A Quantified Distributed Constraint Optimization Problem

Toshihiro Matsui, Hirohsi Matsuo
Nagoya Institute of Technology
Gokiso-cho, Showa-ku
Nagoya 466-8555, Japan
{matsui.t, matsuo}@nitech.ac.jp

Marius Călin Silaghi
Florida Institute of Technology
150 W. University Blvd.
Melbourne, FL 32901
msilaghi@fit.edu

Katsutoshi Hirayama
Kobe University
5-1-1 Fukaeminami-machi,
Higashinada-ku,
Kobe 658-0022, Japan
hirayama@maritime.kobe-u.ac.jp

Makoto Yokoo, Satomi Baba
Kyusyu University
744 Motooka, Nishi-ku
Fukuoka 819-0395, Japan
yokoo@is.kyushu-u.ac.jp,
s-baba@agent.is.kyushu-u.ac.jp

ABSTRACT

In this paper, we propose a Quantified Distributed Constraint Optimization problem (QDCOP) that extends the framework of Distributed Constraint Optimization problems (DCOPs). DCOPs have been studied as a fundamental model of multi-agent cooperation. In traditional DCOPs, all agents cooperate to optimize the sum of their cost functions. However, in practical systems some agents may desire to select the value of their variables without cooperation. In special cases, such agents may take the values with the worst impact on the quality of the result reachable by the optimization process. We apply existential/universal quantifiers to distinct uncooperative variables. A universally quantified variable is left unassigned by the optimization as the result has to hold when it takes any value from its domain, while an existentially quantified variable takes exactly one of its values for each context. Similar classes of problems have recently been studied as (Distributed) Quantified Constraint Problems, where the variables of the CSP have quantifiers. All constraints should be satisfied independently of the value taken by universal variables. We propose a QDCOP that applies the concept of game tree search to DCOP. If the original problem is a minimization problem, agents that own universally quantified variables may intend to maximize the cost value in the worst case. Other agents normally intend to optimize the minimizing problems. Therefore, only the bounds, especially the upper bounds, of the optimal value are guaranteed. The purpose of the new class of problems is to compute such bounds, as well as to compute sub-optimal solutions. For the QDCOP, we also propose several methods that are based on min-max/alpha-beta and ADOPT algorithms.

Categories and Subject Descriptors

I.2.11 [**ARTIFICIAL INTELLIGENCE**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Algorithms

Cite as: Quantified Distributed Constraint Optimization Problem, Toshihiro Matsui, Marius Călin Silaghi, Katsutoshi Hirayama, Makoto Yokoo, Hiroshi Matsuo and Satomi Baba, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. XXX-XXX.

Keywords

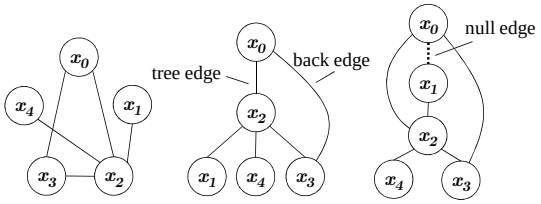
quantified, Distributed Constraint Optimization, multi-agent, cooperation

1. INTRODUCTION

Distributed Constraint Optimization problems (DCOPs) have been studied as a fundamental model of multi-agent cooperation [9, 11, 12, 14, 15, 16]. With DCOPs, a multi-agent system is represented as a discrete optimization problem distributed among agents. The decisions to be made by agents are modeled as variables. Relationships between agents are represented by cost functions. Distributed search algorithms are employed to compute a solution that globally optimizes the aggregated value of these functions. Distributed meeting scheduling, resource allocation in power plants and coordination in sensor networks are modeled as DCOPs [5, 7, 8].

In traditional DCOPs, all agents cooperate to optimize the sum of their cost functions. However, in practical systems some agents may desire to select the value of their variables without cooperation. One motivating domain is a contingency planning problem in a smart grid system, which contains provider nodes and consumer nodes. The provider nodes try to find a robust plan that can handle any requests from consumers. Another domain is a surveillance problem by multiple sensors/cameras. These sensors/cameras try to find a surveillance plan against an intruder. In special cases, such agents may take the values with the worst impact on the quality of the result reachable by the optimization process. We apply existential/universal quantifiers to distinct uncooperative variables. A universally quantified variable is left unassigned by the optimization as the result has to hold when it takes any value from its domain, while an existentially quantified variable takes exactly one of its values for each context. Similar classes of problems have recently been studied as (Distributed) Quantified Constraint Problems (QCSP, QDCSP)[4, 2], where the variables of the CSP have quantifiers. All constraints should be satisfied independently of the value taken by universal variables. In [2], an extension of the asynchronous backtracking search algorithm has been proposed.

A natural extension from DCOP to QDCOP is to introduce the concept of game tree search. For example, if the original problem is a minimization problem, agents that own universally quantified variables may intend to maximize the cost value (in the worst case). Other agents normally intend to optimize the minimizing problems. Therefore, only the bounds, especially upper bounds, of the optimal value are guaranteed. The purpose of the new class of



(a) constraint network (b) pseudo-tree (c) modified pseudo-tree

Figure 1: pseudo-trees

problems is to compute such bounds, as well as to compute sub-optimal solutions. For the QDCOP, we also propose several methods that are based on min-max/alpha-beta and ADOPT algorithms. We show how the pseudo-tree-based DCOP algorithms are generalized into game tree search algorithms. The performance of the proposed methods is evaluated experimentally.

The outline of the paper is as follows. In Section 2, problem definitions including DCOP, QCSP, QDCSP and QDCOP are shown. Then we propose several algorithms for QDCOP in Section 3. The proposed methods are evaluated experimentally in Section 4. In Section 5, related works are considered. We present our conclusion in Section 6.

2. PROBLEM DEFINITIONS

In this section, conventional problem definitions including usual DCOP and Quantified CSP/DCSP are shown. Then we define a Quantified DCOP.

2.1 DCOP

A distributed constraint optimization problem (DCOP) is defined by (A, X, D, C, F) where A is a set of agents, X is a set of variables, D is a set of domains, C is a set of binary constraints, and F is a set of binary functions. Agent i has its own variable $x_i \in X$. x_i takes a value from discrete finite domain $D_i \in D$. The value of x_i is controlled by agent i. Constraint $c_{i,j} \in C$ represents the relationship between x_i and x_j. The cost of an assignment $\{(x_i, d_i), (x_j, d_j)\}$ is defined by a binary function $f_{i,j} \in F$ such that $f_{i,j}(d_i, d_j) : D_i \times D_j \rightarrow \mathbb{N}$. The goal is to find a global optimal solution $\mathcal{A}$ that minimizes the global cost function: $\sum_{f_{i,j} \in F,\ \{(x_i,d_i),(x_j,d_j)\} \subseteq \mathcal{A}} f_{i,j}(d_i, d_j)$. In this paper, to simplify notations, we may not strictly distinguish each agent from its variable.

