Leximin Asymmetric Multiple Objective Distributed Constraint Optimization Problem

Computational Intelligence, Volume 000, Number 000, 0000

Leximin Asymmetric Multiple Objective Distributed Constraint Optimization Problem

Toshihiro Matsui and Hiroshi Matsuo

Nagoya Institute of Technology, Gokiso-cho Showa-ku Nagoya 466-8555, Japan

Marius Silaghi

Florida Institute of Technology, Melbourne FL 32901, United States of America

Katsutoshi Hirayama

Kobe University, 5-1-1 Fukaeminami-machi Higashinada-ku Kobe 658-0022, Japan

Makoto Yokoo

Kyushu University, 744 Motooka Nishi-ku Fukuoka 819-0395, Japan

The Distributed Constraint Optimization Problem (DCOP) lies at the foundations of multiagent cooperation. With DCOPs, the optimization in distributed resource allocation problems is formalized using constraint optimization problems. The solvers for the problem are designed based on decentralized cooperative algorithms that are performed by multiple agents. In a conventional DCOP, a single objective is considered.

The Multiple Objective Distributed Constraint Optimization Problem (MODCOP) is an extension of the DCOP framework, where agents cooperatively have to optimize simultaneously multiple objective functions. In the conventional MODCOPs, a few objectives are globally defined and agents cooperate to find the Pareto optimal solution. However, such models do not capture the interests of each agent. On the other hand, in several practical problems, the share of each agent is important. Such shares are modeled as preference values of agents. This class of problems can be defined using the MODCOP on the preferences of agents. In particular, we define optimization problems based on leximin ordering and Asymmetric DCOPs (Leximin AMODCOPs). The leximin defines an ordering among vectors of objective values. In addition, Asymmetric DCOPs capture the preferences of agents. Since the optimization based on the leximin ordering improves the equality among the satisfied preferences of the agents, this class of problems is important. We propose several solution methods for Leximin AMODCOPs generalizing traditional operators into the operators on sorted objective vectors and leximin. The solution methods applied to the Leximin AMODCOPs are based on pseudo trees. Also, the investigated search methods employ the concept of boundaries of the sorted vectors.

Key words: leximin, preference, multiple objectives, Distributed Constraint Optimization, multiagent, cooperation.

1. INTRODUCTION

The Distributed Constraint Optimization Problem (DCOP) lies at the foundations of multiagent cooperation (Farinelli et al., 2008; Modi et al., 2005; Petcu and Faltings, 2005; Zivan, 2008). With DCOPs, the optimization in distributed resource allocation including distributed sensor networks (Zhang et al., 2005), meeting scheduling (Maheswaran et al., 2004), disaster response (Ramchurn et al., 2010) and smart grids (Miller et al., 2012) is formalized using constraint optimization problems. In a conventional DCOP, a single objective is optimized. The solvers for the problem are designed based on decentralized cooperative algorithms that are performed by multiple agents. The solution methods are categorized into complete and incomplete methods. Several complete methods employ techniques including dynamic programming and tree search that are performed based on a graph structure called pseudo tree. DPOP (Petcu and Faltings, 2005) is a solution method based on dynamic programming, which performs bucket elimination (Dechter, 1999) on a pseudo tree. ADOPT (Modi et al., 2005) performs a tree search with memory-bounded dynamic programming. On the other hand, several incomplete methods are based on local search approach (Zivan, 2008; Vinyals et al., 2011).

The Multiple Objective Distributed Constraint Optimization Problem (MODCOP) is an extension of the DCOP framework, where agents cooperatively have to optimize simul-

taneously multiple objective functions (Delle Fave et al., 2011). For the case of multiple objectives, evaluation values are defined as vectors of objective values. Agents cooperate to find the Pareto optimal solution. In Delle Fave et al. (2011), a bounded Max-Sum algorithm for MODCOPs has been proposed. A solution method based on tree-search and dynamic programming has also been applied to MODCOPs (Matsui et al., 2012).

In conventional MODCOPs, a few objectives are globally defined for the whole system. However, such models do not capture the interests of each agent. In several practical problems, the share of each agent is important. Such shares are modeled as preference values of agents. This point of view recently has been addressed in the context of DCOPs which are designed for dedicated resource allocation problems (Netzer and Meisels, 2011; Matsui and Matsuo, 2012; Netzer and Meisels, 2013a,b). These problems define multiple objective functions, optimizing the preferences for all the agents.

In this work, we address a class of MODCOPs on the preferences of agents. In particular, we focus on problems where the importance of objective functions is based on the leximin ordering (referred to as Leximin AMODCOPs). Leximin (Moulin, 1988; Bouveret and Lemaître, 2009) is a well-known social welfare. Since the optimization based on the leximin ordering improves the equality among the satisfied preferences of the agents, this class of problems is important. The solution methods applied to the Leximin AMODCOPs are based on pseudo trees. Also, the investigated search methods employ the concept of boundaries of the sorted vectors. Our major contributions are as follows. i) We define the Leximin AMODCOP that is similar to classes of Asymmetric DCOPs and MODCOPs in (Netzer and Meisels, 2011, 2013a), while its criteria of optimization is based on leximin ordering. ii) We show that the proposed problem can be solved using a dynamic programming based solution method on a modified pseudo tree, and an extension of DPOP (Petcu and Faltings, 2005) is applied to the problem. iii) To employ a tree search based method, we define boundaries on leximin, and present a solution method which is basically similar to ADOPT (Modi et al., 2005).

The rest of the paper is organized as follows. In Section 2, we introduce the backgrounds of the study including Distributed Constraint Optimization Problem, multiple objective problem, preferences of agents, and related works. We propose the Leximin Asymmetric Multiple Objective DCOP (Leximin AMODCOP) in Section 3. Then in Section 4 we show how solution methods based on pseudo trees can be applied to Leximin AMODCOPs. A dynamic programming method (also known as bucket elimination) and an asynchronous tree search method are generalized for Leximin AMODCOPs. In Section 5, we describe an additional method that employs the individual information of agents. The proposed methods are experimentally investigated in Section 6. In Section 7, we present discussions and future works, before concluding our study in Section 8.

2. PRELIMINARY

2.1. Distributed Constraint Optimization Problem

A Distributed Constraint Optimization Problem (DCOP) is defined as follows.

Definition 1 (Distributed Constraint Optimization Problem): A Distributed Constraint Optimization Problem is defined by (A, X, D, F) where A is a set of agents, X is a set of variables, D is a set of domains of variables, and F is a set of objective functions. Variable $x_i \in X$ represents a state of agent $i \in A$. Domain $D_i \in D$ is a discrete finite set of values for x_i. An objective function $f_{i,j}(x_i, x_j) \in F$ defines a utility extracted for each pair of assignments to x_i and x_j. The objective value of assignment $\{(x_i, d_i), (x_j, d_j)\}$ is defined by the binary function $f_{i,j} : D_i \times D_j \to \mathbb{R}$. For an assignment $\mathcal{A}$ of variables, the global

objective function $F(\mathcal{A})$ is defined as $F(\mathcal{A}) = \sum_{f_{i,j} \in F} f_{i,j}(\mathcal{A}_{|x_i}, \mathcal{A}_{|x_j})$. The value of x_i is controlled by agent i. Agent i locally knows the objective functions that relate to x_i in the initial state. The goal is to find a global optimal assignment $\mathcal{A}^*$ that maximizes the global objective value.

The computation to find the optimal solution is a distributed algorithm. We assume that each pair of agents has a communication route on an overlay network. For the sake of simplicity, we assume that all the objective functions are binary. Also, the state of each agent is represented by only one variable. While several studies address the problems with n-ary functions and agent states represented by multiple variables (Yokoo and Hirayama, 1998; Pecora et al., 2006; Vinyals et al., 2011), we concentrate on the primitive case for simplicity.

2.2. Multiple objective problem

The Multiple Objective DCOP (MODCOP) is a generalization of the DCOP framework (Delle Fave et al., 2011). With MODCOPs, multiple objective functions are defined over the variables. The objective functions are simultaneously optimized based on appropriate criteria. The tuple with the values of all the objective functions for a given assignment is called *objective vector*.

Definition 2 (Objective vector): An objective vector $\mathbf{v}$ is defined as $[v_0, \cdots, v_K]$. Here, v_k is an objective value. The Vector $\mathbf{F}(X)$ of objective functions is defined as $[F^0(X^0), \cdots, F^K(X^K)]$, where X^k is the subset of X on which F^k is defined. $F^k(X^k)$ is an objective function for objective k. For assignment $\mathcal{A}$, the vector $\mathbf{F}(\mathcal{A})$ of the functions returns an objective vector $[v_0, \cdots, v_K]$. Here, $v_k = F^k(\mathcal{A}^k)$ for each objective k.

For two vectors $\mathbf{v}$ and $\mathbf{v}'$ of k objectives, $\mathbf{v} > \mathbf{v}'$ denotes that $v_k > v'_k$ for all objectives k. Similarly, $\mathbf{v} \geqslant \mathbf{v}'$ denotes that $v_k > v'_k \vee v_k = v'_k$ for all objectives k.

Objective vectors are compared based on Pareto dominance. For maximization problems, the dominance between two vectors is defined as follows: Vector $\mathbf{v}$ dominates $\mathbf{v}'$ if and only if $\mathbf{v} \geqslant \mathbf{v}'$, and $v_k > v'_k$ for at least one objective k. Similarly, Pareto optimality on the assignments is defined as follows: Assignment $\mathcal{A}^*$ is Pareto optimal if and only if there is no other assignment $\mathcal{A}$, such that $\mathbf{F}(\mathcal{A}) \geqslant \mathbf{F}(\mathcal{A}^*)$, and $F^k(\mathcal{A}) > F^k(\mathcal{A}^*)$ for at least one objective k. In previous studies of MODCOPs (Delle Fave et al., 2011), each objective function $f_{i,j}(x_i, x_j)$ in the original DCOPs is extended to a vector $[f^0_{i,j}(x_i, x_j), \cdots, f^K_{i,j}(x_i, x_j)]$. $F^k(\mathcal{A}^k)$ is therefore defined as $\sum_{f^k_{i,j} \in F^k} f^k_{i,j}(\mathcal{A}^k_{|x_i}, \mathcal{A}^k_{|x_j})$ for each objective k. Also, all the objectives are evaluated for the same assignment. Namely, $\mathcal{A}^0 = \mathcal{A}^1 = \cdots = \mathcal{A}^K$. Multiple objective problems generally have a set of Pareto optimal solutions that form a Pareto front. With an appropriate social welfare that defines an order on objective vectors, traditional solution methods for single objective problems find a Pareto optimal solution.

2.3. Social welfare

There are several criteria of social welfare (Sen, 1997) and scalarization methods (Marler and Arora, 2004). A well-known social welfare function is defined as the summation $\sum_{k=0}^{K} F^k(\mathcal{A}^k)$ of objectives. The maximization of this summation ensures Pareto optimality. This summation is a 'utilitarian' criterion, since it represents the total value of the objectives, while it does not capture the equality on these objectives. On the other hand, the minimization $\min_{k=0}^{K} F^k(\mathcal{A}^k)$ on objectives emphasizes the objective of the worst value. Although the maximization of the minimum objective (maximin) reduces the worst complaint among

all the objectives, the optimal assignment on the maximin is not Pareto (but weak Pareto) optimal. To improve maximin, the summation welfare function is additionally employed. A social welfare is defined as a vector $[\min_{k=0}^{K} F^k(\mathcal{A}^k), \sum_{k=0}^{K} F^k(\mathcal{A}^k)]$ with an appropriate definition of dominance. When the maximization on the minimization part dominates that on the summation part, it can be considered as a (partial) lexicographical ordering that yields the Pareto optimal solution, similar to the lexicographic weighted Tchebycheff method (Marler and Arora, 2004).

Another social welfare, called *leximin* (Moulin, 1988; Bouveret and Lemaître, 2009), is defined using a lexicographic order on objective vectors whose values are sorted in ascending order.

Definition 3 (Sorted vector): A sorted vector based on vector $\mathbf{v}$ is the vector where all the values of $\mathbf{v}$ are sorted in ascending order.

Definition 4 (Leximin): Let $\mathbf{v}$ and $\mathbf{v}'$ denote vectors of the same length $K+1$. Let $[v_0, \cdots, v_K]$ and $[v'_0, \cdots, v'_K]$ denote sorted vectors of $\mathbf{v}$ and $\mathbf{v}'$, respectively. Also, let $\prec_{leximin}$ denote the relation of the leximin ordering. $\mathbf{v} \prec_{leximin} \mathbf{v}'$ if and only if $\exists t, \forall t' < t, v_{t'} = v'_{t'} \wedge v_t < v'_t$.

Example 1 (Example of leximin): Consider two sorted vectors $\mathbf{v} = [1, 1, 3]$ and $\mathbf{v}' = [1, 2, 2]$. Note that, for both vectors, the summation of elements are the same value 5. While the first elements v_0 and v'_0 of the vectors are the same value 1, the second elements v_1 and v'_1 are compared as $1 < 2$. Therefore, $\mathbf{v} \prec_{leximin} \mathbf{v}'$. On the other hand, for $\mathbf{v} = [2, 2, 2]$ and $\mathbf{v}' = [1, 2, 3]$, the sorted vectors are compared as $\mathbf{v}' \prec_{leximin} \mathbf{v}$, since $v'_0 < v_0$.

The maximization on the leximin ordering ensures Pareto optimality. The leximin is an 'egalitarian' criterion, since it reduces the inequality on objectives. It is also considered as an improved version of maximin. The above property of the leximin is important for the preferences of agents. We focus on the leximin social welfare.

2.4. Preferences of agents

While previous studies address MODCOPs (Delle Fave et al., 2011; Matsui et al., 2012), their goal is to optimize a few global objectives. Agents cooperate with each other to optimize those global objectives. On the other hand, in practical resource allocation problems, such as power supply networks, each agent has a strong interest for its share of the result. Hence there is the need for a more appropriate model where the objectives represent the preferences of agents. This class of problems has two key characteristics: 1) Each agent individually has its set of objective functions whose aggregated value represents its preferences, while several agents are related, since subsets of their variables are in the scope of the same function. 2) The problem is a MODCOP where a solution is characterized by an objective vector consisting of objective values that are individually aggregated for different agents.

In Matsui and Matsuo (2012), a resource constrained DCOP, which is designed for resource allocation on power supply networks, is extended to a MODCOP on the preferences of agents. In that study, min-max as well as min-max with the additional summation was introduced for minimizing problems. In addition, to reduce inequality among agents, a few first methods that consider the variance of objective values were shown. A general representation of the objectives of individual agents has been proposed as Asymmetric DCOP (ADCOP) in Grinshpoun et al. (2013). In the ADCOP, two different objective functions are asymmetrically defined for a pair of two agents. Here, each objective function represents the valuation for one of the agents. Several classes of ADCOPs with multiple objectives for individual agents have been proposed in Netzer and Meisels (2011, 2013a,b). We focus

on a class of ADCOPs optimizing the leximin social welfare. Since the leximin is known to reduce the inequality among agents, it helps define an important class of MODCOPs on preferences of agents.

2.5. **Related works**

In the Asymmetric DCOP shown in Grinshpoun et al. (2013), each value of a function is defined by a pair of values that correspond to different directions on an edge of a constraint graph. Therefore, an agent has its local view based on the direction of connected edges. However, its optimal solution corresponds to the maximum summation over all functions and directions. A major issue of the study is privacy-loss.

