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 \(\{\mathcal{X}, \mathcal{D}, \mathcal{F}, \mathcal{A}, \alpha\}\), 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 \(\alpha : \mathcal{X} \mapsto \mathcal{A}\) 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 \(L_{B_i}\) and \(U_{B_i}\), 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:

\[
OPT_i(d) = \delta_i(d) + \sum_{x_c \in C_i} OPT_c \tag{1}
\]

\[
OPT_i = \min_{d \in D_i} OPT_i(d) \tag{2}
\]

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 \( \infty \), 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:

\[
\delta_i(d) = \sum_{(x_j, d_j) \in X_i} F_{ij}(d, d_j) \tag{3}
\]

\[
LB_i(d) = \delta_i(d) + \sum_{x_c \in C_i} lb_i^c(d) \tag{4}
\]

\[
LB_i = \min_{d \in D_i} \{ LB_i(d) \} \tag{5}
\]

\[
UB_i(d) = \delta_i(d) + \sum_{x_c \in C_i} ub_i^c(d) \tag{6}
\]

\[
UB_i = \min_{d \in D_i} \{ UB_i(d) \} \tag{7}
\]

\[
d_i = \arg \min_{d \in D_i} \{ LB_i(d) \} \tag{8}
\]

\( x_i \) 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 \( X_c \) 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, \( x_i \) 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 \( LB_i(d_i) \geq UB_i \) 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.

¹The 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 = ∞ and like a hybrid of ADOPT and BnB-ADOPT when 1 < k < ∞. ADOPT(k) uses mostly identical data structures and messages as ADOPT and BnB-ADOPT. Each agent x in ADOPT(k) maintains two thresholds, TH_A and TH_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_A(k-1) or LB_i(d_i) ≥ min{TH_A, UB_i}. If k = 1, then the first condition degenerates to LB_i(d_i) > TH_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 = ∞, 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 < ∞, 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, P_C_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_A, each agent maintains TH_B. It is ini-

01 procedure Start() 02 X_c := ((x_p, ValInit(x_p), 0) | x_p ∈ SCP_c); 03 ID_c := 0; 04 forall x_c ∈ C_c and d ∈ D, 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 WhenReceived(msg); 12 Backtrack(); 13 procedure InitChild(x_c, d) 14 lb_i(d) := h_i(d); 15 ub_i(d) := h_i(d); 16 procedure InitSelf() 17 d_i := arg min_{d ∈ D_c} (di(d) + ∑_{x ∈ C_c} lb_i(d)); 18 ID_i := ID_i + 1; 19 TH_Ai := min_{d ∈ D_c} (di(d) + ∑_{x ∈ C_c} lb_i(d)); 20 TH_Bi := ∞; 21 procedure Backtrack() 22 forall d ∈ D_c 23 LB_i(d) := di(d) + ∑_{x ∈ C_c} lb_i(d); 24 UB_i(d) := di(d) + ∑_{x ∈ C_c} ub_i(d); 25 LB_i := ∞; 26 LB_i := min_{d ∈ D_c} (LB_i(d)); 27 UB_i := min_{d ∈ D_c} (UB_i(d)); 28 MaintainThresholdInvariant(); 29 if (TH_Ai = UB_i) 30 d_i := arg min_{d ∈ D_c} (UB_i(d)) 31 else if (LB_i(d_i) > TH_Ai + (k-1)) 32 d_i := arg min_{d ∈ D_c} (LB_i(d_i) = LB_i(UB_i(d_i))); 33 else if (LB_i(d_i) ≥ min{TH_A, UB_i}); 34 d_i := arg min_{d ∈ D_c} (LB_i(d) = LB_i(UB_i(d_i)); 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_c, d, ID_i, th_c(d_i), min(TH_A, UB_i) − δ_i(d_i)) 41 − ∑_{x ∈ C_c} ∕ x ≠ x_i lb_i(d_i) to each x ∈ C_c; 42 Send(VALUE, x_c, d, ID_i, ∞, ∞) to each x_c ∈ PC_c; 43 if (TH_Ai = UB_i) 44 if (x_i is root or termination message received) 45 Send(TERMINATE) to each x_c ∈ C_c; 46 terminate execution; 47 Send(COST, x_i, LB_i, UB_i) to parent; 48 procedure WhenReceived(TERMINATE) 49 record termination message received; 50 procedure WhenReceived(VALUE, x_p, d_p, ID_p, TH_A, TH_B) 51 X_c := X_c; 52 PriorityMerge((x_p, d_p, ID_p), X_c); 53 if (Compatible(X_a, X_a)) 54 forall x_a ∈ C_a and d ∈ D_a 55 if (x_a ∈ SCP_a) 56 if (x_a is parent) 57 TH_A := TH_A; 58 TH_B := TH_B; 59 procedure WhenReceived(COST, x_c, X_a, LB_a, UB_a) 60 X_c := X_c; 61 PriorityMerge(X_c, X_a); 62 if (Compatible(X_c, X_a)) 63 forall x_c ∈ C_c and d ∈ D, 64 if (Compatible((x_c, d, ID_p) ∈ X’ | x_p ∈ SCP_c), X_a)) 65 InitChild(x_c, d); 66 if (Compatible(X_a, X_a)) 67 if (Compatible(X_a, X_a)) 68 lb_i(d) := max(lb_i(d), LB_a) for the unique (a’, d, ID) ∈ X_a with a’ = a; 69 ub_i(d) := min(ub_i(d), UB_a) for the unique (a’, d, ID) ∈ X_a with a’ = a; 70 if (Compatible(X_a, X_a)) 71 MainSelf();
In ADOPT, the proof sketches. Furthermore, 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, 
\[
LB_i = \min_{d \in D_i} \{LB_i(d)\} \quad \text{(Line 26)}
\]
\[
= \min_{d \in D_i} \{OPT_i(d)\} \quad \text{(see above)}
\]
\[
= OPT_i \quad \text{(Eq. 2)}
\]
\[
UB_i = \min_{d \in D_i} \{UB_i(d)\} \quad \text{(Line 27)}
\]
\[
= \min_{d \in D_i} \{OPT_i(d)\} \quad \text{(see above)}
\]
\[
= OPT_i \quad \text{(Eq. 2)}
\]

