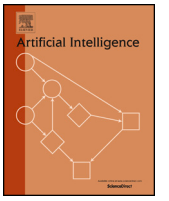
Artificial Intelligence 329 (2024) 104083

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Counterexamples and amendments to the termination and optimality of ADOPT-based algorithms ☆

Koji Noshiro [a,*], Koji Hasebe [b]

[a] Degree Programs in Systems and Information Engineering, University of Tsukuba, Japan
[b] Department of Computer Science, University of Tsukuba, Japan

ARTICLE INFO

Keywords:
Distributed constraint optimization problems
Asynchronous distributed optimization

ABSTRACT

A distributed constraint optimization problem (DCOP) is a framework to model multi-agent coordination problems. Asynchronous distributed optimization (ADOPT) is a well-known complete DCOP algorithm, and many of its variants have been proposed over the last decade. It is considered proven that ADOPT-based algorithms have the key properties of termination and optimality, which guarantee that the algorithms terminate in a finite time and obtain an optimal solution, respectively. In this paper, we present counterexamples to the termination and optimality of ADOPT-based algorithms. They are classified into three types, at least one of which exists in each of ADOPT and eight of its variants that we analyzed. In other words, the algorithms may potentially not terminate or terminate with a suboptimal solution. Furthermore, we show that the bounded-error approximation of ADOPT, which enables the algorithm to terminate faster with the quality of the solution guaranteed within a predefined error bound, also suffers from flaws. Additionally, we propose an amended version of ADOPT that avoids the flaws in existing algorithms and prove that it has the properties of termination and optimality.

1. Introduction

Distributed constraint optimization problems (DCOPs) [2–4] involve agents in a network that must coordinate the values of their variables. In a DCOP, agents communicate with each other and cooperate to find a global solution, which optimizes the sum of cost functions, i.e., constraints, defined between variables. Many multi-agent coordination problems, such as event scheduling [5], sensor network operation [6], and disaster management problems [7], are modeled as DCOPs. For example, resource-constrained task scheduling [8] is formulated by a DCOP as follows: a variable represents the task performed by an agent at a time, and a cost function represents the precedence of tasks or resource usage exceeding its capacity. Owing to the wide range of applications, various algorithms to solve DCOPs have been proposed over the decades.

Asynchronous distributed optimization (ADOPT) [2] is a well-known DCOP algorithm. This is a search-based algorithm, i.e., agents search for optimal values of their variables by sending and receiving several types of messages. Messages report information

☆ This paper extends our conference paper [1]. The main extension consists of four points: (1) the analysis of the flaw in the bounded-error approximation of ADOPT; (2) the analysis of counterexamples in ADOPT-ing; (3) the pseudocode of the amended version of ADOPT that we propose and the discussion of its complexity; and (4) the full proof of the termination and optimality of our proposed version of ADOPT.

* Corresponding author.
E-mail addresses: noshiro@mas.cs.tsukuba.ac.jp (K. Noshiro), hasebe@cs.tsukuba.ac.jp (K. Hasebe).
URL: https://mas.cs.tsukuba.ac.jp/~noshiro (K. Noshiro).

https://doi.org/10.1016/j.artint.2024.104083
Received 28 June 2023; Received in revised form 17 November 2023; Accepted 21 January 2024

Available online 24 January 2024

Table 1
Flaws in ADOPT and eight variant algorithms. N, +, IDB, BnB, BnB+, (k), and BD stand for ADOPT-N, ADOPT$^+$, IDB-ADOPT, BnB-ADOPT, BnB-ADOPT$^+$, ADOPT(k), and BD-ADOPT, respectively. Also, ing(ATC) stands for our modified version of ADOPT-ing with ADOPT's termination condition. The flaws are classified into three types: (1) failure of termination; (2) failure of optimality caused by initialization; and (3) failure of optimality caused by TERMINATE messages.

Flaw	ADOPT	N	+	IDB	BnB	BnB+	(k)	BD	ing(ATC)
(1)	✓	✓	✓	✓	-	-	-	-	-
(2)	✓	✓	✓	✓	-	-	-	-	-
(3)	✓	✓	-	✓	✓	✓	✓	✓	✓

on the state of an agent, such as its current variable value and calculated costs, to other agents. Agents update the cost information based on the received messages and change its variable value using best-first search.

The major advantages of ADOPT are asynchrony, completeness, and bounded-error approximation. Asynchrony allows agents to process their computations without waiting for other agents, which enables the algorithm to use agent resources efficiently. Additionally, completeness guarantees that the algorithm finds an optimal solution. Although complete algorithms require exponential time or space, they are still desirable, especially from a theoretical point of view. On the other hand, ADOPT also provides bounded-error approximation, which allows the algorithm to terminate faster while guaranteeing the quality of the solution within a predefined error bound. This enables a trade-off between solution quality and computational cost.

Many researchers have worked on improving the performance of ADOPT. While the original version could handle only binary constraints, the extension ADOPT-N [8] can deal with n-ary ones. ADOPT$^+$ [9] saves redundant messages exchanged in ADOPT. IDB-ADOPT [10] executes ADOPT iteratively to change the search strategy from best-first to depth-first. The same authors of IDB-ADOPT proposed BnB-ADOPT [11], which employs depth-first branch-and-bound search. BnB-ADOPT$^+$ [9] reduces the number of redundant messages in BnB-ADOPT. Other variants include ADOPT(k) [12], which generalizes ADOPT and BnB-ADOPT, and BD-ADOPT [13], which combines best-first and depth-first searches. ADOPT-ing [14] introduces nogood, which is used in asynchronous backtrack (ABT) [15] to solve distributed constraint satisfaction problems.

For a DCOP algorithm to be complete, it must have two important properties: termination and optimality. Termination guarantees that an algorithm terminates in a finite time, while optimality guarantees that an optimal solution is obtained at termination. Each study of ADOPT and its variant algorithms claims that the algorithm has both of these properties. However, many of the theorems in these studies are invalid or improper to derive termination or optimality.

The objectives of this paper are threefold: to show counterexamples to the termination and optimality of ADOPT and its variant algorithms, to propose an amended version of ADOPT, and to prove the two properties of the proposed version.

First, we provide counterexamples to the termination and optimality of ADOPT and show that similar flaws also exist in many of the variants and that the bounded-error approximation of ADOPT is also incorrect. Table 1 summarizes the flaws in ADOPT and eight of its variants. The flaws are classified into three types by their properties and causes: failure of termination, failure of optimality caused by algorithm initialization, and failure of optimality caused by messages sent at termination. As the table shows, at least one flaw exists in each of the ADOPT-based algorithms that we analyzed. In particular, ADOPT itself has all three flaws, i.e., it may potentially not terminate or terminate with a suboptimal solution. Moreover, most of the variants have the third type of flaw.

Second, we propose three amendments to ADOPT, each of which corresponds to a type of flaw. The first amendment modifies the update rules for lower and upper bounds of costs since the counterexample to termination is caused by the nonmonotonicity of the bounds. The second deals with the initialization of a current context, which is a set of assignments maintained by an agent. Additionally, the third modifies termination messages and their receiving procedure.

Finally, we prove the termination and optimality properties of the amended version of ADOPT. Although these proofs are based on the ones in an existing variant of ADOPT [11], they are modified to derive the properties. Our proofs ensure that all agents in the algorithm terminate in a finite time and then obtain an optimal solution.

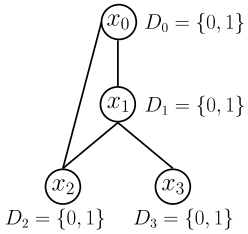
This paper is organized as follows. Section 2 defines DCOPs and summarizes ADOPT. Section 3 presents the counterexamples of ADOPT and its variants in detail. Section 4 describes the amended version of ADOPT, and Section 5 proves its termination and optimality. Finally, Section 6 concludes this paper.

2. Preliminaries

In this section, we provide the preliminaries for this study. Sections 2.1 and 2.2 present the definition of DCOPs and an overview of ADOPT, respectively.

2.1. DCOP

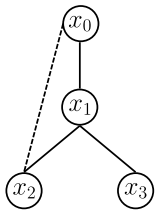
A DCOP is defined as a tuple $\langle A, X, D, F \rangle$. $A = \{a_1, \ldots, a_n\}$ is a finite set of agents, and $X = \{x_1, \ldots, x_n\}$ is a finite set of variables, where x_i is the variable assigned to agent a_i. An agent can control only the value of the variable assigned to it. Following the studies of ADOPT and its variants, we assume that each agent a_i has only one variable, denoted by x_i. Additionally, we use the terms "agent" and "variable" interchangeably. Here, $D = \{D_1, \ldots, D_n\}$ is a set of domains of variables, where D_i is the finite domain of variable



(a) Constraint graph

d_i	d_j	$f_{i,j}$
0	0	2
0	1	5
1	0	3
1	1	0

(b) Cost function



(c) DFS pseudo-tree

Fig. 1. Constraint graph, cost functions, and DFS pseudo-tree of a DCOP. In the DFS pseudo-tree, the solid lines indicate parent/child relationships, and the dashed line indicates a pseudo-parent/pseudo-child relationship.

x_i. Furthermore, F is a finite set of binary cost functions. A cost function for two variables x_i and x_j is defined as $f_{i,j} : D_i \times D_j \to \mathbb{N}$. Each cost function is known only to the involved agents. Agents aim to find an assignment $\boldsymbol{d}^*$ for all variables such that it minimizes the global objective function, which is the sum of the cost functions in F. This assignment is an optimal solution and is expressed as follows:

$$\boldsymbol{d}^* := \arg \min_{\boldsymbol{d} \in \prod_k D_k} \sum_{f_{i,j} \in F} f_{i,j}(d_i, d_j).$$

A DCOP can be expressed as a constraint graph. The nodes correspond to variables in a DCOP, and each edge indicates that the two connected variables share a cost function. A depth-first search (DFS) pseudo-tree extends a spanning tree of a constraint graph. Here, the edges are classified into tree edges and back edges. Tree edges exist in the spanning tree and establish parent/child relationships, while back edges do not exist in the spanning tree but exist in the constraint graph and establish pseudo-parent/pseudo-child relationships. The important property of a DFS pseudo-tree is that no edges appear between nodes in different branches. Therefore, a DCOP can be divided into independent subproblems.

Fig. 1 shows an example of DCOPs. The DCOP consists of four variables (or agents) x_0, x_1, x_2, and x_3, where the domain of each variable x_i is $D_i = \{0, 1\}$, and four cost functions $f_{0,1}, f_{0,2}, f_{1,2}$, and $f_{1,3}$. When variable x_1's value is 0 and variable x_2's value is 1, the cost of $f_{1,2}$ is 5. The optimal solution of this DCOP is $(x_0, x_1, x_2, x_3) = (1, 1, 1, 1)$, and its cost is 0. Fig. 1c shows a DFS pseudo-tree of the DCOP. In the tree, x_0 is the root agent as well as the parent of x_1 and the pseudo-parent of x_2.

We use the following notations concerning agents related to agent $x_i \in X$ in a DFS pseudo-tree.[1] $CD(x_i) \subseteq X$ is the set of the children and pseudo-children of x_i; $C(x_i) \subseteq CD(x_i)$ is the set of the children of x_i; $pa(x_i) \in X$ is the parent of x_i; $P(x_i) \subseteq X$ is the set of the ancestors of x_i; $SCP(x_i) \subseteq P(x_i)$ is the set of the ancestors of x_i that are the parents or pseudo-parents of agents in the subtree rooted at x_i; $CP(x_i) \subseteq SCP(x_i)$ is the set of the ancestors of x_i that are the parent or pseudo-parents of x_i.

2.2. ADOPT

ADOPT [2] is known as an asynchronous complete algorithm to solve DCOPs.[2] In this algorithm, agents asynchronously conduct best-first search by sending and receiving four types of messages: VALUE, COST, THRESHOLD, and TERMINATE. Since these messages are exchanged based on a DFS pseudo-tree, ADOPT requires constructing it in the preprocessing.

The pseudocode of ADOPT is presented in Algorithm 1. The notations concerning agent x_i in this study are as follows: d_i denotes the current value of x_i; CX_i denotes the current context of x_i; $\delta_i(d, CX)$ denotes the local cost of x_i when x_i takes value d and its higher neighbors (i.e., its parent and pseudo-parents) take the values given in context CX; $LB_i(d)$ and $UB_i(d)$ denote the lower and upper bounds of x_i for value d; LB_i and UB_i denote the lower and upper bounds of x_i; TH_i denotes the threshold of x_i; $cx_i(d, x_c), lb_i(d, x_c)$, and $ub_i(d, x_c)$ denote the context, lower bound, and upper bound, respectively, stored by x_i when it receives a COST message in which the context contains assignment (x_i, d) from child x_c; and $th_i(d, x_c)$ denotes the threshold allocated to child x_c when x_i takes value d.