In Netzer and Meisels (2011, 2013a,b); Matsui and Matsuo (2012), resource allocation problems similar to ones in this study have been addressed. On the other hand, we addressed an extension of ADCOPs based on the leximin social welfare. While Theil based social welfare has been addressed in Netzer and Meisels (2011), that solution method is a local search. A search algorithm based on a total order on variables has been proposed in Netzer and Meisels (2013a), while here we generalize the common computation to pseudo trees.

Several variations of DCOPs (Maheswaran et al., 2004; Petcu et al., 2008) model private interests of agents. In Petcu et al. (2008), the VCG mechanism is employed with DCOPs. On the other hand, our interest in this study is the optimization problems on leximin social welfare.

Several solution methods for a centralized constraint optimization problem on the leximin ordering have been proposed in Bouveret and Lemaître (2009). In the previous study, the definition of the problem is an extension of constraint optimization problem, where a part of variables represent the values of objectives, while we mainly focus on a variant of Asynchronous DCOPs with objective vectors. The previous solution methods employ several techniques including branch-and-bound with leximin ordering and several types of constraints to represent relationships of leximin. The techniques are integrated into solution methods for centralized solvers. While how those techniques can be decomposed into distributed algorithms are an interesting issue, we mainly investigate a dynamic programming approach on objective vectors that can be applied to several existing techniques for DCOP solvers, as the first study.

3. LEXIMIN MULTIPLE OBJECTIVE OPTIMIZATION ON PREFERENCES OF AGENTS

3.1. Problem definition

A Leximin MODCOP on preferences of agents (Leximin AMODCOP) is defined as follows.

Definition 5 (Leximin Asymmetric MODCOP on preferences of agents): A leximin MODCOP on preferences of agents is defined by (A, X, D, F), where A, X and D are similarly defined as for the DCOP in Definition 1. Agent $i \in A$ has its local problem defined on $X_i \subseteq X$. Here, $\exists(i,j), i \neq j \wedge X_i \cap X_j \neq \emptyset$. F is a set of objective functions $f_i(X_i)$. The function $f_i(X_i) : D_{i_0} \times \cdots \times D_{i_k} \rightarrow \mathbb{R}$ represents the objective value for agent i based on the variables in $X_i = \{x_{i_0}, \cdots, x_{i_k}\}$. For an assignment $\mathcal{A}$ of variables, the global objective function $\mathbf{F}(\mathcal{A})$ is defined as $[f_0(\mathcal{A}_0), \cdots, f_{|A|-1}(\mathcal{A}_{|A|-1})]$. Here, $\mathcal{A}_i$ denotes the projection of the assignment $\mathcal{A}$ on X_i . The goal is to find the assignment $\mathcal{A}^*$ that maximizes the global objective function based on the leximin ordering.

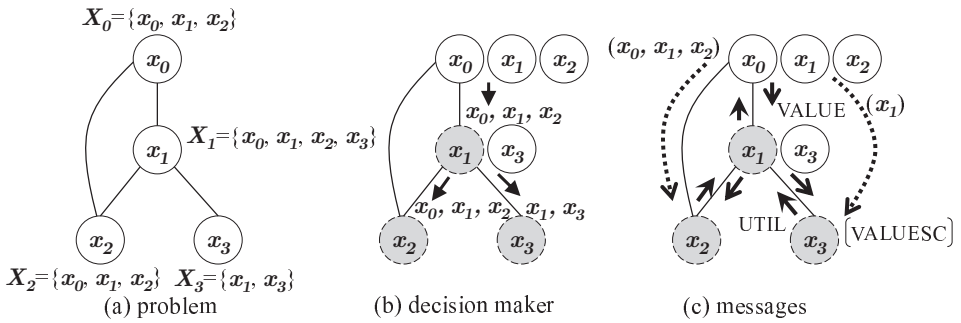


FIGURE 1. Pseudo tree for local problems

We concentrate on the case where each agent has a single variable and primitively relates to its neighborhood agents with binary functions, for simplicity. On the other hand, the functions are asymmetrically defined and locally aggregated.

As shown in Definition 5, each agent i has a function $f_i(X_i)$ that represents i's local problem. In a simple case, the local problem is defined as a part of an ADCOP where $f_i(X_i)$ is the summation of the corresponding functions in the ADCOP. In an ADCOP, variable x_i of agent i relates to other variables by objective functions. When x_i relates to x_j, agent i evaluates an objective function $f_{i,j}(x_i, x_j)$. On the other hand, j evaluates another function $f_{j,i}(x_j, x_i)$. Based on this ADCOP, a local problem is represented as $f_i(X_i) = \sum_{j \in Nbr_i} f_{i,j}(x_i, x_j)$ for agent i, aggregating objective functions among i and its neighborhood agents Nbr_i.

4. SOLUTION METHOD BASED ON PSEUDO TREE

In this section, we propose solution methods for Leximin AMODCOPs. The solution methods are extensions of DCOP solution methods on pseudo trees. For our solution methods, a modified pseudo tree is described in Subsection 4.1. Then, in Subsection 4.2, we present a dynamic programming based method, which is an extension of DPOP (Petcu and Faltings, 2005). Several operations are generalized with sorted vectors, and we show that the sorted vectors can be decomposed and aggregated with dynamic programming. As another approach, we propose a search method similar to ADOPT (Modi et al., 2005) by adding several components. In Subsection 4.3, we present a tree search method, which is a reorganization in time of the solution based on dynamic programming. Here, we introduce boundaries on sorted vectors to compare partially unknown vectors. The search method is extended in Subsection 4.4 with pruning that employ the global lower bound of objective vectors. To reduce the delay of the search method, in Subsection 4.5 it is shown how to employ top-down shortcut messages. In the rest of the section, we also address several issues including the representation of objective vectors, correctness and complexity. Further, the search method can be extended with bottom-up shortcut messages, integrating individual preferences of agents. This extension will be shown in Section 5.

4.1. Pseudo tree for local problems

Several solution methods for DCOPs are based on pseudo trees on constraint networks (Modi et al., 2005; Petcu and Faltings, 2005). A pseudo tree of the problem is a depiction of its constraint network (adding directions to edges and levels for the nodes), based on a spanning tree in which there are no edges between different sub-trees of the

corresponding spanning tree. Such pseudo trees can be generated using several algorithms, including the depth-first traversal on the constraint network. Edges of the spanning tree are called tree-edges, while other edges are called back-edges. Based on the pseudo tree, the following notations are defined for each agent i.

- p_i: parent agent.
- Ch_i: set of child agents.
- $Nbrs_i$: set of neighborhood agents.
- $Nbrs_i^l$: set of lower neighborhood agents, i.e. the child and pseudo child nodes.
- $Nbrs_i^u$: set of upper neighborhood agents, i.e. the parent and pseudo parent nodes.

A partial order on a set of agents is defined based on the tree edges of a pseudo tree. The priorities induced by this order are used for breaking ties during decision making.

Figure 1(a) shows a pseudo tree for a problem. In the figure, four nodes represent agents/variables, while four edges represent functions. In our problem, each edge stands for a pair of two asymmetric objective functions. Since an objective function is evaluated by only one related agent, each agent has to evaluate all the related objective functions. Namely, each agent has to manage all the assignments for its local problem. Therefore, the value of a variable x_i is decided by the highest neighborhood agent whose variable relates to the variable x_i with an edge. Hence a modification of pseudo trees is necessary. Figure 1(b) shows the pseudo tree modified from (a). The priority on decisions of assignments is represented as shown in (b).

To set up the data structures needed for this pseudo tree, agent i computes the following information.

- XX_i^{upr}: A set of pairs of variables. $(x_k, x_h) \in XX_i^{upr}$ specifies that x_k relates to x_h in a higher level, with a function. This data structure is employed to compute the highest agent that has to be a decision maker for each variable.
- X_i^{dcd}: The set of variables whose values are determined by agent i.
- X_i^{sep}: The set of *separator* variables that are shared between the sub-tree rooted at i and another part of the problem.

Each agent needs X_i^{dcd} and X_i^{sep} to perform solution methods shown later. To compute these sets, XX_i^{upr} is updated with X_i^{dcd} and X_i^{sep} in a bottom-up manner on a pseudo tree. Except at the root agent in the pseudo tree, the information is recursively computed as follows.

$$XX_i^{upr} = \bigcup_{h \in Nbrs_i^u} \{(x_i, x_h)\} \cup \{(x_a, x_b) | (x_a, x_b) \in \bigcup_{j \in Ch_i} XX_j^{upr} \wedge x_b \neq x_i\} \quad (1)$$

$$X_i^{dcd} = \{x_a | (x_a, x_i) \in \bigcup_{j \in Ch_i} XX_j^{upr} \wedge \nexists b, (x_a, x_b) \in XX_i^{upr}\} \quad (2)$$

$$X_i^{sep} = \left(\{x_i\} \cup \bigcup_{h \in Nbrs_i^u} \{x_h\} \cup \bigcup_{j \in Ch_i} X_j^{sep} \right) \setminus X_i^{dcd} \quad (3)$$

Equation (1) enables defining the agent assigning x_i as the highest placed agent in the set of those having a relation with some node in the sub-tree rooted as x_i (upper neighbors of i and upper neighbors of variables in sub-trees defined by its children, and found above i). Equation (2) defines the variables assigned by agent i as the lower neighbors of x_i in sub-trees defined by children, and which do not have upper neighbors above i. Equation (3) defines the separator variables as those in the upper neighbors of x_i and its sub-tree, and of that are not controlled by agent i or its children. Intuitively, $(x_k, x_h) \in XX_i^{upr}$ means that the value of x_k may be determined by x_h, since x_k and x_h relate by an asymmetric

function, and x_h is in a higher level than x_k. Note that for the same x_k, different x_h may exist at the same time. XX_i^{upr} is aggregated in a bottom-up manner for the sub-tree rooted at agent i, while agent i eliminates all (x_a, x_i) from XX_i^{upr} (Equation (1)). If at least one child agent j of i has $(x_a, x_i) \in XX_j^{upr}$ and i does not have any $(x_a, x_b) \in XX_i^{upr}$ such that $x_b \neq x_i$, x_a is never referred by any agents in higher levels than x_i. Therefore, X_i^{dcd} must contain x_a (Equation (2)). Other variables that will be determined by agents in higher levels are contained in X_i^{sep} (Equation (3)). Note that $x_i \in X_i^{sep}$ and $x_i \notin X_i^{dcd}$, unlike the standard definition of separators on pseudo trees.

On the other hand, in the root agent, $XX_i^{upr} = \emptyset$, $X_i^{dcd} = \{x_i\} \cup \{x_a | (x_a, x_i) \in \bigcup_{j \in Ch_i} XX_j^{upr}\}$ and $X_i^{sep} = \emptyset$. Note that the root agent also determines the value of its own variable. The actual computation is performed as a distributed processing, after the preprocessing of generating a pseudo tree. Each non-root agent i sends XX_i^{upr}, X_i^{dcd} and X_i^{sep} to its parent agent p_i in a bottom-up manner.

For the example shown in Figure 1, XX_i^{upr}, X_i^{dcd} and X_i^{sep} are as follows.

- $XX_2^{upr} = \{(x_2, x_0), (x_2, x_1)\}$, $X_2^{dcd} = \emptyset$, $X_2^{sep} = \{x_0, x_1, x_2\}$.
- $XX_3^{upr} = \{(x_3, x_1)\}$, $X_3^{dcd} = \emptyset$, $X_3^{sep} = \{x_1, x_3\}$.
- $XX_1^{upr} = \{(x_1, x_0), (x_2, x_0)\}$, $X_1^{dcd} = \{x_3\}$, $X_1^{sep} = \{x_0, x_1, x_2\}$.
- $XX_0^{upr} = \emptyset$, $X_1^{dcd} = \{x_0, x_1, x_2\}$, $X_1^{sep} = \emptyset$.

To generate pseudo trees, several heuristics with depth first search traversal can be applied. Here, we employ maximum-degree heuristic (Hamadi et al., 1998) as a well-known one, where the node connected to the largest number of other nodes, ignoring directions of edges, is selected at first in the traversal. Since variables of high-degree nodes are influential in the modification of decision makers, we prefer them to be the decision makers in higher levels, if possible.

4.2. Computation of the optimal objective vector

We apply a computation of the optimal objective value, which is employed in the solution method DPOP (Petcu and Faltings, 2005), to the Leximin AMODCOP. The computation is performed on the modified pseudo tree shown in Subsection 4.1.

In this computation, a problem is decomposed into sub-problems with parts of objectives, and objective vectors for the sub-problems are aggregated. When the total number of objectives is K, a vector $\mathbf{v}$ such that $|\mathbf{v}| \leqslant K$ is called a *partial objective vector* .

For the aggregation of objective values, we define an addition on vectors that is different from the common definition. The addition is the operator concatenating all the values.

Definition 6 (Addition on vectors): Let $\mathbf{v}$ and $\mathbf{v}'$ denote vectors $[v_0, \cdots, v_K]$ and $[v'_0, \cdots, v'_{K'}]$. The addition $\mathbf{v} \oplus \mathbf{v}'$ of the two vectors gives a vector $\mathbf{v}'' = [v''_0, \cdots v''_{K+K'+1}]$ where each value in $\mathbf{v}''$ is a distinct value in $\mathbf{v}$ or $\mathbf{v}'$. Namely, $\mathbf{v}''$ consists of all values in $\mathbf{v}$ and $\mathbf{v}'$. As a normalization, the values in $\mathbf{v}''$ are sorted in ascending order.

The computation of the optimal objective vector is recursively defined. The optimal objective vector $g_i^*(\mathcal{A}_i^{sep})$ for assignment $\mathcal{A}_i^{sep}$ of variables X_i^{sep} whose values are determined by i's ancestor nodes and parent node is represented as follows.

$$g_i^*(\mathcal{A}_i^{sep}) = \max_{\mathcal{A}_i^{dcd} \text{ for } X_i^{dcd}} g_i(\mathcal{A}_i^{sep} \cup \mathcal{A}_i^{dcd}) \tag{4}$$

$$g_i(\mathcal{A}) = [f_i(\mathcal{A}_{|X_i})] \oplus \bigoplus_{j \in Ch_i, \mathcal{A}_j^{sep} \subseteq \mathcal{A}} g_j^*(\mathcal{A}_j^{sep}) \tag{5}$$

Here, $\mathcal{A}_i^{dcd}$ denotes an assignment of the variables in X_i^{dcd} whose values are determined by i. The operator $\oplus$ denotes aggregation of objective values. While the summation operator is used in common DCOPs, we aggregate objective vectors using the operator shown in Definition 6. Similarly, $\max$ denotes the maximization on the leximin ordering. This computation is a dynamic programming based on the following proposition.

Proposition 1 (Invariance on leximin relation): Let $\mathbf{v}$ and $\mathbf{v}'$ denote vectors of the same length. Also, let $\mathbf{v}''$ denote another vector. If $\mathbf{v} \prec_{leximin} \mathbf{v}'$, then $\mathbf{v} \oplus \mathbf{v}'' \prec_{leximin} \mathbf{v}' \oplus \mathbf{v}''$.

