Generalizing ADOPT and BnB-ADOPT

Patricia Gutierrez[1], **Pedro Meseguer**[1] and **William Yeoh**[2]
[1]IIIA - CSIC, Universitat Autònoma de Barcelona, Bellaterra, Spain
[2]Computer Science Department, University of Massachusetts, Amherst, MA, USA
{patricia|pedro}@iiia.csic.es wyeoh@cs.umass.edu

Abstract

ADOPT and BnB-ADOPT are two optimal DCOP search algorithms that are similar except for their search strategies: the former uses best-first search and the latter uses depth-first branch-and-bound search. In this paper, we present a new algorithm, called ADOPT(k), that generalizes them. Its behavior depends on the k parameter. It behaves like ADOPT when $k = 1$, like BnB-ADOPT when $k = \infty$ and like a hybrid of ADOPT and BnB-ADOPT when $1 < k < \infty$. We prove that ADOPT(k) is a correct and complete algorithm and experimentally show that ADOPT(k) outperforms ADOPT and BnB-ADOPT on several benchmarks across several metrics.

1 Introduction

Distributed Constraint Optimization Problems (DCOPs) [Modi *et al.*, 2005; Petcu and Faltings, 2005] are well-suited for modeling multi-agent coordination problems where interactions are primarily between subsets of agents, such as meeting scheduling [Maheswaran *et al.*, 2004], sensor network [Farinelli *et al.*, 2008] and coalition structure generation [Ueda *et al.*, 2010] problems. DCOPs involve a finite number of agents, variables and binary cost functions. The cost of an assignment of a subset of variables is the evaluation of all cost functions on that assignment. The goal is to find a complete assignment with minimal cost. Researchers have proposed several distributed search algorithms to solve DCOPs optimally. They include ADOPT [Modi *et al.*, 2005], which uses best-first search, and BnB-ADOPT [Yeoh *et al.*, 2010], which uses depth-first branch-and-bound search.

We present a new algorithm, called ADOPT(k), that generalizes ADOPT and BnB-ADOPT. Its behavior depends on the k parameter. It behaves like ADOPT when $k = 1$, like BnB-ADOPT when $k = \infty$ and like a hybrid of ADOPT and BnB-ADOPT when $1 < k < \infty$. The main difference between ADOPT(k) and its predecessors is the condition by which an agent changes its value. While an agent in ADOPT changes its value when another value is more promising by at least 1 unit, an agent in ADOPT(k) changes its value when another value is more promising by at least k units. When $k = \infty$, like agents in BnB-ADOPT, an agent in ADOPT(k) changes its value when the optimal solution for that value is provably no better than the best solution found so far. We prove that ADOPT(k) is correct and complete and experimentally show that ADOPT(k) outperforms ADOPT and BnB-ADOPT on several benchmarks across several metrics.

2 Preliminaries

In this section, we formally define DCOPs and summarize ADOPT and BnB-ADOPT.

2.1 DCOP

A DCOP is defined by $\langle \mathcal{X}, \mathcal{D}, \mathcal{F}, \mathcal{A}, \alpha \rangle$, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of variables; $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of finite domains, where D_i is the domain of variable x_i; $\mathcal{F}$ is a set of binary cost functions, where each cost function $F_{ij} : D_i \times D_j \mapsto \mathbb{N} \cup \{0, \infty\}$ specifies the cost of each combination of values of variables x_i and x_j; $\mathcal{A} = \{a_1, \ldots, a_p\}$ is a set of agents and $\alpha : \mathcal{X} \rightarrow \mathcal{A}$ maps each variable to one agent. We assume that each agent has only one variable mapped to it, and we thus use the terms variable and agent interchangeably. The cost of an assignment of a subset of variables is the evaluation of all cost functions on that assignment. Agents communicate through messages, which are never lost and delivered in the order that they were sent.

A *constraint graph* visualizes a DCOP instance, where nodes in the graph correspond to variables and edges connect pairs of variables appearing in the same cost function. A *depth-first search* (DFS) *pseudo-tree* arrangement has the same nodes and edges as the constraint graph and satisfies that (i) there is a subset of edges, called *tree edges*, that form a rooted tree and (ii) two variables in a cost function appear in the same branch of that tree. The other edges are called *backedges*. Tree edges connect parent-child nodes, while backedges connect a node with its pseudo-parents and its pseudo-children. DFS pseudo-trees can be constructed using distributed DFS algorithms [Hamadi *et al.*, 1998].