Each type of message reports the information of the sender agent to its parent, children, or pseudo-children. A VALUE message is sent from agent x_i to its children and pseudo-children x_c and reports the current value d_i of x_i (lines 14–21 and 52); a COST message is sent from x_i to its parent x_p and reports the current context CX_i, the lower bound LB_i, and the upper bound UB_i (lines 22–37 and 58); a THRESHOLD message is sent from x_i to its children x_c and reports the threshold allocated to x_c, denoted by $th_i(d_i, x_c)$ (lines 38–42 and 71); and a TERMINATE message is sent from x_i to its children x_c and reports the termination of x_i (lines 43–46 and 56).

After initializing or receiving a message, the agent runs procedure Backtrack (lines 47–58). This procedure includes updating the variable value, checking the termination condition (and executing termination), and sending messages. Additionally, after updating bounds or thresholds, the agent modifies the thresholds to maintain the following three invariants (lines 7, 20, 35–36, 41, 53, and 59–78):

[1] These notations follow the study of BnB-ADOPT [11].

[2] In this section, we show a brief description of ADOPT. See the original study [2] for more details.

Algorithm 1 Pseudocode of ADOPT.

```
 1: procedure Start()
 2:     TH_i := 0;
 3:     CX_i := ∅;
 4:     for all d ∈ D_i, x_c ∈ C(x_i) do
 5:         InitChild(d, x_c);
 6:     d_i := arg min_{d∈D_i} LB_i(d);
 7:     MaintainThresholdInvariant();
 8:     Backtrack();

 9: procedure InitChild(d, x_c)
10:     lb_i(d, x_c) := 0;
11:     ub_i(d, x_c) := ∞;
12:     th_i(d, x_c) := 0;
13:     cx_i(d, x_c) := ∅;

14: procedure Received(VALUE, x_p, d)
15:     if TERMINATE not received from parent then
16:         add (x_p, d) to CX_i (and remove the old assignment of x_p);
17:         for all d ∈ D_i, x_c ∈ C(x_i) do
18:             if ¬Compatible(CX_i, cx_i(d, x_c)) then
19:                 InitChild(d, x_c);
20:         MaintainThresholdInvariant();
21:         Backtrack();

22: procedure Received(COST, x_c, CX, LB, UB)
23:     d := d'_i  s.t. (x_i, d'_i) ∈ CX;
24:     remove (x_i, d) from CX;
25:     if TERMINATE not received from parent then
26:         for all (x_j, d_j) ∈ CX and x_j is not x_i's neighbor do
27:             add (x_j, d_j) to CX_i (and remove the old assignment of x_j);
28:         for all d' ∈ D_i, x_c' ∈ C(x_i) do
29:             if ¬Compatible(CX_i, cx_i(d', x_c')) then
30:                 InitChild(d', x_c');
31:     if Compatible(CX, CX_i) then
32:         lb_i(d, x_c) := LB;
33:         ub_i(d, x_c) := UB;
34:         cx_i(d, x_c) := CX;
35:         MaintainChildThresholdInvariant();
36:         MaintainThresholdInvariant();
37:     Backtrack();

38: procedure Received(THRESHOLD, th, CX)
39:     if Compatible(CX, CX_i) then
40:         TH_i := th;
41:         MaintainThresholdInvariant();
42:         Backtrack();

43: procedure Received(TERMINATE, CX)
44:     record TERMINATE received;
45:     CX_i := CX
46:     Backtrack();

47: procedure Backtrack()
48:     if TH_i = UB_i then
49:         d_i := arg min_{d∈D_i} UB_i(d) (choose the previous d_i if possible);
50:     else if LB_i(d_i) > TH_i then
51:         d_i := arg min_{d∈D_i} LB_i(d);
52:     Send(VALUE, x_i, d_i) to each x_c ∈ CD(x_i);
53:     MaintainAllocationInvariant();
54:     if TH_i = UB_i then
55:         if TERMINATE received or x_i is root then
56:             Send(TERMINATE,
        CX_i ∪ {(x_i, d_i)}) to each x_c ∈ C(x_i);
57:             terminate execution;
58:     Send(COST, x_i, CX_i, LB_i, UB_i) to pa(x_i);

59: procedure MaintainThresholdInvariant()
60:     if TH_i < LB_i then
61:         TH_i := LB_i;
62:     if TH_i > UB_i then
63:         TH_i := UB_i;

64: procedure MaintainAllocationInvariant()
65:     while TH_i > δ_i(d_i) + Σ_{x_c∈C(x_i)} th_i(d_i, x_c) do
66:         choose x_c ∈ C(x_i) where ub_i(d_i, x_c) > th_i(d_i, x_c);
67:         th_i(d_i, x_c) := th_i(d_i, x_c) + 1;
68:     while TH_i < δ_i(d_i) + Σ_{x_c∈C(x_i)} th_i(d_i, x_c) do
69:         choose x_c ∈ C(x_i) where lb_i(d_i, x_c) < th_i(d_i, x_c);
70:         th_i(d_i, x_c) := th_i(d_i, x_c) − 1;
71:     Send(THRESHOLD, th_i(d_i, x_c), CX_i) to each x_c ∈ C(x_i);

72: procedure MaintainChildThresholdInvariant()
73:     for all d ∈ D_i, x_c ∈ C(x_i) do
74:         while lb_i(d, x_c) > th_i(d, x_c) do
75:             th_i(d, x_c) := th_i(d, x_c) + 1;
76:     for all d ∈ D_i, x_c ∈ C(x_i) do
77:         while th_i(d, x_c) > ub_i(d, x_c) do
78:             th_i(d, x_c) := th_i(d, x_c) − 1;
```

Definition 1 *(ThresholdInvariant).*

$$LB_i \le TH_i \le UB_i.$$

Definition 2 *(AllocationInvariant).*

$$TH_i = \delta_i(d_i, CX_i) + \sum_{x_c \in C(x_i)} th_i(d_i, x_c).$$

Definition 3 *(ChildThresholdInvariant).* $\forall d \in D_i, \forall x_c \in C(x_i)$,

$$lb_i(d, x_c) \le th_i(d, x_c) \le ub_i(d, x_c).$$

Furthermore, ADOPT provides bounded-error approximation, which enables a trade-off between solution quality and computation time. The algorithm guarantees the cost error at termination within the given error bound b, i.e., it terminates fast if b is a large value. This is implemented by modifying the threshold TH_r of the root agent x_r. ThresholdInvariant in the root agent is redefined as follows:

Definition 4 *(ThresholdInvariant [Bounded Error]).*

$$TH_r = \min\{LB_r + b, UB_r\},$$

where x_r is the root agent.

The original study of ADOPT provides three properties (as theorems in the study) in terms of its termination and optimality. Before showing the properties, we define the optimal costs for the subtree rooted at agent x_i given a context. The optimal costs $\gamma_i(CX)$ and $\gamma_i(d, CX)$[3] are defined recursively as follows:

$$\gamma_i(d, CX) := \delta_i(d, CX) + \sum_{x_c \in C(x_i)} \gamma_i\left(CX \cup \{(x_i, d)\}\right),$$
$$\gamma_i(CX) := \min_{d \in D_i} \gamma_i(d, CX),$$

where d represents a value of x_i and CX denotes a context of x_i. The following three properties are presented in the study of ADOPT. Property 1 shows the relations between the bounds and optimal cost; Property 2 claims to guarantee termination; and Property 3 claims to guarantee optimality.

Property 1. $\forall x_i \in X, LB_i \leq \gamma_i(CX_i) \leq UB_i$.

Property 2. $\forall x_i \in X$, *if the current context* CX_i *is fixed, then* $TH_i = UB_i$ *will eventually occur.*

Property 3. $\forall x_i \in X$, x_i*'s final threshold value* TH_i *is equal to* $\gamma_i(CX_i)$.

There are counterexamples to these properties. We describe them in the next section.

3. Counterexamples to the termination and optimality of ADOPT-based algorithms

We present counterexamples to the theorems for termination and optimality in the original study of ADOPT and their causes. These counterexamples indicate that the proofs presented in the study are incorrect. As mentioned in Section 1, they are classified into three types by their properties and causes. The first is the counterexample to termination, described in Section 3.1, which is caused by the nonmonotonicity of the bounds. The second and third are counterexamples to optimality, described in Sections 3.2 and 3.3, respectively; the second is due to initialization, while the third is due to TERMINATE messages.

In Sections 3.1–3.3, as well as describing the traces[4] and causes of the counterexamples in ADOPT, we show that similar flaws exist in seven variant algorithms: ADOPT-N, ADOPT$^+$, IDB-ADOPT, BnB-ADOPT, BnB-ADOPT$^+$, ADOPT(k), and BD-ADOPT. These flaws exist because the studies attempt to prove the desired properties relying on ADOPT's invalid theorems or introduce theorems that are not strong enough to derive the properties. Additionally, in Section 3.4, we show that the bounded-error approximation of ADOPT is incorrect, i.e., the cost error at termination can exceed the given error bound, as with the counterexample to optimality. Finally, in Section 3.5, we analyze flaws in ADOPT-ing, which requires some modifications to discuss the flaws since its termination process is different from ADOPT and the other variants and problematic in practice.

In the counterexamples, we make some assumptions in terms of message transfer. First, a finite random delay exists between sending a message and receiving it. Second, messages exchanged between a pair of agents are received in the order they were sent, while an agent receives messages from different agents in any order. Finally, agents send messages after processing all received messages, which is called a cycle [2]. These assumptions are adopted in the studies of ADOPT and its variants.

Before presenting the counterexamples, we discuss their generalities. For the sake of simplicity, the counterexamples shown below contain variables that take only one value and constant functions whose costs are 0. These cost functions, in particular, are negligible in practice since they affect neither other costs nor a selection of variable values. However, there also exist more complex counterexamples that do not have such variables and functions. For example, we can easily obtain other counterexamples by adding a constant to all the values of the cost functions in the counterexamples presented below.

3.1. Counterexample to termination

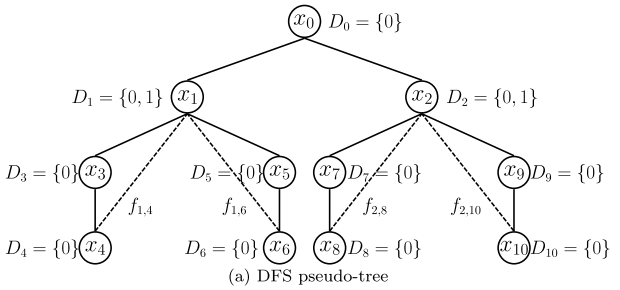
Fig. 2 shows the DFS pseudo-tree and cost functions of a DCOP in which the counterexample to termination occurs. The cost functions not specified in Fig. 2b are defined as constant functions whose values are 0. In the DCOP, only two agents x_1 and x_2 have the domain $\{0, 1\}$, and the other agents always take the same value, 0.

3.1.1. Trace of the counterexample

In the trace of this counterexample, agents repeat the transition of their states; in particular, agents x_1 and x_2 change their variable values infinitely, which results in nontermination and contradicts Property 2 in the original study of ADOPT. Here, the states of some agents at the initial step of the iteration (and the trace itself) are shown in Fig. 3.

[3] $\gamma_i(CX)$ corresponds to $OPT(x_i, context)$ in the original study of ADOPT.

[4] Figures that show the detailed traces of the counterexamples and the implementation of the counterexamples are available at https://mas.cs.tsukuba.ac.jp/~noshiro/.



(a) DFS pseudo-tree

d_1	d_4	$f_{1,4}$
0	0	5
1	0	6

d_1	d_6	$f_{1,6}$
0	0	2
1	0	3

d_2	d_8	$f_{2,8}$
0	0	5
1	0	6

d_2	d_{10}	$f_{2,10}$
0	0	2
1	0	3

(b) Cost functions

Fig. 2. DFS pseudo-tree and cost functions of a DCOP for the counterexample to termination.

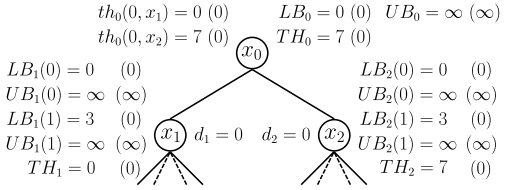


Fig. 3. States of agents at the initial step of the iteration in the counterexample to termination. The states in Step 0 are shown inside parentheses.

In the following description, we only focus on the variable values, contexts, lower bounds, and thresholds, which are directly related to this trace, and ignore the rest unless otherwise noted.