Proof. Let $[v_0, \cdots, v_K]$ and $[v'_0, \cdots, v'_K]$ denote values in the sorted vectors of $\mathbf{v}$ and $\mathbf{v}'$, respectively. From the definition of leximin, there is a value t such that $\forall t' < t, v_{t'} = v'_{t'} \wedge v_t < v'_t$. Let t'' denote the value such that $v_{t''} < v_t \wedge v_{t''+1} = v_t$. Namely, $v_{t''}$ is the value just before the sequence of values equal to v_t. Note that $t'' + 1 \leqslant t$. In the case of $t = 0$, the value of t'' is generalized using -1. Consider the values in the sorted vectors of $\mathbf{v} \oplus \mathbf{v}''$ and $\mathbf{v}' \oplus \mathbf{v}''$. When vector $\mathbf{v}''$ contains k values smaller than v_t, then there are $t'' + k$ such values in both sorted vectors of $\mathbf{v} \oplus \mathbf{v}''$ and $\mathbf{v}' \oplus \mathbf{v}''$. Namely, the sequences of values less than v_t are the same in both of the sorted vectors. When vector $\mathbf{v}''$ contains k' values equal to v_t, $\mathbf{v} \oplus \mathbf{v}''$ contains a sequence of at least $(t - t'') + k'$ values equal to v_t. On the other hand, $\mathbf{v}' \oplus \mathbf{v}''$ contains a sequence of $(t - 1 - t'') + k'$ values equal to v_t. The above property also holds in the cases where $k = 0$ and/or $k' = 0$. Now, we can conclude that the sequences of the first $(t'' + k) + (t - 1 - t'') + k'$ values are the same in both sorted vectors of $\mathbf{v} \oplus \mathbf{v}''$ and $\mathbf{v}' \oplus \mathbf{v}''$, while the next values are the value equal to v_t and a value greater than v_t, respectively. Therefore, $\mathbf{v} \oplus \mathbf{v}'' \prec_{leximin} \mathbf{v}' \oplus \mathbf{v}''$. □

The maximization in Expression (4) compares objective vectors for the same assignment $\mathcal{A}_i^{sep}$ that will produce the same partial objective vector. The above computation therefore correctly calculates the globally optimal objective vector.

After the computation of the optimal objective vector, the root agent i determines its optimal assignment $\mathcal{A}_i^{dcd*}$ such that $g_i(\emptyset \cup \mathcal{A}_i^{dcd*}) = g_i^*(\emptyset)$. $\mathcal{A}_j^{sep*} \subseteq \mathcal{A}_i^{dcd*}$ is then computed for each child $j \in Ch_i$. Similarly, any non-root agent i computes $\mathcal{A}_i^{dcd*}$ such that $g_i(\mathcal{A}_i^{sep*} \cup \mathcal{A}_i^{dcd*}) = g_i^*(\mathcal{A}_i^{sep*})$, and $\mathcal{A}_j^{sep*} \subseteq \mathcal{A}_i^{sep*} \cup \mathcal{A}_i^{dcd*}$ for each child $j \in Ch_i$. The protocol of the modified version of DPOP is basically the same as the original one. The original DPOP performs two phases of computation on a pseudo tree. In the first phase, each agent computes the optimal objective value for each assignment to its separator variables and all assignments to the variables in the sub-tree rooted at the agent, propagating UTIL messages in a bottom-up manner. Then, in the second phase, each agent computes the optimal assignment propagating VALUE messages in a top-down manner. See Petcu and Faltings (2005) for details of the protocol. The modified version of DPOP also employs two types of messages UTIL and VALUE shown in Figure 1(c). In the modified version, a UTIL messages contains a table of assignments and sorted objective vectors, while the original one contains a table of assignments and objective values. In addition, separator variables are also different from original ones due to the modification of decision makers. After the processing of the modified pseudo tree, agents compute the optimal objective vector. In this computation, UTIL messages are propagated in a bottom-up manner. Each agent i sends $g_i^*(\mathcal{A}_i^{sep})$ to its parent p_i using a UTIL message. Then the optimal assignment is computed propagating VALUE messages in a top-down manner. Each agent i sends $\mathcal{A}_j^{sep*}$ to its child agents $j \in Ch_i$ using VALUE messages. While the protocol of DPOP is simple, the size of UTIL messages and memory used to store $g_i^*(\mathcal{A}_i^{sep})$ of all the assignments exponentially increases with the size $|X_i^{sep}|$ of i's separator.

4.3. Search method

As another type of solution methods on pseudo trees, we apply solution methods based on tree search and partial dynamic programming to the Leximin AMODCOPs. The methods are variations of ADOPT (Modi et al., 2005; Yeoh et al., 2008; Matsui and Matsuo, 2012). First, we show a simple search method, which is basically a time division of DPOP. While this method employs messages named VALUE and UTIL shown in Figure 1(c), they are different from those of DPOP. Similar to DPOP, the method consists of two phases of computations.

In the first phase, the optimal objective vector is computed in a manner of tree search. The root agent i chooses an assignment $\mathcal{A}_{i,j}^{dcd}$ for variables in $X_i^{dcd} \cap X_j^{sep}$ for its child $j \in Ch_i$. Then the root agent sends the current assignment $\mathcal{A}_j^{sep} = \mathcal{A}_{i,j}^{dcd}$ to its child node j using a VALUE message. When non-root agent i receives $\mathcal{A}_i^{sep}$ from its parent p_i, agent i chooses an assignment $\mathcal{A}_{i,j}^{dcd}$ for variables in $X_i^{dcd} \cap X_j^{sep}$ for its child j. Agent i then sends $\mathcal{A}_j^{sep} \subseteq \mathcal{A}_i^{sep} \cup \mathcal{A}_{i,j}^{dcd}$ for variables in X_j^{sep} to its child j. Namely, an assignment is expanded for all children of a node in a pseudo tree, in the same time. The current assignment $\mathcal{A}_i^{sep}$ is called *current context*. In the root agent, the current context is always $\emptyset$.

For the current context $\mathcal{A}_i^{sep}$, each agent computes $g_i^*(\mathcal{A}_i^{sep})$. Then $g_i^*(\mathcal{A}_i^{sep})$ is sent to i's parent p_i using a UTIL message. When agent i receives $g_j^*(\mathcal{A}_j^{sep})$ from its child j, $g_i^*(\mathcal{A}_j^{sep})$ is stored in the agent, if $\mathcal{A}_j^{sep}$ is compatible with $\mathcal{A}_i^{sep}$. When the current context changes to new assignment ${\mathcal{A}^{sep}}'_i$, objective vector $g_j^*(\mathcal{A}_j^{sep})$ whose $\mathcal{A}_j^{sep}$ is incompatible with ${\mathcal{A}^{sep}}'_i$ is deleted.

While the computation of $g_i^*(\mathcal{A}_i^{sep})$ is based on Equations (4) and (5), the computation is generalized to the case where agent i has not received $g_j^*(\mathcal{A}_j^{sep})$ from child j. In such cases, the lower and upper limit values of unknown objective values are introduced. With the limit values, the objective values are separated into lower and upper bound values. For the leximin ordering, we define the upper and lower bounds of objective vectors.

Definition 7 (Boundaries of unknown vector): For an objective vector $\mathbf{v}$ of K unknown values, lower bound $\mathbf{v}^{\perp}$ and upper bound $\mathbf{v}^{\top}$ are vectors of K values, whose values are $-\infty$ and ∞, respectively.

These boundaries are obviously reasonable, since they are the minimum vector and the maximum vector on the leximin ordering. Operators $\oplus$ and $\prec_{leximin}$ are applied to the boundaries of vectors without any modifications. For a vector $\mathbf{v} = [v_0, \cdots, v_K]$ and the lower bound $\mathbf{v'}^{\perp} = [-\infty, \cdots, -\infty]$ of unknown vector $\mathbf{v'}$, the vector $\mathbf{v} \oplus \mathbf{v'}^{\perp}$ consists of $-\infty, \cdots, -\infty$ and $v_0, \cdots, v_K$. Similarly, $\mathbf{v} \oplus \mathbf{v'}^{\top}$ consists of $v_0, \cdots, v_K$ and $\infty, \cdots, \infty$. We consider these vectors as $(\mathbf{v} \oplus \mathbf{v'})^{\perp}$ and $(\mathbf{v} \oplus \mathbf{v'})^{\top}$, respectively.

Proposition 2 (Lower bound of partially unknown vector): Let $\mathbf{v}^{\perp}$ denote a vector whose values are $v_0, \cdots, v_K$ and K' values of $-\infty$. For any vector $\mathbf{v}$ whose values are $v_0, \cdots, v_K$ and K' values greater than $-\infty$, $\mathbf{v}^{\perp} \prec_{leximin} \mathbf{v}$.

Proof. While the first value in the sorted vector of $\mathbf{v}^{\perp}$ is $-\infty$, that of $\mathbf{v}$ is greater than $-\infty$. Therefore, $\mathbf{v}^{\perp} \prec_{leximin} \mathbf{v}$. □

Proposition 3 (Upper bound of partially unknown vector): Let $\mathbf{v}^{\top}$ denote a vector whose values are $v_0, \cdots, v_K$ and K' values of ∞. For any vector $\mathbf{v}$ whose values are $v_0, \cdots, v_K$ and K' values less than ∞, $\mathbf{v} \prec_{leximin} \mathbf{v}^{\top}$.

Proof. Consider a vector $\mathbf{v}^{\top[v'_0]}$ where one of values ∞ in $\mathbf{v}^{\top}$ is replaced by a value v'_0 less than ∞. Both sorted vectors of $\mathbf{v}^{\top[v'_0]}$ and $\mathbf{v}^{\top}$ contain the same sequence of k values less than v'_0, since v'_0 does not affect this sequence. When $\mathbf{v}^{\top}$ contains k' values of v'_0, $\mathbf{v}^{\top[v'_0]}$ contains $k'+1$ values of v'_0. We can conclude that the sequences of the first $k+k'$ values are the same in both sorted vectors of $\mathbf{v}^{\top[v'_0]}$ and $\mathbf{v}^{\top}$, while the next values are the value equal to v'_0 and a value greater than v'_0, respectively. Therefore, $\mathbf{v}^{\top[v'_0]} \prec_{leximin} \mathbf{v}^{\top}$. Consider a vector $\mathbf{v}^{\top[v'_0,v'_1]}$ where one of values ∞ in $\mathbf{v}^{\top[v'_0]}$ is replaced by a value v'_1 less than ∞. Similar to $\mathbf{v}^{\top[v'_0]} \prec_{leximin} \mathbf{v}^{\top}$, we can conclude $\mathbf{v}^{\top[v'_0,v'_1]} \prec_{leximin} \mathbf{v}^{\top[v'_0]}$. Based on the mathematical induction, we can conclude that $\mathbf{v} = \mathbf{v}^{\top[v'_0,\cdots v'_{K'-1}]} \prec_{leximin} \cdots \prec_{leximin} \mathbf{v}^{\top[v'_0]} \prec_{leximin} \mathbf{v}^{\top}$ for any combination $[v'_0, \cdots v'_{K'-1}]$ of values that replace the values of ∞ in $\mathbf{v}^{\top}$. □

In addition, with a bottom-up preprocessing, the lower and upper limit values for each function $f_i(X_i)$, (i.e. $\min f_i(X_i)$ and $\max f_i(X_i)$) can be aggregated to vectors of limit values instead of the vectors of $-\infty$ and ∞. Namely, each agent i learns a pair of limit of limit vectors $g_j^{\perp lmt}$ and $g_j^{\top lmt}$for each child j using a bottom-up preprocessing. $g_j^{\perp lmt}$ is consisting of values $\min f_k(X_k)$ for all agents k in the sub-tree rooted at i's child j. Similarly, $g_j^{\top lmt}$ consists of maximum values of objectives. $g_j^{\perp/\top lmt}$ are employed as the default lower and upper bound vectors. As a result of the bottom-up preprocessing, each agent i also knows the limit vectors $g_i^{\perp/\top lmt}$ for its own sub-tree.

$g_i^*(\mathcal{A}_i^{sep})$ is extended to a pair of $g_i^{*\perp}(\mathcal{A}_i^{sep})$ and $g_i^{*\top}(\mathcal{A}_i^{sep})$ that are simultaneously computed. To introduce the boundaries, an agent has to know the number of descendants of each sub-tree rooted at each child. The information of the descendants is additionally computed in the preprocessing. When the number of descendants for a child j is dcd_j, $g_j^{*\perp}(\mathcal{A}_j^{sep})$ for unknown $g_j^*(\mathcal{A}_j^{sep})$ is a vector of dcd_j values of $-\infty$. Similarly, $g_j^{*\top}(\mathcal{A}_j^{sep})$ is a vector of dcd_j values of ∞.

Based on the boundaries, agents complete the tree search for sub problems. When $g_j^{*\perp}(\mathcal{A}_j^{sep}) = g_j^{*\top}(\mathcal{A}_j^{sep})$ for child $j \in Ch_i$, agent i completes the tree search for the assignment $\mathcal{A}_j^{sep}$. Then i chooses another assignment $\mathcal{A}^{sep\prime}{}_j$ such that $g_j^{*\perp}(\mathcal{A}^{sep\prime}{}_j) \prec_{leximin} g_j^{*\top}(\mathcal{A}^{sep\prime}{}_j)$. While there are several search strategies on the assignments, we employ a depth-first search based on the pseudo tree.

Now, a UTIL message carries a pair of vectors for both boundaries. Since the boundaries are narrowed with the true objective values that are propagated in a bottom-up manner on the pseudo tree, agents repeatedly send UTIL messages. When agent i receives new vectors of $g^{*\perp}{}_j(\mathcal{A}_j^{sep})$ and $g^{*\top}{}_j(\mathcal{A}_j^{sep})$ from child $j \in Ch_i$, those vectors update the previous vectors. While $g^{*\perp}{}_j(\mathcal{A}_j^{sep})$ is maximized, $g^{*\top}{}_j(\mathcal{A}_j^{sep})$ is minimized with the new vectors based on the leximin ordering.

When $g_i^{*\perp}(\emptyset) = g_i^{*\top}(\emptyset)$ in the root agent i, agent i compute the optimal assignment $\mathcal{A}_i^{dcd*}$ such that $g_i^{\perp}(\emptyset \cup \mathcal{A}_i^{dcd*}) = g_i^{\top}(\emptyset \cup \mathcal{A}_i^{dcd*}) = g_i^{*\perp}(\emptyset) = g_i^{*\top}(\emptyset)$. $\mathcal{A}_j^{sep*} \subseteq \mathcal{A}_i^{dcd*}$ is then sent to each child $j \in Ch_i$ using a VALUE message with a flag of the termination. When $g_i^{*\perp}(\mathcal{A}_i^{sep*}) = g_i^{*\top}(\mathcal{A}_i^{sep*})$ in non-root agent i, the agent similarly computes the optimal assignment $\mathcal{A}_i^{dcd*}$ such that $g_i^{\perp}(\mathcal{A}_i^{sep*} \cup \mathcal{A}_i^{dcd*}) = g_i^{\top}(\mathcal{A}_i^{sep*} \cup \mathcal{A}_i^{dcd*}) = g_i^{*\perp}(\mathcal{A}_i^{sep*}) = g_i^{*\top}(\mathcal{A}_i^{sep*})$, and $\mathcal{A}_j^{sep*} \subseteq \mathcal{A}_i^{sep*} \cup \mathcal{A}_i^{dcd*}$ for each child $j \in Ch_i$ under $\mathcal{A}_i^{sep*}$. As a result, all the agents determine their optimal assignment.