2.2 ADOPT

ADOPT [Modi *et al.*, 2005] is a distributed search algorithm that solves DCOPs optimally. ADOPT first constructs a DFS pseudo-tree, after which each agent knows its parent, pseudo-parents, children and pseudo-children. Each agent x_i maintains: its current value d_i; its current context X_i, which is its assumption on the current value of its ancestors; the lower and upper bounds LB_i and UB_i, which are bounds on the optimal cost OPT_i given that its ancestors take on their respective values in X_i; the lower and upper bounds $LB_i(d)$ and $UB_i(d)$ for all values $d \in D_i$, which are bounds on the optimal costs $OPT_i(d)$ given that x_i takes on the value d and its ancestors take on their respective values in X_i; the lower and upper bounds $lb_i^c(d)$ and $ub_i^c(d)$ for all values $d \in D_i$ and children x_c, which are its assumption on the bounds LB_c and UB_c of its children x_c with context $X_i \cup (x_i, d)$; and the thresholds TH_i and $th_i^c(d)$ for all values $d \in D_i$ and children x_c, which are used to speed up the solution reconstruction process. The optimal costs are calculated using:

$$OPT_i(d) = \delta_i(d) + \sum_{x_c \in C_i} OPT_c \quad (1) \qquad OPT_i = \min_{d \in D_i} OPT_i(d) \quad (2)$$

for all values $d \in D_i$, where C_i is the set of children of agent x_i and $\delta_i(d)$ is the sum of the costs of all cost functions between x_i and its ancestors given that x_i takes on the value d and the ancestors take on their respective values in X_i.

ADOPT agents use four types of messages: VALUE, COST, THRESHOLD and TERMINATE. At the start, each agent x_i initializes its current context X_i to $\emptyset$, lower and upper bounds $lb_i^c(d)$ and $ub_i^c(d)$ to user-provided heuristic values $h_i^c(d)$ and ∞, respectively. For all values $d \in D_i$ and all children x_c, x_i calculates the remaining lower and upper bounds and takes on its best value using:

$$\delta_i(d) = \sum_{(x_j, d_j) \in X_i} F_{ij}(d, d_j) \quad (3)$$

$$LB_i(d) = \delta_i(d) + \sum_{x_c \in C_i} lb_i^c(d) \quad (4) \qquad LB_i = \min_{d \in D_i} \{LB_i(d)\} \quad (5)$$

$$UB_i(d) = \delta_i(d) + \sum_{x_c \in C_i} ub_i^c(d) \quad (6) \qquad UB_i = \min_{d \in D_i} \{UB_i(d)\} \quad (7)$$

$$d_i = \arg\min_{d \in D_i} \{LB_i(d)\} \quad (8)$$

x_i sends a VALUE message containing its value d_i to its children and pseudo-children. It also sends a COST message containing its context X_i and its bounds LB_i and UB_i to its parent. Upon receipt of a VALUE message, if its current context X_i is compatible with the value in the VALUE message, it updates its context to reflect the new value of its ancestor and reinitializes its lower and upper bounds $lb_i^c(d)$ and $ub_i^c(d)$. Contexts are *compatible* iff they agree on common agent-value pairs. Upon receipt of a COST message from child x_c, if its current context X_i is compatible with the context in the message, then it updates its lower and upper bounds $lb_i^c(d)$ and $ub_i^c(d)$ to the lower and upper bounds in the message, respectively. Otherwise, the COST message is discarded. After processing either message, it recalculates the remaining lower and upper bounds and takes on its best value using the above equations and sends VALUE and COST messages. This process repeats until the root agent x_r reaches the termination condition $LB_r = UB_r$, which means that it has found the optimal cost. It then sends a TERMINATE message to each of its children and terminate. Upon receipt of a TERMINATE message, each agent does the same.

Due to memory limitations, each agent x_i can only store lower and upper bounds for *one* context. Thus, it reinitializes its bounds each time the context changes. If its context changes back to a previous one, it has to update its bounds from scratch. ADOPT optimizes this process by having the parent of x_i send x_i the lower bound computed earlier as threshold TH_i in a THRESHOLD message. This optimization changes the condition for which an agent changes its value. Each agent x_i now changes its value d_i only when $LB_i(d_i) \geq TH_i$.

2.3 BnB-ADOPT

BnB-ADOPT [Yeoh *et al.*, 2010] shares most of the data structures and messages of ADOPT. The main difference is their search strategies. ADOPT employs a *best-first search* strategy while BnB-ADOPT employs a *depth-first branch-and-bound search* strategy. This difference in search strategies is reflected by how the agents change their values. While each agent x_i in ADOPT eagerly takes on the value that minimizes its lower bound $LB_i(d)$, each agent x_i in BnB-ADOPT changes its value only when it is able to determine that the optimal solution for that value is provably no better than the best solution found so far for its current context. In other words, when $LB(d_i) \geq UB_i$ for its current value d_i.

The role of thresholds in the two algorithms is also different. As described earlier, each agent in ADOPT uses thresholds to store the lower bound previously computed for its current context such that it can reconstruct the partial solution more efficiently. On the other hand, each agent in BnB-ADOPT uses thresholds to store the cost of the best solution found so far for all contexts and uses them to change its values more efficiently. Therefore, each agent x_i now changes its value d_i only when $LB_i(d_i) \geq \min\{TH_i, UB_i\}$.