2.2 QCSP/QDCSP

The Quantified Constraint Satisfaction problem (QCSP)[4] is an extension of classical CSP. The classical CSP is defined by (X, D, C) where X is a set of variables, D is a set of domains, and C is a set of constraints. x_i takes a value from domain $D_i \in D$. A solution of CSP is a set of assignment $\{(x_0, d_0), \cdots, (x_n, d_n)\}$ that satisfies all constraints in C. In addition to the definition of the classical CSP, QCSP defines a sequence of quantified variables. A QCSP has the form $Q.C = q_0 x_0 \cdots q_n x_n.C$. Q is a sequence of variables where q_i is the existential quantifier $\exists$ or the universal quantifier $\forall$. The semantics of a QCSP $Q.C$ is recursively defined as follows. If C is empty, $Q.C$ is true. If Q is of the form $\exists x_0 q_1 x_1 \cdots q_n x_n$, then $Q.C$ is true iff there exists a value $d \in D_0$ such that $q_1 x_1 \cdots q_n x_n.(C \cup \{x_0 = d\})$ is true. If Q is of the form $\forall x_0 q_1 x_1 \cdots q_n x_n$, then $Q.C$ is true iff $q_1 x_1 \cdots q_n x_n.(C \cup \{x_0 = d\})$ is true for all values $d \in D_0$. Otherwise, $Q.C$ is false.

In a Quantified Distributed CSP(QDCSP)[2], the variables are distributed among agents. In [2], an extension of the asynchronous backtracking search algorithm has been proposed.

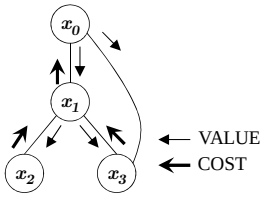


Figure 2: message paths of ADOPT

2.3 QDCOP

The class of Quantified DCOPs (QDCOPs) is introduced based on that of a DCOP. In addition to definition of the DCOP, QDCOP defines a sequence of quantified variables similarly to QCSP/QDCSP. A QDCOP has the form $Q.(C, F) = q_0 x_0 \cdots q_n x_n.(C, F)$. Q is a sequence of variables where q_i is the existential quantifier $\exists$ or the universal quantifier $\forall$. Basically, the goal of QDCOP is to find a global optimal (minimal) solution in the corresponding DCOP. However, its semantics is modified due to quantifiers. Existentially quantified variables are usual variables. On the other hand, universally quantified variables can take any values. Therefore, its optimal solution is different from that of the DCOP. A QDCOP defines boundaries of the optimal cost value, while a DCOP defines a unique optimal cost value. The usual optimal cost is now the cost of the best case. Therefore, the best case defines the lower bound. In the worst case, universally quantified variables take values that increase costs as possible. Therefore, the worst case defines the upper bound. This class of problem is similar to the problem in game tree search [13]. We focus on the worst case problem as a QDCOP. While any solutions between the best and the worst cases can be chosen, we believe that the worst cost is informative in most of the practical problems. We assume there exists a virtual agent for each universally quantified variable, who imitates the adversary's actions but cooperates in searching for the bound with its team of cooperative agents, i.e., they are calculating the bound *off-line*.

3. SEARCH ALGORITHMS FOR QDCOP

3.1 Modified pseudo-tree

A pseudo-tree [12] is a graph structure that defines a partial order on variables. The pseudo-tree contains a spanning tree of the constraint network. For example, the pseudo-tree in Figure 1 (b) is generated from the constraint network in Figure 1 (a). The edges of the original constraint network are categorized into either the tree or the back edges of the pseudo-tree. The tree edges represent the partial order relation between the two variables. We use some notations to represent agents related to agent i. Let $Chld_i$, $Dcnd_i$ and $parent_i$ denote child nodes, descendant nodes, and a parent node respectively. And PP_i represents a subset of ancestor agents whose variables are directly related to other variables of several agents in the subtree rooted at i. There are no back edges between different sub-trees. Therefore, a divide-and-conquer strategy can be applied to the search processing for different sub-trees. By employing this property, search processing can be performed in parallel.

A typical pseudo-tree is based on a depth first search (DFS) tree on a constraint network. The DFS tree is generated in a top down manner from a root variable node. However, in the case of QDCOPs, the order of variables cannot be easily changed. Otherwise, quantifiers are evaluated in the wrong order. Therefore, the pseudo-tree must be modified to keep the ordering. This modification is applied by inserting an extra *null* edge for each pair of a parent and a child if the edge is necessary. A simple method for generating such a modified pseudo-tree is a well known bottom up computation as follows. In the initial state, agent i knows set Nbr_i of neighbor-

hood nodes and the order relationship of agents. Then Nbr_i is immediately separated into Nbr_i^u and Nbr_i^l which represent upper and lower neighborhood nodes respectively. $Chld_i$, $Dcnd_i$, PP_i and $parent_i$ are recursively computed using the following equations.

$$Chld_i = \{j | parent_j = i\} \quad (1)$$

$$Dcnd_i = Chld_i \cup \bigcup_{j \in Chld_i} Dcnd_j \quad (2)$$

$$PP_i = Nbrs_i^u \cup \bigcup_{j \in Chld_i} (PP_j \backslash \{i\}) \quad (3)$$

$$parent_i = \begin{cases} \text{lowest } j \in PP_i & Nbrs_i^l \subseteq Dcnd_i \\ \text{undefined} & \text{otherwise} \end{cases} \quad (4)$$

If i is a leaf node, PP_i and $parent_i$ are immediately computed because $Nbrs_i^l$ is an empty set. Note that $Chld_i$ is an empty set because no agent's parent is decided yet. Therefore $Dcnd_i$ is also an empty set. Also, PP_i equals $Nbrs_i^u$. We consider that $Nbrs_i^l \subseteq Dcnd_i$ in Equation 4 is true if both sets are empty. This computation eventually converges. If i is the root node, the final value of $parent_i$ is considered as an *empty* value because PP_i is an empty set. In actual distributed algorithms, each agent i sends $(Chld_i, Dcnd_i, PP_i, parent_i)$ to $parent_i$ when $parent_i$ is decided. An example of a modified pseudo-tree is shown in Figure 1 (c).

We first decide a total order on agents, so that the total order satisfies a partial order that is specified in the sequence of quantifiers. For a given total order, we can uniquely create a pseudo-tree by adding some null-edges if necessary. Due to the existence of the predefined partial order, the choice of total orders is rather limited.

3.2 Computation of cost value and solution

Cost values are computed according to pseudo-trees. The computation is based on ADOPT [11], except that several agents make choices contrary to the optimization goal. We assume that each agent knows the values of the variables and the cost functions of other agents that share constraints with the agent. Basically, agent i's computation is based on the partial solution s_i of PP_i. s_i is called *context*. $s \approx s'$ represents compatibility, and is defined as :

$$s \approx s' \triangleq \forall (d, d') \text{ s.t. } (x, d) \in s \wedge (x, d') \in s', d = d' \quad (5)$$

The aggregation of costs in agent i is shown as follows. The local cost $\delta_i(s_i, d)$ for value d of variable x_i and context s_i is defined as:

$$\delta_i(s_i, d) = \sum_{(x_j, d_j) \in s_i,\ j \in Nbr_i^u} f_{i,j}(d, d_j) \quad (6)$$

The local optimal cost for value d of variable x_i, context s_i and the sub-tree routed at x_i are recursively defined as:

$$g_i^*(s_i) = \begin{cases} \min_{d \in D_i} g_i(s_i, d) & q_i = \exists \\ \max_{d \in D_i} g_i(s_i, d) & q_i = \forall \end{cases} \quad (7)$$

$$g_i(s_i, d) = \delta_i(s_i, d) + \sum_{j \in Chld_i} g_j^*(s_j) \text{ s.t. } (x_i, d) \in s_j, s_j \approx s_i \quad (8)$$

Note that the calculation of $g_i^*(s_i)$ depends on the quantifier q_i of variable x_i. For existentially quantified variables, the cost values are minimized as with usual optimization. On the other hand, to compute the worst case values, cost values are maximized for universally quantified variables.