4.4. Pruning

Next, we introduce the pruning based on the global lower bound of objective vectors. The global lower bound is $g_r^{*\perp}(\emptyset)$ in the root agent r. $g_r^{*\perp}(\emptyset)$ is propagated in a top-down manner using VALUE messages. An assignment $\mathcal{A}_j^{sep}$ for agent j is pruned if $g_r^{*\perp}(\emptyset) \not\prec_{leximin} g_j^{*\top}(\mathcal{A}_j^{sep})$. However, the length of $g_j^{*\top}(\mathcal{A}_j^{sep})$ is the number of agents in the sub-tree rooted at j, while the length of $g_r^{*\perp}(\emptyset)$ equals the number of all the agents $|A|$. In this case, $\prec_{leximin}$ is applied as follows. Since $g_j^{*\top}(\mathcal{A}_j^{sep})$ is an upper bound, unknown objective values are represented by ∞. Therefore, with padding of ∞, $g_j^{*\top}(\mathcal{A}_j^{sep})$ and $g_r^{*\perp}(\emptyset)$ can be compared as the same length of vectors. Let $g_j^{*\top\top}(\mathcal{A}_j^{sep})$ denote the vector $g_j^{*\top}(\mathcal{A}_j^{sep})$ with the padding of ∞. In actual computation, the padding can be omitted, since the sequence of ∞ is the last part of vectors. When $g_j^{*\perp}(\mathcal{A}_j^{sep}) = g_j^{*\top}(\mathcal{A}_j^{sep}) \vee g_r^{*\perp}(\emptyset) \not\prec_{leximin} g_j^{*\top\top}(\mathcal{A}_j^{sep})$ for child $j \in Ch_i$, agent i completes the tree search for the assignment $\mathcal{A}_j^{sep}$.

Moreover, to improve the effects of the pruning, we introduce an upper bound for other parts of the problem. Namely, for each child agent $j \in Ch_i$, agent i computes the upper bound of objective vector $h_j^{+\top}(\mathcal{A}_j^{sep})$ for sub-trees except one rooted at j.

$$h_j^{+\top}(\mathcal{A}_j^{sep}) = h_i^{+\top}(\mathcal{A}_i^{sep}) \oplus \max_{\mathcal{A}^{dcd\prime}{}_i \text{ for } X_i^{dcd} \setminus X_j^{sep}} h_i^{\top}(\mathcal{A}_i^{sep} \cup \mathcal{A}^{dcd\prime}{}_i \cup \mathcal{A}_j^{sep}) \tag{6}$$

$$h_i^{\top}(\mathcal{A}) = [f_i(\mathcal{A}_{|X_i})] \oplus \bigoplus_{k \in Ch_i \setminus \{j\}, \mathcal{A}_k^{sep} \subseteq \mathcal{A}} g_k^{*\top}(\mathcal{A}_k^{sep}) \tag{7}$$

Note that the maximization in Equation (6) is not the maximization of objective values but the selection of the widest boundary. Since $\mathcal{A}_j^{sep}$ is a part of an assignment for $X_i^{sep} \cup X_j^{dcd}$, there are several assignments compatible with $\mathcal{A}_j^{sep}$. For such compatible assignments, the widest boundary prevents an over estimation. $h_j^{+\top}(\mathcal{A}_j^{sep})$ is sent from agent i to its child j using VALUE messages. When $g_j^{*\perp}(\mathcal{A}_j^{sep}) = g_j^{*\top}(\mathcal{A}_j^{sep}) \vee g_r^{*\perp}(\emptyset) \not\prec_{leximin} h_j^{+\top}(\mathcal{A}_j^{sep}) \oplus g_j^{*\top}(\mathcal{A}_j^{sep})$ for child $j \in Ch_i$, agent i completes the tree search for the assignment $\mathcal{A}_j^{sep}$.

Similar to $g_j^{\perp/\top lmt}$ shown in Subsection 4.3, for child j and $h_j^{+\top}$, a default upper bound limit vector $h_j^{+\top lmt}$ can be computed. The aggregation is performed and propagated in a top-down manner using $g_j^{\perp/\top lmt}$ as follows.

$$h_j^{+\top lmt} = h_i^{+\top lmt} \oplus [\max f_i(X_i)] \oplus \bigoplus_{k \in Ch_i \setminus \{j\}} g_k^{\top lmt} \tag{8}$$

As a result, each agent i knows $h_i^{+\top lmt}$ except the root agent whose $h_i^{+\top lmt}$ and $h_i^{+\top}(\mathcal{A}_i^{sep})$ are empty vectors. Similarly, $h_i^{+\perp lmt}$ is computed for minimum objective values. With $h_i^{+\perp lmt}$, the default values of $g_r^{*\perp}(\emptyset)$ is represented as $h_i^{+\perp lmt} \oplus g_i^{+\perp lmt}$ instead of the vector of $-\infty$.

Actually, in this computation, we choose an inexact context $\mathcal{A}_i^{sep}$ for $h_i^{+\top}$. Since $h_i^{+\top}$ depends on other sub-trees rooted at i's siblings, the context should contain assignments for all ancestor agents. However, such a context reveals assignments for other sub-trees in the case where several ancestors are decision makers for other sub-trees. Therefore, we still employ partial assignment $\mathcal{A}_i^{sep}$. With the partial assignments, agent i may fail to reset $h_i^{+\top}$ to a default upper limit vector. There are two cases of incorrect pruning. 1) If the pruning does not work, the search processing continues. 2) If the pruning works, the search processing is blocked. However, $g_i^{*\perp/\top}$ is correct, while the current context is compatible. In addition,

$g_i^{*\perp/\top}$ will be reset if the current context is changed. Eventually, an agent correctly computes $h_i^{+\top}$ and propagates the correct $h_i^{+\top}$. Therefore, the search processing resumes. We need to keep the propagation of 'fresh' information including $h_i^{+\top}$.

4.5. Shortcut VALUE messages for modified pseudo tree

In several search methods (Modi et al., 2005; Yeoh et al., 2008), additional VALUE messages are sent from ancestor agents to descendant agents taking shortcut paths. The shortcut VALUE messages directly carry assignments to deep levels of the pseudo tree. Then the assignments are propagated in a bottom-up manner using extended UTIL messages to update contexts. Similarly, we introduce shortcut VALUE messages (VALUESC messages). In our solution methods, the shortcut messages are particularly important to reduce the delay in updating the contexts, since the decision makers of most variables are the agents in higher levels of the pseudo tree. In the conventional methods, the paths of shortcut VALUE messages are back edges. On the other hand, in our cases, back edges may not directly connect the decision maker and the deepest agent which relate to the same variable. In the example of Figure 1(c), with VALUESC messages, the root agent sends x_0, x_1 and x_2 to the agent of x_2, and sends x_1 to the agent of x_3, respectively. Note that the root agent and the agent of x_3 are not directly connected. Therefore, we compute the deepest related agent for each variable in a bottom-up preprocessing, which is integrated to the preprocessing. The information on the deepest agent is stored in the corresponding decision maker. Agent i knows a set Sc_i of agents, to which VALUESC messages are sent. For each agent $k \in Sc_i$, i computes $\mathcal{A}_k^{sc}$ containing assignments for k based on $\mathcal{A}_j^{sep}$, where child $j \in Ch_i$ is an ancestor of k. In addition, we employ timestamps based on the logical clock of the assignment for each variable, to compare the freshness of the assignment.

4.6. Pseudo code of search method

Figure 2 shows the pseudo code of the search method for agent i. Here ${}^i*$ denotes agent i's copy of $*$. Also, $*^{\perp/\top}$ denotes a pair of $*^{\perp}$ and $*^{\top}$. Here we represent limit vectors of infinity values. $\overrightarrow{-\infty}$ and $\overrightarrow{\infty}$ denote the vectors consisting of $-\infty$ and ∞, respectively. The length of these vectors is the same as the length of the vectors to be assigned. In addition, $null$ represents the undefined element. $p_i = null$ means that agent i is the root agent (line 2). For non-root agent i, assignment $\mathcal{A}_i^{sep}$ is initialized as $null$ to show that no assignment to the separator variables has been received (line 3). Agent i also maintains frags $ptrm_i$ and trm_i that take Boolean values. $ptrm_i$ shows that parent agent p_i has terminated its processing. At the root agent i, $ptrm_i$ is always true. trm_i inherits the value of $ptrm_i$ and represents the termination of agent i. After the initialization (lines 2-4), agents repeatedly receive messages and maintain their status (lines 5-8). Note that the message passing is initiated by the root agent when it first enters the Maintenance state (line 8). When an agent receives a message, the agent updates its status based on the type of messages (lines 10-22). Then the agent maintains other data structures ${}^i g_r^{*\perp}(\emptyset)$, $\mathcal{A}_i^{*dcd}$, $\mathcal{A}_j^{sep}$ for each child j, and $\mathcal{A}_k^{sc}$ for each lower agent in $k \in Sc_i$ (lines 25-29, 31 and 32). The root agent i updates the global lower bound ${}^i g_r^{*\perp}(\emptyset)$, when it evaluates new global lower bound $g_i^{*\perp}(\emptyset)$ greater than the current one (line 25). ${}^i g_r^{*\perp}(\emptyset)$ is propagated to all agents by VALUE messages (lines 30 and 12). If the termination condition is achieved, each agent i determines its optimal assignment $\mathcal{A}_i^{*dcd}$, and changes its frag trm_i to $true$ (lines 26 and 27). $\mathcal{A}_i^{*dcd}$ is employed to determine the optimal assignment to the separators for child agents (line 29). trm_i is sent to each child agent j, and stored as $ptrm_j$ (lines 30 and 12). Then, agent i determines assignment $\mathcal{A}_j^{sep}$ to the separators for each child agent j. When agent i is terminating, $\mathcal{A}_j^{sep}$ is determined

1 Main{
2 **if**($p_i = null$){ $\mathcal{A}_i^{sep} \leftarrow \emptyset$. $h_i^{+\top}(\mathcal{A}_i^{sep}) \leftarrow [\,]$. $ptrm_i \leftarrow true$. }
3 **else**{ $\mathcal{A}_i^{sep} \leftarrow null$. $ptrm_i \leftarrow false$. }
4 ${}^i g_r^{*\perp}(\emptyset) \leftarrow \overrightarrow{-\infty}$. $trm_i \leftarrow false$.
5 **while**(forever){
6 **until**(receive loop exits){
7 **if**($\neg trm_i$){ receive a message. }**else**{ purge all messages. } }
8 **if**($\mathcal{A}_i^{sep} \neq null \wedge \neg trm_i$){ Maintenance. } }

10 Receive(VALUE, $\mathcal{A}$, g, h, trm){
11 update $\mathcal{A}_i^{sep}$ by $\mathcal{A}$. **if**($\mathcal{A}_i^{sep}$=$\mathcal{A}$){ $h_i^{+\top}(\mathcal{A}_i^{sep}) \leftarrow h$.}**else**{ $h_i^{+\top}(\mathcal{A}_i^{sep}) \leftarrow \overrightarrow{\infty}$. }
12 ${}^i g_r^{*\perp}(\emptyset) \leftarrow g$. $ptrm_i \leftarrow trm$. Consistent. **return**. }

14 Receive(VALUESC, $\mathcal{A}$){
15 **if**($\mathcal{A}_i^{sep} \neq null$){
16 update $\mathcal{A}_i^{sep}$ by $\mathcal{A}$. **if**($\mathcal{A}_i^{sep}$ is updated){ $h_i^{+\top}(\mathcal{A}_i^{sep}) \leftarrow \overrightarrow{\infty}$. }
17 Consistent. } **return**. }

19 Receive(UTIL, j, $\mathcal{A}$, $g^{\perp/\top}$){
20 update $\mathcal{A}_i^{sep}$ by $\mathcal{A}$. **if**{$\mathcal{A}_i^{sep}$ is updated){$h_i^{+\top}(\mathcal{A}_i^{sep}) \leftarrow \overrightarrow{\infty}$. }
21 **if**($\mathcal{A}_i^{sep}$ is compatible with $\mathcal{A}$){store/update ${}^i g_j^{*\perp/\top}(\mathcal{A})$ by $g^{\perp/\top}$. }
22 Consistent. **return**. }

24 Maintenance{
25 **if**($p_i = null \wedge {}^i g_r^{*\perp}(\emptyset) \prec_{leximin} g_i^{*\perp}(\emptyset)$){${}^i g_r^{*\perp}(\emptyset) \leftarrow g_i^{*\perp}(\emptyset)$.}
26 **if**($ptrm_i \wedge g_i^{*\perp}(\mathcal{A}_i^{sep}) = g_i^{*\top}(\mathcal{A}_i^{sep})$){
27 determine $\mathcal{A}_i^{*dcd}$ corresponding to the termination condition. $trm_i \leftarrow true$. }
28 **foreach**($j \in Ch_i$){
29 **if**(trm_i){ determine $\mathcal{A}_j^{sep}$ from $\mathcal{A}_i^{*dcd}$ and $\mathcal{A}_i^{*sep}$. }**else**{ choose $\mathcal{A}_j^{sep}$ with a strategy. }
30 send (VALUE, $\mathcal{A}_j^{sep}$, ${}^i g_r^{*\perp}(\emptyset)$, $h_j^{+\top}(\mathcal{A}_j^{sep})$, trm_i) to j. }
31 **foreach**($k \in Sc_i$){
32 determine $\mathcal{A}_k^{sc}$ from $\mathcal{A}_j^{sep}$ of k's ancestor j. send (VALUESC, $\mathcal{A}_k^{sc}$) to k. }
33 **if**($\neg ptrm_i$){ send (UTIL i, $\mathcal{A}_i^{sep}$, $g_i^{*\perp/\top}(\mathcal{A}_i^{sep})$) to p_i. }
34 **return**. }

36 Consistent{
37 **foreach**($\mathcal{A}$ incompatible with $\mathcal{A}_i^{sep}$){delete ${}^i g_j^{*\perp/\top}(\mathcal{A})$.}
38 **return**. }

FIGURE 2. Distributed search for leximin AMODCOP (agent i)

from the optimal assignment $\mathcal{A}_i^{*dcd}$ and $\mathcal{A}_i^{*sep}$ (the last $\mathcal{A}_i^{sep}$ from p_i). Otherwise, $\mathcal{A}_j^{sep}$ is determined with a strategy (line 29). Each $\mathcal{A}_j^{sep}$ is sent to child agent j with corresponding bounds of objective vectors $h_j^{+\top}(\mathcal{A}_j^{sep})$ using a VALUE message (line 30). The child agent j updates its $\mathcal{A}_j^{sep}$ based on the assignment from its parent p_j, while it also updates $h_j^{+\top}(\mathcal{A}_j^{sep})$ if $\mathcal{A}_j^{sep}$ is compatible with the one from p_j (line 11). Note that a part of $\mathcal{A}_j^{sep}$ may be updated to a newer (and prior) one by a VALUESC message. In such a case, $\mathcal{A}_j^{sep}$ may be partially updated by the assignment from parent p_j. As a result, $\mathcal{A}_j^{sep}$ can be temporally incompatible with (delaying) assignment from parent p_j, and $h_j^{+\top}(\mathcal{A}_j^{sep})$ is reset to the default bounds (line 11). In addition to assignments that are sent using VALUE messages, agent i determines an assignment for a shortcut message to each lower agent $k \in Sc_i$. Then the assignments are sent using VALUESC messages (lines 31 and 32). Finally, agent i sends bounds of objective vectors $g_i^{*\perp/\top}(\mathcal{A}_i^{sep})$ for the current $\mathcal{A}_i^{sep}$ to its parent agent using a UTIL message (line 33).