BnB-ADOPT also has several optimizations that can be applied to ADOPT: (1) Agents in BnB-ADOPT processes messages differently compared to agents in ADOPT. Each agent in ADOPT updates its lower and upper bounds and takes on a new value, if necessary, after each message that it receives. On the other hand, each agent in BnB-ADOPT does so only after it processes all its messages. (2) BnB-ADOPT includes thresholds in VALUE messages such that THRESHOLD messages are no longer required. (3) BnB-ADOPT includes a time stamp for each value in contexts such that their recency can be compared.[1]

Researchers recently observed that some of the messages in BnB-ADOPT are redundant and thus introduced BnB-ADOPT$^+$, an extension of BnB-ADOPT without most of the redundant messages [Gutierrez and Meseguer, 2010b]. BnB-ADOPT$^+$ is shown to outperform BnB-ADOPT in a variety of metrics, especially in the number of messages sent. Researchers have also applied the same message reduction techniques to extend ADOPT to ADOPT$^+$ [Gutierrez and Meseguer, 2010a]. However, it is not as competitive since ADOPT has fewer redundant messages than BnB-ADOPT.

3 ADOPT(k)

Each agent in ADOPT always changes its value to the most promising value. This strategy requires the agent to repeatedly reconstruct partial solutions that it previously found,

[1]The first two optimizations were in the implementation of ADOPT [Yin, 2008] but not in the publication [Modi *et al.*, 2005].

which can be computationally inefficient. On the other hand, each agent in BnB-ADOPT changes its value only when the optimal solution for that value is provably no better than the best solution found so far, which can be computationally inefficient if the agent takes on bad values before good values. Therefore, we believe that there should be a good trade off between the two extremes, where an agent keeps its value longer than it otherwise would as an ADOPT agent and shorter than it otherwise would as a BnB-ADOPT agent.

With this idea in mind, we developed ADOPT(k), which generalizes ADOPT and BnB-ADOPT. It behaves like ADOPT when $k = 1$, like BnB-ADOPT when $k = \infty$ and like a hybrid of ADOPT and BnB-ADOPT when $1 < k < \infty$. ADOPT(k) uses mostly identical data structures and messages as ADOPT and BnB-ADOPT. Each agent x_i in ADOPT(k) maintains two thresholds, TH_i^A and TH_i^B, which are the thresholds in ADOPT and BnB-ADOPT, respectively. They are initialized and updated in the same way as in ADOPT and BnB-ADOPT, respectively.

The main difference between ADOPT(k) and its predecessors is the condition by which an agent changes its value. Each agent x_i in ADOPT(k) changes its value d_i when $LB_i(d_i) > TH_i^A + (k-1)$ or $LB_i(d_i) \geq \min\{TH_i^B, UB_i\}$. If $k = 1$, then the first condition degenerates to $LB_i(d_i) > TH_i^A$, which is the condition for agents in ADOPT. The agents use the second condition, which remains unchanged, to determine if the optimal solution for their current value is provably no better than the best solution found so far. If $k = \infty$, then the first condition is always false and the second condition, which remains unchanged, is the condition for agents in BnB-ADOPT. If $1 < k < \infty$, then each agent in ADOPT(k) keeps its current value until the lower bound of that value is at least k units larger than the lower bound of the most promising value, at which point it takes on the most promising value.

3.1 Pseudocode

Figures 1 and 2 show the pseudocode of ADOPT(k), where x_i is a generic agent, C_i is its set of children, PC_i is its set of pseudo-children and SCP_i is the set of agents that are either ancestors of x_i or parent and pseudo-parents of either x_i or its descendants. The pseudocode uses the predicate **Compatible**(X,X') to determine if two contexts X and X' are compatible and the procedure **PriorityMerge**(X,X') to replace the values of agents in context X' with more recent values, if available, of the same agents in context X (see [Yeoh *et al.*, 2010] for more details). The pseudocode is similar to ADOPT's pseudocode with the following changes:

- The pseudocode includes the optimizations described in Section 2.3 that was presented for BnB-ADOPT but can be applied to ADOPT (Lines 03, 08-12, 35-36 and 40-41).
- In ADOPT, the **MaintainThresholdInvariant()**, **MaintainChildThresholdInvariant()** and **MaintainAllocationInvariant()** procedures are called after each message is processed. Here, they are called in the **Backtrack()** procedure (Lines 28 and 38-39). The invariants are maintained only after all incoming messages are processed.
- In addition to TH_i^A, each agent maintains TH_i^B. It is ini-

```
01 procedure Start()
02 $X_i := \{(x_p, \textbf{ValInit}(x_p), 0) \mid x_p \in SCP_i\}$;
03 $ID_i := 0$;
04 forall $x_c \in C_i$ and $d \in D_i$ InitChild($x_c$, $d$);
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();

13 procedure InitChild($x_c$, $d$)
14 $lb_i^c(d) := h_i^c(d)$;
15 $ub_i^c(d) := \infty$;
16 $th_i^c(d) := lb_i^c(d)$;

17 procedure InitSelf()
18 $d_i := \arg\min_{d \in D_i}\{\delta_i(d) + \sum_{x_c \in C_i} lb_i^c(d)\}$;
19 $ID_i := ID_i + 1$;
20 $TH_i^A := \min_{d \in D_i}\{\delta_i(d) + \sum_{x_c \in C_i} lb_i^c(d)\}$;
21 $TH_i^B := \infty$;

22 procedure Backtrack()
23 forall $d \in D_i$
24     $LB_i(d) := \delta_i(d) + \sum_{x_c \in C_i} lb_i^c(d)$;
25     $UB_i(d) := \delta_i(d) + \sum_{x_c \in C_i} ub_i^c(d)$;
26 $LB_i := \min_{d \in D_i}\{LB_i(d)\}$;
27 $UB_i := \min_{d \in D_i}\{UB_i(d)\}$;
28 MaintainThresholdInvariant();
29 if ($TH_i^A = UB_i$)
30     $d_i := \arg\min_{d \in D_i}\{UB_i(d)\}$
31 else if ($LB_i(d_i) > TH_i^A + (k-1)$)
32     $d_i := \arg\min_{d \in D_i | LB_i(d) = LB_i}\{UB_i(d)\}$
33 else if ($LB_i(d_i) \geq \min\{TH_i^B, UB_i\}$)
34     $d_i := \arg\min_{d \in D_i | LB_i(d) = LB_i}\{UB_i(d)\}$
35 if (a new $d_i$ has been chosen)
36     $ID_i := ID_i + 1$;
37 MaintainCurrentValueThresholdInvariant();
38 MaintainChildThresholdInvariant();
39 MaintainAllocationInvariant();
40 Send(VALUE, $x_i, d_i, ID_i, th_i^c(d_i), \min(TH_i^B, UB_i) - \delta_i(d_i)$
       $- \sum_{x_{c'} \in C_i | x_{c'} \neq x_c} lb_i^{c'}(d_i)$) to each $x_c \in C_i$;
41 Send(VALUE, $x_i, d_i, ID_i, \infty, \infty$) to each $x_c \in PC_i$;
42 if ($TH_i^A = UB_i$)
43     if ($x_i$ is root or termination message received)
44         Send(TERMINATE) to each $x_c \in C_i$;
45         terminate execution;
46 Send(COST, $x_i, X_i, LB_i, UB_i$) to parent;

47 procedure When Received(TERMINATE)
48 record termination message received;

49 procedure When Received(VALUE, $x_p, d_p, ID_p, TH_p^A, TH_p^B$)
50 $X' := X_i$;
51 PriorityMerge($(x_p, d_p, ID_p)$, $X_i$);
52 if (!Compatible($X'$, $X_i$))
53     forall $x_c \in C_i$ and $d \in D_i$
54         if ($x_p \in SCP_c$)
55             InitChild($x_c$, $d$);
56     InitSelf();
57 if ($x_p$ is parent)
58     $TH_i^A := TH_p^A$;
59     $TH_i^B := TH_p^B$;

60 procedure When Received(COST, $x_c, X_c, LB_c, UB_c$)
61 $X' := X_i$;
62 PriorityMerge($X_c$, $X_i$);
63 if (!Compatible($X'$, $X_i$))
64     forall $x_c \in C_i$ and $d \in D_i$
65         if (!Compatible($\{(x_p, d_p, ID_p) \in X' \mid x_p \in SCP_c\}$,$X_i$))
66             InitChild($x_c$, $d$);
67 if (Compatible($X_c$, $X_i$))
68     $lb_i^c(d) := \max\{lb_i^c(d), LB_c\}$ for the unique $(a', d, ID) \in X_c$ with $a' = a$;
69     $ub_i^c(d) := \min\{ub_i^c(d), UB_c\}$ for the unique $(a', d, ID) \in X_c$ with $a' = a$;
70 if (!Compatible($X'$, $X_i$))
71     InitSelf();
```

Figure 1: Pseudocode of ADOPT(k) (1)

```
72 procedure MaintainChildThresholdInvariant()
73 forall $x_c \in C_i$ and $d \in D_i$
74     while($th_i^c(d) < lb_i^c(d)$)
75         $th_i^c(d) := th_i^c(d) + \epsilon$;
76 forall $c \in C_i$ and $d \in D_i$
77     while($th_i^c(d) > ub_i^c(d)$)
78         $th_i^c(d) := th_i^c(d) - \epsilon$;
79 procedure MaintainThresholdInvariant()
80 if ($TH_i^A < LB_i$)
81     $TH_i^A = LB_i$;
82 if ($TH_i^A > UB_i$)
83     $TH_i^A = UB_i$;
84 procedure MaintainCurrentValueThresholdInvariant()
85 $TH_i^A(d_i) := TH_i^A$;
86 if ($TH_i^A(d_i) < LB_i(d_i)$)
87     $TH_i^A(d_i) = LB_i(d_i)$;
88 if ($TH_i^A(d_i) > UB_i(d_i)$)
89     $TH_i^A(d_i) = UB_i(d_i)$;
90 procedure MaintainAllocationInvariant()
91 while($TH_i^A(d_i) > \delta_i(d_i) + \sum_{x_c \in C_i} th_i^c(d_i)$)
92     $th_i^{c'}(d_i) := th_i^{c'}(d_i) + \epsilon$ for any $x_{c'} \in C_i$ with $ub_i^{c'}(d_i) > th_i^{c'}(d_i)$;
93 while($TH_i^A(d_i) < \delta_i(d_i) + \sum_{x_c \in C_i} th_i^c(d_i)$)
94     $th_i^{c'}(d_i) := th_i^{c'}(d_i) - \epsilon$ for any $x_{c'} \in C_i$ with $lb_i^{c'}(d_i) < th_i^{c'}(d_i)$;
```