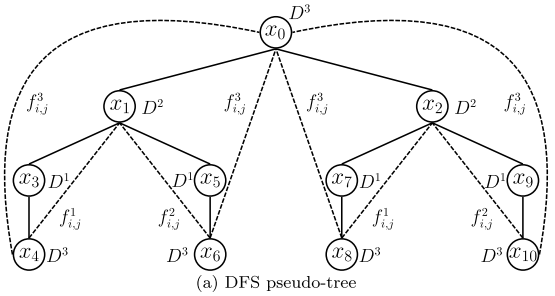
Step 0. As the initial state, we assume that agents x_1 and x_2 set their values (i.e., d_1 and d_2) to 0. In the initialization, the thresholds allocated to the children of x_0 (i.e., $th_0(0,x_1)$ and $th_0(0,x_2)$) are set to 0; the lower bounds of x_1 and x_2 (i.e., $LB_1(0), LB_1(1), LB_2(0)$, and $LB_2(1)$) are set to 0; and the thresholds of x_1 and x_2 (i.e., TH_1 and TH_2) are set to 0. The states of agents at this point are presented inside the parentheses in Fig. 3.

Step 1. Let us begin with the process in the subtree rooted at x_1. First, x_1 sends VALUE messages to its children and pseudo-children x_3, x_4, x_5, and x_6. Afterwards, x_4 receives the VALUE message and computes the lower bound using the updated context $CX_4 \ni (x_1, 0)$, and thus $LB_4 = 5$. Then, x_4 reports the lower bound to x_3 by sending a COST message. Subsequently, x_3 receives the VALUE message from x_1 and the COST message from x_4 and computes the lower bound as $LB_3 = 5$. Then, x_3 sends a COST message with $CX_3 \ni (x_1, 0)$ and $LB_3 = 5$ to x_1. After x_1 receives the COST message, x_1 updates $LB_1(0)$ to 5. Since $LB_1(0) = 5 > TH_1 = 0$, x_1 changes its value d_1 to 1 and sends VALUE messages to its children and pseudo-children.

Step 2. Next, x_5 receives the VALUE messages from x_1 and sends a COST message to x_1, but this COST message does not affect the bounds of x_1 because $LB_5 = 0$ and $UB_5 = \infty$, which are the initial bounds. This message is necessary for sending the reinitialized bounds of x_5 to x_1 in the second or subsequent iteration. Similar to the procedure in Step 1, x_1, x_3, and x_4 receive and send messages, and then x_1 updates the lower bound as $LB_1(1) = 6$. Additionally, the threshold of x_1 increases as $TH_1 = LB_1 = \min\{LB_1(0), LB_1(1)\} = 5$ because of ThresholdInvariant. Since $LB_1(1) = 6 > TH_1 = 5$, x_1 changes its value d_1 back to 0.

Step 3. Subsequently, similar cost calculations and value changes are performed between x_1, x_5, and x_6, and the results are summarized as follows. First, x_1 computes the lower bound as $LB_1(0) = lb_1(0,x_3) + lb_1(0,x_5) = 5 + 2 = 7$ and then changes its value d_1 from 0 to 1. Next, x_1 also computes the lower bound as $LB_1(1) = lb_1(1,x_3) + lb_1(1,x_5) = 6 + 3 = 9$, and then the value d_1 returns to 0. After the cost calculations, x_1 sends VALUE messages with the current value $d_1 = 0$ to the lower neighbors and a COST message with $LB_1 = 7$ to x_0. Afterwards, x_0 receives the COST message and then increases LB_0, TH_0, and $th_0(0,x_1)$ to 7 (the thresholds change due to ThresholdInvariant and ChildThresholdInvariant). As the result of the processes from Step 0 to Step 3, the current contexts of the lower neighbors of x_1 (i.e., x_3, x_4, x_5, and x_6) contain assignment $(x_1, 1)$. Note that the lower neighbors of x_1 still do not receive the last VALUE messages with $d_1 = 0$ from x_1; in particular, x_3 and x_4 still do not receive the VALUE messages that x_1 sent during the processes of x_5 and x_6 after x_3 had sent the COST message to x_1 in Step 2.

Step 4. Here, x_3 receives the three VALUE messages from x_1, in which the values are $d_1 = 0, d_1 = 1$, and $d_1 = 0$, in the order of sending. When x_3 processes the first VALUE message, the lower bound of x_3 is reinitialized as $LB_3 = LB_3(0) = lb_3(0,x_4) = 0$ since context $cx_3(0,x_4) \ni (x_1, 1)$, received from x_4 through the COST message, is incompatible with the updated context $CX_3 \ni (x_1, 0)$. After x_3 processes the remaining messages, x_3 sends two types of COST messages to x_1, i.e., the message with $LB_3 = 0$ and $CX_3 = \{(x_1, 0)\}$ and the message with $LB_3 = 0$ and $CX_3 = \{(x_1, 1)\}$. Similarly, x_5 receives the VALUE message with $d_1 = 0$ from x_1 and reinitializes the bounds. Additionally, x_5 sends a COST message with $LB_5 = 0$ and $CX_5 = \{(x_1, 0)\}$ to x_1.



(a) DFS pseudo-tree

d_i	d_j	$f^1_{i,j}$
0	0	5
0	1	8
0	2	1
1	0	6
1	1	9
1	2	1

d_i	d_j	$f^2_{i,j}$
0	0	2
0	1	2
0	2	1
1	0	3
1	1	3
1	2	1

(b) Cost functions

Fig. 4. DFS pseudo-tree and cost functions of a DCOP for the counterexample to termination in ADOPT$^+$ and IDB-ADOPT. The domains of variables are defined as $D^1 = \{0\}, D^2 = \{0,1\}$, and $D^3 = \{0,1,2\}$. Additionally, $i < j$ holds for each $f^k_{i,j}$, and the cost of $f^3_{i,j}$ is 0 if $d_i = d_j$, otherwise, it is 1000.

Step 5. Next, x_1 receives the COST messages from x_3 and x_5 and updates the lower bounds as $LB_1(0) = lb_1(0,x_3) + lb_1(0,x_5) = 0$, $LB_1(1) = lb_1(1,x_3) + lb_1(1,x_5) = 3$. Thus, x_1 obtains the lower bound as $LB_1 = 0$ and keeps the value $d_1 = 0$. After x_1 sends a COST message with $LB_1 = 0$ to x_0, x_0 receives it and updates the lower bound as $LB_0 = LB_0(0) = lb_0(0,x_1) + lb_0(0,x_2) = 0$. Although LB_0 decreases, the thresholds for the children are kept as $th_0(0,x_1) = 7$ and $th_0(0,x_2) = 0$.

Step 6. After the procedure is completed in the subtree rooted at x_1, the same process is performed in the subtree rooted at x_2. The states of agents after performing the process are presented in Fig. 3 (outside the parentheses). In this process, x_2 changes its value d_2 repeatedly, and eventually the lower bounds of x_2 are obtained as $LB_2(0) = 0$ and $LB_2(1) = 3$, and thus $LB_2 = 0$. Here, the thresholds of x_0 are crucial. Since x_2 once increased LB_2 to 7, x_0 also increased $th_0(0,x_2)$ to 7. However, the threshold TH_0 has not been changed from 7 because $LB_0 = lb_0(0,x_1) + lb_0(0,x_2) = 0 + 7 = 7$ when $th_0(0,x_2)$ was changed. Thus, x_0 decreases $th_0(0,x_1)$ to 0 by AllocationInvariant. Hence, x_0 does not satisfy the termination condition since $TH_0 = 7 < UB_0 = \infty$ (as x_0 received the reinitialized upper bound $UB_1 = \infty$ from x_1). Thus, no agent terminates at this point.

Subsequent steps Afterwards, the subtrees rooted at x_1 and x_2 alternately repeat the same processes as described from Step 1 to Step 6. The processes cause x_1 and x_2 to change their values (i.e., d_1 and d_2) infinitely; ADOPT never terminates because the root agent x_0 never satisfies the termination condition. This result contradicts Property 2.

3.1.2. Cause of the counterexample

The cause of this counterexample is the nonmonotonicity of lower bounds. In the trace demonstrated above, the lower bounds of x_1 and x_2 (i.e., LB_1 and LB_2) increase from 0 to 7 and then decrease to 0. This transition causes x_0 to repeatedly change the thresholds allocated to x_1 and x_2 (i.e., $th_0(0,x_1)$ and $th_0(0,x_2)$) without altering TH_0. Thus, x_1 and x_2 keep changing their values, and x_0 does not satisfy the termination condition.

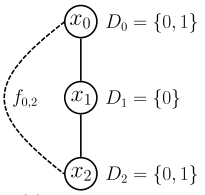
3.1.3. Occurrence in variants

Similar counterexamples to termination appear in the variants ADOPT-N, ADOPT$^+$, and IDB-ADOPT. ADOPT-N is equivalent to the original ADOPT if the given DCOP does not contain n-ary ($n > 2$) cost functions. Since the problem consists of only binary functions, ADOPT-N has this flaw. ADOPT$^+$ and IDB-ADOPT also suffer from similar flaws but require another DCOP. A DCOP for the flaws in these algorithms is presented in Fig. 4. ADOPT$^+$ is a variant that saves redundant messages in ADOPT. Although the trace in Section 3.1.1 contains redundant messages, a similar trace can be performed without such messages by using the presented DCOP. Therefore, this flaw can appear in ADOPT$^+$. In IDB-ADOPT, which repeats the execution of ADOPT until the obtained solution quality is not improved, the root agent sets its initial threshold as the value less by 1 than the cost obtained in the previous iteration. Since the optimal cost of the DCOP in Fig. 4 is 4, the initial threshold of the root agent x_0 at the final iteration is 3. No matter how the threshold is allocated to its children x_1 and x_2, the allocated threshold must be small enough for x_1 and x_2 to change their values when they receive COST messages if the value of x_0 is 0 or 1. Consequently, a trace similar to the one in Section 3.1.1 can be conducted, which causes the counterexample to termination.

On the other hand, the counterexample does not occur in BnB-ADOPT, BnB-ADOPT$^+$, ADOPT(k), and BD-ADOPT. Since the bounds are updated monotonically in these algorithms, the cause of the counterexample is removed.

3.2. Counterexample to optimality caused by initialization

The DFS pseudo-tree and cost function of a DCOP for the second counterexample are shown in Fig. 5. The domains of agents x_0 and x_2 are $\{0,1\}$, and that of agent x_1 is $\{0\}$. The cost between x_0 and x_2 is 0 when the two variable values are 1, otherwise, it is 100. Similar to the DCOP for the counterexample to termination, the other cost functions are constant functions whose values are 0. Thus, the optimal solution of this DCOP is $(x_0, x_1, x_2) = (1, 0, 1)$, whose cost is 0.



(a) DFS pseudo-tree

d_0	d_2	$f_{0,2}$
0	0	100
0	1	100
1	0	100
1	1	0

(b) Cost function

Fig. 5. DFS pseudo-tree and the cost function of a DCOP for the counterexample to optimality caused by initialization.

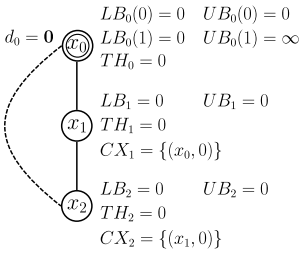


Fig. 6. States of agents in Step 2 in the counterexample to optimality caused by initialization. The terminated agent is indicated with a double circle.

3.2.1. Trace of the counterexample

In this counterexample, a delay of a VALUE message causes termination with a suboptimal solution. The states of agents when x_0 terminates (in Step 2) are shown in Fig. 6.

Step 0. We assume that all agents set their variable values to 0 in the initialization. At this point, their current contexts are initialized to be empty.

Step 1. First, agents send VALUE messages to their children and pseudo-children. Then, x_2 receives the VALUE message only from its parent x_1, which means that the message from the pseudo-parent x_0 is delayed. Here, x_2 updates the current context as $CX_2 = \{(x_1, 0)\}$. Since local cost $\delta_i(d, CX)$ is defined as $\delta_i(d, CX) := \sum_{(x_j, d_j) \in CX} f_{i,j}(d, d_j)$, x_2 computes the local costs as $\delta_2(0, \{(x_1, 0)\}) = f_{1,2}(0, 0) = 0$ and $\delta_2(1, \{(x_1, 0)\}) = f_{1,2}(0, 1) = 0$. Thus, the bounds of x_2 are obtained as $LB_2 = UB_2 = 0$. Then, x_2 sends a COST message with $LB_2 = UB_2 = 0$ to x_1.

Step 2. After x_1 receives the VALUE message from x_0 and the COST message from x_2, x_1 computes the bounds as $LB_1 = UB_1 = 0$. Similarly, x_0 updates the bounds as $LB_0(0) = UB_0(0) = 0$ through the COST message sent from x_1. Since $LB_0 = TH_0 = UB_0 = UB_0(0) = 0$ due to ThresholdInvariant, x_0 keeps its value as $d_0 = 0$ and satisfies the termination condition. However, the variable value $d_0 = 0$ is suboptimal. This is a contradiction to Property 3.