When agent i receives messages, it updates assignment $\mathcal{A}_i^{sep}$ (lines 11, 16 and 20). Then, it resets the information of bounds of objective vectors ${}^i g_j^{*\perp/\top}(\mathcal{A})$, if $\mathcal{A}$ is incompatible with $\mathcal{A}_i^{sep}$ (lines 12, 17, 22 and 36-38). Similarly, $h_i^{+\top}(\mathcal{A}_i^{sep})$ is reset, if $\mathcal{A}_i^{sep}$ is updated (lines 11, 16 and 20).

4.7. Representation of objective vectors

In the whole computation of objective vectors, sorted vector can be employed. With the sorted vectors, the objective values of individual agents are not directly identified. The length of the objective vectors is upper bounded by the number of agents $|A|$. On the other hand, the sorted vector is compressed with run-length encoding, as a sequence of pairs $(objective\ value,\ length)$. This reduces both the size of the representation and the computation of $\prec_{leximin}$, when there are a number of the same objective values.

4.8. Correctness and Complexity

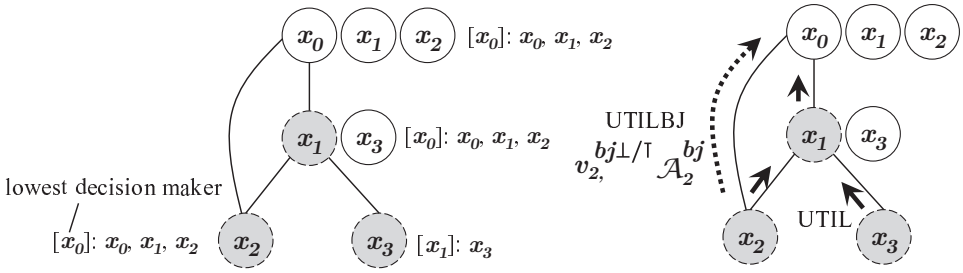
Both, the extended DPOP and the search method, are variations of previous solution methods (Modi et al., 2005; Petcu and Faltings, 2005), while we use a representation without any subtraction. Therefore, their correctness is proven with the same reasoning as for the previous methods, replacing the assignment concept with the proposed vectors (since we proved above that it satisfies the same additive properties). We have addressed how the computation is extended to Leximin AMODCOPs. Propositions 1, 2 and 3 show that the monotonicity in the computation resembles the conventional solution methods based on addition. The properties on the computational/communication complexity of the proposed methods are also the same as those of the previous methods. On the other hand, the modified pseudo tree implicitly increases the induced width (Petcu and Faltings, 2005), which is $\prod_{x_i \in X_i^{sep}} |D_i|$ for agent i. The worst case of the basic tree search is as follows. 1) The tree is a single sequence of agents. 2) The decision maker is only the root node. 3) The evaluation is only made in the single leaf node. 4) No pruning works. Therefore, the maximum number of message cycles is $2(|A| - 1) \prod_{x_j \in X} |D_j|$. However, this is an inherent property of the AMODCOPs. One can address large size problems using approximation methods. The maximum length of objective vector is the same as the number of agents $|A|$. With the representation using pairs of a value and its length, the size of the representation is between 2 and $2|A|$. This representation can be implemented with several tree structures, including Red-Black trees whose major operations are performed in $O(\log n)$ time. The size of messages increases, since their scalar values are replaced by vectors.

5. INDIVIDUAL PREFERENCES

In the definition of objective vectors, objective values are sorted regardless of the individuality of agents. While such objective vectors potentially can help to improve privacy of agents, they also restrict effective pruning methods in search algorithms. When solution methods distinguish the preferences of individual agents, additional pruning methods are available. In this section, we apply a partial inference method based on the individuality of objectives.

5.1. Basic idea

In the tree search algorithm, we introduced top-down shortcut VALUE messages to reduce the delay of search processing. As another approach, a type of bottom-up shortcut



(a) lowest decision maker of current context (b) UTILBJ message

FIGURE 3. Short cut message in bottom-up aggregation

messages are employed to reduce backtracking and related search processing (Matsui et al., 2005). This kind of message propagation relates to back jumping on (Distributed) Constraint Satisfaction Problems. In such methods, each agent knows/detects its ancestor agent that effectively improve the current partial assignment. However, this approach needs several extensions for asymmetric DCOPs and leximin optimization problems due to the following reasons.

i) Since an objective vector inferred by an agent i is a part of another vector in i's ancestor agent, the shortcut needs identification of conflicting objective values to integrate both vectors.

ii) The asymmetricity restricts the evaluation in the agents. In the asymmetric problems, each agent can locally evaluate assignments for its variable and its neighborhood variables including several lower neighborhood variables. On the other hand, for an agent i other than the root, i's parent/ancestor agents are the ones who decide assignments for i and i's lower neighborhood agents. Therefore, the destination of a shortcut message is determined considering the decision makers. In addition, the inference of partial objective vector for a shortcut is complex. Therefore, it is reasonable to employ the shortcut messages for single objectives.

iii) We employ a pair of lower and upper bounds of a partial sorted objective vector to prune the tree search. If one aggregates a bound of partial sorted vector inferred by agent i with another bound of partial sorted vector in i's ancestor agent h, ignoring the aggregations in intermediate agents between h and i, the resulting bound of partial sorted vector does not assure the monotonic increase/decrease of the lower/upper bound. Therefore, it needs a safe lower/upper bound that assure the monotonicity. As shown below, it also supports the inferences used to restrict the shortcut messages for single objectives.

Considering the above reasons, we employ the following techniques.

5.2. Individuality on Objectives

To employ the individuality of agents, we relate identifiers of agents to objective values.

Definition 8 (Indexed Sorted Vector): An indexed sorted vector is a sorted vector which allows for identifying each objective with a set of indices.

There are opportunities to employ run-length encoding for a part of a vector by modifying an indexed sorted vector, since the identifiers are necessary only for the objectives related to the shortcut. For simplicity, we used a implementation that has sorted objective values and another array of non-sorted objective values In the following, we focus on the effect of the identification of objectives employing indexed sorted vectors.

5.3. Additional relationship among agents

For the shortcut messages, each agent has to determine a destination ancestor agent. Here we consider the shortcut for single objectives. Since each objective directly corresponds to an agent, a single objective relates with a variable of an agent and its neighborhood variables. Namely, a single objective depends on the assignments for its local problem. Therefore, the destination agent of the shortcut is determined as the lowest decision maker for variables in the local problem.

As shown in Subsection 4.1, each agent i knows a set X_i^{dcd} of variables whose values are assigned by agent i. After that, an additional top-down preprocessing is performed. Here, for each agent i, a list of ancestor agents $Ancst_i$, including a parent, and $XX_i^{dcd} = \{(x_h, X_h^{dcd}) | x_h \in Ancst_i\}$ is propagated. Each agent i aggregates decision makers x_d in Dcd_i^{nbrs} for each $x_k \in \{i\} \cup Nbrs_i$ as follows.

$$Dcd_i^{nbrs} = \{(x_k, x_d) | x_k \in \{i\} \cup Nbrs_i, (x_d, X_d^{dcd}) \in XX_i^{dcd}, x_k \in X_d^{dcd}\} \tag{9}$$

Each agent i also aggregates depth $depth(x_d)$ of each decision maker x_d in Dcd_i^{nbrs} in a pseudo tree. For agent i, the destination of the shortcut message is the agent of x_i^{dcd} such that

$$x_i^{dcd} = \text{argmax}_{x_d \text{in } Dcd_i^{nbrs}} depth(x_d). \tag{10}$$

If such a decision maker is the parent agent (the root agent if the sender agent is also the root agent), there is no opportunity for the shortcut. Figure 3(a) shows an example of the lowest decision maker in each agent. While the lowest decision maker in the agent of x_2 is its ancestor x_0, other agents' decision makers are their parents or the root agent. Therefore, only the agent of x_2 sends shortcut messages on backtracking.

5.4. Shortcut on Backtracking

In addition to UTIL messages, each agent sends shortcut messages (UTILBJ messages) to one of its ancestor agents. Since we consider the shortcut for single objectives, a UTILBJ message transfers a pair of lower and upper bounds for a single objective value of the sender agent. Each agent i computes its objective $v_i^{bj\perp}$ and $v_i^{bj\top}$ for the UTILBJ message as follows.

$$v_i^{bj\perp} = \min_{\mathcal{A}_i^{dcd}} f_i((\mathcal{A}_i^{sep} \cup \mathcal{A}_i^{dcd})_{|X_i}) \tag{11}$$

$$v_i^{bj\top} = \max_{\mathcal{A}_i^{dcd}} f_i((\mathcal{A}_i^{sep} \cup \mathcal{A}_i^{dcd})_{|X_i}) \tag{12}$$

$v_i^{bj\perp}$ is the minimum value of objective i for the current context, since $v_i^{bj\perp}$ is an lower bound value. Similarly, upper bound value $v_i^{bj\top}$ is the maximum value of objective i for the current context.

In addition, a partial assignment $\mathcal{A}_i^{bj}$ that supports v_i^{bj} is computed.

$$\mathcal{A}_i^{bj} = (\mathcal{A}_i^{sep})_{|X_i \setminus X_i^{dcd}} \tag{13}$$

$v_i^{bj\perp/\top}$ depends on assignment $\mathcal{A}_i^{bj}$ that is a part of the current context. Note that the agent of x_i^{dcd} shown in Expression (10) should determine one of assignments in $\mathcal{A}_i^{sc}$. If the lowest decision maker (agent of x_i^{dcd}) is i's ancestor agent, agent i sends $v_i^{bj\perp/\top}$ and corresponding $\mathcal{A}_i^{bj}$ as a UTILBJ message to the agent of x_i^{dcd}. In the example of Figure 3(b) the agent of x_2 sends a UTILBJ message to x_0.

When an agent i receives a UTILBJ message from a descendant agent l, agent i stores the copy $({}^i v_l^{bj\perp/\top}, {}^i\mathcal{A}_l^{bj})$ of the received information for each l. Since ${}^i v_l^{bj\perp/\top}$ depends on

assignment ${}^i\mathcal{A}_l^{bj}$, the assignment ${}^i\mathcal{A}_l^{bj}$ must be compatible with the current context. When ${}^i\mathcal{A}_l^{bj}$ is incompatible with the current context, the pair $({}^iv_l^{bj\perp/\top}, {}^i\mathcal{A}_l^{bj})$ is eliminated. When a new compatible pair $(v_l^{bj\perp/\top}, \mathcal{A}_l^{bj})$ is received, ${}^iv_l^{bj\perp/\top}$ is updated by a maximization/minimization to narrow a partial lower/upper bound.

5.5. Integration of Shortcut Utility

Now each agent i has a pair $({}^iv_l^{bj\perp/\top}, {}^i\mathcal{A}_l^{bj})$ for descendant agent l. ${}^iv_l^{bj\perp/\top}$ is a correct lower/upper bound value for the objective of agent l. Let us concentrate on the case of upper bound. For upper bound ${}^iv_l^{bj\top}$, there is an upper bound vector ${}^ig_j^{*\top}(\mathcal{A})$ that has been received from i's child j and objective l is an element of ${}^ig_j^{*\top}(\mathcal{A})$. Note that $\mathcal{A}$ and ${}^i\mathcal{A}_l^{bj}$ must be compatible.

However, in general cases, ${}^iv_l^{bj\top}$ cannot be directly integrated with ${}^ig_j^{*\top}(\mathcal{A})$. One can incorrectly think that objective value v_l in ${}^ig_j^{*\top}(\mathcal{A})$ is narrowed by $\min({}^iv_l^{bj\top}, v_l)$. Such an integration is incorrect, since it ignores the aggregation and maximization of sorted vectors between agents i and l. It also reveals the fact that the integration of "shortcut" partial sorted vectors is more complicated than the case of single objectives. Therefore, we choose the shortcut for single objectives as shown above.

To integrate ${}^iv_l^{bj\top}$ into ${}^ig_j^{*\top}(\mathcal{A})$, an additional step is necessary. In Subsection 4.3, we introduced default lower/upper limit vectors $g_j^{\perp/\top lmt}$ for i's child agent j. The integration is defined using the default upper limit vector $g_j^{\top lmt}$.

Definition 9 (Upper Bound Vector with Shortcut): Let ${}^iv_l^{bj\top}$, ${}^ig_j^{*\top}(\mathcal{A})$ and $g_j^{\top lmt}$ denote an upper bound objective value received with a shortcut message, an upper bound objective vector containing a value v_l for objective l, and the default upper limit objective vector for ${}^ig_j^{*\top}(\mathcal{A})$, respectively. The integration of ${}^iv_l^{bj\top}$ and ${}^ig_j^{*\top}(\mathcal{A})$ is represented as follows

(1) $\mathbf{v}^\top \leftarrow g_j^{\top lmt}$. Update v_l in $\mathbf{v}^\top$ by ${}^iv_l^{bj\top}$ (Note that ${}^iv_l^{bj\top} = \min(v_l, {}^iv_l^{bj\top})$).
(2) $\mathbf{v}^{\top\top} \leftarrow \min_{leximin}(\mathbf{v}^\top, {}^ig_j^{*\top}(\mathcal{A}))$.

$\mathbf{v}^{\top\top}$ is a upper bound vector with shortcut.

Proposition 4 (Monotonicity of Upper Bound Vector with Shortcut): Upper bound vectors with shortcut do not exceed the correct upper bound vectors.

Proof. Let denote $g_j^{**\top}(\mathcal{A})$ a new vector of ${}^ig_j^{*\top}(\mathcal{A})$ that will be received in future. Also, let v_l denote the value of objective l in $g_j^{**\top}(\mathcal{A})$. Since $g_j^{**\top}(\mathcal{A})$ is a new vector, $g_j^{**\top}(\mathcal{A}) \preceq_{leximin} {}^ig_j^{*\top}(\mathcal{A})$. In particular, consider the case that an objective l has been evaluated in the aggregation of $g_j^{**\top}(\mathcal{A})$. Namely, the value v_l of objective l in $g_j^{**\top}(\mathcal{A})$ is not a default limit value. If an upper bound value ${}^iv_l^{bj\top}$ has been received, v_l is equal to or less than ${}^iv_l^{bj\top}$. Note that ${}^iv_l^{bj\top}$ is the maximum value of objective l that has been maximized ignoring the aggregation of sorted vectors.

