Flaws of Termination and Optimality in ADOPT-based Algorithms

Koji Noshiro and Koji Hasebe
University of Tsukuba
noshiro@mas.cs.tsukuba.ac.jp, hasebe@cs.tsukuba.ac.jp

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 owing to its superior characteristics, many 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. The flaws are classified into three types, at least one of which exists in each of ADOPT and seven of its variants that we analyzed. In other words, the algorithms may potentially not terminate or terminate with a suboptimal solution. We also 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) [Modi et al., 2005; Weiss, 2013; Fioretto et al., 2018] involve agents in a network that must coordinate the values of their variables. Many multi-agent coordination problems, such as meeting scheduling [Maheswaran et al., 2004], sensor network operation [Jain et al., 2009], and disaster management problems [Lass et al., 2008], are modeled as DCOPs. Owing to this wide range of applications, various algorithms to solve DCOPs have been proposed over the decades.

Asynchronous distributed optimization (ADOPT) [Modi et al., 2005] is a well-known DCOP algorithm. The major advantages of ADOPT are asynchrony and completeness. Asynchrony allows agents to process their computations without waiting for other agents, which enables the algorithm to use agent resources efficiently, while 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.

Many researchers have worked on improving the performance of ADOPT. While the original version could handle only binary constraints, the extension ADOPT-N [Pecora et al., 2006] is able to deal with n-ary ones. ADOPT+ [Gutierrez and Meseguer, 2010] saves redundant messages exchanged in ADOPT. IDB-ADOPT [Yeoh et al., 2009] executes ADOPT iteratively to change the search strategy from best-first to depth-first. The same authors of IDB-ADOPT proposed BnB-ADOPT [Yeoh et al., 2010], which employs depth-first branch-and-bound search. BnB-ADOPT+ [Gutierrez and Meseguer, 2010] reduces the number of redundant messages in BnB-ADOPT. Other variants include ADOPT(k) [Gutierrez et al., 2011], which generalizes ADOPT and BnB-ADOPT, and BD-ADOPT [Chen et al., 2018], which combines best-first and depth-first search. ADOPT-ing [Silaghi and Yokoo, 2009] introduces nogood, which is used in asynchronous backtrack (ABT) [Yokoo et al., 1998] 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 inappropriate to derive termination or optimality.

The objectives of this paper are threefold: to show the flaws in ADOPT and its variant algorithms, to propose an amended version of ADOPT, and to prove its termination and optimality. First, we provide counterexamples to the termination and optimality of ADOPT and show that similar flaws also exist in many of the variants. Table 1 summarizes the flaws in ADOPT and seven of its variants. They 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, that is, 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
is a solution to a DCOP and is expressed as follows:

\[ \text{which is the sum of the cost functions in} \]

variables such that it minimizes the global objective function,

edges and back edges. Tree edges exist in the spanning tree
of a constraint graph. Here, the edges are classified into tree
depth-first search (DFS) pseudo-tree extends a spanning tree
that the two connected variables share a cost function. A
definition of Agent 

A DCOP is defined as a tuple \((A, X, D, F)\). \( A = \{a_1, \ldots, a_n\}\) 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 \( x_i \). Furthermore, \( F \) is a finite set of binary cost functions. A cost function for two variables \( x_i \) and \( x_j \) is defined as \( f_{i,j} : D_i \times D_j \rightarrow \mathbb{N} \). Each cost function is known only to the agents involved. Agents aim to find an assignment \( d^* \) for all variables such that it minimizes the global objective function, which is the sum of the cost functions in \( F \). This assignment is a solution to a DCOP and is expressed as follows:

\[
d^* := \arg \min_{d \in \prod_{i=1}^{n} D_i} \sum_{f_{i,j} \in F} f_{i,j}(d_i, d_j).
\]

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 constraint graph but exist in the constraint graph and establish pseudo-parent/pseudo-child relationships.

### 2.2 ADOPT

ADOPT [Modi et al., 2005] is an asynchronous and complete algorithm for solving DCOPs. In ADOPT, agents are arranged in a DFS pseudo-tree and exchange four types of messages, namely VALUE, COST, THRESHOLD, and TERMINATE. Agents update their variable values and compute the bounds of costs through messages. Here, we only introduce some notations used in subsequent sections and review the properties of termination and optimality in the original study. A detailed description of the algorithm is presented in Appendix A.