3.2.2. Cause of the counterexample

This counterexample occurs due to the initialization of the current context CX_i of agent x_i. Since CX_i is initialized as an empty set, it may not contain the assignments of some higher neighbors, resulting in underestimations of the local cost $\delta_i(d, CX)$ and the bounds of x_i's ancestors. This fact contradicts Property 1. Moreover, the underestimation results in premature termination.

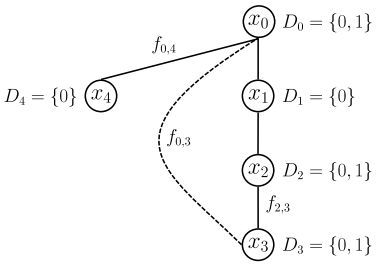
3.2.3. Occurrence in variants

This counterexample can occur in ADOPT-N and ADOPT$^+$ since the DCOP contains only binary cost functions and no redundant messages exist in the trace. IDB-ADOPT also encounters this flaw because the initial threshold of the root agent at the final iteration is less than (or essentially equal to) that in ADOPT.

In BnB-ADOPT, BnB-ADOPT$^+$, ADOPT(k), and BD-ADOPT, the counterexample does not occur since the current context of an agent is initialized to contain the assignments of all of the higher neighbors.

3.3. Counterexample to optimality caused by TERMINATE messages

The third counterexample occurs in a DCOP whose DFS pseudo-tree and cost functions are shown in Fig. 7. In this DCOP, three agents $x_0, x_2,$ and x_3 have the domain $\{0, 1\}$, while the other agents only take values 0. As shown in Fig. 7c, the cost functions $f_{0,3}$ and $f_{2,3}$ involved by x_3 can be considered as one cost function $f_{0,2}$ between x_0 and x_2 since x_3 chooses its variable value to minimize the sum of the cost functions, that is, x_3 chooses the same value as x_0. Note that $f_{0,2}$ is calculated by x_3 and depends on the assignments of x_0 and x_2 in the current context of x_3. Similar to the counterexamples previously shown in this section, the cost functions not specified in Fig. 7b are constant functions whose values are 0. The optimal solution of the DCOP is $(x_0, x_1, x_2, x_3, x_4) = (0, 0, 0, 0, 0)$, whose cost is 1.



(a) DFS pseudo-tree

d_0	d_3	$f_{0,3}$
0	0	1
0	1	1000
1	0	1000
1	1	100

d_2	d_3	$f_{2,3}$
0	0	0
0	1	100
1	0	10
1	1	0

d_0	d_4	$f_{0,4}$
0	0	0
1	0	1000

(b) Cost functions

d_0	d_2	$f_{0,2}$
0	0	1
0	1	11
1	0	200
1	1	100

(c) Cost function induced by $f_{0,3}, f_{2,3}$

Fig. 7. DFS pseudo-tree and the cost functions of a DCOP for the counterexample to optimality caused by TERMINATE messages.

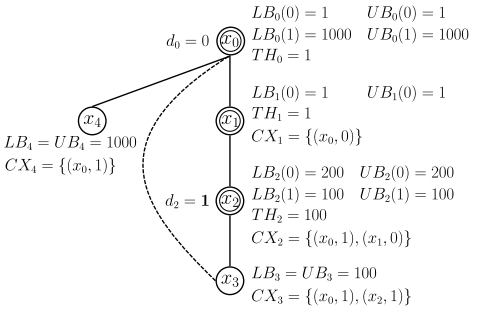


Fig. 8. States of agents in Step 6 in the counterexample to optimality caused by TERMINATE message.

3.3.1. Trace of the counterexample

In the DCOP, agent x_2 can terminate with a suboptimal value because the termination condition is satisfied despite an incorrect threshold. The states of agents in Step 6, where x_2 terminates, are shown in Fig. 8.

Step 0. Similar to the other counterexamples, let us assume that all agents set the variable values to 0 as the initial states. First, agents calculate the cost in the case where x_0 takes the value 0. Although we omit this procedure, the calculation of the cost results in the following: $LB_0(0) = UB_0(0) = 1$; $LB_0(1) = 0$; $UB_0(1) = \infty$; $TH_0 = 0$; $TH_1 = 1$; $LB_2 = UB_2 = 1$; $TH_2 = 1$; and $d_2 = 0$. Furthermore, the contexts of the agents except x_0 contain assignment $(x_0, 0)$. After the calculation, since $LB_0(0) = 1 > TH_0 = 0$, x_0 changes its value d_0 to 1 and sends VALUE messages to x_1, x_3, and x_4.

Step 1. Next, x_3 and x_4 receive the VALUE messages from x_0. At this moment, x_3 updates the context and bounds as $CX_3 = \{(x_0, 1), (x_2, 0)\}$ and $LB_3 = UB_3 = 200$, and x_4 also updates them as $CX_4 = \{(x_0, 1)\}$ and $LB_4 = UB_4 = 1000$. Then, they send COST messages to their parents: from x_3 to x_2 and from x_4 to x_0. After x_0 receives the COST message from x_4, x_0 changes its value d_0 back to 0 since the bounds are obtained as $LB_0(1) = 1000$ and $LB_0 = TH_0 = UB_0 = UB_0(0) = 1$ due to ThresholdInvariant. Therefore, x_0 satisfies the termination condition. Here, x_0 sends VALUE messages with $d_0 = 0$ to its lower neighbors (i.e., x_1, x_3, and x_4) and THRESHOLD and TERMINATE messages to its children (i.e., x_1 and x_4) in this order, and then x_0 executes termination.

Step 2. Afterwards, x_1 receives the VALUE messages from x_0, including the message that x_0 sent when $d_0 = 1$, in the order of sending. Then, x_1 changes CX_1 from $\{(x_0, 0)\}$ into $\{(x_0, 1)\}$ and returns it to $\{(x_0, 0)\}$. Since $\{(x_0, 1)\}$ is incompatible with $cx_1(0, x_2) = \{(x_0, 0)\}$, x_1 reinitializes the bounds as $LB_1 = 0$ and $UB_1 = \infty$. By contrast, TH_1 is not changed from 1. Furthermore, x_1 receives the THRESHOLD and TERMINATE messages from x_0, and then x_1 records receiving the TERMINATE message but does not terminate because $TH_1 = 1 < UB_1 = \infty$. Subsequently, x_1 sends a VALUE message to x_2.

Step 3. Afterwards, x_2 receives only the message from x_1, not from x_3, which means that the messages from x_3 are delayed. Then, x_2 sends a COST message to x_1 with $CX_2 = \{(x_0, 0), (x_1, 0)\}$ and $LB_2 = UB_2 = 1$, which are the same states as in Step 0.

Step 4. Next, x_2 receives the COST message from x_3 with $CX_3 = \{(x_0, 1), (x_2, 0)\}$ and $LB_3 = UB_3 = 200$. Since x_2 is not a neighbor of x_0, x_2 updates CX_2 from $\{(x_0, 0), (x_1, 0)\}$ to $\{(x_0, 1), (x_1, 0)\}$. Then, the bounds of x_2 are reinitialized and updated by the bounds in the message: $LB_2(0) = UB_2(0) = 200$, $LB_2(1) = 0$, and $UB_2(1) = \infty$. x_2 also changes its value d_2 to 1 because $LB_2(0) > TH_2 = 1$, and sends a VALUE message to x_3. After x_3 receives only the VALUE message from x_2 but not the message from x_0, x_3 updates the current context and the bounds as $CX_3 = \{(x_0, 1), (x_2, 1)\}$ and $LB_3 = UB_3 = 100$. Next, x_3 sends a COST message to x_2 again,

and x_2 receives it. Then, x_2 computes the bounds as $LB_2(1) = UB_2(1) = 100$ and updates the threshold as $TH_2 = 100$ because of ThresholdInvariant.

Step 5. Here, x_1 receives the COST message with $CX_2 = \{(x_0, 0), (x_1, 0)\}$ and $LB_2 = UB_2 = 1$, sent from x_2 in Step 3. Since this context is compatible with $CX_1 = \{(x_0, 0)\}$, x_1 updates the bounds as $LB_1 = UB_1 = 1$. At this moment, the termination condition is satisfied because $TH_1 = UB_1 = 1$. Thus, x_1 sends two messages to x_2: a THRESHOLD message with $th_1(0, x_2) = 1$ and $CX_1 = \{(x_0, 0)\}$ and a TERMINATE message with $CX_1 \cup \{(x_1, 0)\} = \{(x_0, 0), (x_1, 0)\}$. Then, x_1 terminates.

Step 6. When x_2 receives the THRESHOLD message from x_1, x_2 does not update TH_2 since context $\{(x_0, 0)\}$ in the message is incompatible with $CX_2 = \{(x_0, 1), (x_1, 0)\}$, and therefore retains its threshold as $TH_2 = 100$. Next, x_2 receives the TERMINATE message from x_1. Although CX_2 is changed to $\{(x_0, 0), (x_1, 0)\}$, the bounds of x_2 are not changed since reinitialization is not performed when an agent receives a TERMINATE message. Therefore, x_2 terminates with the suboptimal value $d_2 = 1$ because x_2 has already satisfied the termination condition with $UB_2 = UB_2(1) = TH_2 = 100$ and received the TERMINATE message. This result contradicts Property 3.

3.3.2. Cause of the counterexample

The counterexample presented above is attributed to the following two factors. The first cause is that an agent does not reinitialize its bounds when it receives a TERMINATE message whose context is incompatible with the current context.

However, as shown below, termination with a suboptimal solution can occur even if such reinitialization is performed. To demonstrate this fact, let us consider a scenario that the bounds of x_2 are reinitialized when x_2 receives the TERMINATE message from x_1 in Step 6. In this case, the bounds of x_2 are obtained as $LB_2(0) = LB_2(1) = 0$ and $UB_2(0) = UB_2(1) = \infty$; and the threshold of x_2 is maintained as $TH_2 = 100$. After x_3 receives the VALUE message with the value $d_0 = 0$ from x_0, x_3 updates the current context as $CX_3 = \{(x_0, 0), (x_2, 1)\}$ and the bounds as $LB_3 = UB_3 = 11$. Then, x_3 sends a COST message to x_2, and x_2 receives it. At this point, x_2 computes the bounds as $LB_2(1) = UB_2(1) = 11$ because context $\{(x_0, 0), (x_2, 1)\}$ in the COST message is compatible with $CX_2 = \{(x_0, 0), (x_1, 0)\}$. Then, x_2 updates the threshold as $TH_2 = 11$ due to ThresholdInvariant. Since $UB_2 = UB_2(1) = TH_2$, x_2 does not change its variable value d_2 from 1. Therefore, x_2 satisfies the termination condition and terminates with the suboptimal value $d_2 = 1$.

The second cause of the counterexample is an incorrect threshold at termination. Even if the threshold of an agent differs from that in the THRESHOLD message sent just before a TERMINATE message, the termination condition of the agent can be satisfied. Therefore, the agent can terminate with a suboptimal value.

3.3.3. Occurrence in variants

In ADOPT-N and IDB-ADOPT, this counterexample occurs for the same reasons as the flaws previously shown. Moreover, BnB-ADOPT, BnB-ADOPT$^+$, ADOPT(k), and BD-ADOPT all have similar flaws. The causes of them are that agents do not send their current contexts in VALUE messages, which include thresholds, or TERMINATE messages. Thus, the agents can terminate with incorrect contexts.

The incorrectness of contexts means that the statements of the proofs in the studies of ADOPT(k) and BD-ADOPT are not proper. The statements imply that agents obtain the optimal costs for their own current contexts, which may be incorrect at termination. The study of BnB-ADOPT also presents an improper statement. It implies that only the root agent obtains the optimal cost at termination, but the other agents can terminate with incorrect contexts, resulting in a suboptimal solution.

By contrast, ADOPT$^+$ does not have this flaw because it is obtained by fixing the causes of the flaw in ADOPT. Specifically, in ADOPT$^+$, a VALUE message contains the current context and threshold of the agent, and furthermore, a time stamp for a variable value is introduced. These modifications guarantee the correct context and threshold at termination, and agents can reinitialize the bounds if the current context is changed.

3.4. Flaw in the bounded-error approximation of ADOPT

The bounded-error approximation of ADOPT also suffers from flaws similar to the counterexamples to optimality. Under the approximation, it is desired to satisfy the following inequation at the termination of the algorithm:

$$c \leq c^* + b, \tag{1}$$

where c is the obtained global cost at termination, c^* is the optimal global cost, and b is the given error bound. However, counterexamples to this property exist, which are caused by the same or similar DCOPs and traces described in Sections 3.2 and 3.3.

