Fundamenta Informaticae 158 (2018) 63–91
DOI 10.3233/FI-2018-1642
IOS Press

Leximin Multiple Objective DCOPs on Factor Graphs for Preferences of Agents

Toshihiro Matsui*
Nagoya Institute of Technology
Gokiso-cho Showa-ku Nagoya 466-8555, Japan
matsui.t@nitech.ac.jp

Marius Silaghi
Florida Institute of Technology
Melbourne FL 32901, United States of America
msilaghi@fit.edu

Tenda Okimoto
Kobe University
5-1-1 Fukaeminami-machi Higashinada-ku Kobe 658-0022, Japan
tenda@maritime.kobe-u.ac.jp

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

Makoto Yokoo
Kyushu University
744 Motooka Nishi-ku Fukuoka 819-0395, Japan
yokoo@is.kyushu-u.ac.jp

Hiroshi Matsuo
Nagoya Institute of Technology
Gokiso-cho Showa-ku Nagoya 466-8555, Japan
matsuo@nitech.ac.jp

Abstract. Distributed Constraint Optimization Problem (DCOP) has been studied as a fundamental component of multiagent systems. With DCOPs, various applications on multiagent systems are formalized as constraint optimization problems where variables and functions are distributed among agents. Leximin AMODCOP has been proposed as a class of Multiple Objective DCOPs, where multiple objectives for individual agents are optimized based on the leximin operator. This problem also relates to Asymmetric DCOPs based on its the criteria of fairness among agents. Previous studies explore only Leximin AMODCOPs on constraint graphs limited to functions with unary or binary scopes. We address the Leximin AMODCOPs on factor graphs that directly represent n-ary functions. A dynamic programming method on factor graphs is investigated as an exact solution method. In addition, for relatively dense problems, we also investigate several approximate/inexact algorithms.

Keywords: distributed constraint optimization, asymmetric, multiple objectives, leximin, egalitarian

*Address for correspondence: Nagoya Institute of Technology, Gokiso-cho Showa-ku Nagoya 466-8555, Japan

1. Introduction

1.1. Distributed Constraint Optimization Problem and preferences of agents

The Distributed Constraint Optimization Problem (DCOP) [1, 2, 3, 4] has been studied as a fundamental component of multiagent systems. With DCOPs, various applications on multiagent systems, including sensor network, power smart grid, and disaster response, are formalized as constraint optimization problems where variables and functions are distributed among agents [5, 4, 6]. The agents cooperatively solve these problems in a decentralized manner. While the goal of the original DCOP is the optimization on the summation of all functions, that traditional criterion does not capture the preferences of agents. In practical resource allocation tasks, the individual preference of each agent is a natural requirement. Such problems are considered here as a class of multiple objective problems on the preferences of agents. Moreover, to manage the preferences among agents, fairness (or inequality) is an important criterion.

The Multiple Objective Distributed Constraint Optimization Problem (MODCOP) [7, 8] has been studied as an extension to DCOPs. With MODCOPs, agents cooperatively solve multiple objective problems. While the original MODCOP addresses a few objectives that are shared among all agents, the concept of the multiple objectives is generalized into the preferences of individual agents. The preferences of agents also relate to Asymmetric DCOPs [9, 10] where each agent differently evaluates asymmetric functions. The concept of the Asymmetric DCOP is also extended with MODCOPs where each agent differently evaluates its own local problem that represents the preference of the agent. Such MODCOPs can be optimized with several criteria including fairness or inequality. Recently, several existing studies focused on these classes of problems [11, 12]. The fairness among agents is an important requirement in practical resource allocation tasks [13, 14, 11, 12].

1.2. Motivating domains

For example, in a distributed sensor network, which consists of multiple sensors and multiple observation areas (or targets), the sensors are separately allocated to their neighboring observation areas to cover the areas cooperatively [4]. When an observation area is modeled as an agent (several sensor nodes may perform as such agents), the agent requires some sensor resources to obtain its necessary observation quality. In this case, fairness among observation areas are important to assure a baseline observation quality over the whole system.

As another example, in a micro smart grid with distributed power sources, a power transformer station or a small group of buildings can be considered as an agent who provides or consumes power resources [5]. Assume that they have to share the limited power resources with other agents connected by power lines. This resource allocation can be formalized as a distributed optimization problem. In general, agents have different resource requirements. These may be represented by individual preference values so that the requirements can be compared with each other. In this case, the fairness among preference values of agents should be optimized.

In a representation of Coalition Structure Generation (CSG) problems [15], agents generate several groups in a distributed manner. In this problem, each agent determines whether it belongs to one of the groups or it does not belong to any groups. When an agent belongs to a group, it should collaborate

with other agents in the group, and it obtains a utility based on an interaction of the agents. Otherwise, the agent does not collaborate with any agents, and obtains a different utility. These can be considered as task allocation problems or team generation problems. In this case, it is natural that each agent desires to improve its own utility individually[1].