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 \(q - 1\), 
\[
LB_i(d) = \delta_i(d) + \sum_{x_e \in C_i} LB_e \quad \text{(Lines 24 and 68)}
\]
\[
\leq \delta_i(d) + \sum_{x_e \in C_i} OPT_e \quad \text{(induction ass.)}
\]
\[
= OPT_i \quad \text{(Eq. 1)}
\]
\[
UB_i(d) = \delta_i(d) + \sum_{x_e \in C_i} UB_e \quad \text{(Line 25 and 69)}
\]
\[
\geq \delta_i(d) + \sum_{x_e \in C_i} OPT_e \quad \text{(induction ass.)}
\]
\[
= OPT_i \quad \text{(Eq. 1)}
\]

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 \(q - 1\), 
\[
LB_i = \min_{d \in D_i} \{LB_i(d)\} \quad \text{(Line 26)}
\]
\[
= \min_{d \in D_i} \{OPT_i(d)\} \quad \text{(see above)}
\]
\[
= OPT_i \quad \text{(Eq. 2)}
\]
\[
UB_i = \min_{d \in D_i} \{UB_i(d)\} \quad \text{(Line 27)}
\]
\[
= \min_{d \in D_i} \{OPT_i(d)\} \quad \text{(see above)}
\]
\[
= OPT_i \quad \text{(Eq. 2)}
\]

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 \(TH_i^A(d)\) 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,
\[
TH_i^A = UB_i \quad \text{(Line 42)}
\]
\[
= UB_i(d_i) \quad \text{(Lines 29-30)}
\]
\[
\geq LB_i(d_i) \quad \text{(Lemma 1)}
\]
Thus, $LB_i(d_i) \leq TH^A_i = UB_i$ and $TH^A_i(d_i)$ is not set to a different value later (Lines 86-89). Then, $TH^A_i(d) = TH^A_i$ on termination.

**Theorem 1** For all agents $x_i$, $TH^A_i = OPT_i$ 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 $TH^A_i = UB_i$ on termination (Lines 42-45), $TH^A_i = LB_i$ 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 $q-1$. Let $x_p$ be an arbitrary agent 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. The value of $\delta_p(d_p)$ is its current value on termination. Then,

$$\sum_{x_c \in C_p} ub^p_c(d_p) = UB_p(d_p) - \delta_p(d_p) \quad \text{(Line 25)}$$

$$= UB_p - \delta_p(d_p) \quad \text{(Lines 29-30)}$$

$$= TH^A_p - \delta_p(d_p) \quad \text{(Lines 42-45)}$$

$$= TH^A_p(d_p) - \delta_p(d_p) \quad \text{(Lemma 3)}$$

$$= \sum_{x_c \in C_p} th^A_p(d_p) \quad \text{(Lines 91-94)}$$

Thus, $\sum_{x_c \in C_p} ub^p_c(d_p) = \sum_{x_c \in C_p} th^A_p(d_p)$. Furthermore, for all agents $x_c \in C_p, th^A_p(d_p) \leq ub^p_c(d_p)$ (Lines 77-78). Combining the inequalities, we get $th^A_p(d_p) = ub^p_c(d_p)$. Additionally, $TH^A_c = th^A_p(d_p)$ (Lines 57-58) and $UB_c = ub^p_c(d_p)$ (Line 69). Therefore, $TH^A_c = UB_c$. Next,