The counterexample in Section 3.2 can occur under the bounded-error approximation. Assume that the error bound b is 9, i.e., the threshold of the root agent x_0 is set as $TH_0 = 9$. This assumption does not affect the processes of x_1 and x_2 in Steps 1 and 2 since they only send their updated bounds to their parents. Thus, x_0 receives the COST message with $LB_1 = UB_1 = 0$ from x_1 and updates the upper bound as $UB_0 = UB_0(0) = 0$. Then, x_0 decreases threshold TH_0 to 0 due to ThresholdInvariant, and hence it terminates with $d_0 = 0$. Therefore, the final cost must be 100, which contradicts the inequation (1) as $100 > 0 + 9$, where the optimal global cost is 0.

The counterexample in Section 3.3 also causes a problem in the bounded-error approximation. Note that, unlike the counterexample in Section 3.2, the cost function in this counterexample must be modified to perform the same trace under the approximation.

For example, the minimum cost of $f_{0,4}$ is changed to be greater than or equal to the given error bound. Let us consider a setting where error bound b is 9 and $f_{0,4}(0,0) = 10$. In this case, the same trace presented in Section 3.3 can be conducted, and as a result, the obtained global cost must be greater than or equal to 21 since $d_2 = 1$ and the cost of $f_{0,2}$ (this is the virtual cost function induced by $f_{0,3}$ and $f_{2,3}$) is 11 or 100. Therefore, the obtained cost does not satisfy the inequation (1) as $21 > 11 + 9$, where the optimal global cost is 11.

3.5. Remark on ADOPT-ing

ADOPT-ing [14] is different from ADOPT and the other variants, especially in terms of termination. The study of ADOPT-ing assumes that the quiescence of all agents is observable and seen as the termination of the algorithm. However, this assumption is not realistic in many practical cases. Thus, we adopt the termination condition, or the termination detection, of ADOPT to ADOPT-ing.

Although the method to implement the termination detection is provided in the study, it requires additional modifications. Here, the termination condition of ADOPT is divided into two parts: (1) agent x_i satisfies $TH_i = UB_i$, and (2) x_i receives a TERMINATE message from its parent (or is the root agent). The modifications of ADOPT-ing also consist of two parts, each of which corresponds to one of the conditions. First, we introduce a boolean value *exact* into two notions concerning costs, namely valued nogood and cost assessment. This modification is proposed in the study of ADOPT-ing. Here, *exact* indicates whether the cost in a valued nogood or a cost assessment is exact, which corresponds to equation $LB_i = UB_i$ in ADOPT. This value is sent to the parent of the agent, and the parent also uses the value to calculate its own *exact*. Second, we also introduce a boolean value *terminate* into an ok? message, which corresponds to a VALUE message in ADOPT. The value *terminate* is set to true if the agent satisfies the termination condition. This value enables an ok? message to work as a TERMINATE message in ADOPT. With the two modifications above, ADOPT-ing is allowed to use a termination condition similar to that of ADOPT, which is as follows: (1) agent x_i has a cost assessment with the minimum cost and $exact = true$,[5] and (2) x_i receives an ok? message with $terminate = true$ from its parent (or is the root agent). We refer to this modified version of ADOPT-ing as ADOPT-ing(ATC).

We analyze the existence of flaws in ADOPT-ing(ATC) similarly to the discussions in Sections 3.1–3.3. Note that the cost functions must be modified since ADOPT-ing assumes they have non-zero and positive costs, and thus let us assume that the same positive constant is added to each cost in the analysis below. The first two counterexamples, i.e., the one to termination and the one to optimality due to initialization, do not occur because of the following reasons. In ADOPT-ing(ATC), the cost of cost assessment $h[d]$ for variable value d, which corresponds to the lower bound in ADOPT, monotonically increases. Thus, the counterexample in Section 3.1 does not occur. Additionally, because of the definition of *exact*, agents cannot satisfy the termination condition with underestimated costs. Therefore, the counterexample in Section 3.2 also does not occur.

On the contrary, the counterexample to termination due to TERMINATE messages (ok? messages in ADOPT-ing(ATC)), described in Section 3.3, can occur. The trace is similar to the one presented in the section. First, agents calculate the costs in the case where $d_0 = 0$, and then x_0 changes its value to 1. Subsequently, x_4 sends a cost of $f_{0,4}(1,0)$ to x_0, and as a result, x_0 returns its value to 0, sends ok? messages with $d_0 = 0$ and $exact = true$ to the lower agents, and terminates. Next, x_2 and x_3 receive $d_0 = 1$ through the ok? messages from x_0 but still do not receive $d_0 = 0$. They calculate the costs under $d_0 = 1$, and x_2 updates its value as $d_2 = 1$. Afterwards, x_1 receives the ok? message from x_0 with $d_0 = 0$ and $terminate = true$. Since x_1 obtains a valued nogood with $exact = true$ from the valued nogood in the ok? message, which corresponds to a threshold in ADOPT, x_1 satisfies the termination condition and sends an ok? message with $terminate = true$ to x_2, and then the agent terminates. When x_2 receives the ok? message, x_2 also satisfies the termination condition without changing its value d_2 from 1 because the agent has already had the minimum cost with $exact = true$ calculated under $d_0 = 1$. However, $d_2 = 1$ is suboptimal, and hence this contradicts optimality. Similarly to the counterexample in ADOPT, this flaw occurs because the cost for an incompatible context can satisfy the termination condition. Therefore, the termination process in ADOPT is critical and must be modified to guarantee the termination condition is satisfied with the optimal cost for the correct context.

4. Amendments to ADOPT

In this section, we propose an amended version of ADOPT to avoid the counterexamples. The amendments consist of three parts, each of which corresponds to one of the causes described in Section 3: the nonmonotonicity of bounds, the initialization of a current context, and TERMINATE messages and their receiving procedure. We present the pseudocode of the proposed version in Algorithm 2 only for the modified part; the other part is the same as Algorithm 1.

First, we modify the update rules for the lower and upper bounds $lb_i(d, x_c)$ and $ub_i(d, x_c)$ for a child so as to change monotonically (lines 19–20). In the amended version, when agent x_i receives a COST message from child x_c with bounds LB and UB and context CX that contains (x_i, d) and is compatible with CX_i, x_i updates the lower and upper bounds for d and x_c by the following:

$$lb_i(d, x_c) := \max\left\{lb_i(d, x_c), LB\right\},$$
$$ub_i(d, x_c) := \min\left\{ub_i(d, x_c), UB\right\}.$$

The other calculations of the lower and upper bounds (i.e., $LB_i(d), UB_i(d), LB_i$, and UB_i) are not changed from the original definitions.

[5] More precisely, cost assessment $h[d]$ whose cost is equal to $\min_{d'}\left\{cost(h[d'])\right\}$ contains $exact = true$.

Algorithm 2 Pseudocode of the amended version of ADOPT.

```
 1: procedure Start()
 2:     TH_i := 0;
 3:     CX_i := {(x_p, ValInit(x_p)) | x_p ∈ SCP(x_i)};
 4:     for all d ∈ D_i, x_c ∈ C(x_i) do
 5:         InitChild(d, x_c);
 6:     d_i := arg min_{d∈D_i} LB_i(d);
 7:     MaintainThresholdInvariant();
 8:     Backtrack();
 9: procedure Received(COST, x_c, CX, LB, UB)
10:     d := d'_i   s.t. (x_i, d'_i) ∈ CX;
11:     remove (x_i, d) from CX;
12:     if TERMINATE not received from parent then
13:         for all (x_j, d_j) ∈ CX and x_j is not x_i's neighbor do
14:             add (x_j, d_j) to CX_i (and remove the old assignment of x_j);
15:         for all d' ∈ D_i, x_c' ∈ C(x_i) do
16:             if ¬Compatible(CX_i, cx_i(d', x_c')) then
17:                 InitChild(d', x_c');
18:     if Compatible(CX, CX_i) then
19:         lb_i(d, x_c) := max{lb_i(d, x_c), LB};
20:         ub_i(d, x_c) := min{ub_i(d, x_c), UB};
21:         cx_i(d, x_c) := CX;
22:         MaintainChildThresholdInvariant();
23:         MaintainThresholdInvariant();
24:     Backtrack();
25: procedure Received(TERMINATE, th, CX)
26:     record TERMINATE received;
27:     CX_i := CX
28:     TH_i := th;
29:     for all d ∈ D_i, x_c ∈ C(x_i) do
30:         if ¬Compatible(CX_i, cx_i(d, x_c)) then
31:             InitChild(d, x_c);
32:     Backtrack();
33: procedure Backtrack()
34:     if TH_i = UB_i then
35:         d_i := arg min_{d∈D_i} UB_i(d) (choose the previous d_i if possible);
36:     else if LB_i(d_i) > TH_i then
37:         d_i := arg min_{d∈D_i} LB_i(d);
38:     Send(VALUE, x_i, d_i) to each x_c ∈ CD(x_i);
39:     MaintainAllocationInvariant();
40:     if TH_i = UB_i then
41:         if TERMINATE received or x_i is root then
42:             Send(TERMINATE, th_i(d_i, x_c),
        CX_i ∪ {(x_i, d_i)}) to each x_c ∈ C(x_i);
43:             terminate execution;
44:     Send(COST, x_i, CX_i, LB_i, UB_i) to pa(x_i);
```

Second, we modify the initialization of a current context so as to store the assignments of all higher neighbors (line 3). Agent x_i initializes the current context as follows:

$$CX_i := \{(x_p, ValInit(x_p)) \mid x_p \in SCP(x_i)\},$$

where $ValInit(x_p) \in D_p$ is the initial value for x_p. Note that for x_i to avoid flaws, it is sufficient to initialize the current context only with the assignments of the parent and pseudo-parents of x_i. However, our amended version uses the assignments of all agents in $SCP(x_i)$ since the current context CX_i eventually stores their assignments. The two aforementioned amendments are already introduced in BnB-ADOPT, BnB-ADOPT$^+$, ADOPT(k), and BD-ADOPT.

Finally, we modify TERMINATE messages (lines 25 and 42) and their receiving procedure (lines 28–31). The modified procedure when agent x_i receives a TERMINATE message from the parent x_p is as follows: if the context $cx_i(d, x_c)$ for value d and child x_c is incompatible with the current context CX_i updated by the TERMINATE message, x_i reinitializes the lower and upper bounds $lb_i(d, x_c)$ and $ub_i(d, x_c)$, threshold $th_i(d, x_c)$, and context $cx_i(d, x_c)$. Additionally, we attach the threshold $th_i(d_i, x_c)$ for the current value d_i and child x_c to the TERMINATE message sent to x_c. These modifications guarantee the threshold of an agent after receiving a TERMINATE message to be correct, which is proven by a theorem in the next section. Therefore, the termination condition is satisfied with the correct threshold and optimal value.

As mentioned above, these amendments fix the causes of the counterexamples in Section 3. In the trace of the counterexample demonstrated in Section 3.1, the first amendment prevents the lower bounds of x_1 and x_2 from decreasing. Thus, the thresholds allocated to them are also not decreased, the repeated changes of the variable values do not occur, and finally, all the agents satisfy the termination condition. The second amendment avoids the counterexample in Section 3.2 since x_2's current context at initialization contains the assignments sufficient to prevent the underestimations of the local cost and bounds. Furthermore, the third amendment also avoids the counterexample presented in Section 3.3. When x_2 receives the TERMINATE message from x_1 in Step 6, x_2 updates its threshold as $TH_2 = 1$ and its current context as $CX_2 = \{(x_0, 0), (x_1, 0)\}$, both of which are contained in the message. Additionally, x_2 reinitializes its bounds since the current context is changed. Afterwards, the threshold and current context are never changed, and x_2 eventually satisfies the termination condition with the correct threshold and optimal value.

The complexity of the amended version is the same as that of the original ADOPT since the amendments affect neither the search strategy nor the data structure stored by agents of ADOPT. Therefore, the time complexity is exponential in the number of variables n, while the space complexity is polynomial in n.

5. Proof of termination and optimality for the amended version of ADOPT

We prove the termination and optimality of the amended version of ADOPT. The arguments of the proofs are based on the study of BnB-ADOPT [11]. Briefly, Lemmata 1–5 and Corollary 1 prove properties related to an agent's optimal cost, context, and bounds. Lemmata 6–8 show half of the termination condition (i.e., $TH_i = UB_i$; the other is to receive a TERMINATE message) eventually holds. These lemmata lead to proving the termination of the algorithm in Theorem 1. Subsequently, Lemma 9 shows a property of the threshold of the root agent, and Theorem 2 proves the optimality.

We modify some statements and arguments from the proofs presented in the previous study because of the differences between the amended version of ADOPT and BnB-ADOPT, e.g., threshold TH_i is more crucial for ADOPT's termination than BnB-ADOPT. Ad-

ditionally, we also modify the statement of Theorem 2 to guarantee that all agents obtain the optimal values and costs at termination since the study of BnB-ADOPT only guarantees that the root agent obtains the optimal cost.