Consider $\mathbf{v}^\top$ in Definition 9. For each objective $m \neq l$, $v_m^\top$ in $\mathbf{v}^\top$ is the upper limit value of objective m. Therefore, for v_m in $g_j^{**\top}(\mathcal{A})$, $v_m \leqslant v_m^\top$. In addition, for v_l in $g_j^{**\top}(\mathcal{A})$, $v_l \leqslant {}^iv_l^{bj\top} = v_l^\top$. Namely, all pairs of corresponding objectives are bounded. When the values in $g_j^{**\top}(\mathcal{A})$ are sorted in ascending order from position $t = 0$, for each value v_l of

```
1  Main{
2    if(p_i = null){ A_i^sep ← ∅. h_i^{+⊤}(A_i^sep) ← [ ]. ptrm_i ← true. }
3    else{ A_i^sep ← null. ptrm_i ← false. }
4    {}^i g_r^{*⊥}(∅) ← h_i^{+⊥lmt} ⊕ g_i^{⊥lmt}. trm_i ← false.
5    while(forever){
6      until(receive loop exits){
7        if(¬trm_i){ receive a message. }else{ purge all messages. } }
8      if(A_i^sep ≠ null ∧ ¬trm_i){ Maintenance.} }

10 Receive(VALUE, A, g, h, trm){
11   update A_i^sep by A.
12   if(A_i^sep=A){ h_i^{+⊤}(A_i^sep) ← h. }else{ h_i^{+⊤}(A_i^sep) ← h_i^{+⊤lmt}. }
13   {}^i g_r^{*⊥}(∅) ← g. ptrm_i ← trm. Consistent. return. }

15 Receive(VALUESC, A){
16   if(A_i^sep ≠ null){
17     update A_i^sep by A. if(A_i^sep is updated){ h_i^{+⊤}(A_i^sep) ← h_i^{+⊤lmt}. }
18     Consistent. } return. }

20 Receive(UTIL, j, A, g^{⊥/⊤}){
21   update A_i^sep by A. if(A_i^sep is updated){h_i^{+⊤}(A_i^sep) ← h_i^{+⊤lmt}. }
22   if(A_i^sep is compatible with A){ store/update {}^i g_j^{*⊥/⊤}(A) by g^{⊥/⊤}. }
23   Consistent. return. }

25 Receive(UTILBJ, j, A, v^{⊥/⊤}){
26   update A_i^sep by A. if(A_i^sep is updated){ h_i^{+⊤}(A_i^sep) ← h_i^{+⊤lmt}. }
27   if(A_i^sep is compatible with A){ store/update {}^i A_j^bj and {}^i v_j^{bj⊥/⊤} by A and v^{⊥/⊤}. }
28   Consistent. return. }

30 Maintenance{
31   if(p_i = null ∧ {}^i g_r^{*⊥}(∅) ≺_{leximin} g_i^{*⊥}(∅)){ {}^i g_r^{*⊥}(∅) ← g_i^{*⊥}(∅). }
32   if(ptrm_i ∧ g_i^{*⊥}(A_i^sep) = g_i^{*⊤}(A_i^sep)){
33     determine A_i^{*dcd} corresponding to the termination condition. trm_i ← true. }
34   foreach(j ∈ Ch_i){
35     if(trm_i){ determine A_j^sep from A_i^{*dcd} and A_i^{*sep}. }else{ choose A_j^sep with a strategy. }
36     send (VALUE, A_j^sep, {}^i g_r^{*⊥}(∅), h_j^{+⊤}(A_j^sep), trm_i) to j. }
37   foreach(k ∈ Sc_i){
38     determine A_k^sc from A_j^sep of k's ancestor j. send (VALUESC, A_k^sc) to k. }
39   if(¬ptrm_i){
40     send (UTIL, i, A_i^sep, g_i^{*⊥/⊤}(A_i^sep)) to p_i.
41     send (UTILBJ, i, A_i^bj, v_i^{bj⊥/⊤}) to the agent of x_i^dcd. }
42   return. }

44 Consistent{
45   foreach(A incompatible with A_i^sep){ delete {}^i g_j^{*⊥/⊤}(A). }
46   foreach({}^i A_j^bj incompatible with A_i^sep){ delete {}^i A_j^bj and {}^i v_j^{bj⊥/⊤}. }
47   return. }
```

FIGURE 4. Distributed search for leximin AMODCOP with UTILBJ (agent i)

objective l in $g_j^{**\top}(\mathcal{A})$ at position t, the corresponding value $v_l^\top$ in $\mathbf{v}^\top$ cannot be $v_l^\top < v_l$ at any position t' such that $t' \leqslant t$. Therefore, $g_j^{**\top}(\mathcal{A}) \preceq_{leximin} \mathbf{v}^\top$.

From $g_j^{**\top}(\mathcal{A}) \preceq_{leximin} {}^i g_j^{*\top}(\mathcal{A})$, $g_j^{**\top}(\mathcal{A}) \preceq_{leximin} \mathbf{v}^\top$ and step (2) in Definition 9, it is concluded that $g_j^{**\top}(\mathcal{A}) \preceq_{leximin} \mathbf{v}^{\top\top} = \min_{leximin}(\mathbf{v}^\top, {}^i g_j^{*\top}(\mathcal{A}))$. □

Similarly, An integration is correctly defined for lower bound ${}^i v_l^{bj\perp}$ with a maximization. In the case that a decision maker receives multiple shortcut messages from different descendants, the above integration is naturally extended to the case of multiple objectives. Since ${}^i v_l^{bj\perp/\top}$ is the minimum/maximum value of each objective l under the current context, it is independent from the minimum/maxim values of other objectives. Therefore, the integration of Definition 9 is correctly applied for each ${}^i v_l^{bj\top}$ (and ${}^i v_l^{bj\perp}$) at the same time.

As shown above, for a correct upper bound, only the single objective in a default upper limit vector is replaced. However, it may prune the search when an upper bound vector has been reset into the default upper limit vector due to a change of the current context. Since leximin compares the first different elements of sorted vectors, it generates opportunities for pruning.

5.6. Pseudo code of search method with shortcut on backtracking

Figure 4 shows an extended version of the search method with shortcut messages on backtracking. The pseudo code is based on the algorithm shown in Figure 2. Here we represent limit vectors $h_i^{+\perp lmt} \oplus g_i^{\perp lmt}$ and $h_i^{+\top lmt}$ instead of $\overrightarrow{-\infty}$ and $\overrightarrow{\infty}$. Each agent sends additional messages UTILBJ (line 41). UTILBJ messages are handled similar to UTIL messages (lines 25-28). Since assignments in $\mathcal{A}$ of a UTILBJ message are also employed to update the current context (line 26), $\mathcal{A}$ actually has a corresponding set of logical time stamps, similar to assignments of UTIL messages. ${}^i\mathcal{A}_j^{bj}$ and ${}^i v_j^{bj}$ are stored while ${}^i\mathcal{A}_j^{bj}$ is compatible with current context $\mathcal{A}_i^{sep}$ (lines 27 and 46).

In the algorithm, indexed sorted vectors are employed instead of sorted vectors. Moreover, as shown in in Definition 9, each ${}^i g_j^{*\top}$ is replaced by $v^{\top\top}$ in the comparison computation if corresponding ${}^i v_j^{bj}$ exists. Similarly, ${}^i g_j^{*\perp}$ is replaced. Therefore, the overhead of the algorithm is larger than that of the base algorithm. Our main interest in this method is a technical challenge to generalize shortcut utility messages for the operations on sorted vectors and leximin.

6. EVALUATION

The proposed method was experimentally evaluated. In our experiments with Leximin AMODCOPs (see Subsection 3.1) each problem consists of n ternary variables (i.e. $|D_i| = 3$) and c pairs of asymmetric objective functions. Here we employed $(n, c) \in \{(10, 9), (10, 12), (10, 15), (20, 19), (20, 22), (40, 39)\}$. The constraint network is randomly generated by first creating a spanning tree and then adding additional edges[1].

For each assignment, the objective function $f_{i,j}(x_i, x_j)$ returns an integer value w from $[0, 1]$ or $[0, 10]$ based on a uniform distribution. Note that we treat the aggregated function $f_i(X_i)$ as a black-box which cannot be decomposed. For each type of problem, the results are averaged over 50 instances. Since the difficulty of the problems cannot be exactly controlled, we show the average performance on these instances.

Tables 1 and 2 shows the comparison between leximin and other optimization criteria. The following optimization criteria are compared.

- max-leximin: leximin

[1]The spanning tree assures that the constraint network is connected. For the selection of edges, we randomly weighted all pairs of variables (i.e. edges of a complete graph) with uniform distribution. Based on the weight values, a minimum spanning tree is generated. Similarly, additional edges are selected with ascending order on the weight values.

TABLE 1. Comparison between max-leximin and other optimization criteria ($w = [0, 1]$)

$\prec$: the number of cases where (result of max-leximin)$\prec$(result of max-sum/min/LWT) on sum/min/max/variance/leximin/Pareto dominance.

comparison	sum									min								
optimization	max-sum			max-min			max-LWT			max-sum			max-min			max-LWT		
n, c	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$
10, 15	25	25	**0**	0	5	**45**	19	31	**0**	0	35	**15**	0	50	**0**	0	50	**0**
20, 22	33	17	**0**	0	0	**50**	29	21	**0**	0	31	**19**	0	50	**0**	0	50	**0**
40, 39	33	17	**0**	0	0	**50**	16	34	**0**	0	9	**41**	0	50	**0**	0	50	**0**
comparison	max									variance								
optimization	max-sum			max-min			max-LWT			max-sum			max-min			max-LWT		
n, c	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$
10, 15	25	25	0	3	24	23	18	32	0	**37**	13	0	**13**	6	31	**26**	24	0
20, 22	25	25	0	3	20	27	23	27	0	**44**	6	0	**9**	0	41	**43**	7	0
40, 39	20	30	0	7	14	29	14	36	0	**50**	0	0	**1**	0	49	**34**	16	0
comparison	leximin									Pareto								
optimization	max-sum			max-min			max-LWT			max-sum			max-min			max-LWT		
n, c	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$
10, 15	0	13	**37**	0	4	**46**	0	24	**26**	0	50	**0**	0	23	**27**	0	50	**0**
20, 22	0	6	**44**	0	0	**50**	0	7	**43**	0	50	**0**	0	19	**31**	0	50	**0**
40, 39	0	0	**50**	0	0	**50**	0	16	**34**	0	50	**0**	0	26	**24**	0	50	**0**

- max-sum: summation
- max-min: maximin
- max-LWT: maximin with additional summation

The results are computed using a dynamic programming based solution method shown in Subsection 4.2. Other criteria were also applied to the solvers based on pseudo trees, similar to the previous solvers (Matsui and Matsuo, 2012).

Since these criteria can be decomposed and aggregated using a dynamic programming manner, we employed variants of DPOP whose objective values and aggregation operators are replaced for the criteria.

Each cell of Tables 1 and 2 shows the number of cases of dominance ($\prec$ or $\succ$) or tie ($=$). Here, max-leximin is in the left hand side of the comparison symbols. For each comparison, we emphasized the preferred side of three columns, except the results of the maximum values 'max' that are shown as a reference. Namely, in the criteria except variance, *greater* values are preferred. On the summation of objective values, max-sum and max-LWT are never dominated by max-leximin. The results are reasonable, since those methods maximize the criteria including summation values. Max-leximin, max-min and max-LWT give the same minimum objective value. The results show that these criteria are subsets of maximin. For max-sum and max-LWT, max-leximin relatively decreases the variance of objective values. Max-min is not Pareto optimal, while the other criteria are Pareto optimal.

We also evaluated search methods on the modified pseudo trees and the leximin ordering. The following solution methods were evaluated.

- `b`: the basic search method shown in Subsection 4.3.
- `gl`: `b` with the pruning based on the global lower bound shown in Subsection 4.4.

TABLE 2. Comparison between max-leximin and other optimization criteria ($w = [0, 10]$)

$\prec$: the number of cases where (result of max-leximin)$\prec$(result of max-sum/min/LWT) on sum/min/max/variance/leximin/Pareto dominance.

comparison	sum									min								
optimization	max-sum			max-min			max-LWT			max-sum			max-min			max-LWT		
n, c	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$
10, 15	48	2	**0**	5	6	**39**	38	12	**0**	0	13	**37**	0	50	**0**	0	50	**0**
20, 22	50	0	**0**	5	0	**45**	48	2	**0**	0	4	**46**	0	50	**0**	0	50	**0**
40, 39	50	0	**0**	1	0	**49**	49	1	**0**	0	0	**50**	0	50	**0**	0	50	**0**

comparison	max									variance								
optimization	max-sum			max-min			max-LWT			max-sum			max-min			max-LWT		
n, c	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$
10, 15	42	5	3	25	8	17	33	15	2	**48**	2	0	**31**	4	15	**39**	9	2
20, 22	45	4	1	23	6	21	39	8	3	**50**	0	0	**27**	0	23	**49**	1	0
40, 39	47	1	2	19	7	24	36	9	5	**50**	0	0	**21**	0	29	**49**	0	1

comparison	leximin									Pareto								
optimization	max-sum			max-min			max-LWT			max-sum			max-min			max-LWT		
n, c	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$	$\prec$	=	$\succ$
10, 15	0	2	**48**	0	4	**46**	0	9	**41**	0	50	**0**	0	46	**4**	0	50	**0**
20, 22	0	0	**50**	0	0	**50**	0	1	**49**	0	50	**0**	0	47	**3**	0	50	**0**
40, 39	0	0	**50**	0	0	**50**	0	0	**50**	0	50	**0**	0	49	**1**	0	50	**0**

- `glou`: `bl` with the upper bound for other part of the problem shown in Subsection 4.4.
- `glousv`: `glou` with shortcut VALUE (VALUESC) messages shown in Subsection 4.5.
- `lvb`, `lvgl`, `lvglou`, `lvglousv`: solution methods with the vectors of lower and upper limit values for each function $f_i(X_i)$ addressed in Subsection 4.3 [2].
- `lvglousvbj`: `lvglousv` with shortcut UTIL (UTILBJ) messages shown in Section 5.

We employed a depth first strategy for each variables. On the other hand, a value of each variable is optimistically selected so that the value corresponding to largest upper bound vector is chosen.

The experiments were performed using simulation programs based on message cycles. Here we focused on the main processing of proposed methods, since it iteratively performs relatively complex message passing, and mainly relates the consumption of computational resources. The preprocessing is emulated using an equivalent sequential processing that performs depth first search traversal on constraint network and propagation on a pseudo tree to determine decision makers and default bounds of objective vectors. The simulation program repeats message cycles, where all agents are executed in a round robin manner. In the first message cycle, agents initialize their status. In a message cycle, each agent receives messages from its message queue. Then the agent updates its status and sends messages if necessary. The messages are exchanged by the simulator. A simulation is interrupted

[2]In this case, to avoid over estimation, we modified the condition of the pruning in the second phase using a flag. Agent i completes the tree search for the assignment $\mathcal{A}_j^{sep}$ when $g_j^{*\perp}(\mathcal{A}_j^{sep}) = g_j^{*\top}(\mathcal{A}_j^{sep}) \vee h_j^{+\top}(\mathcal{A}_j^{sep}) \oplus g_j^{*\top}(\mathcal{A}_j^{sep}) \prec_{leximin} g_r^{*\perp}(\emptyset)$ for child $j \in Ch_i$ (e.g. $\prec_{leximin}$ is used instead of $\preceq_{leximin}$ when agents determine the optimal assignment). Additional flags in VALUE messages switch the pruning mode.