In actual computation, some cost values may be yet unknown. In such cases, upper and lower limit values are used as default values. In this work, we use 0 and ∞ as lower and upper limits respectively.

Figure 3: min-max ADOPT

```
1  Main(){
2   Initialize().
3   until(forever){
4    until(receive loop is broken){
5      if(¬trm_i){ receive messages. }else{ purge messages. }}
6    if(¬((waiting initial s_i)∨trm_i)){ Maintenance(). }}
7  }
8  Initialize(){
9   d_i ← d ∈ D_i. s_i ← φ. trm_i ←false. ptrm_i ←false.
10  foreach d ∈ D_i, j ∈ Chld_i{ (s_{d,j}, lb_{d,j}, ub_{d,j})←(φ, 0, ∞). }
11  if(i is root){ trm_i ←true. Maintenance(). }
12 }
13 Receive(VALUE, j, s, ptrm){
14  update s_i using s.
15  ensure child_cost_consistency.
16  if(s is not old){
17   if(j = parent_i){ ptrm_i ← ptrm. }}
18 }
19 Receive(COST, j, s, lb, ub){
20  d ← d' s.t. (x_i, d') ∈ s. s ← s\{(x_i, d')}.
21  update s_i using s.
22  ensure child_cost_consistency.
23  if(s is not old){
24   update (s_{d,j}, lb_{d,j}, ub_{d,j}) using(s, lb, ub). }
25 }
26 Maintenance(){
27  if(ptrm_i ∧ lb_i^* = ub_i^*){
28   if(q_i = ∃){ d_i ← d s.t. ub_i(d) = lb_i^*. }
29   else{ d_i ← d s.t. lb_i(d) = ub_i^*. }
30   trm_i ←true.
31  }else if(lb_i(d_i) = ub_i(d_i)){
32   d_i ← d s.t. lb_i(d) ≠ ub_i(d) if such d exists. }
33  foreach j ∈ Chld_i{
34   send (VALUE, i, s_i ∪ {(x_i, d_i)}, trm_i) to j. }
35  foreach j ∈ Nbr_i^l\Chld_i{
36   send (VALUE, i, {(x_i, d_i)}, φ) to j. }
37  send (COST, i, s_i, lb_i^*, ub_i^*) to parent_i.
38 }
```

These default values separate a cost value into upper and lower bound values. Once a globally optimal cost $g_r^*(\phi)$ is computed for the root variable x_r, a sub-optimal solution can be computed in a top down manner.

3.3 min-max ADOPT

Pseudo-tree based distributed search algorithms can be naturally extended to address QDCOPs. First, we describe the min-max ADOPT algorithm as a basic scheme. As shown in Figure 2, the algorithm works using two types of messages. The VALUE messages announce the value of variables. In addition to the top down path of tree edges, short cut paths of back edges are also used. The short cut messages can contain only the assignments of source agents. The COST messages announce cost values.

Received values are locally stored in each agent i. Therefore, context s_i now denotes the newest copy of the assignments of the ancestors. Since ADOPT is a memory-bounded algorithm, only the current partial solution is referred to in cost computations. By $lb_{d,j}$ we denote a copy of the lower bound value of $g_j^*(s_j)$ such that $(x_i, d) \in s_j$ where j is a child agent of i. Similarly, $ub_{d,j}$ denotes the upper bound. $lb_i(d)$ and $ub_i(d)$ denote lower and upper bounds of $g_i(s_i, d)$. Furthermore, lb_i^* and ub_i^* denote boundaries of $g_i^*(s_i)$ respectively. $lb_{d,j}$ and $ub_{d,j}$ are maintained with related context $s_{d,j}$. The initial values of $(s_{d,j}, lb_{d,j}, ub_{d,j})$ are $(\phi, 0, \infty)$. They are updated using received cost values and the related context. When $s_i \not\approx s_{d,j}$, $(s_{d,j}, lb_{d,j}, ub_{d,j})$ are reset to initial values.

The pseudo code of min-max ADOPT is shown in Figure 3. In addition to notations shown above, several notations are used. d_i

denotes the current assignment of x_i. For the termination sequence, $ptrm_i$ and trm_i are used. They specify whether $parent_i$ and i have been terminated or not. In the pseudo codes, we implicitly use the vector of logical time. Each logical time is associated with a variable's value. When the value is changed, its logical time is increased. It is necessary to decide the newest value in updating contexts. Cost values that are computed for obsolete variables' values are reset to initial values. The removal of the vector clock is not addressed in this paper. In the algorithm, the following invariant and update operations are used.

child_cost_consistency: For each $d \in D_i$ and $j \in Chld_i$, $s_{d,j} \approx s_i$ must hold. If they are different, ($s_{d,j}$, $lb_{d,j}$, $ub_{d,j}$) are reset to their initial value (ϕ, 0, ∞).

update s_i using s: For each (x, d) and (x, d') such that $(x, d) \in s_i \wedge (x, d') \in s$, if the logical time of d' is newer than d, (x, d) is replaced by (x, d'). For each (x, d') such that $(x, d') \notin s_i \wedge (x, d') \in s$, (x, d') is appended to s_i.

update ($s_{d,j}$, $lb_{d,j}$, $ub_{d,j}$) using(s, lb, ub): $s_{d,j}$ is updated by s. $lb_{d,j}$ is updated by lb if $lb > lb_{d,j}$. $ub_{d,j}$ is updated by ub if $ub < ub_{d,j}$.

The outline of the processing is as follows. The processing is initiated by the root agent's VALUE messages. When each agent i receives messages, its context s_i is updated. Then consistencies of ($s_{d,j}$, $lb_{d,j}$, $ub_{d,j}$) are maintained. Their values are reset if necessary. ($s_{d,j}$, $lb_{d,j}$, $ub_{d,j}$) and $ptrm_i$ are also updated according to the types of messages. After receiving messages, agent i computes its new state. If $parent_i$ has been already terminated and $lb_i^* = ub_i^*$, i selects its optimal value of x_i and terminates. Otherwise, i selects x_i's value to search the rest of the solution space. Finally, all agents terminate and the system reaches quiescence. While it may be possible to select a more desirable solution in the termination sequence, we prefer the solution on the upper bounds of optimal cost in the minimizing problem.

This algorithm is clearly a simplified version of the original ADOPT. There are no major pruning methods such as *backtracking thresholds*. Additionally, there are opportunities to remove several redundancies of the algorithm. If a new message is scheduled to carry exactly the same information as a previous message, the new message is not necessary. In particular, in the case of the min-max method, each agent only depends on the assignments of PP_i. Therefore, the contexts of VALUE messages can be reduced to it. It reduces extra flipping of variables' values. In our implementation, redundant messages are partially limited, and we minimize the context of VALUE messages.

3.4 alpha-beta ADOPT

The alpha-beta method is a fundamental pruning method for game tree search [13]. This method employs two boundary parameters, alpha and beta, that represent the lower bound and upper bound of possible cost values. Alpha represents the lower bound, controlled by the maximizing player, and beta represents the upper bound controlled by the minimizing players. Neither player can modify a type of boundary other than the one given by its type. In QDCOP, each agent performs either as a minimizing or as a maximizing player. The pruning is driven in a top down manner. When an agent reports the cost value of the current partial solution, its parent agent narrows alpha/beta according to the cost value. Then the new alpha/beta value is used to prune the child agent's search. This pruning can be applied to pseudo-tree based min-max ADOPT. Alpha/beta values are managed using a technique similar to the *backtracking threshold* in the original ADOPT.