As mentioned in Section 3, we assume that message transfer is based on a cycle. ϵ is the largest duration of a cycle, i.e., the largest duration between receiving a message and sending new messages. Additionally, Δ is the largest duration between sending a message and processing it.

Lemma 1 implies that compatible contexts containing sufficient assignments give equal optimal costs of an agent. Although this may seem trivial, the premise in the lemma is important since two compatible contexts can potentially give different costs if one lacks some assignments contained in the other.

Lemma 1. *If two contexts CX and CX' of any agent $x_i \in X$ contain the values of all agents $x_p \in SCP(x_i)$, and if these values coincide for each $x_p \in SCP(x_i)$, then $\gamma_i(CX) = \gamma_i(CX')$.*

Proof. By induction on the height (denoted by k) of the subtree rooted at x_i.

Base Case (for $k = 0$). Consider the case where x_i is a leaf agent. By definition,

$$\begin{aligned}\gamma_i(CX) &= \min_{d \in D_i} \gamma_i(d, CX) \\ &= \min_{d \in D_i} \delta_i(d, CX).\end{aligned}$$

Since $\delta_i(d, CX)$ is the local cost between x_i and its higher neighbors, $\delta_i(d, CX)$ is determined by d and the values of $x_p \in CP(x_i) \subseteq SCP(x_i)$ contained in CX. By assumption, given CX contains all the values of $x_p \in SCP(x_i)$,

$$\gamma_i(CX) = \min_{d \in D_i} \left(\sum_{x_p \in CP(x_i)} f_{i,p}(d, d_p) \right)$$

such that $(x_p, d_p) \in CX$. Similarly,

$$\gamma_i(CX') = \min_{d \in D_i} \left(\sum_{x_p \in CP(x_i)} f_{i,p}(d, d'_p) \right)$$

such that $(x_p, d'_p) \in CX'$. By assumption, the values of any $x_p \in SCP(x_i)$ in CX and CX' coincide. Thus, for any value $d \in D_i$,

$$\sum_{x_p \in CP(x_i)} f_{i,p}(d, d_p) = \sum_{x_p \in CP(x_i)} f_{i,p}(d, d'_p).$$

Therefore, $\gamma_i(CX) = \gamma_i(CX')$.

Indution Step (for $k > 0$). By definition and a similar argument as in the base case,

$$\begin{aligned}\gamma_i(CX) &= \min_{d \in D_i} \gamma_i(d, CX) \\ &= \min_{d \in D_i} \left(\delta_i(d, CX) + \sum_{x_c \in C(x_i)} \gamma_c \left(CX \cup \{(x_i, d)\} \right) \right) \\ &= \min_{d \in D_i} \left(\sum_{x_p \in CP(x_i)} f_{i,p}(d, d_p) + \sum_{x_c \in C(x_i)} \gamma_c \left(CX \cup \{(x_i, d)\} \right) \right)\end{aligned}$$

such that $(x_p, d_p) \in CX$. Similarly,

$$\gamma_i(CX') = \min_{d \in D_i} \left(\sum_{x_p \in CP(x_i)} f_{i,p}(d, d'_p) + \sum_{x_c \in C(x_i)} \gamma_c \left(CX' \cup \{(x_i, d)\} \right) \right)$$

such that $(x_p, d'_p) \in CX'$. By assumption, the values contained in CX and CX' coincide for any $x_p \in SCP(x_i)$. Thus, for any value $d \in D_i$,

$$\sum_{x_p \in CP(x_i)} f_{i,p}(d, d_p) = \sum_{x_p \in CP(x_i)} f_{i,p}(d, d'_p).$$

Also, for any value $d \in D_i$ and any child $x_c \in C(x_i)$, both $CX \cup \{(x_i, d)\}$ and $CX' \cup \{(x_i, d)\}$ contain the value for any $x_{p'} \in SCP(x_c) \subseteq SCP(x_i) \cup \{x_i\}$, and $CX \cup \{(x_i, d)\}$ and $CX' \cup \{(x_i, d)\}$ coincide for the values of any $x_{p'} \in SCP(x_c)$. Thus, by the induction hypothesis,

$$\gamma_c\left(CX \cup \{(x_i, d)\}\right) = \gamma_c\left(CX' \cup \{(x_i, d)\}\right).$$

Therefore, $\gamma_i(CX) = \gamma_i(CX')$. □

Lemma 2 proves the monotonicity of lower and upper bounds.

Lemma 2. *If context CX_i does not change for any agent $x_i \in X$ from a time T to a time $t \geq T$, then for any time t' such that $T \leq t' \leq t$, lower bounds $lb_i(d, x_c)$, $LB_i(d)$, and LB_i and upper bounds $ub_i(d, x_c)$, $UB_i(d)$, and UB_i are monotonically non-decreasing and monotonically non-increasing, respectively, for any value $d \in D_i$ and any child $x_c \in C(x_i)$.*

Proof. Since CX_i does not change at any time t' such that $T \leq t' \leq t$, the local cost $\delta_i(d, CX_i)$ also does not change for any $d \in D_i$. In addition, for any $d \in D_i$ and any $x_c \in C(x_i)$, the lower and upper bounds are updated by the following equations after the initialization:

$$\begin{aligned}
lb_i(d, x_c) &:= \max\left\{lb_i(d, x_c), LB\right\},\\
ub_i(d, x_c) &:= \min\left\{ub_i(d, x_c), UB\right\},\\
LB_i(d) &:= \delta_i(d, CX_i) + \sum_{x_c \in C(x_i)} lb_i(d, x_c),\\
UB_i(d) &:= \delta_i(d, CX_i) + \sum_{x_c \in C(x_i)} ub_i(d, x_c),\\
LB_i &:= \min_{d \in D_i} LB_i(d),\\
UB_i &:= \min_{d \in D_i} UB_i(d),
\end{aligned}$$

where LB and UB are the bounds received via a COST message from x_c. Therefore, the lower and upper bounds are monotonically non-decreasing and monotonically non-increasing, respectively, for any $d \in D_i$ and any $x_c \in C(x_i)$. □

Next, we define correctness of the current context of an agent, which implies that the context contains sufficient assignments and each of the assignments equals the value taken by the corresponding agent. Especially, this notion is crucial to proving the optimality in Theorem 2 because it must guarantee obtaining the optimal cost for a proper current context.

Definition 5. The current context CX_i of agent x_i is correct iff CX_i contains the values of all agents in $SCP(x_i)$, and the values of all agents in CX_i are equal to the values taken by the agents, i.e.,

$$\forall_{x_p \in SCP(x_i)} \exists_{d \in D_p}\left((x_p, d) \in CX_i\right) \wedge \forall_{(x'_j, d'_j) \in CX_i} \exists_{x_j \in X}\left(x_j = x'_j \wedge d_j = d'_j\right).$$

Lemma 3 proves that if a higher agent of agent x_i does not change its value during a certain period, the corresponding assignment in x_i's current context is equal to that value from some point to the end of the period.

Lemma 3. *If the value of any agent $x_p \in SCP(x_i)$ for any agent $x_i \in X$ does not change from a time T to a time t such that $T + |X| \cdot (\Delta + \epsilon) \leq t$, then the value of x_p contained in the context of x_i is equal to the value taken by x_p between some time $t' \leq t$ and t.*

Proof.

Case 1. Consider the case where x_p is a parent or a pseudo-parent of x_i. Since x_p took its current value before or at time T and the duration of one cycle is less than or equal to ϵ, x_p sends a VALUE message containing the value to x_i at a time t'' such that $t'' \leq T + \epsilon$. Also, since the maximum delay for a message to be processed is Δ, x_i receives and processes the VALUE message before time $t'' + \Delta$. Then, the value of x_p contained in the context of x_i is updated. Thus, there exists a time t' with $t'' \leq t' \leq t'' + \Delta \leq T + \Delta + \epsilon$ such that the value of x_p contained in the context of x_i is equal to the value taken by x_p.

Case 2. Consider the case where x_p is neither the parent nor the pseudo-parent of x_i. The value of x_p is propagated from lower to higher agents via COST messages or from higher to lower agents via TERMINATE messages. First, we consider the propagation of COST messages. Since $x_p \in SCP(x_i)$, there exist descendants of x_i that are pseudo-children of x_p. Among them, let x_c be the highest agent in the DFS pseudo-tree. As in Case 1, x_p sends a VALUE message containing its own value to x_c at a time t'' such that $t'' \leq T + \epsilon$, and then x_c receives and processes this message before time $t'' + \Delta$. At this point, the value of x_p contained in the context of x_c is updated, and then x_c sends a COST message containing the updated context to $pa(x_c)$ before $t'' + \Delta + \epsilon$. Subsequently, $pa(x_c)$ receives and processes the message before $t'' + 2\Delta + \epsilon$. At this point, the value of x_p contained in the context of $pa(x_c)$ is updated, and then $pa(x_c)$ sends a COST message containing the updated context to $pa(pa(x_c))$ before $t'' + 2 \cdot (\Delta + \epsilon)$. A similar process continues until the value of x_p contained in the context of x_i is updated. Specifically, for the number (denoted by $k \leq |X|$) of chains of message passing, the context of x_i is updated by

time $t'' + k \cdot \Delta + (k-1) \cdot \epsilon$. Therefore, from time t' to time t such that $t'' \leq t' \leq t'' + k \cdot \Delta + (k-1) \cdot \epsilon \leq T + |X| \cdot (\Delta + \epsilon) \leq t$, the value of x_p contained in the context of x_i is equal to the value taken by x_p. Furthermore, the case of the propagation of TERMINATE messages follows the argument above since the number of chains of message passing is less than $|X|$. □

Although Corollary 1 can be mostly derived from Lemma 3, it should be noted that the current context of agent x_i can contain the values of the higher agents not in $SCP(x_i)$. These values are contained in the current context only after x_i receives a TERMINATE message from its parent. Here, the higher agents do not change their own values after sending their TERMINATE messages, which must be sent before x_i receives the TERMINATE message from its parent, and x_i also does not change the values in its current context after receiving the message. Thus, the correctness of the current context only depends on the values of the agents in $SCP(x_i)$, which is discussed in Lemma 3.

Corollary 1. *If the values of all agents $x_p \in SCP(x_i)$ for any agent $x_i \in X$ do not change between a time T and a time t such that $T + |X| \cdot (\Delta + \epsilon) \leq t$, then the context of x_i is correct between some time $t' \leq t$ and t.*

Lemmata 4 and 5 show invariants between the bounds and optimal costs of agents.

Lemma 4. *For any agent $x_i \in X$, any value $d \in D_i$, and any child $x_c \in C(x_i)$, if $LB_c \leq \gamma_c(CX_c) \leq UB_c$ holds at any time, then $lb_i(d, x_c) \leq \gamma_c(CX_i \cup \{(x_i, d)\}) \leq ub_i(d, x_c)$ also holds at any time.*

Proof. By induction on the number (denoted by $k \geq 0$) of times that agent x_i changes its context or updates its bounds $lb_i(d, x_c)$ and $ub_i(d, x_c)$ for any $d \in D_i$ and any $x_c \in C(x_i)$ after x_i (re)initializes the bounds. (We use "initialize" and "reinitialize" interchangeably below.)

Base Case (for $k = 0$). In this case, x_i initializes $lb_i(d, x_c)$ and $ub_i(d, x_c)$, where

$$\begin{aligned} lb_i(d, x_c) &= 0 \\ &\leq \gamma_c\left(CX_i \cup \{(x_i, d)\}\right) \\ &\leq \infty \\ &= ub_i(d, x_c). \end{aligned}$$

Induction Step (for $k > 0$). For the case where x_i changes the context or updates $lb_i(d, x_c)$ and $ub_i(d, x_c)$ for any $d \in D_i$ and any $x_c \in C(x_i)$ without reinitializing these bounds, there are the following two cases: one is when the context is changed, i.e., when x_i receives a VALUE or TERMINATE message, or receives a COST message and executes lines 12–14 of Algorithm 2; the other is when the bounds are updated, i.e., when x_i receives a COST message and executes lines 18–20 of Algorithm 2.

Case 1. Let us assume x_i receives either VALUE, COST, or TERMINATE message, and then changes the context from CX_i to $\hat{CX}_i$ without reinitializing the bounds. This case is divided into the following subcases (i) and (ii).