In the above problems, the agents evaluate their own objectives. Therefore, an asymmetric representation is necessary for the individual objectives. In addition, a criterion, which is different from the conventional summation, is necessary to aggregate/compare the objectives considering fairness.

1.3. Asymmetric problem with leximin and extension on factor graphs

As a class of MODCOPs, Leximin AMODCOP, where multiple objectives for individual agents are optimized based on the leximin operator, has been proposed [16]. This problem is also a class of Asymmetric DCOPs. Leximin is a well-known egalitarian social welfare that represents the fairness/inequality among agents. Since the maximization on leximin ordering improves equality among agents, the Leximin AMODCOP is considered as a fundamental class of DCOPs. While the leximin is defined on vectors of objectives, the optimization on leximin can be decomposed using a dynamic programming approach for DCOPs [16, 2]. The previous study [16] has proposed the Leximin AMODCOP on constraint graphs for binary and unary functions. Constraint graphs of Asymmetric DCOPs are represented as directed arc graphs where nodes and directed arcs/edges stand for variables and functions, respectively [9, 13]. Therefore, the direction of edges should be handled in solution methods.

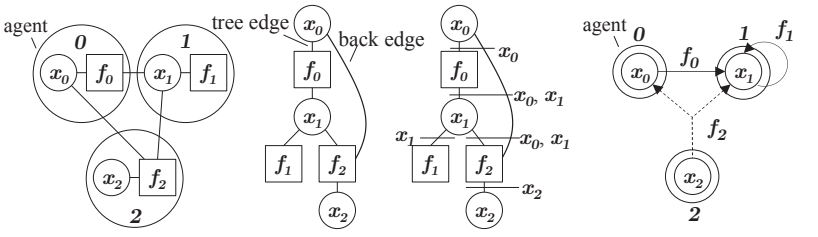
On the other hand, this class of problems is well represented with factor graphs. In factor graphs, nodes stand for variables or functions, while non-directed edges stand for scopes of functions. Since a function is separately treated as a node in factor graphs, the function node is owned by an agent, where the function represents the preferences of the agent. Therefore, there are no directions of edges that represent ownership of the functions. Namely, asymmetric functions are naturally represented as factor graphs without any modifications. In addition, factor graphs directly represent n-ary functions.

For the Asymmetric DCOPs, the n-ary function can be considered as a general representation, since an agent is interested in its own objective values which are aggregated for the agent. Even if an n-ary asymmetric objective function can be originally defined by multiple asymmetric binary functions, the agent firstly aggregates the functions to evaluate the true objective values, in the case of solution methods without dedicated pruning technics. Moreover, the n-ary function generally represents preferences of decisions among three or more agents including preferences partially depending on the number of agents or complicated friendships.

1.4. Our contributions

In this paper, we propose several solution methods for Leximin AMODCOPs on factor graphs. First we present a complete solution method based on a dynamic programming on factor graphs. Then several approximation methods and a local search are proposed to address the issue of combinational explosion in the complete solution method. The effects and influences of the proposed methods are

[1]Here we do not address the solution concepts such as the Shapley value or the nucleolus.



(a) factor graph/AMODCOP (b) pseudo tree (c) separators (d) constraint graph/AMODCOP

x_0	x_1	x_2	a_0	a_1	a_2
a	a	a	1	2	1
a	a	b	1	2	2
a	b	a	2	1	3
a	b	b	2	1	4
b	a	a	3	2	1
b	a	b	3	2	2
b	b	a	4	1	3
b	b	b	4	1	4

(d) asymmetric function

x_0	x_1	f_0
a	a	1
a	b	2
b	a	3
b	b	4

x_1	f_1
a	2
b	1

x_0	x_1	x_2	f_2
a	a	a	1
a	a	b	2
a	b	a	3
a	b	b	4
b	a	a	1
b	a	b	2
b	b	a	3
b	b	b	4

(e) individual functions

Figure 1. AMODCOP on factor graph

experimentally evaluated. Our contributions are as follows. (1) With factor graphs, we generalized the representation of Leximin AMODCOPs and a solution method based on dynamic programming. (2) To reduce the size of partial problems in the dynamic programming based solution method, two approximation methods are introduced. These methods employ a heuristic to eliminate assignments to variables, and a mini-buckets approach. (3) A local search method is applied to Leximin AMODCOPs on factor graphs. (4) We experimentally evaluate the effect of optimization based on leximin criterion in several classes of problems. (5) For the approximation approaches and inexact solution methods, we also experimentally show several influences on solution quality.

This paper is an extended work of the previous study [17]. We refined several descriptions and experimental results including new evaluations on a structured problem domain. In addition, a new solution method that employs the mini-buckets approach is introduced.