Figure 2: Pseudocode of ADOPT(k) (2)

tialized, propagated and used in the same way as in BnB-ADOPT (Lines 21, 33-34, 40 and 59).

- The condition by which each agent x_i changes its value is now $LB_i(d_i) > TH_i^A + (k-1)$ or $LB_i(d_i) \geq \min\{TH_i^B, UB_i\}$ (Lines 31 and 33). Thus, the agent keeps its value until the lower bound of that value is k units larger than the lower bound of the most promising value or the optimal solution for that value is provably no better than the best solution found so far.

- In ADOPT, the **MaintainAllocationInvariant()** procedure ensures that the invariant $TH_i^A = \sum_{x_c \in C_i} TH_c^A$ always hold. This procedure assumes that $TH_i^A \geq LB_i(d_i)$ for the current value d_i of agent x_i, which is always true since the agent would change its value otherwise. However, this assumption is no longer true in ADOPT(k). Therefore, the pseudocode includes a new threshold $TH_i^A(d_i)$, which is set to TH_i^A and updated such that it satisfies the invariant $LB_i(d_i) \leq TH_i^A(d_i) \leq UB_i(d_i)$ in the **MaintainCurrentValueThresholdInvariant()** procedure (Lines 84-89). This new threshold then replaces TH_i^A in the **MaintainAllocationInvariant()** procedure (Lines 91 and 93).

3.2 Correctness and Completeness

The proofs for the following lemmata and theorem closely follow those in [Modi *et al.*, 2005; Yeoh *et al.*, 2010]. We thus only provide proof sketches.

Lemma 1 *For all agents x_i and all values $d \in D_i$, $LB_i \leq OPT_i \leq UB_i$ and $LB_i(d) \leq OPT_i(d) \leq UB_i(d)$ at all times.*

Proof sketch: We prove the lemma by induction on the depth of the agent in the pseudo-tree. It is clear that for each leaf agent x_i, $LB_i(d) = OPT_i(d) = UB_i(d)$ for all values $d \in D_i$ (Lines 24-25 and Eq. 1). Furthermore,

$$\begin{aligned}
LB_i &= \min_{d \in D_i}\{LB_i(d)\} && \text{(Line 26)}\\
&= \min_{d \in D_i}\{OPT_i(d)\} && \text{(see above)}\\
&= OPT_i && \text{(Eq. 2)}\\
UB_i &= \min_{d \in D_i}\{UB_i(d)\} && \text{(Line 27)}\\
&= \min_{d \in D_i}\{OPT_i(d)\} && \text{(see above)}\\
&= OPT_i && \text{(Eq. 2)}
\end{aligned}$$

So, the lemma holds for each leaf agent. Assume that it holds for all agents at depth q in the pseudo-tree. For all agents x_i at depth $q-1$,

$$\begin{aligned}
LB_i(d) &= \delta_i(d) + \sum_{x_c \in C_i} LB_c && \text{(Lines 24 and 68)}\\
&\leq \delta_i(d) + \sum_{x_c \in C_i} OPT_c && \text{(induction ass.)}\\
&= OPT_i && \text{(Eq. 1)}\\
UB_i(d) &= \delta_i(d) + \sum_{x_c \in C_i} UB_c && \text{(Line 25 and 69)}\\
&\geq \delta_i(d) + \sum_{x_c \in C_i} OPT_c && \text{(induction ass.)}\\
&= OPT_i && \text{(Eq. 1)}
\end{aligned}$$

for all values $d \in D_i$. The proof for $LB_i \leq OPT_i \leq UB_i$ is similar to the proof for the base case. Thus, the lemma holds. ■

Lemma 2 *For all agents x_i, if the current context of x_i is fixed, then $LB_i = TH_i^A = UB_i$ will eventually occur.*

Proof sketch: We prove the lemma by induction on the depth of the agent in the pseudo-tree. The lemma holds for leaf agents x_i since $LB_i = UB_i$ (see proof for the base case of Lemma 1) and $LB_i \leq TH_i^A \leq UB_i$ (lines 79-83). Assume that the lemma holds for all agents at depth q in the pseudo-tree. For all agents x_i at depth $q-1$,

$$\begin{aligned}
LB_i &= \min_{d \in D_i}\{\delta_i(d) + \sum_{x_c \in C_i} lb_i^c(d)\} && \text{(Lines 24 and 26)}\\
&= \min_{d \in D_i}\{\delta_i(d) + \sum_{x_c \in C_i} LB_c\} && \text{(Line 68)}\\
&= \min_{d \in D_i}\{\delta_i(d) + \sum_{x_c \in C_i} UB_c\} && \text{(induction ass.)}\\
&= \min_{d \in D_i}\{\delta_i(d) + \sum_{x_c \in C_i} ub_i^c(d)\} && \text{(Line 69)}\\
&= UB_i && \text{(Line 25 and 27)}
\end{aligned}$$