The notations concerning the states of 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 \( \delta_i(d, CX) \) 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, the lower bound, and the upper bound, respectively, stored by \( x_i \) when it receives a COST message in which the context contains assignment \( (x_i, d) \) from child \( x_c \); \( th_i(d, x_c) \) denotes the threshold allocated to child \( x_c \) when \( x_i \) takes value \( d \). Additionally, \( \gamma_i(d, CX) \) and \( \gamma_i(CX) \) denote the optimal costs for the subtree rooted at \( x_i \) given context \( CX \).

The original study of ADOPT provides three properties (as theorems in the paper) in terms of its termination and optimality. Property 2 implies the termination, while Property 3 implies the optimality.

<table>
<thead>
<tr>
<th>Flaw</th>
<th>ADOPT</th>
<th>N + IDB</th>
<th>BnB</th>
<th>BnB+ (k)</th>
<th>BD</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(2)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(3)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 1: Flaws in ADOPT and seven variant algorithms, where N +, IDB, BnB, BnB+, (k), and BD stand for ADOPT-N, ADOPT+, IDB-ADOPT, BnB-ADOPT, BnB-ADOPT+, ADOPT(k), and BD-ADOPT, respectively. The flaws are classified into three types: (1) failure of termination; (2) failure of optimality caused by initialization; (3) failure of optimality caused by TERMINATE messages.

Finally, we prove the termination and optimality properties of our amended version of ADOPT.

This paper is organized as follows. Section 2 defines DCOPs and summarizes ADOPT. Section 3 presents the flaws 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 work.

### 2 Preliminaries

In this section, we define DCOPs and provide a summary of ADOPT.

#### 2.1 DCOP

A DCOP is defined as a tuple \((A, X, D, F)\). \( A = \{a_1, \ldots, a_n\}\) 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 \( x_i \). Furthermore, \( F \) is a finite set of binary cost functions. A cost function for two variables \( x_i \) and \( x_j \) is defined as \( f_{i,j} : D_i \times D_j \rightarrow \mathbb{N} \). Each cost function is known only to the agents involved. Agents aim to find an assignment \( d^* \) for all variables such that it minimizes the global objective function, which is the sum of the cost functions in \( F \). This assignment is a solution to a DCOP and is expressed as follows:

\[
d^* := \arg \min_{d \in \prod_{i=1}^{n} D_i} \sum_{f_{i,j} \in F} f_{i,j}(d_i, d_j).
\]

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 constraint graph but exist in the constraint graph and establish pseudo-parent/pseudo-child relationships.

### 3 Flaws in ADOPT-based Algorithms

We present counterexamples to the theorems for termination and optimality in the original study of ADOPT, which means that the proofs presented in the study are incorrect. We also 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.

As mentioned above, counterexamples are classified into three types by their properties and causes. The first is the counterexample to termination, described in Section 3.1,Appendices and other supplemental materials are available at [https://mas.cs.tsukuba.ac.jp/~noshiro](https://mas.cs.tsukuba.ac.jp/~noshiro).
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 these 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 [Modi et al., 2005]. These assumptions are adopted in the studies of ADOPT and its variants.

In this section, we show the counterexamples with several figures that indicate crucial points. More detailed traces are presented in Appendix B.

3.1 Counterexample to Termination

Figure 1 shows the DFS pseudo-tree and cost functions of a DCOP for the counterexample to termination. The solid lines indicate parent/child relationships, and the dashed lines indicate pseudo-parent/pseudo-child relationships.

![DFS pseudo-tree](image)

![Cost functions](image)

Figure 1: DFS pseudo-tree and cost functions of a DCOP for the counterexample to termination. The solid lines indicate parent/child relationships, and the dashed lines indicate pseudo-parent/pseudo-child relationships.

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., \( \text{th}_0(0, x_1) \) and \( \text{th}_0(0, x_2) \)) are set to 0; the lower bounds of \( x_1 \) and \( x_2 \) (i.e., \( \text{LB}_1(0) \), \( \text{LB}_1(1) \), \( \text{LB}_2(0) \), and \( \text{LB}_2(1) \)) are set to 0; and the thresholds of \( x_1 \) and \( x_2 \) (i.e., \( \text{TH}_1 \) and \( \text{TH}_2 \)) are set to 0. The states of agents at this point are presented inside the parentheses in Figure 2.