In the alpha-beta ADOPT, an agent i employs several values for

Figure 4: alpha-beta ADOPT

```
1  Initialize(){
2   d_i ← d ∈ D_i. s_i ← φ. rlvl_i ← 0. trm_i ←false.
3   ptrm_i ←false. (s_i^αβ, rlvl_i^αβ, α_i, β_i)←(s_i, rlvl_i, 0, ∞).
4   foreach d ∈ D_i, j ∈ Chld_i{
5    (s_{d,j}, rlvl_{d,j}, lb_{d,j}, ub_{d,j}, lb_{d,j}^αβ, ub_{d,j}^αβ, α_{d,j}, β_{d,j})
6     ←(s_i, rlvl_i, 0, ∞, 0, ∞, 0, ∞). }
7   if(i is root){ trm_i ←true. Maintenance(). }
8  }
9  Receive(VALUE, j, s, rlvl, α, β, ptrm){
10  update (s_i, rlvl_i) using (s, rlvl).
11  ensure alpha_beta_consistency and child_cost_consistency.
12  ensure child_alpha_beta_invariant.
13  if(s and rlvl are not old){
14   if(j = parent_i){
15    update (s_i^αβ, rlvl_i^αβ, α_i, β_i) using (s, rlvl, α, β).
16    ensure child_alpha_beta_invariant. ptrm_i ← ptrm. }}
17 }
18 Receive(COST, j, s, rlvl, lb, ub, lb^αβ, ub^αβ){
19  d ← d' s.t. (x_i, d') ∈ s. s ← s\{(x_i, d')}.
20  update (s_i, rlvl_i) using (s, rlvl).
21  ensure alpha_beta_consistency and child_cost_consistency.
22  ensure child_alpha_beta_invariant.
23  if(s and rlvl are not old){
24   update (s_{d,j}, rlvl_{d,j}, lb_{d,j}, ub_{d,j}, lb_{d,j}^αβ, ub_{d,j}^αβ) using
25    (s, rlvl, lb, ub, lb^αβ, ub^αβ). }
26 }
27 Maintenance(){
28  ensure alpha_beta_invariant.
29  ensure child_alpha_beta_invariant.
30  if(ptrm_i ∧ q_i = ∃ ∧ ub_i^* = α_i){
31   d_i ← d s.t. ub_i(d) = α_i. trm_i ←true.
32  }else if(ptrm_i ∧ q_i = ∀ ∧ lb_i^* = β_i){
33   d_i ← d s.t. lb_i(d) = β_i. trm_i ←true.
34  }else if(ptrm_i ∧ rlvl_i < i){
35   α' ← α_i. β' ← β_i. rlvl_i ← i.
36   ensure alpha_beta_consistency.
37   if(q_i = ∃){ α_i ← α'. }else{ β_i ← β'. }
38   ensure child_cost_consistency.
39   ensure alpha_beta_invariant.
40   ensure child_alpha_beta_invariant.
41  }else if(lb_i^{αβ*−} ≠ ub_i^{αβ*−} ∧ lb_i^αβ(d_i) = ub_i^αβ(d_i)){
42   d_i ← d s.t. lb_i^αβ(d) ≠ ub_i^αβ(d) if such d exists. }
43  foreach j ∈ Chld_i{
44   send (VALUE, i, s_i ∪ {(x_i, d_i)}, rlvl_i, α_{d_i,j}, β_{d_i,j}, trm_i)
45    to j. }
46  foreach j ∈ Nbr_i^l\Chld_i{
47   send (VALUE, i, {(x_i, d_i)}, rlvl_i, φ, φ, φ) to j. }
48  send (COST, i, s_i, rlvl_i, lb_i^*, ub_i^*, lb_i^{αβ*−}, ub_i^{αβ*−})
49   to parent_i.
50 }
```

the alpha-beta method. α_i and β_i represent the alpha and beta of i. $lb_{d,j}^{\alpha\beta}$, $ub_{d,j}^{\alpha\beta}$, $lb_i^{\alpha\beta}(d)$, $ub_j^{\alpha\beta}(d)$, $lb_i^{\alpha\beta*}$ and $ub_j^{\alpha\beta*}$ are similar to $lb_{d,j}$, $ub_{d,j}$, $lb_i(d)$, $ub_i(d)$, lb_i^* and ub_i^* respectively. Because the alpha-beta method often computes cost values that exceed true boundaries, we separate these cost values from the true cost values. Additionally, $lb_i^{\alpha\beta*-}$ and $ub_j^{\alpha\beta*-}$ are introduced for restricted values of $lb_i^{\alpha\beta*}$ and $ub_j^{\alpha\beta*}$. $lb_i^{\alpha\beta*-}$ and $ub_j^{\alpha\beta*-}$ are defined as follows.

$$lb_i^{\alpha\beta*-} = \min(\max(lb_i^{\alpha\beta*}, \alpha_i), \beta_i) \quad (9)$$

$$ub_i^{\alpha\beta*-} = \min(\max(lb_i^{\alpha\beta*}, \alpha_i), \beta_i) \quad (10)$$

The alpha/beta values are shared between i and its child agents. $\alpha_{d,j}$ and $\beta_{d,j}$ represent the shared value for child agent j in the case of $x_i = d$.

In addition to the implicit vector clock for the values of vari-

ables, a logical time, *root level*, is introduced. The logical time is used to detect whether search is reset in the termination sequence. The necessity of the reset will be discussed below. $rlvl_i$ represents i's current newest root level. Consistency of context and root level are maintained for alpha, beta and cost values. α_i and β_i are maintained with related context $s_i^{\alpha\beta}$ and $rlvl_i^{\alpha\beta}$. $lb_{d,j}$, $ub_{d,j}$, $lb_{d,j}^{\alpha\beta}$, $ub_{d,j}^{\alpha\beta}$, $\alpha_{d,j}$ and $\beta_{d,j}$ are maintained with the related context $s_{d,j}$ and $rlvl_{d,j}$. In our algorithm we use the following invariants and update operations.

alpha_beta_consistency: $s_i^{\alpha\beta}$ and $rlvl_i^{\alpha\beta}$ must be equal to s_i and $rlvl_i$ respectively. If they are different, ($s_i^{\alpha\beta}$, $rlvl_i^{\alpha\beta}$, α_i, β_i) are reset to their initial value (s_i, $rlvl_i$, 0, ∞).

child_cost_consistency: For each $d \in D_i$ and $j \in Chld_i$, $s_{d,j}$ and $rlvl_{d,j}$ must be equal to s_i and $rlvl_i$ respectively. If they are different, ($s_{d,j}$, $rlvl_{d,j}$, $lb_{d,j}$, $ub_{d,j}$, $lb_{d,j}^{\alpha\beta}$, $ub_{d,j}^{\alpha\beta}$, $\alpha_{d,j}$, $\beta_{d,j}$) are reset to their initial value (s_i, $rlvl_i$, 0, ∞, 0, ∞, 0, ∞).