Additionally, $LB_i \leq TH_i^A \leq UB_i$ (Lines 79-83). Therefore, $LB_i = TH_i^A = UB_i$. ■

Lemma 3 *For all agents x_i, $TH_i^A(d) = TH_i^A$ on termination.*

Proof sketch: Each agent x_i terminates when $TH_i^A = UB_i$ (Line 42). After $TH_i^A(d_i)$ is set to TH_i^A for the current value d_i of x_i (Line 85) in the last execution of the MaintainCurrentValueThresholdInvariant() procedure,

$$\begin{aligned}
TH_i^A &= UB_i && \text{(Line 42)}\\
&= UB_i(d_i) && \text{(Lines 29-30)}\\
&\geq LB_i(d_i) && \text{(Lemma 1)}
\end{aligned}$$

Thus, $LB_i(d_i) \leq TH_i^A = UB_i$ and $TH_i^A(d_i)$ is not set to a different value later (Lines 86-89). Then, $TH_i^A(d) = TH_i^A$ on termination. ■

Theorem 1 *For all agents x_i, $TH_i^A = OPT_i$ on termination.*

Proof sketch: We prove the theorem by induction on the depth of the agent in the pseudo-tree. The theorem holds for the root agent x_i since $TH_i^A = UB_i$ on termination (Lines 42-45), $TH_i^A = LB_i$ at all times (Lines 20 and 28), and $LB_i \leq OPT_i \leq UB_i$ (Lemma 1). Assume that the theorem holds for all agents at depth q in the pseudo-tree. We now prove that the theorem holds for all agents at depth $q+1$. Let x_p be an arbitrary agent at depth q in the pseudo-tree and d_p is its current value on termination. Then,

$$\begin{aligned}
\sum_{x_c \in C_p} ub_p^c(d_p) &= UB_p(d_p) - \delta_p(d_p) && \text{(Line 25)}\\
&= UB_p - \delta_p(d_p) && \text{(Lines 29-30)}\\
&= TH_p^A - \delta_p(d_p) && \text{(Lines 42-45)}\\
&= TH_p^A(d_p) - \delta_p(d_p) && \text{(Lemma 3)}\\
&= \sum_{x_c \in C_p} th_p^c(d_p) && \text{(Lines 91-94)}
\end{aligned}$$

Thus, $\sum_{x_c \in C_p} ub_p^c(d_p) = \sum_{x_c \in C_p} th_p^c(d_p)$. Furthermore, for all agents $x_c \in C_p$, $th_p^c(d_p) \leq ub_p^c(d_p)$ (Lines 77-78). Combining the inequalities, we get $th_p^c(d_p) = ub_p^c(d_p)$. Additionally, $TH_c^A = th_p^c(d_p)$ (Lines 57-58) and $UB_c = ub_p^c(d_p)$ (Line 69). Therefore, $TH_c^A = UB_c$. Next,

$$\begin{aligned}
\sum_{x_c \in C_p} OPT_c &= OPT_p - \delta_p(d_p) && \text{(Eq. 1)}\\
&= TH_p^A - \delta_p(d_p) && \text{(induction ass.)}\\
&= TH_p^A(d_p) - \delta_p(d_p) && \text{(Lemma 3)}\\
&= \sum_{x_c \in C_p} th_p^c(d_p) && \text{(Lines 91-94)}\\
&= \sum_{x_c \in C_p} TH_c^A && \text{(Lines 57-58)}
\end{aligned}$$

Thus, $\sum_{x_c \in C_p} OPT_c = \sum_{x_c \in C_p} TH_c^A = \sum_{x_c \in C_p} UB_c$ (see above). Furthermore, for all agents $x_c \in C_p$, $OPT_c \leq UB_c$ (Lemma 1). Combining the inequalities, we get $OPT_c = UB_c$. Therefore, $TH_c^A = UB_c = OPT_c$. ■

4 Experimental Results

We compare ADOPT$^+(k)$ to ADOPT$^+$ and BnB-ADOPT$^+$. ADOPT$^+(k)$ is an optimized version of ADOPT(k) with the message reduction techniques used by ADOPT$^+$ and BnB-ADOPT$^+$. All the algorithms use the DP2 heuristic values [Ali *et al.*, 2005]. We measure runtimes in (synchronous) cycles [Modi *et al.*, 2005] and non-concurrent constraint checks (NCCCs) [Meisels *et al.*, 2002], and we measure the network load in the number of VALUE and COST messages sent.[2] We do not report the number of TERMINATE messages sent because every algorithm sends the same number, namely $|\mathcal{X}| - 1$. Also, we report the trivial upper bound *UB* as the sum of the maximums over cost functions. Table 1 shows the results. Due to space constraints, we omit ADOPT$^+$ in Tables 1(a) and 1(b) since BnB-ADOPT$^+$ performs better than ADOPT$^+$ across all metrics.