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 \( \text{CX}_4 \supseteq (x_1, 0) \), and thus \( \text{LB}_4 = 5 \). Then \( x_4 \) reports the lower bound to \( x_3 \) by sending a COST message. Subsequently, \( x_3 \) receives the VALUE message from \( x_4 \) and the COST message from \( x_4 \) and computes the lower bound as \( \text{LB}_3 = 5 \). Then \( x_3 \) sends a COST message with \( \text{CX}_3 \supseteq (x_1, 0) \) and \( \text{LB}_3 = 5 \) to \( x_1 \). After \( x_1 \) receives the COST message, \( x_1 \) updates \( \text{LB}_1(0) \) to 5. Since \( \text{LB}_1(0) = 5 > \text{TH}_1 = 0 \), \( x_1 \) 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 \( \text{LB}_3 = 0 \) and \( \text{UB}_3 = \infty \), 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 \( \text{LB}_1(1) = 6 \). Additionally, the threshold of \( x_1 \) increases as \( \text{TH}_1 = \text{LB}_1 = \min\{\text{LB}_1(0), \text{LB}_1(1)\} = 5 \) because of Threshold-Invariant (for more information on the invariants concerning thresholds, see Appendix A or [Modi et al., 2005]). Since \( \text{LB}_1(1) = 6 > \text{TH}_1 = 5 \), \( x_1 \) 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 \( \text{LB}_3(0) = \text{lb}_3(0, x_3) + \text{lb}_1(0, x_3) = 5 + 2 = 7 \) and then changes its value \( d_1 \) from 0 to 1. Next, \( x_1 \) also computes the lower bound as \( \text{LB}_1(1) = \text{lb}_1(1, x_3) + \text{lb}_1(1, x_5) = 6 + 3 = 9 \), and then the value \( d_1 \) returns to 0. After the cost calculation, \( x_1 \) sends VALUE messages with the current value \( d_1 = 0 \) to the lower neighbors and a COST message with \( \text{LB}_1 = 7 \) to \( x_0 \). Afterwards, \( x_6 \) receives the COST message and then increases \( \text{LB}_0, \text{TH}_0, \) and \( \text{th}_0(0, x_1) \)
to 7 (the thresholds change due to ThresholdInvariant and ChildThresholdInvariant). As the result of 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 a 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, $x_2$ 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 $x_1$.

**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 procedures are 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 Figure 2 (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 $x_2$ 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 $LB_0 = lb_0(0, x_1) + lb_0(0, x_2) = 0 + 7 = 7$. 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 process as described from Step 1 to Step 6. This causes $x_1$ 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.

**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.

**Occurrence in Variants**

Similar counterexamples to termination appear in the variants ADOPT-N and ADOPT$^+$. ADOPT-N is equivalent to the original ADOPT if the given DCOP does not contain unary ($n > 2$) cost functions. Since the problem consists of only binary functions, ADOPT-N has this flaw. ADOPT$^+$ is a variant of ADOPT that saves its redundant messages. Although the trace above contains redundant messages, a similar trace can be performed without such messages. Therefore, this flaw can appear in ADOPT$^+$.

On the other hand, the counterexample does not occur in IDB-ADOPT. The algorithm repeats the execution of ADOPT, decreasing the initial threshold of the root agent, and eventually, the initial threshold becomes less than the optimal cost by 1. In the DCOP shown in Figure 1, the initial threshold of the root agent at the final execution is 13, and hence the threshold of $x_1$ or $x_2$ must be equal to or greater than 7. Therefore, the agents cannot change their variable values repeatedly, that is, the counterexample does not appear in IDB-ADOPT. Additionally, BnB-ADOPT, BnB$^+$-ADOPT, ADOPT(k), and BD-ADOPT do not have the flaw because the bounds are updated monotonically in these algorithms.

**3.2 Counterexample to Optimality Caused by Initialization**

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

![DFS pseudo-tree and Cost function](a)

<table>
<thead>
<tr>
<th>$d_0$</th>
<th>$d_2$</th>
<th>$f_{0,2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 3:** DFS pseudo-tree and the cost function of a DCOP for the counterexample to optimality caused by initialization.

**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 Figure 4.
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 Figure 5. 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 Figure 5(c), 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 Figure 5(b) are the 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.

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 Figure 6.