alpha_beta_invariant: α_i and β_i must take the best upper and lower bounds of cost values respectively. That is achieved as follows. First, $lb_i^{\alpha\beta*}$ and $ub_i^{\alpha\beta*}$ are calculated as shown in equations 7 and 8. Then, $lb_i^{\alpha\beta*-}$ and $ub_i^{\alpha\beta*-}$ are calculated as shown in equations 9 and 10. If agent i has an existentially quantified variable, β_i is updated by $\min(\beta_i, ub_i^{\alpha\beta*-})$. Otherwise, α_i is updated by $\max(\alpha_i, lb_i^{\alpha\beta*-})$. Additionally, in the root agent, the opposite side of α_i or β_i is closed to hold $\alpha_i = \beta_i$ when $lb_i^{\alpha\beta*-} = ub_i^{\alpha\beta*-}$ is held. This special feedback rule generalizes other processing of the root agent.

child_alpha_beta_invariant: α_i and β_i are shared between agent i and its child agents. If i has only one child j, $\alpha_{d,j}$ and $\beta_{d,j}$ are respectively set to $\max(0, \alpha_i - \delta_i(d))$ and $\max(0, \beta_i - \delta_i(d))$ for each $d \in D_i$. Otherwise, $\alpha_{d,j}$ and $\beta_{d,j}$ are respectively set to 0 and $\max(0, \beta_i - \delta_i(d))$ for each $d \in D_i$ and $j \in Chld_i$. Here, the lower limit is 0. Note that child agents cannot correct overestimated boundaries. Therefore, the widest boundaries are set for multiple child nodes. In addition, $lb_{d,j}^{\alpha\beta}$ and $ub_{d,j}^{\alpha\beta}$ are restricted as $\min(\max(lb_{d,j}^{\alpha\beta}, \alpha_{d,j}), \beta_{d,j})$ and $\min(\max(ub_{d,j}^{\alpha\beta}, \alpha_{d,j}), \beta_{d,j})$.

update (s_i, $rlvl_i$) using (s, $rlvl$): s_i is updated using s in the same manner as in min-max ADOPT. If $rlvl > rlvl_i$, then $rlvl_i$ is updated by $rlvl$.

update ($s_i^{\alpha\beta}$, $rlvl_i^{\alpha\beta}$, α_i, β_i) using (s, $rlvl$, α, β): $s_i^{\alpha\beta}$ and $rlvl_i^{\alpha\beta}$ are updated by s and $rlvl$ respectively. If agent i has existentially quantified variable, α_i and β_i are updated by α and $\max(\min(\beta_i, \beta), \alpha)$ respectively. Otherwise, α_i and β_i are updated by $\min(\max(\alpha_i, \alpha), \beta)$ and β respectively.

update ($s_{d,j}$, $rlvl_{d,j}$, $lb_{d,j}$, $ub_{d,j}$, $lb_{d,j}^{\alpha\beta}$, $ub_{d,j}^{\alpha\beta}$) using (s, $rlvl$, lb, ub, $lb^{\alpha\beta}$, $ub^{\alpha\beta}$): $s_{d,j}$ and $rlvl_{d,j}$ are updated by s and $rlvl$ respectively. $lb_{d,j}$ and $ub_{d,j}$ are updated using lb and ub in the same manner as in min-max ADOPT. $lb_{d,j}^{\alpha\beta}$ and $ub_{d,j}^{\alpha\beta}$ are updated as follows. First, the values of $lb^{\alpha\beta}$ and $ub^{\alpha\beta}$ are modified as $\min(\max(lb^{\alpha\beta}, \alpha_{d,j}), \beta_{d,j})$ and $\min(\max(ub^{\alpha\beta}, \alpha_{d,j}), \beta_{d,j})$ respectively. These restrictions are necessary to ensure their child_alpha_beta_invariant. Then, $lb_{d,j}^{\alpha\beta}$ is updated by $lb^{\alpha\beta}$ if $lb^{\alpha\beta} > lb_{d,j}^{\alpha\beta}$. $ub_{d,j}^{\alpha\beta}$ is updated by $ub^{\alpha\beta}$ if $ub^{\alpha\beta} < ub_{d,j}^{\alpha\beta}$.

In the case of the alpha-beta method, each agent depends on all the assignments of its ancestors. Therefore, s_i, $s_i^{\alpha\beta}$ and $s_{d,j}$ must be completely equal in alpha_beta_consistency and child_cost_consistency. In child_alpha_beta_invariant, agent i may allocate $\alpha_{d,j}$ and $\beta_{d,j}$ which exceed the original $lb_{d,j}^{\alpha\beta}$ and $ub_{d,j}^{\alpha\beta}$. Such allocation is infeasible. However, in such a case, $lb_{d,j}^{\alpha\beta}$ and $ub_{d,j}^{\alpha\beta}$ are modified to hold $lb_{d,j}^{\alpha\beta} = ub_{d,j}^{\alpha\beta}$ any way, and that causes pruning.

Figure 5: bi-theshold ADOPT

```
1  Initialize(){
2   d_i ← d ∈ D_i. s_i ← φ. rlvl_i ← 0. trm_i ←false.
3   ptrm_i ←false. (s_i^{tαβ}, rlvl_i^{tαβ}, t_i^α, t_i^β)←(s_i, rlvl_i, 0, ∞).
4   foreach d ∈ D_i, j ∈ Chld_i{
5    (s_{d,j}, rlvl_{d,j}, lb_{d,j}, ub_{d,j}, t_{d,j}^α, t_{d,j}^β)
6     ←(s_i, rlvl_i, 0, ∞, 0, ∞). }
7   if(i is root){ trm_i ←true. Maintenance(). }
8  }
9  Receive(VALUE, j, s, rlvl, t^α, t^β, ptrm){
10  update (s_i, rlvl_i) using (s, rlvl).
11  ensure threshold_consistency and child_cost_consistency.
12  if(s and rlvl are not old){
13   if(j = parent_i){
14    update (s_i^{tαβ}, rlvl_i^{tαβ}, t_i^α, t_i^β) using (s, rlvl, t^α, t^β).
15    ptrm_i ← ptrm. }}
16 }
17 Receive(COST, j, s, rlvl, lb, ub){
18  d ← d' s.t. (x_i, d') ∈ s. s ← s\{(x_i, d')}.
19  update (s_i, rlvl_i) using (s, rlvl).
20  ensure threshold_consistency and child_cost_consistency.
21  if(s and rlvl are not old){
22   update (s_{d,j}, rlvl_{d,j}, lb_{d,j}, ub_{d,j}) using (s, rlvl, lb, ub). }
23 }
24 Maintenance(){
25  ensure threshold_invariant.
26  if(ptrm_i ∧ q_i = ∃ ∧ ub_i^* = t_i^α){
27   d_i ← d s.t. ub_i(d) = t_i^α. trm_i ←true.
28  }else if(ptrm_i ∧ q_i = ∀ ∧ lb_i^* = t_i^β){
29   d_i ← d s.t. lb_i(d) = t_i^β. trm_i ←true.
30  }else if(ptrm_i ∧ rlvl_i < i){
31    t^{α'} ← t_i^α. t^{β'} ← t_i^β. rlvl_i ← i.
32    ensure threshold_consistency.
33    if(q_i = ∃){ t_i^α ← t^{α'}. }else{ t_i^β ← t^{β'}. }
34    ensure child_cost_consistency.
35    ensure threshold_invariant.
36    if(t_i^α < lb_i(d) ∨ ub_i(d) < t_i^β){
37     d_i ← d s.t. ¬(t_i^α < lb_i(d) ∨ ub_i(d) < t_i^β). }
38  }else if(t_i^α < lb_i(d) ∨ ub_i(d) < t_i^β){
39   d_i ← d s.t. ¬(t_i^α < lb_i(d) ∨ ub_i(d) < t_i^β). }
40  ensure child_threshold_invariant and child_allcation_invariant.
41  foreach j ∈ Chld_i{
42   send (VALUE, i, s_i ∪ {(x_i, d_i)}, rlvl_i, t_{d_i,j}^α, t_{d_i,j}^β, trm_i)
43    to j. }
44  foreach j ∈ Nbr_i^l\Chld_i{
45   send (VALUE, i, {(x_i, d_i)}, rlvl_i, φ, φ, φ) to j. }
46  send (COST, i, s_i, rlvl_i, lb_i^*, ub_i^*) to parent_i.
47 }
```

