as constraint graphs, whose vertices are the agents and whose edges are the constraints. Most DCOP algorithms operate on pseudo-trees, which are spanning trees of fully connected constraint graphs such that no two vertices in different subtrees of the spanning tree are connected by edges in the constraint graph.

**DDCOPs:** We define a DDCOP to be a sequence of (static) DCOPs with changes from one DCOP to the next one in the sequence. Solving a DDCOP optimally means finding a cost-minimal solution for all DCOPs in the sequence. This approach is a *reactive* approach since it does not consider future changes. The advantage of this approach is that solving DDCOPs is no harder than solving multiple DCOPs.

**DCOP Algorithms:** ADOPT [2] and BnB-ADOPT [3] transform the constraint graph to a pseudo-tree and then

*This material is based upon work supported by NSF (while Sven Koenig was serving at NSF). It is also based upon work supported by ARL/ARO under contract/grant number W911NF-08-1-0468, ONR in form of a MURI under contract/grant number N00014-09-1-1031 and DOT under contract/grant number DTFH61-11-C-00010. 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 or the U.S. government.

**Cite as:** Incremental DCOP Search Algorithms for Solving Dynamic DCOPs (Extended Abstract), W. Yeoh, P. Varakantham, X. Sun, S. Koenig, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. XXX-XXX.

| | Any-space ADOPT | | | | | | Any-space BnB-ADOPT | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cache Factor | 0.0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 | 0.0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 |
| With ReuseBounds (cycles) | 86301 | 21395 | 9207 | 5117 | 3386 | 2615 | 1653 | 1573 | 1556 | 1481 | 1427 | 1383 |
| Without ReuseBounds (cycles) | 86401 | 22096 | 9825 | 5618 | 3810 | 2976 | 1654 | 1578 | 1577 | 1573 | 1570 | 1568 |
| Speedup (%) | 0.12 | 3.17 | 6.29 | 8.92 | 11.13 | 12.13 | 0.06 | 0.32 | 1.33 | 5.84 | 9.10 | 11.80 |

**Table 1: Experimental Results**

search for a cost-minimal solution. ADOPT uses *best-first search* while BnB-ADOPT uses *depth-first branch-and-bound search*. For ADOPT and BnB-ADOPT, each agent $a_i$ maintains at all times *one* context $X^{a_i}$ and lower bounds $LB^{a_i}_{X^{a_i}}(d)$ and upper bounds $UB^{a_i}_{X^{a_i}}(d)$ for all values $d \in d_i$ and the context $X^{a_i}$. For any-space ADOPT and any-space BnB-ADOPT, each agent maintains *multiple* contexts and the bounds for these contexts [4]. A context is the assumption of agent $a_i$ on the agent-value assignments of all of its ancestors in the pseudo-tree. The bounds $LB^{a_i}_{X^{a_i}}(d)$ and $UB^{a_i}_{X^{a_i}}(d)$ are bounds on the optimal cost $OPT^{a_i}_{X^{a_i}}(d)$, which is the cost of a cost-minimal solution in case agent $a_i$ takes on value $d$ and each of its ancestors takes on its respective value in $X^{a_i}$. The optimal cost $OPT^{a_i}_{X^{a_i}}(d)$ is defined by

$$OPT^{a_i}_{X^{a_i}}(d) = \delta^{a_i}_{X^{a_i}} + \sum_{c \in C(a_i)} OPT^{c}_{X^{a_i} \cup (a_i,d)} \quad (1)$$

$$OPT^{a_i}_{X^{a_i}} = \min_{d \in d_i} OPT^{a_i}_{X^{a_i}}(d) \quad (2)$$

where $\delta^{a_i}_{X^{a_i}}$ is the sum of the costs of all constraints between agents whose values are defined in context $X^{a_i}$, and $C(a_i)$ is the set of children of agent $a_i$ in the pseudo-tree.

# 3. REUSEBOUNDS PROCEDURE

When solving the next DCOP in the sequence, one constructs the pseudo-tree for the next DCOP, uses the ReuseBounds procedure to identify the lower and upper bounds that were cached by any-space ADOPT or any-space BnB-ADOPT when solving the previous DCOP and can be reused for the next DCOP, initializes the other bounds and finally uses any-space ADOPT or any-space BnB-ADOPT to solve the next DCOP optimally. The ReuseBounds procedure identifies affected agents, which are those agents whose optimal costs can be different for the previous and next DCOPs. They have one or more of the following properties:

- **Property 1:** Agent $a_i$ shares an added constraint, deleted constraint or constraint with changed constraint costs with another agent. If the agent shares the constraint with a descendant, then it is an affected agent (see Property 3). If the agent shares the constraint with an ancestor, then the cost $\delta^{a_i}_{X^{a_i}}(d)$ for some value $d$ and context $X^{a_i}$ can change, which in turn can change its optimal cost $OPT^{a_i}_{X^{a_i}}(d)$ (see Equation 1).
- **Property 2:** Agent $a_i$ has a different set of children $C(a_i)$ in the previous and next DCOPs, which can change its optimal cost $OPT^{a_i}_{X^{a_i}}(d)$ (see Equation 1).
- **Property 3:** Agent $a_i$ has a descendant $a_j$ that is an affected agent, which means that the optimal cost $OPT^{a_j}_{X^{a_j}}(d)$ for some value $d$ and context $X^{a_j}$ can change, which in turn can change the optimal cost $OPT^{a_j}_{X^{a_j}}$ (see Equation 2) and thus also the optimal cost $OPT^{a_k}_{X^{a_k}}(d')$ of its parent $a_k$ (see Equation 1), and so on. Therefore, the optimal costs of all ancestors of agent $a_j$ (including the one of agent $a_i$) can change.

The affected agents cannot reuse their lower and upper bounds for the next DCOP because the optimal costs can be different for the previous and next DCOPs and the bounds on the optimal costs of the previous DCOP might thus no longer be bounds on the optimal costs of the next DCOP.

# 4. EXPERIMENTAL RESULTS

We now compare the runtimes of any-space ADOPT and any-space BnB-ADOPT with and without the ReuseBounds procedure. We use the distributed DFS algorithm with the max-degree heuristic [1] to construct the pseudo-trees. We measure the runtimes in cycles [2], vary the amount of memory of each agent with the cache factor metric [4] and use the MaxEffort and MaxPriority caching schemes [4] for any-space ADOPT and any-space BnB-ADOPT, respectively. We consider the following changes: (1) change in the costs of a random constraint, (2) removal of a random constraint, (3) addition of a random constraint, (4) removal of a random agent and (5) addition of a random agent. We averaged our experimental results over 50 DDCOP instances with the above five changes in random order and used a randomly generated graph coloring problem of density 2, domain cardinality 5 and constraint costs in the range of 0 to 10,000 as the initial DCOP for each DDCOP.

Table 1 shows our experimental results. The runtimes of both DCOP algorithms decrease as the cache factor increases. The reason for this behavior is that they reduce the amount of duplicated search effort when they cache more information [4]. The runtimes of both DCOP algorithms are smaller with the ReuseBounds procedure than without it, and the speedup increases as the cache factor increases. The reason for this behavior is that the unaffected agents can cache and reuse more lower and upper bounds from the previous DCOPs as the cache factor increases.

# 5. REFERENCES


[1] Y. Hamadi, C. Bessière, and J. Quinqueton. Distributed intelligent backtracking. In *ECAI*, pages 219–223, 1998.

[2] P. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005.

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

[4] W. Yeoh, P. Varakantham, and S. Koenig. Caching schemes for DCOP search algorithms. In *AAMAS*, pages 609–616, 2009.