$$\sum_{x_c \in C_p} OPT_c = OPT_p - \delta_p(d_p) \quad \text{(Eq. 1)}$$

$$= TH^A_p - \delta_p(d_p) \quad \text{(induction ass.)}$$

$$= TH^A_p(d_p) - \delta_p(d_p) \quad \text{(Lemma 3)}$$

$$= \sum_{x_c \in C_p} th^A_p(d_c) \quad \text{(Lines 91-94)}$$

$$= \sum_{x_c \in C_p} TH^A_c \quad \text{(Lines 57-58)}$$

Thus, $\sum_{x_c \in C_p} OPT_c = \sum_{x_c \in C_p} TH^A_c = \sum_{x_c \in C_p} UB_c$ (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, $TH^A_c = UB_c = OPT_c$. ■

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 $|X| - 1$. 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 $(100, \ldots, 200)$. 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 $(0, \ldots, 200)$. 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 almost the same. Agents in ADOPT$^+$($k$) send more VALUE messages because they need to send VALUE messages when $TH^B_p$ changes even if $TH^A_p$ 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.
Table 1: The number of messages, cycles and NCCCs of ADOPT, BnB-ADOPT and ADOPT+(k) on several benchmarks.

<table>
<thead>
<tr>
<th>p1</th>
<th>Trivial UB</th>
<th>Algorithm</th>
<th>Total Msgs</th>
<th>VALUE</th>
<th>COST</th>
<th>Cycles</th>
<th>NCCCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>45,766</td>
<td>ADOPT+ (k = 1,000)</td>
<td>2,085,732</td>
<td>1,000,000</td>
<td>13,403</td>
<td>313,768</td>
<td>45,766</td>
</tr>
<tr>
<td>0.5</td>
<td>45,766</td>
<td>ADOPT+ (k = 4,000)</td>
<td>197,342</td>
<td>109,111</td>
<td>88,212</td>
<td>17,100</td>
<td>3,969,104</td>
</tr>
<tr>
<td>0.6</td>
<td>53,722</td>
<td>BnB-ADOPT</td>
<td>1,017,939</td>
<td>500,514</td>
<td>517,407</td>
<td>99,969</td>
<td>26,191,249</td>
</tr>
<tr>
<td>0.7</td>
<td>63,654</td>
<td>BnB-ADOPT</td>
<td>3,716,766</td>
<td>1,825,322</td>
<td>1,891,416</td>
<td>38,774</td>
<td>116,050,941</td>
</tr>
<tr>
<td>0.8</td>
<td>71,624</td>
<td>ADOPT+ (k = 5,000)</td>
<td>10,122,499</td>
<td>5,174,611</td>
<td>5,007,746</td>
<td>313,697</td>
<td>192,355,298</td>
</tr>
<tr>
<td>0.8</td>
<td>71,624</td>
<td>ADOPT+ (k = 10,000)</td>
<td>6,484,296</td>
<td>3,663,938</td>
<td>2,821,339</td>
<td>628,362</td>
<td>195,216,439</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Trivial UB</th>
<th>Algorithm</th>
<th>Total Msgs</th>
<th>VALUE</th>
<th>COST</th>
<th>Cycles</th>
<th>NCCCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>15,324,868,488</td>
<td>ADOPT+ (k = 30,000,000)</td>
<td>9,493,156</td>
<td>4,684,177</td>
<td>4,808,961</td>
<td>1,032,767</td>
<td>324,217,538</td>
</tr>
<tr>
<td>15,595,397,524</td>
<td>BnB-ADOPT</td>
<td>10,722,499</td>
<td>5,174,611</td>
<td>5,007,746</td>
<td>313,697</td>
<td>192,355,298</td>
</tr>
</tbody>
</table>

(c) Sensor Network Instances (100 variables)

<table>
<thead>
<tr>
<th>Trivial UB</th>
<th>Algorithm</th>
<th>Total Msgs</th>
<th>VALUE</th>
<th>COST</th>
<th>Cycles</th>
<th>NCCCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>98,000,000</td>
<td>ADOPT+ (k = 1,000)</td>
<td>25,731</td>
<td>12,769</td>
<td>12,764</td>
<td>259</td>
<td>10,840</td>
</tr>
<tr>
<td>98,000,000</td>
<td>BnB-ADOPT</td>
<td>3,764</td>
<td>1,239</td>
<td>1,236</td>
<td>827</td>
<td>38,749</td>
</tr>
<tr>
<td>98,000,000</td>
<td>ADOPT+ (k = 30)</td>
<td>35,915</td>
<td>12,953</td>
<td>12,964</td>
<td>259</td>
<td>12,581</td>
</tr>
<tr>
<td>98,000,000</td>
<td>ADOPT+ (k = 50)</td>
<td>10,502</td>
<td>5,351</td>
<td>4,963</td>
<td>550</td>
<td>25,196</td>
</tr>
</tbody>
</table>

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