(i) Consider the case where x_i has not updated $lb_i(d, x_c), ub_i(d, x_c)$, and $cx_i(d, x_c)$ since their initialization, in other words, x_i has only changed its context. As in the base case,

$$\begin{aligned} lb_i(d, x_c) &= 0 \\ &\leq \gamma_c\left(\hat{CX}_i \cup \{(x_i, d)\}\right) \\ &\leq \infty \\ &= ub_i(d, x_c). \end{aligned}$$

(ii) Consider the case where x_i has updated $lb_i(d, x_c), ub_i(d, x_c)$, and $cx_i(d, x_c)$ at least once since their initialization. Since $lb_i(d, x_c)$ and $ub_i(d, x_c)$ are not reinitialized, $cx_i(d, x_c)$ is compatible with CX_i and $\hat{CX}_i$. Moreover, by the initialization of the context of x_i, each CX_i and $\hat{CX}_i$ contains the values of all $x_p \in SCP(x_i)$. Similarly, since CX_c contains the values of all $x_{p'} \in SCP(x_c) \subseteq SCP(x_i) \cup \{x_i\}$, $cx_i(d, x_c)$ contains the values of all $x_{p'} \in SCP(x_c) \backslash \{x_i\}$. Therefore, each $CX_i \cup \{(x_i, d)\}$ and $\hat{CX}_i \cup \{(x_i, d)\}$ contains the values of all $x_{p'} \in SCP(x_c)$, and the values of any $x_{p'} \in SCP(x_c)$ contained in $CX_i \cup \{(x_i, d)\}$ and $\hat{CX}_i \cup \{(x_i, d)\}$ coincide. Thus, by Lemma 1 and the induction hypothesis,

$$\begin{aligned} lb_i(d, x_c) &\leq \gamma_c\left(CX_i \cup \{(x_i, d)\}\right) \\ &= \gamma_c\left(\hat{CX}_i \cup \{(x_i, d)\}\right) \\ ub_i(d, x_c) &\geq \gamma_c\left(CX_i \cup \{(x_i, d)\}\right) \\ &= \gamma_c\left(\hat{CX}_i \cup \{(x_i, d)\}\right). \end{aligned}$$

Case 2. Let us assume x_i receives a COST message with bounds LB_c and UB_c and context CX_c from child x_c, and then updates $lb_i(d, x_c)$ and $ub_i(d, x_c)$ to $\hat{lb}_i(d, x_c)$ and $\hat{ub}_i(d, x_c)$, respectively, without reinitialization. In this case, $CX_i \cup \{(x_i, d)\}$ is compatible with CX_c in the COST message. Moreover, by the initialization of the contexts of x_i and x_c, $CX_i \cup \{(x_i, d)\}$ and CX_c contain the values of all $x_{p'} \in SCP(x_c) \subseteq SCP(x_i) \cup \{x_i\}$, and the values of any $x_{p'} \in SCP(x_c)$ contained in $CX_i \cup \{(x_i, d)\}$ and CX_c coincide. Thus, by Lemma 1 and the induction hypothesis,

$$\begin{aligned}
\hat{lb}_i(d, x_c) &= \max\left\{lb_i(d, x_c), LB_c\right\} \\
&\leq \max\left\{\gamma_c\left(CX_i \cup \left\{(x_i, d)\right\}\right), \gamma_c(CX_c)\right\} \\
&= \max\left\{\gamma_c\left(CX_i \cup \left\{(x_i, d)\right\}\right), \gamma_c\left(CX_i \cup \left\{(x_i, d)\right\}\right)\right\} \\
&= \gamma_c\left(CX_i \cup \left\{(x_i, d)\right\}\right) \\
\hat{ub}_i(d, x_c) &= \min\left\{ub_i(d, x_c), UB_c\right\} \\
&\geq \min\left\{\gamma_c\left(CX_i \cup \left\{(x_i, d)\right\}\right), \gamma_c(CX_c)\right\} \\
&= \min\left\{\gamma_c\left(CX_i \cup \left\{(x_i, d)\right\}\right), \gamma_c\left(CX_i \cup \left\{(x_i, d)\right\}\right)\right\} \\
&= \gamma_c\left(CX_i \cup \left\{(x_i, d)\right\}\right).
\end{aligned}$$

Therefore, for any value $d \in D_i$ and any child $x_c \in C(x_i)$, $lb_i(d, x_c) \leq \gamma_c(CX_i \cup \{(x_i, d)\}) \leq ub_i(d, x_c)$ always holds. □

Lemma 5. *For any value $d \in D_i$ of any agent $x_i \in X$, $LB_i(d) \leq \gamma_i(d, CX_i) \leq UB_i(d)$ and $LB_i \leq \gamma_i(CX_i) \leq UB_i$ hold at any time.*

Proof. By induction on the height (denoted by k) of the subtree rooted at agent x_i.

Base Case (for $k = 0$). For the case where x_i is a leaf agent. For any value $d \in D_i$,

$$\begin{aligned}
LB_i(d) &= \delta_i(d, CX_i) \\
&= \gamma_i(d, CX_i) \\
UB_i(d) &= \delta_i(d, CX_i) \\
&= \gamma_i(d, CX_i).
\end{aligned}$$

Thus, for any value $d \in D_i$, $LB_i(d) \leq \gamma_i(d, CX_i) \leq UB_i(d)$ always holds. Also,

$$\begin{aligned}
LB_i &= \min_{d \in D_i} LB_i(d) \\
&= \min_{d \in D_i} \gamma_i(d, CX_i) \\
&= \gamma_i(CX_i) \\
UB_i &= \min_{d \in D_i} UB_i(d) \\
&= \min_{d \in D_i} \gamma_i(d, CX_i) \\
&= \gamma_i(CX_i).
\end{aligned}$$

Thus, $LB_i \leq \gamma_i(CX_i) \leq UB_i$ always holds.

Induction Step (for $k > 0$). By the induction hypothesis and Lemma 4, for any $d \in D_i$,

$$\begin{aligned}
LB_i(d) &= \delta_i(d, CX_i) + \sum_{x_c \in C(x_i)} lb_i(d, x_c) \\
&\leq \delta_i(d, CX_i) + \sum_{x_c \in C(x_i)} \gamma_c\left(CX_i \cup \left\{(x_i, d)\right\}\right) \\
&= \gamma_i(d, CX_i) \\
UB_i(d) &= \delta_i(d, CX_i) + \sum_{x_c \in C(x_i)} ub_i(d, x_c) \\
&\geq \delta_i(d, CX_i) + \sum_{x_c \in C(x_i)} \gamma_c\left(CX_i \cup \left\{(x_i, d)\right\}\right) \\
&= \gamma_i(d, CX_i).
\end{aligned}$$

Thus, for any $d \in D_i$, $LB_i(d) \leq \gamma_i(d, CX_i) \leq UB_i(d)$ always holds. Also,

$$\begin{aligned} LB_i &= \min_{d \in D_i} LB_i(d) \\ &\leq \min_{d \in D_i} \gamma_i(d, CX_i) \\ &= \gamma_i(CX_i) \\ UB_i &= \min_{d \in D_i} UB_i(d) \\ &\geq \min_{d \in D_i} \gamma_i(d, CX_i) \\ &= \gamma_i(CX_i). \end{aligned}$$

Thus, $LB_i \leq \gamma_i(CX_i) \leq UB_i$ always holds. □

Additionally, we define a notion of an agent's potential, which is the sum of the differences between the upper and lower bounds. As shown in Lemma 6 below, the potential is monotonically non-increasing if the current context does not change.

Definition 6. For agent $x_i \in X$, the potential of x_i is $\sum_{d \in D_i} \left(UB_i(d) - LB_i(d)\right)$.

Lemma 6. *If the context CX_i of any agent $x_i \in X$ does not change after a time t, then the potential of the agent is monotonically non-increasing and decreases by more than a positive constant every time the agent changes its value after t.*

Proof. Since CX_i does not change after t, by Lemma 2, the lower bound $LB_i(d)$ is monotonically non-decreasing, and the upper bound $UB_i(d)$ is monotonically non-increasing for all $d \in D_i$. Therefore, the potential of x_i is monotonically non-increasing. In addition, x_i changes the value d_i to a new value $\hat{d_i}$ only if $LB_i(\hat{d_i}) = \min_{d \in D_i} LB_i(d) < LB_i(d_i)$ or $UB_i(\hat{d_i}) = \min_{d \in D_i} UB_i(d) < UB_i(d_i)$. Thus, $LB_i(d_i)$ increases or $UB_i(\hat{d_i})$ decreases when d_i changes to $\hat{d_i}$. Therefore, the potential of x_i decreases by more than a positive constant when x_i changes its value after t. □

Lemmata 7 and 8 prove that agents change their values only a finite number of times and eventually satisfy half of the termination condition.

Lemma 7. *All agents change their values only a finite number of times.*

Proof. Assume that agent $x_i \in X$ infinitely changes its value. Since $SCP(x_r) = \varnothing$ holds for the root agent x_r and we can prove the lemma by the induction on the depth of x_i in the DFS pseudo-tree based on the argument below, without loss of generality, we can assume that all agents $x_p \in SCP(x_i)$ change their values only a finite number of times. By this assumption, there exists a time t such that all $x_p \in SCP(x_i)$ do not change their values after t. Thus, by Corollary 1, there exists a time $t' \geq t$ such that the context CX_i of x_i is correct and the agent does not change the values in the context after t'. Furthermore, by Lemma 6, every time agent x_i changes its value afterwards, its potential decreases by more than a positive constant. By assumption, since x_i infinitely changes the value, the potential of x_i decreases towards $-\infty$. However, by Lemma 5, $LB_i(d) \leq UB_i(d)$ holds for all $d \in D_i$. Thus, the potential of x_i cannot become negative, which is a contradiction. Therefore, all agents change their values only a finite number of times. □

Lemma 8. *There exists a time t such that $TH_i = UB_i$ holds for all agents x_i after t.*

Proof. By induction on the height (denoted by k) of the subtree rooted at x_i.

Base Case (for $k = 0$). For the case where x_i is a leaf agent. By definition,

$$\begin{aligned} LB_i &= \min_{d \in D_i} LB_i(d) = \min_{d \in D_i} \delta_i(d, CX_i) \\ UB_i &= \min_{d \in D_i} UB_i(d) = \min_{d \in D_i} \delta_i(d, CX_i). \end{aligned}$$

Thus, $LB_i = UB_i$. Also, by ThresholdInvariant, $LB_i \leq TH_i \leq UB_i$. Therefore, $TH_i = UB_i$ always holds.

Induction Step (for $k > 0$). By the induction hypothesis, there exists a time t_0 such that for all x_i's descendants x_s, $TH_s = UB_s$ holds after t_0. Also, by Lemma 7, there exists a time t_1 such that any agent x_j does not change its value d_j after t_1. Thus, by Corollary 1, there exists a time $t_2 \geq t_1$ such that the context CX_j of any x_j is correct and does not change after t_2. Additionally, for the following reasons (1)–(3), there exists a time $t_3 \geq t_2$ such that for any x_j and for all its values $d \in D_j$ and all its children $x_c \in C(x_j)$, $lb_j(d, x_c), LB_j(d), LB_j, ub_j(d, x_c), UB_j(d)$, and UB_j do not change after t_3.

(1) By Lemma 2, the lower bounds $lb_j(d, x_c), LB_j(d)$, and LB_j are monotonically non-decreasing, and the upper bounds $ub_j(d, x_c), UB_j(d)$, and UB_j are monotonically non-increasing.

(2) By Lemma 5, $LB_j(d) \leq \gamma_j(d, CX_j) \leq UB_j(d)$ and $LB_j \leq \gamma_j(CX_j) \leq UB_j$.
(3) By Lemma 4, $lb_j(d, x_c) \leq ub_j(d, x_c)$.

In the following, we prove by contradiction, i.e., let us assume that for any $t \geq t_0$, there exists a time $t' \geq t$ such that $TH_i \neq UB_i$ holds at t'. By this assumption, x_i and its descendants never terminate. Thus, all $x_c \in C(x_i)$ send COST messages containing LB_c, UB_c, and $CX_c \ni \{(x_i, d_i)\}$ to x_i after t_0 and t_3, and x_i receives all of the messages at a time t_4 such that $t_4 \geq t_0$ and $t_4 \geq t_3$. After t_4, since the bounds of x_i and x_c do not change and the contexts CX_c in the COST messages are compatible with CX_i, for all $x_c \in C(x_i)$,

$$\begin{aligned} lb_i(d_i, x_c) &= \max\left\{lb_i(d_i, x_c), LB_c\right\} \\ &\geq LB_c, \\ ub_i(d_i, x_c) &= \min\left\{ub_i(d_i, x_c), UB_c\right\} \\ &\leq UB_c. \end{aligned}$$

Also, by the assumption, $TH_i \neq UB_i$ holds at some $t' \geq t_4$. Here, by definition, ThresholdInvariant, and the induction hypothesis,

$$\begin{aligned} TH_i &< UB_i \\ &= \min_{d \in D_i} UB_i(d) \\ &\leq UB_i(d_i) \\ &= \delta_i(d_i, CX_i) + \sum_{x_c \in C(x_i)} ub_i(d_i, x_c) \\ &\leq \delta_i(d_i, CX_i) + \sum_{x_c \in C(x_i)} UB_c \\ &= \delta_i(d_i, CX_i) + \sum_{x_c \in C(x_i)} TH_c. \end{aligned}$$

Thus, by AllocationInvariant, that is, $TH_i = \delta_i(d_i, CX_i) + \sum_{x_c \in C(x_i)} th_i(d_i, x_c)$, for some $x_{c'} \in C(x_i)$,

$$th_i(d_i, x_{c'}) < TH_{c'}.$$

Furthermore, by ChildThresholdInvariant,

$$\begin{aligned} th_i(d_i, x_{c'}) &\geq lb_i(d_i, x_{c'}) \\ &\geq LB_{c'}. \end{aligned}$$

Additionally, x_i sends a THRESHOLD message containing $th_i(d_i, x_{c'})$ to $x_{c'}$ after t'. When $x_{c'}$ receives the message, $x_{c'}$ changes $TH_{c'}$ to the received threshold $th_i(d_i, x_{c'})$ since the context CX_i in the message is compatible with $CX_{c'}$ and the threshold satisfies ThresholdInvariant. At this point, $TH_{c'} \neq UB_{c'}$ holds, which contradicts the induction hypothesis. □

Using the lemma above, Theorem 1 proves the termination of the amended version of ADOPT.