The paper is organized as follows. In the next section, we present several preliminaries of our study including DCOPs, factor graphs and Leximin AMODCOPs. Then we present several solution methods for LeximinAMODCOPs on factor graphs in Section 3. The proposed methods are experimentally evaluated in Section 4. We address related works and discussions in Section 5 and conclude our study in Section 6.

2. Preliminaries

In the following, we present preliminaries of our study. Several definitions and notations are inherited from previous literature [16, 18].

2.1. DCOP

A distributed constraint optimization problem (DCOP) is defined as follows.

Definition 2.1. (DCOP)
A DCOP 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. The variables and functions are distributed to the agents in A. A variable $x_n \in X$ takes values from its domain defined by the discrete finite set $D_n \in D$. A function $f_m \in F$ is an objective function defining valuations of a constraint among several variables. Here f_m represents utility values that are maximized. We also call the utility values of f_m, objective values. $X_m \subset X$ defines the set of variables that are included in the scope of f_m. $F_n \subset F$ similarly defines a set of functions that include x_n in its scope. f_m is defined as $f_m(x_{m0}, \cdots, x_{mk}) : D_{m0} \times \cdots \times D_{mk} \to \mathbb{N}_0$, where $\{x_{m0}, \cdots, x_{mk}\} = X_m$. $f_m(x_{m0}, \cdots, x_{mk})$ is also simply denoted by $f_m(X_m)$. The aggregation $F(X)$ of all the objective functions is defined as follows: $F(X) = \sum_m \text{s.t. } _{f_m \in F, X_m \subseteq X} f_m(X_m)$. The goal is to find a globally optimal assignment that maximizes the value of $F(X)$.

Each agent locally knows its own variables and related functions. A distributed optimization algorithm is performed to compute the globally optimal solution.

2.2. Factor graph, Max-Sum algorithm and bounded Max-Sum algorithm

The factor graph [3] is a representation of DCOPs, and is a bipartite graph consisting of variable nodes, function nodes and edges. An edge represents a relationship between a variable and a function. Figure 1(a) shows a factor graph consisting of three variable nodes and three function nodes. As shown in the case of a ternary function f_2, the factor graph directly represents n-ary functions.

The Max-Sum algorithm [3] is a method for solving a DCOP by exploiting its factor graph. Each node of the factor graph corresponds to an 'agent' referred to as variable node or function node. Each such node communicates with neighborhood nodes using messages to compute globally optimal solutions. A message represents an evaluation function for a variable. A node computes/sends a message for each variable that corresponds to a neighborhood node. Here the nodes of functions in F_n are called the *neighborhood function nodes* of variable node x_n. Similarly, the nodes of variables in X_m are called the *neighborhood variable nodes* of function node f_m. A message payload $q_{x_n \to f_m}(x_n)$ that is sent from variable node x_n to function node f_m is represented as follows.

$$q_{x_n \to f_m}(x_n) = \begin{cases} 0 & \text{if } F_n = \{f_m\} \\ \sum\limits_{f_{m'} \in F_n \setminus \{f_m\}} r_{f_{m'} \to x_n}(x_n) & \text{otherwise} \end{cases} \tag{1}$$

A message payload $r_{f_m \to x_n}(x_n)$ that is sent from function node f_m to variable node x_n is represented as follows.

$$r_{f_m \to x_n}(x_n) = \max_{\varepsilon \in D_{X_m \setminus \{x_n\}}} \left(f_m(\varepsilon, x_n) + \sum_{x_{n'} \in X_m \setminus \{x_n\}} q_{x_{n'} \to f_m}(\varepsilon \| _{x_{n'}}) \right) \tag{2}$$

Here $\max_{\varepsilon \in D_{X_m \setminus \{x_n\}}}$ denotes the maximization for all assignments of variables in $X_m \setminus \{x_n\}$. A variable node x_n computes a marginal function that is represented as $z_n(x_n) = \sum_{m \text{ s.t. } f_m \in F_n} r_{f_m \to x_n}(x_n)$. Since $z_n(x_n)$ corresponds to global objective values for variable x_n, the variable node of x_n chooses the value of x_n that maximizes $z_n(x_n)$ as its solution. See [3] for the details of the algorithm.

In the cases where a factor graph contains cycles, the Max-Sum algorithm is an inexact method that may not converge, since the computation on different paths cannot be separated. In Bounded Max-Sum algorithm [19], a cyclic factor graph is approximated to a maximum spanning tree (MST) using a preprocessing that eliminates the cycles. For the computation of MST, the impact of edge e_{ij} between function f_i and variable x_j is evaluated as weight value $w_{ij} = \max_{X_i \setminus \{x_j\}} \left(\max_{x_j} f_i(X_i) - \min_{x_j} f_i(X_i)\right)$. When a set of variables $X_i^c \in X_i$ is eliminated from the scope of function f_i, the function is approximated to $\tilde{f}_i = \min_{X_i^c} f_i(X_i)$. Then, the Max-Sum algorithm is applied to the spanning tree as an exact solution method. In this computation, a couple of bottom-up and top-down processing steps based on a rooted tree are performed similarly to DPOP [2].