Table 1(a) shows the results on random binary DCOP instances with 10 variables of domain size 10. The costs are randomly chosen over the range $\langle 1000, \ldots, 2000 \rangle$. We impose $n(n-1)/2 * p_1$ cost functions, where n is the number of variables and p_1 is the network connectivity. We vary p_1 from 0.5 to 0.8 in 0.1 increments and average our results over 50 instances for each value of p_1. The table shows that ADOPT$^+(k)$ requires a large number of messages, cycles and NCCCs when k is small. These numbers decrease as k increases until a certain point where they increase again. For the best value of k, ADOPT$^+(k)$ performs significantly better than BnB-ADOPT$^+$ across all metrics.

Table 1(b) shows the results on sensor network instances from a publicly available repository [Yin, 2008]. We use instances from all four available topologies and average our results over 30 instances for each topology. We observe the same trend as in Table 1(a) but only report the results for the best value of k due to space constraints.

Lastly, Table 1(c) shows the results on sensor network instances of 100 variables arranged into a chain following the [Maheswaran *et al.*, 2004] formulation. All the variables have a domain size of 10. The cost of hard constraints is 1,000,000. The cost of soft constraints is randomly chosen over the range $\langle 0, \ldots, 200 \rangle$. Additionally, we use discounted heuristic values, which we obtain by dividing the DP2 heuristic values by two, to simulate problems where well informed heuristics are not available due to privacy reasons. We average our results over 30 instances. The table shows that ADOPT$^+$ terminates earlier than BnB-ADOPT$^+$ but sends more messages. When $k = 1$, the results for ADOPT$^+(k)$ and ADOPT$^+$ are almoust the same. Agents in ADOPT$^+(k)$ sends more VALUE messages because they need to send VALUE messages when TH_p^B changes even if TH_p^A remains unchanged. These additional messages then trigger the need for more constraint checks. Agents in ADOPT$^+$ do not need to send VALUE messages in such a case. We observe that as k increases, the runtime of ADOPT$^+(k)$ increases but the number of messages sent decreases. Therefore, ADOPT$^+(k)$ provides a good mechanism for balancing the tradeoff between runtime and network load.

5 Conclusions

We introduced ADOPT(k), which generalizes ADOPT and BnB-ADOPT. The behavior of ADOPT(k) depends on the parameter k. It behaves like ADOPT when $k = 1$, like BnB-ADOPT when $k = \infty$ and like a hybrid of the two algorithms when $1 < k < \infty$. Our experimental results show that ADOPT(k) can outperform ADOPT and BnB-ADOPT in terms of runtime and network load on random DCOP instances and sensor network instances. Additionally, ADOPT(k) provides a good mechanism for balancing the tradeoff between runtime and network load. It is future work to better understand the characteristics of ADOPT(k) such that the best value of k can be chosen automatically.

[2] We differentiate them because the size of VALUE messages is $O(1)$ and the size of COST messages is $O(|\mathcal{X}|)$.

p_1	Trivial *UB*	Algorithm	Total Msgs	VALUE	COST	Cycles	NCCCs
0.5	45,766	BnB-ADOPT$^+$	262,812	131,785	131,009	23,646	5,353,423
		ADOPT$^+$(k = 1,000)	413,711	225,788	187,905	34,402	8,491,909
		ADOPT$^+$(k = 4,000)	**197,342**	**109,111**	**88,212**	**17,100**	**3,969,104**
		ADOPT$^+$(k = 6,000)	197,486	109,193	88,275	17,117	3,972,960
0.6	53,722	BnB-ADOPT$^+$	1,017,939	500,514	517,407	99,969	26,191,249
		ADOPT$^+$(k = 1,000)	1,864,165	1,019,709	844,438	160,365	45,101,673
		ADOPT$^+$(k = 4,500)	**701,374**	**387,658**	**313,697**	**62,977**	**16,454,689**
		ADOPT$^+$(k = 6,000)	701,529	387,742	313,768	62,994	16,459,453
0.7	63,654	BnB-ADOPT$^+$	3,716,766	1,825,332	1,891,416	387,744	116,050,941
		ADOPT$^+$(k = 1,000)	6,846,289	3,809,015	3,037,255	591,271	187,172,366
		ADOPT$^+$(k = 6,000)	**2,558,658**	**1,427,249**	**1,131,391**	**241,102**	**71,495,744**
		ADOPT$^+$(k = 10,000)	2,559,603	1,427,739	1,131,845	241,169	71,503,823
0.8	71,624	BnB-ADOPT$^+$	9,493,156	4,684,177	4,808,961	1,032,767	324,271,538
		ADOPT$^+$(k = 5,000)	10,911,176	6,123,650	4,787,507	1,056,531	339,276,384
		ADOPT$^+$(k = 10,000)	**6,395,945**	**3,614,771**	**2,781,156**	**619,431**	**192,355,298**
		ADOPT$^+$(k = 20,000)	6,484,296	3,663,938	2,820,339	628,362	195,216,439