TABLE 3. Number of iterations ($w = [0, 1]$) (trm.: number of completed instances)

n, c	10, 9				20, 19			
alg.	msg. cyc.	ncop.	ncst. [s]	trm.	msg. cyc.	ncop.	ncst. [s]	trm.
b	901	118380	0.022	**50**	13820	1504919	0.367	43
gl	715	110374	0.0215	**50**	11151	1382353	0.338	45
glou	404	295626	0.049	**50**	5926	4599000	0.752	49
glousv	243	261805	0.040	**50**	2620	3326015	0.575	**50**
lvb	285	66132	0.012	**50**	8200	922458	0.231	46
lvgl	246	**63040**	**0.011**	**50**	7001	**865591**	**0.204**	47
lvglou	122	148461	0.0210	**50**	1343	2154010	0.339	**50**
lvglousv	**88**	146986	0.023	**50**	**622**	1851827	0.280	**50**
lvglousvbj	**88**	146986	0.064	**50**	**622**	1851827	1.007	**50**
n, c	10, 12				20, 22			
b	14864	5406728	1.015	44	41136	15711705	3.163	16
gl	9352	4303904	0.843	49	35555	13993149	3.082	25
glou	4208	6472761	0.969	**50**	27399	38337847	6.722	33
glousv	2909	6679108	1.037	**50**	21177	39958924	7.591	41
lvb	12943	4536151	0.866	46	37863	15003390	3.110	23
lvgl	7799	**3533306**	0.707	**50**	31895	**13381399**	**3.023**	30
lvglou	2332	4410715	**0.656**	**50**	9067	23771553	4.011	49
lvglousv	1742	5043808	0.807	**50**	6364	25350356	4.380	**50**
lvglousvbj	**985**	4220513	1.781	**50**	**4548**	23614979	13.895	**50**
n, c	10, 15				40, 39			
b	43056	44511576	6.706	17	40099	6669717	1.706	16
gl	31963	40362659	6.921	34	37604	6516241	1.743	17
glou	21423	51906214	7.895	45	30422	26902314	3.430	28
glousv	18450	67587148	10.444	48	20754	23319660	3.369	40
lvb	42309	42243988	7.171	17	30576	5662956	1.538	26
lvgl	28172	37222882	6.858	39	28815	**5580486**	1.631	29
lvglou	14682	43229508	**6.644**	48	9615	14243382	1.754	47
lvglousv	13159	58524454	9.209	48	**3802**	9221296	**1.221**	**50**
lvglousvbj	**3919**	**16839045**	8.205	**50**	**3802**	9221296	5.490	**50**

Due to space limitation, we show the results of pairwise t-test for the number of message cycles as a verification. Except the same number of message cycles of 'lvglousv' and 'lvglousvbj', in the cases of trees, the following pairs are undistinguished under 95 percent confidential interval. (n, c , alg., alg.)=(10, 9, glousv, lvb), (10, 9, glousv, lvgl), (20, 19, glou, lvgl), (40, 39, glou, lvb), (40, 39, glou, lvgl).

after a number of 50000 cycles. Additionally, the number of non-concurrently performed operations (ncops) relating to objective functions and assignments is also evaluated. While it resembles ncccs (Meisels et al., 2002), we also consider several operations that involve a (partial) assignment. Moreover, the longest chain of actual computation time of agents, ignoring communication time, (denoted by 'ncst') is shown. For this result, five trials for the same instance are averaged. We performed the experiments on a single computer with Core i7-4930 @ 3.40GHz, 28GB memory, Linux 2.6.32 and g++ 4.4.7.

Note that our current implementations of search methods redundantly repeat similar computations in the internal processing of agents. Namely, several equations on bounded vectors including evaluation of objective vectors and pruning methods are separately computed, while they contain similar parts. Moreover, the search methods also repeat the distributed search processing. Therefore, the ncops obtained for the search methods is larger than that obtained with simple implementation of dynamic programming based solution methods when the dynamic programming is applicable. For example, in the case of $(n, c) = (20, 22)$ and $w = [0, 1]$, the dynamic programming method required 99874 ncops, while `lvglou` required 25350356 ncops in average. Since our current interest is the interaction among the agents on new boundaries based on sorted vectors and leximin, the optimized internal processing will be addressed in future work as a common issue of similar search methods. On the other hand, we reduced the number of redundant messages that transfer the same information. We mainly focus on the impact that new solutions have on the number of message cycles and messages, as a measure of the interaction among agents.

Tables 3 and 4 shows the number of iterations and several results on computational performances. In these results, the best value in each column is emphasized. The efficient methods reduce the number of message cycles. `gl` prunes the search, and its effect is improved by `glou` and `glousv`. In particular, `glou` is effective, since it prunes branches with full information of boundaries. Although `glou` employs the limit values $-\infty$ and ∞, the pruning works. The effect comes from the property that leximin partially compares values in two vectors. Also, the additional shortcut VALUE messages (`glousv` and `lvglousv`) reduces message cycles. In the case of asymmetric DCOPs, there are opportunities to send the shortcut VALUE messages even if an original pseudo tree is an exact tree as shown in the cases of (n, c)=(10, 9), (20, 19) and (40, 39) in both tables. Note that this is caused by the modification of decision makers. As a simple example, consider a tree of linear graph consisting of nodes/variables x_i, x_j, x_k, where x_i and x_k are the root and leaf nodes, respectively. Since x_i refers the values of its child x_j, x_i decides the value of x_j. On the other hand, x_k refers the values of its parent x_j. Hence, there are the opportunities of the shortcut VALUE messages from x_i to x_j.

In addition, `lv*` that employ the lower and upper limit values for each function $f_i(X_i)$ are effective in the case of trees and less effective for cyclic networks. For example, in the case of (n, c)=(10, 9) in Table 3, the number of message cycles of `lvb` is less than that of `b`. However, in the case of (n, c)=(10, 12) and (10, 15) the ratio of those values is relatively small. This reveals the need for better bounding methods, as available with conventional DCOP solvers. Such methods are, however, domain specific, since the decomposition of $f_i(X_i)$ and the identification of the preferences of the agents is necessary.

`lvglousvbj` employs such an identification of the preferences of the agents to transfer shortcut UTIL (UTILBJ) messages. For the problem instances of cyclic graphs, the shortcut UTILBJ messages reduce message cycles, as shown in the cases of (n, c)=(10, 12), (10, 15) and (20, 22). On the other hand, in the case of trees, there are no opportunities to send the UTILBJ messages, and `lvglousvbj` is the same as `lvglousv`.

Advanced methods need more ncops than basic methods. For example, as shown in the most results of `b` and `lvglousv`, `lvglousv` needs more ncops than `b` even if `lvglousv` reduces the number of message cycles. Therefore, there are several trade-offs between computation and communication. In these results, `lvgl` relatively well reduced both the number of message cycles and ncops. For *heavy* methods such as `lvglousvbj`, the effect on the number of message cycles is important to mitigate the trade-offs, as shown in the case of (n, c)=(10, 15). On the other hand, there are opportunities to reduce ncops in our implementation.

The results of computation time 'ncst' is still affected by the perturbation of computer environment due to the limitation of trials. On the other hand, those relatively similar to

TABLE 4. Number of iterations ($w = [0, 10]$) (trm.: number of completed instances)

n, c	10, 9				20, 19			
alg.	msg. cyc.	ncop.	ncst. [s]	trm.	msg. cyc.	ncop.	ncst. [s]	trm.
b	578	108958	**0.02445**	**50**	10827	1480711	0.474	46
gl	483	100642	0.02790	**50**	8432	1314528	0.436	46
glou	285	249952	0.051	**50**	3253	4106505	0.974	**50**
glousv	185	231108	0.043	**50**	1385	3224723	0.665	**50**
lvb	412	88892	0.02789	**50**	8532	1233105	0.492	47
lvgl	344	**82610**	0.02448	**50**	6481	**1082643**	**0.433**	48
lvglou	199	202010	0.040	**50**	1262	2851531	0.679	**50**
lvglousv	**135**	191823	0.044	**50**	**641**	2519020	0.607	**50**
lvglousvbj	**135**	191823	0.096	**50**	**641**	2519020	1.688	**50**
n, c	10, 12				20, 22			
b	13300	5661150	1.464	46	39620	18234099	5.335	20
gl	5705	3715226	0.889	**50**	30975	13432645	**4.149**	29
glou	2918	4649116	**0.825**	**50**	21114	30493686	7.169	39
glousv	2050	4964571	0.915	**50**	14751	33158647	8.224	46
lvb	12697	5422566	1.626	47	38172	17700952	6.866	23
lvgl	4923	**3409258**	1.018	**50**	28846	**12981207**	5.233	32
lvglou	2000	3571027	0.833	**50**	10480	20413559	6.388	47
lvglousv	1499	4199580	1.014	**50**	7085	22026501	7.333	49
lvglousvbj	**915**	3539162	1.945	**50**	**6035**	21069540	17.487	**50**
n, c	10, 15				40, 39			
b	42199	52022404	9.682	18	37055	7540898	**2.763**	20
gl	26976	38610222	8.000	40	33627	7202239	2.902	26
glou	19449	44551099	**7.544**	46	19496	25948196	7.292	39
glousv	16721	56891746	9.347	47	10826	20088109	4.763	48
lvb	41956	51168367	13.207	18	33544	7200254	3.229	25
lvgl	23694	37399351	10.836	42	30144	**6833475**	3.220	31
lvglou	15064	42115210	10.201	49	11851	20341194	5.650	46
lvglousv	12987	54435512	13.316	49	**5581**	14697514	3.556	**49**
lvglousvbj	**3525**	**14889248**	9.333	**50**	**5581**	14697514	10.430	**49**

Due to space limitation, we show the results of pairwise t-test for the number of message cycles as a verification. Except the same number of message cycles of 'lvglousv' and 'lvglousvbj', in the cases of trees, the following pairs are undistinguished under 95 percent confidential interval. (n, c , alg., alg.)=(10, 9, gl, lvb), (10, 9, glousv, lvglou), (10, 12, glousv, lvglou), (10, 15, glousv, lvglou), (20, 19, gl, lvb), (40, 39, gl, lvb), (40, 39, glousv, lvglou).

the results of ncops, except the cases of `lvglousvbj`. For example, in the case of (n, c)=(10, 12) in Table 3, `lvglousvbj` needs more ncst than `lvglous`, while its ncops is less than that of `lvglous`. This result reveals an implementation issue of the vectors that is not captured by ncops. In `lvglousvbj`, the run-length encoding of vectors cannot be made available, since it has to distinguish individual objectives for the UTILBJ messages. Moreover, indices of the objectives are also necessary. Hence, its trade-offs are tighter than other methods.

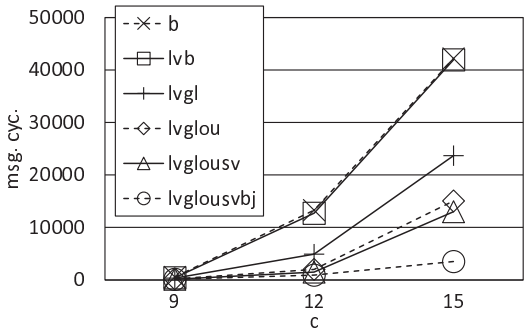


FIGURE 5. Number of message cycles ($w = [0, 10], n = 10$)

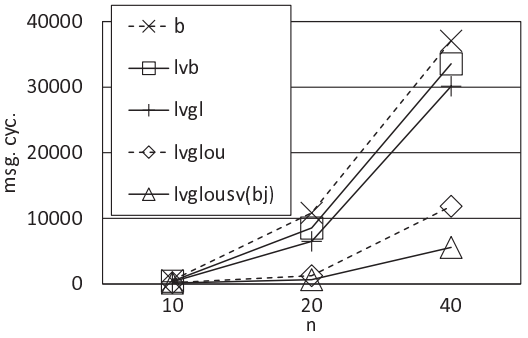


FIGURE 6. Number of message cycles ($w = [0, 10]$, tree)

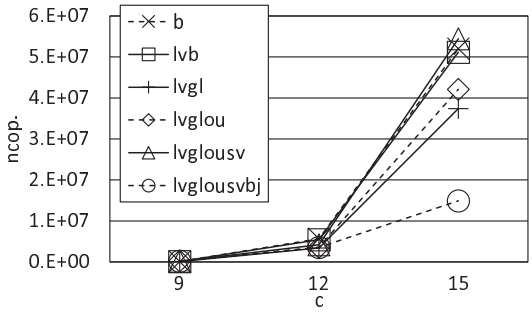


FIGURE 7. ncops. ($w = [0, 10], n = 10$)

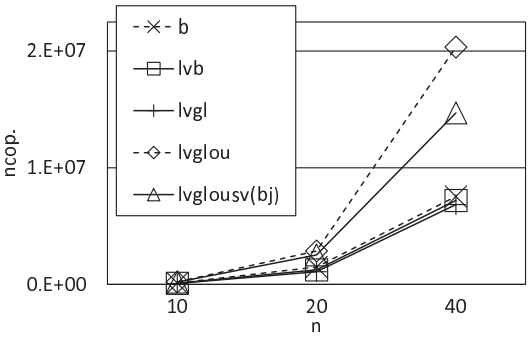


FIGURE 8. ncops. ($w = [0, 10]$, tree)

Table 4 shows the case of $w = [0, 10]$. The results resemble the ones in the case of $w = [0, 1]$.

We also show the number of message cycles and ncops for several problem settings in the case of $w = [0, 10]$, and solution methods in Figs. 5-8. Figures 5 and 6 show the number of message cycles for the density of constraints in the case of $n = 10$, and the number of nodes in the case of trees, respectively. These results show that the scalability of the solution methods resemble conventional ones. The number of message cycles exponentially grows with the density of the problems and the number of agents. Figures 7 and 8 show ncops for the same settings. While the results resemble the cases of the number of message cycles, the efficiencies of solution method are various, since those depend on the trade-offs between ncops in a message cycle, and the number of message cycles.

Tables 5 and 6 shows the number of messages per cycle. Since we employed a rule that reduce redundant messages that repeatedly transfer the same information, the number of messages is relatively small. While the number of shortcut VALUE (VALUESC) messages are relatively large even in the cases of trees, relatively small number of shortcut UTIL (UTILBJ) messages are transferred.

Table 7 shows the size of the pseudo trees. There are a number of agents with an empty X_i^{dcd}. These agents only evaluate their objective values. While there are opportunities to reduce this redundancy by revealing the objective functions of the agents, it will also be domain specific.

Table 8 shows the size of the vectors. The actual size (2sz.) of the representation of the vectors is relatively smaller than the length (len.) of the vectors in the case of $w = [0, 1]$. In these results, the computation of leximin is reduced, since the number of pairs (sz.) to be

TABLE 5. Number of messages per cycle ($w = [0, 1]$)

n, c	10, 9				10, 12				10, 15			
alg.	VAL	VALSC	UTL	UTLBJ	VAL	VALSC	UTL	UTLBJ	VAL	VALSC	UTL	UTLBJ
b	2		2		2		2		2		2	
gl	2		2		2		2		2		2	
glou	2		2		2		2		2		2	
glousv	3	2	3		3	4	3		3	4	3	
lvb	2		2		2		2		2		2	
lvgl	2		2		2		2		2		2	
lvglou	3		2		2		2		2		3	
lvglousv	4	3	3		3	5	4		3	4	4	
lvglousvbj	4	3	3	0	4	5	4	1	4	6	4	1
n, c	20, 19				20, 22				40, 39			
alg.	VAL	VALSC	UTL	UTLBJ	VAL	VALSC	UTL	UTLBJ	VAL	VALSC	UTL	UTLBJ
b	2		2		2		2		3		2	
gl	2		2		2		2		3		3	
glou	3		3		3		3		4		4	
glousv	4	4	4		5	6	5		6	6	5	
lvb	3		3		2		2		4		3	
lvgl	3		3		3		2		4		4	
lvglou	4		4		4		4		7		7	
lvglousv	7	6	6		7	8	7		12	11	10	
lvglousvbj	7	6	6	0	8	9	7	1	12	11	10	0

enumerated is less than the length of vectors. On the other hand, `lvglousvbj` employs indexed sorted vectors. In our experiment, we simply implemented the indexed objective values using an additional array of length "lng.".