Theorem 1. *The amended version of ADOPT terminates after a finite amount of time.*

Proof. By Lemma 8, there exists a time t such that $TH_i = UB_i$ holds for all agents x_i after t. When $TH_r = UB_r$ holds at the root agent x_r, the agent sends TERMINATE messages to all $x_c \in C(x_r)$ and then terminates. These TERMINATE messages arrive at all x_c in a finite time. At that time, since $TH_c = UB_c$ also holds at all x_c, the agent sends TERMINATE messages to all $x_{c'} \in C(x_c)$ and then terminates. By the same process as above, all agents terminate after a finite amount of time. □

Lemma 9 implies that the lower bound and threshold of the root agent are always equal.

Lemma 9. *$LB_r = TH_r$ always holds at the root agent x_r.*

Proof. Since x_r does not receive THRESHOLD and TERMINATE messages, TH_r changes so as to satisfy ThresholdInvariant only if LB_r or UB_r changes when the agent receives a COST message. Furthermore, for any $d \in D_r$ and any $x_c \in C(x_r)$, $cx_r(d, x_c) = \emptyset$ always holds. Since $cx_r(d, x_c)$ is compatible with any context, the bounds $lb_r(d, x_c)$ and $ub_r(d, x_c)$ are initialized only at the beginning of the algorithm execution. In the following, we use the induction on the number (denoted by $k \geq 0$) of changes of LB_r and UB_r after x_r initializes $lb_r(d, x_c)$ and $ub_r(d, x_c)$.

Base Case (for $k = 0$). In this case, $lb_r(d, x_c)$ and $ub_r(d, x_c)$ are initialized for all $d \in D_r$ and all $x_c \in C(x_r)$. Thus, $lb_r(d, x_c) = 0$. Also, since $CX_r = \emptyset$, $\delta_r(d, CX_r) = 0$. Therefore,

$$\begin{aligned} LB_r &= \min_{d \in D_r} LB_r(d) \\ &= \min_{d \in D_r} \left(\delta_r(d, CX_r) + \sum_{x_c \in C(x_r)} lb_r(d, x_c) \right) \\ &= 0. \end{aligned}$$

Furthermore, $TH_r = 0$ holds when x_r initializes $lb_r(d, x_c)$ and $ub_r(d, x_c)$ at the beginning of the algorithm execution. Therefore, $LB_r = TH_r$.

Induction Step (for $k > 0$). Since $CX_r = \emptyset$ always holds at x_r, by Lemma 2, LB_r and UB_r are always monotonically non-decreasing and monotonically non-increasing, respectively. Thus, the following two cases should be considered.

Case 1. For the case where LB_r increases. By the procedure MaintainThresholdInvariant, TH_r is updated as $TH_r := LB_r$.

Case 2. For the case where UB_r decreases. By the induction hypothesis and Lemma 5, $LB_r = TH_r \leq UB_r$. Thus, TH_r does not change.

Therefore, $LB_r = TH_r$ always holds. □

Finally, Theorem 2 proves the optimality of the amended version of ADOPT. The theorem shows that when the algorithm terminates, any agent has the correct current context, its threshold is equal to the optimal cost for the context, and its final value is optimal. These properties are stronger than the statement of the proof in the study of BnB-ADOPT, which only shows that the bounds of the root agent at termination are equal to the optimal cost.

Theorem 2. *For any agent $x_i \in X$ when the amended version of ADOPT terminates, the current context CX_i is correct, and $TH_i = \gamma_i(d_i, CX_i) = \gamma_i(CX_i)$.*

Proof. By Theorem 1, the amended version of ADOPT terminates after a finite amount of time, and then $TH_i = UB_i$ holds for any $x_i \in X$. At that time, by $d_i = \arg\min_{d \in D_i} UB_i(d)$, $TH_i = UB_i(d_i)$ also holds. Moreover, by definition and Lemma 5,

$$\begin{aligned} \gamma_i(CX_i) &= \min_{d \in D_i} \gamma_i(d, CX_i) \\ &\leq \gamma_i(d_i, CX_i) \\ &\leq UB_i(d_i) \\ &= TH_i. \end{aligned}$$

Therefore, if $TH_i = \gamma_i(CX_i)$, then $\gamma_i(d_i, CX_i) = \gamma_i(CX_i)$. In the following, we use the induction on the depth (denoted by k) of agent x_i in the DFS pseudo-tree.

Base Case (for $k = 0$). In this case, since $SCP(x_i) = \emptyset$ and $CX_i = \emptyset$ always hold, CX_i is correct. Furthermore, by Lemma 9, $LB_i = TH_i$. Thus, when the amended version of ADOPT terminates,

$$LB_i = TH_i = UB_i.$$

By Lemma 5, $LB_i \leq \gamma_i(CX_i) \leq UB_i$. Thus, $TH_i = \gamma_i(CX_i)$ holds.

Induction Step (for $k > 0$). By the induction hypothesis, for $x_p = pa(x_i)$, $TH_p = \gamma_p(d_p, CX_p)$. Moreover, since $TH_p = UB_p(d_p)$, we have the following:

$$\begin{aligned} &\delta_p(d_p, CX_p) + \sum_{x_i \in C(x_p)} th_p(d_p, x_i) \\ =&\delta_p(d_p, CX_p) + \sum_{x_i \in C(x_p)} \gamma_i\left(CX_p \cup \{(x_p, d_p)\}\right) \\ =&\delta_p(d_p, CX_p) + \sum_{x_i \in C(x_p)} ub_p(d_p, x_i). \end{aligned}$$

Furthermore, by Lemma 4 and Lemma 5, and ChildThresholdInvariant, for all $x_i \in C(x_p)$,

$$\gamma_i\left(CX_p \cup \{(x_p, d_p)\}\right) \leq ub_p(d_p, x_i),$$

$$th_p(d_p, x_i) \leq ub_p(d_p, x_i).$$

Thus, for all $x_i \in C(x_p)$,

$$th_p(d_p, x_i) = \gamma_i\left(CX_p \cup \{(x_p, d_p)\}\right) = ub_p(d_p, x_i). \tag{2}$$

When x_p terminates at a time t, the agent sends TERMINATE messages to all $x_i \in C(x_p)$. At a time $t' \geq t$, if any x_i receives the TERMINATE message, then the agent updates the context CX_i as $CX_p \cup \{(x_p, d_p)\}$, which is contained in the TERMINATE message. By the induction hypothesis, CX_p is correct at t. Also, when x_p terminates, since all $x_{p'} \in P(x_p)$ have already terminated, the values of all $x_{p'}$ do not change after t, and thus CX_p is correct after t. Moreover, since the value of x_p does not change after t and $SCP(x_i) \subseteq SCP(x_p) \cup \{x_p\}$, and CX_i does not change after a TERMINATE message is received, CX_i is also correct after t'. Furthermore, when x_i receives the TERMINATE message from x_p, TH_i is updated. Thus, $TH_i = th_p(d_p, x_i)$. By formula (2) and Lemma 1, at t',

$$TH_i = \gamma_i\left(CX_p \cup \{(x_p, d_p)\}\right) = \gamma_i(CX_i).$$

Since x_i does not receive any THRESHOLD message and TERMINATE message after t', TH_i may change only if LB_i or UB_i changes. By Lemma 5,

$$LB_i \leq TH_i = \gamma_i(CX_i) \leq UB_i.$$

Thus, since ThresholdInvariant is always satisfied after t', TH_i does not change. Therefore, at termination, CX_i is correct, and

$$TH_i = \gamma_i(CX_i) = \gamma_i(d_i, CX_i). \quad \square$$

6. Conclusion

We presented counterexamples to the termination and optimality of ADOPT and its variants, which have been believed to be correct. The counterexamples are classified into three types, one of which is related to termination, while the other two are related to optimality. Their causes are the nonmonotonicity of bounds, the initialization of a current context, and TERMINATE messages and their receiving procedure. We also showed that the bounded-error approximation of ADOPT suffers from flaws similar to those of optimality. Additionally, we proposed an amended version of ADOPT that avoids these flaws, whose amendments correspond to the causes. Furthermore, we proved termination and optimality in the amended version of ADOPT. For future work, we plan to analyze other DCOP algorithms and develop a method to verify the properties of the algorithms automatically.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] K. Noshiro, K. Hasebe, Flaws of termination and optimality in ADOPT-based algorithms, in: E. Elkind (Ed.), Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23, International Joint Conferences on Artificial Intelligence Organization, 2023, pp. 1996–2003, main Track.

[2] P.J. Modi, W.-M. Shen, M. Tambe, M. Yokoo, Adopt: asynchronous distributed constraint optimization with quality guarantees, Artif. Intell. 161 (1–2) (2005) 149–180, https://doi.org/10.1016/j.artint.2004.09.003.

[3] G. Weiss, Multiagent Systems, 2nd edition, EBSCO Ebook Academic Collection, MIT Press, 2013.

[4] F. Fioretto, E. Pontelli, W. Yeoh, Distributed constraint optimization problems and applications: a survey, J. Artif. Intell. Res. 61 (2018) 623–698, https://doi.org/10.1613/jair.5565.

[5] R.T. Maheswaran, M. Tambe, E. Bowring, J.P. Pearce, P. Varakantham, Taking DCOP to the real world: efficient complete solutions for distributed multi-event scheduling, in: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '04, vol. 1, IEEE Computer Society, USA, 2004, pp. 310–317.

[6] M. Jain, M.E. Taylor, M. Tambe, M. Yokoo, DCOPs meet the real world: exploring unknown reward matrices with applications to mobile sensor networks, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, 2009, pp. 181–186.

[7] R.N. Lass, J.B. Kopena, E.A. Sultanik, D.N. Nguyen, C.P. Dugan, P.J. Modi, W.C. Regli, Coordination of first responders under communication and resource constraints, in: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, vol. 3, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2008, pp. 1409–1412.

[8] F. Pecora, P.J. Modi, P. Scerri, Reasoning about and dynamically posting n-ary constraints in ADOPT, in: Proceedings of Seventh Workshop on Distributed Constraint Reasoning, 2006, pp. 57–71.

[9] P. Gutierrez, P. Meseguer, Saving messages in ADOPT-based algorithms, in: AAMAS 2010 Workshop: Distributed Constraint Reasoning, Toronto, Canada, 2010, pp. 53–64.

[10] W. Yeoh, A. Felner, S. Koenig, IDB-ADOPT: a depth-first search DCOP algorithm, in: A. Oddi, F. Fages, F. Rossi (Eds.), Recent Advances in Constraints, in: Lecture Notes in Artificial Intelligence, vol. 5655, Springer, Rome, Italy, 2009, pp. 132–146.
[11] W. Yeoh, A. Felner, S. Koenig, BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm, J. Artif. Intell. Res. 38 (2010) 85–133, https://doi.org/10.1613/jair.2849.
[12] P. Gutierrez, P. Meseguer, W. Yeoh, Generalizing ADOPT and BnB-ADOPT, in: T. Walsh (Ed.), Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, 2011, pp. 554–559.
[13] Z. Chen, C. He, Z. He, M. Chen, BD-ADOPT: a hybrid DCOP algorithm with best-first and depth-first search strategies, Artif. Intell. Rev. 50 (2) (2018) 161–199, https://doi.org/10.1007/s10462-017-9540-z.
[14] M.C. Silaghi, M. Yokoo, ADOPT-ing: unifying asynchronous distributed optimization with asynchronous backtracking, Auton. Agents Multi-Agent Syst. 19 (2) (2009) 89–123, https://doi.org/10.1007/s10458-008-9069-2.
[15] M. Yokoo, E.H. Durfee, T. Ishida, K. Kuwabara, The distributed constraint satisfaction problem: formalization and algorithms, IEEE Trans. Knowl. Data Eng. 10 (5) (1998) 673–685, https://doi.org/10.1109/69.729707.