The pseudo code of the algorithm is shown in Figure 4. The main procedure is the same as min-max ADOPT. Basically, the processing is similar to min-max ADOPT except for pruning.

In the root node r, $\alpha_r = \beta_r \wedge (ub_r^* = \alpha_r \vee lb_r^* = \beta_r)$ will eventually hold. Note that α_r and β_r are closed by the special feedback rule in the alpha_beta_invariant. Then, the root node selects its optimal or worst solution. Termination at the root node is announced by VALUE messages. The VALUE message contains α and β such that $\alpha = \beta$. When a parent of an agent i has been terminated, the agent i can terminate in a similar manner if i's boundaries are closed. On the other hand, if i's boundaries are still open, additional search is necessary to close the boundaries. However, $\alpha_i = \beta_i$ has already been held.

In the original ADOPT, a greedy strategy is used for such additional search. Each agent i intends to select its variable's value d such that $ub_i(d) = threshold_i$. Here $threshold_i$ represents the best lower bound in ADOPT. $threshold_i$ can be considered as α_i

except for overestimation. In a leaf node l, $ub_l(d) = threshold_l$ immediately hold. In other agents, a similar equation will eventually hold. However, such a strategy may not converge in the min-max methods because of the non-monotonicity of computation. Agents intend to take variables' values whose costs are on opposite sides of the boundaries according to their quantifiers. This often causes a situation in which an agent keeps its boundaries open ignoring the value of its variable that globally closes boundaries.

To avoid such a problem, we use a reset of search in the termination sequence. If boundaries are still open in an agent i when i's parent has been terminated, i resets the search by increasing the *root level* $rlvl_i$. The value of $rlvl_i$ is propagated to the descendant nodes of i by messages. After the reset of search, i now becomes a new root node. In the new problem, the variables of ancestor nodes of i have been fixed. Eventually, the boundary of cost converges in i. Then, i terminates the search. Although the reset of the search seems to be an inefficient way, it creates an opportunity to reduce search spaces. Instead of a complete reset, one side of alpha/beta that is not controlled by i is restored in i. In the case of the alpha-beta method, the boundary that is forced by i's root node is the exact value.

3.5 bi-threshold ADOPT

The idea of the alpha-beta method leads to another version of ADOPT with two *backtracking thresholds*. The backtracking threshold is a pruning parameter similar to alpha in alpha-beta method. However, while the alpha-beta method computes excessive costs for non-optimal solutions, ADOPT with backtracking threshold does not overestimate costs for all solutions. In the original ADOPT, a single backtracking threshold is employed. The backtracking threshold is mainly driven by lower bounds of costs. We insert another backtracking threshold that is mainly driven by upper bounds. Each agent i employs the following values for the thresholds. t_i^α and t_i^β represent backtracking thresholds of i. $t_{d,j}^\alpha$ and $t_{d,j}^\beta$ represent backtracking thresholds that are allocated for $j \in Chld_i$ and $d \in D_i$. For the computation of costs, $lb_{d,j}$, $ub_{d,j}$, $lb_i(d)$, $ub_i(d)$, lb_i^* and ub_i^* are employed. The key idea of the bi-threshold ADOPT is to hold $lb_i^* \leq t_i^\alpha \leq t_i^\beta \leq ub_i^*$. t_i^α and t_i^β are maintained with related $s_i^{t\alpha\beta}$ and $rlvl_i^{t\alpha\beta}$. $lb_{d,j}$, $ub_{d,j}$, $t_{d,j}^\alpha$ and $t_{d,j}^\beta$ are maintained with related $s_{d,j}$ and $rlvl_{d,j}$. Invariants and update operations of the algorithm are as follows.

threshold_consistency: $s_i^{t\alpha\beta}$ and $rlvl_i^{t\alpha\beta}$ must be equal to s_i and $rlvl_i$ respectively. If they are different, ($s_i^{t\alpha\beta}$, $rlvl_i^{t\alpha\beta}$, t_i^α, t_i^β) are reset to their initial value (s_i, $rlvl_i$, 0, ∞).

child_cost_consistency: For each $d \in D_i$ and $j \in Chld_i$, $s_{d,j}$ and $rlvl_{d,j}$ must be equal to s_i and $rlvl_i$ respectively. If they are different, ($s_{d,j}$, $rlvl_{d,j}$, $lb_{d,j}$, $ub_{d,j}$, $t_{d,j}^\alpha$, $t_{d,j}^\beta$) are reset to their initial value (s_i, $rlvl_i$, 0, ∞, 0, ∞, 0, ∞).

threshold_invariant: $lb_i^* \leq t_i^\alpha \leq t_i^\beta \leq ub_i^*$ must hold. t_i^α and t_i^β are updated by $\min(\max(t_i^\alpha, lb_i^*), ub_i^*)$ and $\min(\max(t_i^\beta, lb_i^*), ub_i^*)$.

child_threshold_invariant: For each $d \in D_i$ and $j \in Chld_i$, $lb_{d,j} \leq t_{d,j}^\alpha \leq t_{d,j}^\beta \leq ub_{d,j}$ must hold. $t_{d,j}^\alpha$ and $t_{d,j}^\beta$ are updated by $\min(\max(t_{d,j}^\alpha, lb_{d,j}), ub_{d,j})$ and $\min(\max(t_{d,j}^\beta, lb_{d,j}), ub_{d,j})$.

child_allocation_invariant: t_i^α and t_i^β are shared between agent i and its child agents. For d_i and $j \in Chld_i$, $t_{d_i,j}^\alpha$ and $t_{d_i,j}^\beta$ are maintained to hold $t_i^\alpha = \delta_i(d_i) + \sum_{j \in Chld_i} t_{d_i,j}^\alpha$ and $t_i^\beta = \delta_i(d_i) + \sum_{j \in Chld_i} t_{d_i,j}^\beta$. When the equations are not satisfied, several $t_{d_i,j}^\alpha$ and $t_{d_i,j}^\beta$ are increased or decreased until the equations are satisfied. child_threshold_invariant is also satisfied in the reallocation.

update (s_i, $rlvl_i$) using (s, $rlvl$): This operation is the same as alpha-beta ADOPT.

update ($s_i^{t\alpha\beta}$, $rlvl_i^{t\alpha\beta}$, t_i^α, t_i^β) using (s, $rlvl$, t^α, t^β): $s_i^{t\alpha\beta}$ and $rlvl_i^{t\alpha\beta}$ are updated by s and $rlvl$ respectively. t_i^α and t_i^β are updated by t^α and t^β respectively. When t^α or t^β exceeds lb_i^* or ub_i^*, they will be corrected by threshold_invariant. That is the main difference from alpha-beta ADOPT.

update ($s_{d,j}$, $rlvl_{d,j}$, $lb_{d,j}$, $ub_{d,j}$) using (s, $rlvl$, lb, ub): $s_{d,j}$ and $rlvl_{d,j}$ are updated by s and $rlvl$ respectively. $lb_{d,j}$ and $ub_{d,j}$ are updated using lb and ub in the same manner as in min-max ADOPT.