The results show that the sorted objective vectors and leximin ordering are successfully generalized with the pseudo tree based solution methods. Also, the monotonicity on boundaries of the sorted objective vectors correctly works with search methods. On the other hand, the results reveal the necessity of more sophisticated methods including several implementation techniques to reduce overheads.

7. DISCUSSIONS AND FUTURE WORKS

In this study, we investigated two types of solution methods based on pseudo trees. The aim of our study is to clarify the decomposition of problems and the generalization of boundaries on Leximin AMODCOPs through these solution methods. In the current results shown in the evaluations, the dynamic programming method is promising for the problems with relatively small size of separators, while the search methods require high computational overhead. Since several causes of overhead in search methods relate to optimizations of algorithms and implementation techniques, they are above the main scope of this study. However, the additional computations for the search increase computation time. The efficiency of the techniques to handle sorted objective vectors should be improved. The optimized internal processing and the identification of appropriate pruning operations on promising classes of problems will be included in future work.

In our proposed search methods, the high induced width exponentially increases the

TABLE 6. Number of messages per cycle ($w = [0, 10]$)

n, c	10, 9				10, 12				10, 15			
alg.	VAL	VALSC	UTL	UTLBJ	VAL	VALSC	UTL	UTLBJ	VAL	VALSC	UTL	UTLBJ
b	2		2		2		2		2		2	
gl	2		2		2		2		2		2	
glou	2		2		2		2		2		3	
glousv	3	2	3		3	4	3		3	5	3	
lvb	2		2		2		2		2		2	
lvgl	2		2		2		2		2		2	
lvglou	2		2		2		2		2		3	
lvglousv	4	3	3		3	5	4		3	5	4	
lvglousvbj	4	3	3	0	4	5	4	1	4	6	4	1
n, c	20, 19				20, 22				40, 39			
alg.	VAL	VALSC	UTL	UTLBJ	VAL	VALSC	UTL	UTLBJ	VAL	VALSC	UTL	UTLBJ
b	2		2		2		2		3		3	
gl	3		2		3		3		3		3	
glou	3		3		3		3		5		5	
glousv	5	5	4		5	7	5		8	8	7	
lvb	3		2		2		2		3		3	
lvgl	3		3		3		3		4		3	
lvglou	4		4		4		4		6		5	
lvglousv	6	6	5		6	8	6		10	10	8	
lvglousvbj	6	6	5	0	7	8	6	1	10	10	8	0

TABLE 7. Size of pseudo tree

n, c	max. depth	max. $\lvert Ch_i \rvert$	max. $\lvert X_i^{sep} \rvert$	max. $\lvert X_i \rvert$	max. $\lvert X_i^{dcd} \rvert$	num. of agents s.t. $\lvert X_i^{dcd} \rvert = 0$	max. $\lvert Sc_i \rvert$
10, 9	5	3	2	4	4	4	4
10, 12	6	2	5	5	5	6	5
10, 15	7	2	7	6	6	6	6
20, 19	8	4	2	5	5	9	5
20, 22	9	3	5	6	6	10	6
40, 39	11	5	2	6	6	17	6

number of search iterations. For addressing this issue, a promising direction is to investigate more aggressive modifications of graphs (Vinyals et al., 2011). Also, there are opportunities to approximate the problems (Rogers et al., 2011; Delle Fave et al., 2011). While existing efficient techniques including forward-bounding may improve the efficiency of solution methods (Netzer and Meisels, 2013a), it will require higher computational/communication overhead.

In Netzer and Meisels (2013a), a class of Asymmetric DCOP similar to ones in this study has been addressed, where its objective is the minimization of envy among agents. In that study, a synchronous search algorithm based on a total order on variables employs a forward bounding technique with shortcut messages, while we employed shortcut messages that are

TABLE 8. Size of vector

w	$[0,1]$						$[0,10]$					
n, c	20, 22			40, 39			20, 22			40, 39		
alg.	len.	sz.	2sz.	len.	sz.	2sz.	len.	sz.	2sz.	len.	sz.	2sz.
lvb	4	2	**4**	3	2	**3**	4	3	**7**	3	3	**5**
lvgl	9	3	**5**	16	2	**5**	10	6	**12**	16	7	**14**
lvglou	11	3	**6**	21	3	**6**	11	7	**15**	21	10	**19**
lvglousv	11	3	**6**	22	3	**6**	11	7	**15**	22	10	**20**
lvglousvbj	11	3	**6**	22	3	**6**	11	7	**15**	22	10	**20**

sent beside an asynchronous search on a pseudo tree. The comparison of both approaches is an interesting issue and will be included in future works.

In Bouveret and Lemaître (2009), several solution methods for a centralized constraint optimization problem on the leximin ordering have been proposed. The solution methods employ several techniques including branch-and-bound with leximin ordering and several types of constraints to represent relationships of leximin. On the other hand, the solution methods are dedicated centralized algorithm for constant solvers. How those techniques can be decomposed into distributed algorithms to construct efficient solution methods is also an interesting issue.

In general cases of optimization problems with criteria of equality, agents are interested in the comparison between their utilities. On the other hand, identifying the utility of a particular agent is useful for the solution methods that handles objective vectors. This tradeoff between the privacy of agents and effectiveness of the solution method will be important.

We discussed our solution methods on a basic Leximin AMODCOP for the sake of simplicity. For future works, we address several motivation domains below.

Example 2 (Resource allocation on a power supply network): In a resource allocation problem on a power supply network (Miller et al., 2012; Matsui and Matsuo, 2012), each agent represents a node of the network. An agent i has several input links, output links and its resource. Given the amount $x^l_{i,j}$ of transferred resource on each input/output link (i,j) and x^r_i of its own resource, the total amount must satisfy resource constraint c_i : $\sum_{x^l_{j,i} \in X^{in}_i} x^l_{j,i} = x^r_i + \sum_{x^l_{i,k} \in X^{out}_i} x^l_{i,k}$. Here, X^{in}_i and X^{out}_i corresponds to input and output links, respectively. In addition, agent i has an objective function $f^r_i(x^r_i)$ of its own resource use x^r_i. Using a sufficiently small objective value for the violation of hard constraint c_i, this problem is represented by $f_i(X_i)$ for agent i, where X_i consists of $\{x^i_r\} \cup X^{in}_i \cup X^{out}_i$. The value of $f_i(X_i)$ is $f^r_i(x^r_i)$ if assignments for X_i satisfy c_i. Otherwise, $f_i(X_i)$ takes the sufficiently small value. Each agent desires to improve its local objective value under the resource constraints and preferences of other agents.

While this modeling is very simplistic, it represents a basic structure in resource constraint DCOPs related to (leximin) Asynchronous Multi-objective problems.

Example 3 (Variation of Coalition Structure Generation): A Coalition Structure Generation problem is represented as a DCOP (Ueda et al., 2010). An agent i has two variables x_i and x^g_i. x^g_i represents a group to which agent i belongs. x^g_i takes a value 'alone' when agent i belongs to a group of single agent. x_i represents i's decision. Depending on x^g_i, utility values that relate to x_i are defined as follows. $f^v_{i,j}(x_i, x_j, x^g_i, x^g_j) = v_{i,j}(x_i, x_j)$ if $x^g_i \neq$

$`alone' \wedge x_i^g = x_j^g$. Otherwise, $f_{i,j}^v(x_i, x_j, x_i^g, x_j^g) = 0$. $f_i^v(x_i, x_i^g) = v_i(x_i)$ if $x_i^g = `alone'$. Otherwise, $f_i^v(x_i, x_i^g) = 0$. Based on this DCOP, a local problem is represented as $f_i(X_i) = f_i^v(x_i, x_i^g) + \sum_{j \in Nbr_i} f_{i,j}^v(x_i, x_j, x_i^g, x_j^g)$ for agent i aggregating utility functions among i and its neighborhood agents Nbr_i.

For the problems above, several extensions of the solution method will be necessary to handle specific structures and large solution spaces efficiently.

8. CONCLUSIONS

In this work, we presented a multiple objective DCOP that considers preferences of agents, and its solution method based on the leximin ordering on multiple objectives. To solve the problem, the solution methods based on a pseudo tree of a constraint network are applied to the leximin ordering on objective vectors.

As addressed in the previous section, our future work will include improvements of solution methods, reducing redundant computation, evaluations in practical domains, and analysis on various types of problems.

ACKNOWLEDGMENT

This work was supported in part by KAKENHI Grant-in-Aid for Scientific Research (C), 25330257.

REFERENCES

BOUVERET, SYLVAIN, and MICHEL LEMAÎTRE. 2009. Computing leximin-optimal solutions in constraint networks. Artificial Intelligence, **173**(2):343–364.

DECHTER, RINA. 1999. Bucket elimination: A unifying framework for reasoning. Artificial Intelligence, **113**(1-2):41–85. ISSN 0004-3702.

DELLE FAVE, FRANCESCO M., RUBEN STRANDERS, ALEX ROGERS, and NICHOLAS R. JENNINGS. 2011. Bounded decentralised coordination over multiple objectives. *In* 10th International Conference on Autonomous Agents and Multiagent Systems, Volume 1, pp. 371–378.

FARINELLI, A., A. ROGERS, A. PETCU, and N. R. JENNINGS. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. *In* 7th International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 639–646.

GRINSHPOUN, TAL, ALON GRUBSHTEIN, ROIE ZIVAN, ARNON NETZER, and AMNON MEISELS. 2013. Asymmetric distributed constraint optimization problems. Journal of Artificial Intelligence Research, **47**:613–647.

HAMADI, YOUSSEF, CHRISTIAN BESSIÈRE, and JOËL QUINQUETON. 1998. Distributed intelligent backtracking. *In* 13th European Conference on Artificial Intelligence, pp. 219–223.

MAHESWARAN, R. T., M. TAMBE, E. BOWRING, J. P. PEARCE, and P. VARAKANTHAM. 2004. Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. *In* AAAMAS04, pp. 310–317.

MARLER, R. TIMOTHY, and JASBIR S. ARORA. 2004. Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization, **26**:369–395.

MATSUI, TOSHIHIRO, and HIROSHI MATSUO. 2012. Considering equality on distributed constraint optimization problem for resource supply network. *In* 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology, Volume 2, pp. 25–32.

MATSUI, TOSHIHIRO, HIROSHI MATSUO, and AKIRA IWATA. 2005. Efficient methods for asynchronous distributed constraint optimization algorithm. *In* IASTED International Conference on Artificial Intelligence and Applications, part of the 23rd Multi-Conference on Applied Informatics, pp. 727–732.

MATSUI, TOSHIHIRO, MARIUS SILAGHI, KATSUTOSHI HIRAYAMA, MAKOTO YOKOO, and HIROSHI MAT-

SUO. 2012. Distributed search method with bounded cost vectors on multiple objective dcops. *In* Principles and Practice of Multi-Agent Systems - 15th International Conference, pp. 137–152.

MEISELS, A., E. KAPLANSKY, I. RAZGON, and R. ZIVAN. 2002. Comparing performance of distributed constraints processing algorithms. *In* 3rd International Workshop on Distributed Constraint Reasoning, p. (no page numbers).

MILLER, SAM, SARVAPALI D. RAMCHURN, and ALEX ROGERS. 2012. Optimal decentralised dispatch of embedded generation in the smart grid. *In* 11th International Conference on Autonomous Agents and Multiagent Systems, Volume 1, pp. 281–288.

MODI, P. J., W. SHEN, M. TAMBE, and M. YOKOO. 2005. Adopt: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence, **161**(1-2):149–180.

MOULIN, HERVE. 1988. Axioms of Cooperative Decision Making. Cambridge : Cambridge University Press.

NETZER, ARNON, and AMNON MEISELS. 2011. SOCIAL DCOP - Social Choice in Distributed Constraints Optimization. *In* 5th International Symposium on Intelligent Distributed Computing, pp. 35–47.

NETZER, ARNON, and AMNON MEISELS. 2013a. Distributed Envy Minimization for Resource Allocation. *In* 5th International Conference on Agents and Artificial Intelligence, Volume 1, pp. 15–24.

NETZER, ARNON, and AMNON MEISELS. 2013b. Distributed Local Search for Minimizing Envy. *In* 2013 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp. 53–58.

PECORA, F., P.J. MODI, and P. SCERRI. 2006. Reasoning about and dynamically posting n-ary constraints in adopt. *In* 7th International Workshop on Distributed Constraint Reasoning, at AAMAS, 2006.

PETCU, ADRIAN, and BOI FALTINGS. 2005. A scalable method for multiagent constraint optimization. *In* IJCAI05, pp. 266–271.

PETCU, ADRIAN, BOI FALTINGS, and DAVID PARKES. 2008. M-DPOP: Faithful Distributed Implementation of Efficient Social Choice Problems. Journal of Artificial Intelligence Research, **32**:705–755.

RAMCHURN, SARVAPALI D., ALESSANDRO FARINELLI, KATHRYN S. MACARTHUR, and NICHOLAS R. JENNINGS. 2010. Decentralized coordination in robocup rescue. Comput. J., **53**(9):1447–1461.

ROGERS, A., A. FARINELLI, R. STRANDERS, and N. R. JENNINGS. 2011. Bounded approximate decentralised coordination via the Max-Sum algorithm. Artificial Intelligence, **175**(2):730–759.

SEN, AMARTYA KUMAR. 1997. Choice, Welfare and Measurement. Harvard University Press.

UEDA, SUGURU, ATSUSHI IWASAKI, MAKOTO YOKOO, MARIUS SILAGHI, KATSUTOSHI HIRAYAMA, and TOSHIHIRO MATSUI. 2010. Coalition structure generation based on distributed constraint optimization.

VINYALS, MERITXELL, JUAN A. RODRIGUEZ-AGUILAR, and JESÚS CERQUIDES. 2011. Constructing a unifying theory of dynamic programming dcop algorithms via the generalized distributive law. Autonomous Agents and Multi-Agent Systems, **22**(3):439–464.

VINYALS, MERITXELL, ERIC SHIEH, JESUS CERQUIDES, JUAN ANTONIO RODRIGUEZ-AGUILAR, ZHENGYU YIN, MILIND TAMBE, and EMMA BOWRING. 2011. Quality guarantees for region optimal dcop algorithms. *In* 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1, pp. 133–140.

YEOH, WILLIAM, ARIEL FELNER, and SVEN KOENIG. 2008. Bnb-adopt: an asynchronous branch-and-bound dcop algorithm. *In* 7th International Joint Conference on Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-1-6. pp. 591–598.

YOKOO, MAKOTO, and KATSUTOSHI HIRAYAMA. 1998. Distributed constraint satisfaction algorithm for complex local problems. *In* 3rd International Conference on Multiagent Systems, pp. 372–381.

ZHANG, WEIXIONG, GUANDONG WANG, ZHAO XING, and LARS WITTENBURG. 2005. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. Artificial Intelligence, **161**(1-2):55–87. ISSN 0004-3702.

ZIVAN, ROIE. 2008. Anytime local search for distributed constraint optimization. *In* AAAI08, pp. 393–398.