(a) Random Binary DCOP Instances (10 variables)

	Trivial *UB*	Algorithm	Total Msgs	VALUE	COST	Cycles	NCCCs
A	15,234,868,488	BnB-ADOPT$^+$	5,090,410	2,708,370	2,381,903	228,784	43,595,024
		ADOPT$^+$(k = 30,000,000)	**2,005,732**	**1,230,851**	**774,745**	**88,556**	**24,068,428**
B	15,355,044,866	BnB-ADOPT$^+$	23,911,475	12,979,404	10,931,932	1,024,435	249,771,051
		ADOPT$^+$(k = 30,000,000)	**9,869,280**	**6,054,524**	**3,814,618**	**459,540**	**166,542,715**
C	3,997,096,838	BnB-ADOPT$^+$	311,738	165,302	146,346	17,571	3,386,651
		ADOPT$^+$(k = 15,000,000)	**178,301**	**104,141**	**74,070**	**10,815**	**2,625,136**
D	15,595,397,524	BnB-ADOPT$^+$	10,722,499	5,714,611	5,007,746	575,613	156,019,351
		ADOPT$^+$(k = 30,000,000)	**3,812,541**	**2,231,332**	**1,581,066**	**196,424**	**66,347,439**

(b) Sensor Network Instances (70, 70, 50 and 70 variables)

Trivial *UB*	Algorithm	Total Msgs	VALUE	COST	Cycles	NCCCs
98,000,000	ADOPT$^+$	25,731	12,769	12,764	**259**	**10,840**
	BnB-ADOPT$^+$	**3,764**	**1,239**	**2,326**	827	38,704
	ADOPT$^+$(k = 1)	25,915	12,953	12,764	259	12,381
	ADOPT$^+$(k = 30)	22,092	13,403	8,490	449	20,591
	ADOPT$^+$(k = 50)	10,502	5,351	4,963	550	25,196
	ADOPT$^+$(k = 100)	6,290	3,061	3,031	550	25,196
	ADOPT$^+$(k = 1,000)	5,050	2,439	2,413	827	38,441

(c) Sensor Network Instances (100 variables)

Table 1: The number of messages, cycles and NCCCs of ADOPT$^+$, BnB-ADOPT$^+$ and ADOPT$^+$(k) on several benchmarks.

Acknowledgments

Patricia Gutierrez has an FPI scholarship BES-2008-006653. She and Pedro Meseguer are partially supported by the Spanish project TIN-2009-13591-C02-02.

References

[Ali *et al.*, 2005] Syed Ali, Sven Koenig, and Milind Tambe. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *Proc. of AAMAS*, pages 1041–1048, 2005.

[Farinelli *et al.*, 2008] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas Jennings. Decentralised coordination of low-power embedded devices using the Max-Sum algorithm. In *Proc. of AAMAS*, pages 639–646, 2008.

[Gutierrez and Meseguer, 2010a] Patricia Gutierrez and Pedro Meseguer. Saving redundant messages in ADOPT-based algorithms. In *Proc. of DCR*, pages 53–64, 2010.

[Gutierrez and Meseguer, 2010b] Patricia Gutierrez and Pedro Meseguer. Saving redundant messages in BnB-ADOPT. In *Proc. of AAAI*, pages 1259–1260, 2010.

[Hamadi *et al.*, 1998] Youssef Hamadi, Christian Bessière, and Joël Quinqueton. Distributed intelligent backtracking. In *Proc. of ECAI*, pages 219–223, 1998.

[Maheswaran *et al.*, 2004] Rajiv Maheswaran, Milind Tambe, Emma Bowring, Jonathan Pearce, and Pradeep Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. In *Proc. of AAMAS*, pages 310–317, 2004.

[Meisels *et al.*, 2002] Amnon Meisels, Eliezer Kaplansky, Igor Razgon, and Roie Zivan. Comparing performance of distributed constraint processing algorithms. In *Proc. of DCR*, pages 86–93, 2002.

[Modi *et al.*, 2005] Pragnesh Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005.

[Petcu and Faltings, 2005] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *Proc. of IJCAI*, pages 1413–1420, 2005.

[Ueda *et al.*, 2010] Suguru Ueda, Atsushi Iwasaki, and Makoto Yokoo. Coalition structure generation based on distributed constraint optimization. In *Proc. of AAAI*, pages 197–203, 2010.

[Yeoh *et al.*, 2010] William Yeoh, Ariel Felner, and Sven Koenig. BnB-ADOPT: Asynchronous branch-and-bound dcop algorithm. *Journal of Artificial Intelligence Research*, 38:85–133, 2010.

[Yin, 2008] Zhengyu Yin. USC DCOP repository, 2008.