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. We omit this procedure, which is presented in Appendix B. The calculation of the cost results in the following:

\[
LB_0(0) = U_B_0(0) = 1; \quad LB_0(1) = 0; \quad UB_0(1) = \infty; \quad TH_0 = 0; \quad TH_1 = 1; \quad LB_2 = UB_2 = 1; \quad TH_2 = 1; \quad \text{and } d_2 = 0.
\]

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 and then terminates. Here, \(x_0\) sends VALUE messages with \(d_0 = 0\) to its lower neighbors (i.e., \(x_1, x_3,\) and \(x_4\)) and TERMINATE and THRESHOLD 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 \(CX_2\) from \(\{(x_0, 0)\}\) into \(\{(x_0, 1)\}\) and returns it to \(\{(x_0, 0)\}.\) Since \(\{(x_0, 1)\}\) is incompatible with \(\{(x_0, 0)\}\), \(x_1\) 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 TERMINATE and THRESHOLD 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 messages from $x_1$, $x_3$ 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 $CX_2 = \{ (x_0, 0), (x_1, 0) \}$, 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.

**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 reinitialization was performed. To demonstrate this fact, let us consider a scenario that the solution can occur even if reinitialization was performed.
fore a TERMINATE message, the termination condition of the agent can be satisfied. Therefore, the agent can terminate with a suboptimal value.

**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, they 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 appropriate. They imply that agents obtain the optimal costs for each of the current contexts, which may be incorrect at termination. The study of BnB-ADOPT also presents an inappropriate statement. It implies that only the root agent obtains the optimal cost at termination, however, 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.

**4 Amendment to ADOPT**

In this section, we propose an amended version of ADOPT to avoid the flaws. The amendment consists of three parts, each of which corresponds to one of the causes described in Section 3. The pseudocode of our version of ADOPT is presented in Appendix C.

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. In the amended version, when agent \(x_i\) receives a COST message from child \(x_c\) with bounds \(LB_c\) and \(UB_c\) and context \(CX_c \ni (x_i, d)\), \(x_i\) updates the lower and upper bounds for \(d\) and \(x_c\) by the following:

\[
\begin{align*}
lb_i(d, x_c) &:= \max\{lb_i(d, x_c), LB_c\}, \\
ub_i(d, x_c) &:= \min\{ub_i(d, x_c), UB_c\}.
\end{align*}
\]

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

Second, we modify the initialization of a current context so as to store the assignments of all higher neighbors. 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 \(x_p\) and \(SCP(x_i)\) is a set of the ancestors of \(x_i\) that are parents or pseudo-parents of agents (including \(x_i\)) in the subtree rooted at \(x_i\). Note that for \(x_i\) to avoid flaws, it is sufficient to initialize the current context 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 have been introduced in BnB-ADOPT, BnB-ADOPT\(^+\), ADOPT(k), and BD-ADOPT.

Finally, we modify TERMINATE messages and their receiving procedure. The amended 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 introduce the threshold \(th_i(d, x_c)\) for the current value \(d\), and child \(x_c\) into the TERMINATE message sent to \(x_c\). The amendment to TERMINATE messages guarantees 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.

**5 Proof of Termination and Optimality**

Our amended algorithm guarantees termination and optimality, which are stated as Theorems 1 and 2 below, and the full proofs and lemmata are shown in Appendix D. The arguments of the proofs are based on the study of BnB-ADOPT. However, the statements of the theorems are modified to derive our aimed properties.

For the theorem of optimality, we define the correctness of the current context of an agent.

**Definition 1.** The current context \(CX_i\) of agent \(x_i\) is correct iff \(CX_i\) contains the assignments of all agents in \(SCP(x_i)\), and the values of all agents in \(CX_i\) are equal to the values of the agents.

The theorems for the amended version of ADOPT are as follows:

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

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

**6 Conclusion**

We presented flaws in ADOPT and its variants with respect to termination and optimality, which have been believed to be correct. The flaws are classified into three types, one of which is related to termination, while the other two of which are related to optimality. The causes of the flaws are the update rules for the bounds, the initialization of a current context, and TERMINATE messages and their receiving procedure. Additionally, we proposed an amended version of ADOPT that avoids these flaws, whose amendment corresponds to the causes of the flaws. Furthermore, we proved termination and optimality in the amended version of ADOPT. For future work, we plan to verify the properties of ADOPT-ing, which was not analyzed in this work.
References