The pseudo code of the algorithm is shown in Figure 5. The main procedure is the same as for min-max ADOPT. The processing is similar to min-max/alpha-beta ADOPT except for pruning.

In the termination sequence, it employs reset of search. Even if bi-threshold ADOPT does not destroy true boundaries of costs, it presents the same problem in termination as alpha-beta ADOPT.

3.6 Correctness of algorithms

Min-max/bi-threshold ADOPT can be considered as straightforward in correctness because they are a natural limitation/extension of the original ADOPT. There are no major modifications in the maintenances of boundaries of costs. Therefore, we concentrate on the correctness of alpha-beta ADOPT. As shown in 3.4, the alpha-beta method enforces candidate boundaries (i.e. alpha and beta) of optimal cost regardless of true boundaries in subtrees in the search tree. In each level of the search tree, alpha and beta are maintained to represent the best boundaries. Then the alpha and beta are applied to its subtree. If such boundaries are infeasible, the cost value of the subtree is limited by the alpha and beta. Therefore, information of true boundaries is lost, except for the optimal costs. In alpha-beta ADOPT, true boundaries are computed beside the computation of alpha and beta. Because true boundaries narrow alpha and beta, at least one of the true boundaries is equal to alpha or beta when the optimal cost converges in the root agent.

4. EVALUATION

We performed experiments to evaluate the efficiency of the proposed methods. In this section, we show a comparison of their efficiency, and related considerations.

4.1 Problem settings

In this work, we show several important characteristics about search iterations as the first result because this class of problems implicitly contains many parameters, including topologies of modified pseudo-trees and placements of quantifiers, which mutually affect each other. We applied the methods to the following class of benchmark problems.

max-CSP: This class of problem represents maximum CSPs. Each binary cost function takes 0 or 1 for each tuple of values. We set the ratio t of tuples, whose cost is 1, to $\frac{1}{3}$ and $\frac{5}{9}$.

COP: In this class of problem, the cost of tuple is randomly set to an integer value between 0 and 10, selected with uniform probability.

Each problem consists of n ternary variables and $l \times n$ binary functions. l is a parameter for the density of binary functions. We show the results in the case of $l = 2$ which illustrate well the characteristics of algorithms. The ratio of universally quantified variables is set by the parameter u. The quantifiers are randomly selected with the parameter u. The results are averaged for fifty problem instances. The algorithms are denoted as follows. **min-max:**

min-max ADOPT. **al-bt:** alpha-beta ADOPT. **bi-thr:** bi-threshold ADOPT. We used simulation programs that iterate message cycles. In a message cycle, each agent reads the messages from its receiving queue. Then the agent writes messages for the sending queue. The messages in each queue are exchanged at the end of the cycle. The number of message cycles was limited to 10^5. Experiments are aborted if the number of message cycles raises above a predefined limit. In that case, the limit was used as the number of message cycles of the instance.

4.2 Results

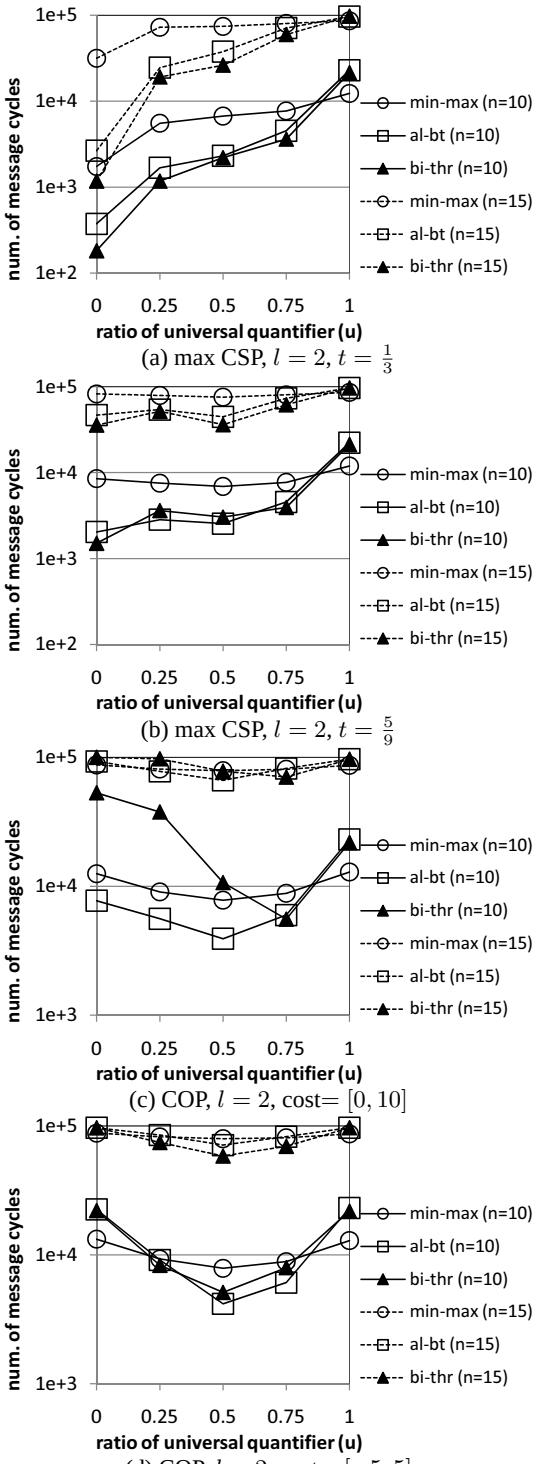
The number of message cycles is shown in Figure 6(a), (b) and (c). In the case of $n = 10$, all instances terminate correctly. While many instances reached to the limit number of message cycles in the case of $n = 15$, the shape of graphs is similar to the case of $n = 10$. In the case of max CSP, $t = \frac{1}{3}$, the number of message cycles of min-max increases according to the ratio of universally quantified variables. In terms of cost functions of these problems, the ratio of 0 and 1 is not even. Additionally, we use 0 as the lower limit of cost value. Therefore, it can be considered that minimizing problems are easier than maximizing problems. Results of other algorithms are similar to min-max. Note that the implementation of min-max is tuned to employ minimal contexts as described in the Section 3.3. Therefore, if pruning methods are insufficient, al-bt and bi-thr, which use contexts for all ancestors, may require more message cycles than min-max. The case of $u = 1$ is such a situation. In the cases of $0 \leq u \leq 0.75$, al-bt and bi-thr are more efficient than min-max. bi-thr is slightly more efficient than al-bt in several cases. The difference can be caused by the behavior of the boundaries. In al-bt, possible boundaries are enforced in a top down manner, Therefore, to avoid overestimation, the boundaries are not divided for multiple child agents. On the other hand, in bi-thr, a speculative division of thresholds is performed. Moreover, while each agent maintains one side of the boundary in al-bt, all agents maintain both backtracking thresholds in bi-thr. That affects the delay of the convergences. In the case of max CSP, $t = \frac{5}{9}$, the number of message cycles of min-max is almost concave up. In contrast, al-bt and bi-thr takes a lower number of message cycles when $u = 0$. That is due to pruning using lower limit 0.

In the case of COP, cost=$[0, 10]$, the number of message cycles of min-max is concave up. The cost functions of these problems take uniform cost values, and, in most cases, the cost values are non-zero. Therefore, when the ratio of minimizing and maximizing agents is even, the upper and lower bounds converge in fewer iterations. In the cases of $0 \leq u \leq 0.5$, bi-thr takes a greater number of message cycles. We believe that the main reason for the drawback is the optimistic search of ADOPT. As shown in [1, 10], when the cost values are in a wide range, ADOPT repeatedly improves lower bounds for already searched solutions. On the other hand, al-bt is better than bi-thr in such cases because of depth first strategy and boundaries enforced by ancestor agents.

Table 1 shows results regarding the number of message cycles in a termination sequence. In the table, root and global denotes number of message cycles at termination in the root and all agents. While search is reset in termination, the ratio of extra search is relatively low in each method. Effects of each sub-algorithm are shown in Table 2. tree-VALUE does not employ VALUE messages along back edges of a pseudo-tree. A linear-tree is based on a sequential graph instead of a pseudo-tree. The result shows effects of additional VALUE messages and pseudo-trees.

4.3 Symmetric problems

Algorithms shown in Section 3 are designed for problems whose



(a) max CSP, $l = 2$, $t = \frac{1}{3}$

(b) max CSP, $l = 2$, $t = \frac{5}{9}$

(c) COP, $l = 2$, cost= $[0, 10]$

(d) COP, $l = 2$, cost= $[-5, 5]$

Figure 6: number of message cycles

lower limit value is 0. We replaced the lower limit value with $-\infty$. For the negative cost values, a modification of child_alpha_beta_ invariant in alpha-beta ADOPT is also necessary. The modified invariant allocates ∞ as $\beta_{d,j}$ when the agent has multiple child nodes. Aggregation of cost value is now non-monotonic. However, the algorithms work because pruning for partial solutions, using a lower limit of 0, is disabled. Figure 6(d) shows the number of message cycles when cost values take from integer values between -5 and 5. The graphs are concave up because pruning does not

Table 1: number of message cycles in termination sequence
(max CSP, $n = 10$, $l = 2$, $t = \frac{5}{9}$)

u	0			0.5			1		
algorithm	root	global	ratio	root	global	ratio	root	global	ratio
min-max	6783	8500	1.25	4941	6905	1.40	9634	11946	1.24
al-bt	1598	2037	1.27	1906	2552	1.34	20215	22234	1.10
bi-thr	1317	1506	1.14	2634	3039	1.15	20221	21265	1.05

Table 2: effect of sub-algorithms (number of message cycles)
(max CSP, $n = 10$, $l = 2$, $t = \frac{5}{9}$, $u = 0.5$)

algorithms	all	tree-VALUE	linear-tree
min-max	6905	8420	12238
al-bt	2552	3047	3972
bi-thr	3039	3705	4585

(average depth of pseudo-trees is 8.8)

work effectively around $u = 0$.

5. RELATED WORKS

The QDCOP defined in this paper can be considered as an extension of QCSP/QDCSP [4, 2]. Indeed, basic QCSPs are represented as Q(D)COP. For example, instead of using hard constraints, $\forall x_0 \exists x_1 . x_0 \neq x_1$ and $\exists x_0 \forall x_1 . x_0 \neq x_1$ are represented using the cost function $f : D_0 \times D_1 \rightarrow \{0, 1\}$ where 0 and 1 denote true and false respectively. When $D_0 = D_1$, the optimal cost values of first and second problems are 0 and 1 respectively. A relaxation of QCSP is shown in [6]. On the other hand, our main purpose is to generalize DCOPs into quantified problems. Extended QCSP named QCOP/QCOP+ is shown in [3]. With QCOP/QCOP+, additional objective functions and constraints are defined for the QCSP. That class of problems is different from the problems in this paper. It is possible to address the min-max method with dynamic programming based algorithms [12]. However, when the induced width of the pseudo-tree is relatively large, simple dynamic programming cannot be applied because of the space complexity. Several methods that reduce message cycles [10, 14] can be applied to the proposed algorithms.

6. CONCLUSION

We proposed a Quantified Distributed Constraint Optimization problem (QDCOP) that extends the framework of Distributed Constraint Optimization problems (DCOPs). In QDCOPs, agents own existentially/universally quantified variables. The obtained bounds have to hold for any value of the universally quantified variables. An existentially quantified variable takes exactly one value for each context. For the QDCOP, we also propose several methods that are based on min-max/alpha-beta and ADOPT algorithms. We have evaluated these algorithms experimentally and describe the obtained results. Future works will include more detailed analysis for several graph structures and combinations of quantified variables, improvements of search algorithms applying efficient methods for DCOP solvers, and examining practical application domains of QDCOPs.

7. ACKNOWLEDGMENTS

This work was supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (B), 19300048.

8. REFERENCES

[1] S. M. Ali, S. Koenig, and M. Tambe. Preprocessing techniques for accelerating the dcop algorithm adopt. In *4th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1041–1048, 2005.

[2] S. Baba, N. Nishimura, A. Iwasaki, and M. Yokoo. Cooperative problem solving against adversary: Quantified distributed constraint satisfaction problem. In *The IJCAI-09 Workshop on Distributed Constraint Reasoning (DCR)*, 2009.

[3] M. Benedetti, A. Lallouet, and J. Vautard. Quantified constraint optimization. In *CP '08: Proceedings of the 14th international conference on Principles and Practice of Constraint Programming*, pages 463–477, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] H. M. Chen. *The computational complexity of quantified constraint satisfaction*. PhD thesis, Ithaca, NY, USA, 2004. Adviser-Kozen, Dexter.

[5] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 639–646, 2008.

[6] A. Ferguson and B. O'Sullivan. Quantified constraint satisfaction problems: from relaxations to explanations. In *In Proc. of Int. Joint. Conf. on Artificial Intelligence (IJCAI*, pages 74–79. Morgan Kaufmann, 2007.

[7] A. Kumar, B. Faltings, and A. Petcu. Distributed constraint optimization with structured resource constraints. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 923–930, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[8] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 310–317, 2004.

[9] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 438–445, 2004.

[10] T. Matsui, M. C. Silaghi, K. Hirayama, M. Yokoo, and H. Matsuo. Directed soft arc consistency in pseudo trees. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 1065–1072, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

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

[12] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *9th International Joint Conference on Artificial Intelligence*, pages 266–271, 2005.

[13] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.

[14] M. C. Silaghi and M. Yokoo. Adopt-ing: unifying asynchronous distributed optimization with asynchronous backtracking. *Journal of Autonomous Agents and Multi-Agent Systems*, 19(2):89–123, 10 2009.

[15] W. Yeoh, A. Felner, and S. Koenig. Bnb-adopt: an asynchronous branch-and-bound dcop algorithm. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 591–598, 2008.

[16] R. Zivan. Anytime local search for distributed constraint optimization. In *Twenty-Third AAAI Conference on Artificial Intelligence*, pages 393–398, 2008.