2.3. Asymmetric DCOP

Asymmetric DCOP (ADCOP) [9, 10] is an extended class of DCOPs, where different objective functions are asymmetrically defined for a set of agents. Based on the definition of DCOP shown in Definition 2.1, an ADCOP is generally defined as follows.

Definition 2.2. (Asymmetric DCOP)
An Asymmetric DCOP is defined by (A, X, D, F), where A, X and D are similarly defined as the original DCOP. For a set of agents $A_m \subseteq A$, a function $f_{A_m} \in F$ is defined as $f_{A_m}(x_{A_{m,0}}, \cdots, x_{A_{m,k}}) : D_{A_{m,0}} \times \cdots \times D_{A_{m,k}} \to \mathbb{N}_0^{|A_m|}$, where each dimension of $\mathbb{N}_0^{|A_m|}$ corresponds to an agent in A_m. For each agent $i \in A_m$, function f_{A_m} is also redefined as $f_{A_{m,i}}(x_{A_{m,0}}, \cdots, x_{A_{m,k}}) : D_{A_{m,0}} \times \cdots \times D_{A_{m,k}} \to \mathbb{N}_0$. $f_{A_m}(x_{A_{m,0}}, \cdots, x_{A_{m,k}})$ and $f_{A_{m,i}}(x_{A_{m,0}}, \cdots, x_{A_{m,k}})$ are also simply denoted by $f_{A_m}(X_{A_m})$ and $f_{A_{m,i}}(X_{A_m})$. The aggregation $F(X)$ of all the objective functions is defined as follows: $F(X) = \sum_{A_m \text{ s.t. } f_{A_m} \in F, X_{A_m} \subseteq X} f_{A_m}(X_{A_m})$. The goal is to find a globally optimal assignment that maximizes the value of $F(X)$.

Here each objective function $f_{A_{m,i}}(X_{A_m})$ represents the valuation for one of the agents. The goal of the problem is to optimize the summation of objective values. In the previous studies on ADCOPs, solution methods for binary asymmetric functions on pairs of agents are presented [9, 10]. In the case of binary asymmetric functions, two different functions $f_i(x_a, x_b)$ and $f_j(x_a, x_b)$ are defined for a pair of variables x_a and x_b in different agents i and j.

2.4. Multiple objective DCOP for preferences of agents

2.4.1. Multiple objective DCOPs

Multiple objective DCOP [7] (MODCOP) is a generalization of the DCOP framework. With MODCOPs, multiple objective functions are defined over the variables. The objective functions are si-

multaneously optimized based on appropriate criteria. The tuple with the values of all the objective functions for a given assignment is called the *objective vector*.

Definition 2.3. (Objective vector)
An objective vector $\mathbf{v}$ is defined as $[v_0, \cdots, v_K]$, where v_j 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^j is the subset of X on which F^j is defined. $F^j(X^j)$ is an objective function for objective j. For assignment $\mathcal{A}$, the vector $\mathbf{F}(\mathcal{A})$ of the functions returns an objective vector $[v_0, \cdots, v_K]$. Here $v_j = F^j(\mathcal{A}^j)$.

Since there is a trade-off among objectives, objective vectors are compared based on Pareto dominance [20, 21]. In the case of maximization problems, the dominance between two objective vectors, and Pareto optimality on the assignments are defined as follows.

Definition 2.4. (Pareto dominance)
A vector $\mathbf{v}$ dominates $\mathbf{v}'$ if and only if $\mathbf{v} \geq \mathbf{v}'$, and $v_k > v'_k$ for at least one objective k.

Definition 2.5. (Pareto optimality)
Assignment $\mathcal{A}^*$ is Pareto optimal if and only if there is no other assignment $\mathcal{A}$, such that $\mathbf{F}(\mathcal{A}) \geq \mathbf{F}(\mathcal{A}^*)$, and $F^k(\mathcal{A}^k) > F^k(\mathcal{A}^{*k})$ for at least one objective k.

Multiple objective problems generally have a set of Pareto optimal solutions that form a Pareto front.

2.4.2. Social welfare

With a social welfare that defines an order on objective vectors, traditional solution methods for single objective problems can be applied to choose a Pareto optimal solution. There are several criteria of social welfare [20] and scalarization methods [21]. A traditional social welfare is defined as the summation $\sum_{j=0}^{K} F^j(\mathcal{A}^j)$ of objectives. The maximization of this summation ensures Pareto optimality. However, it does not capture the equality on these objectives. *Maximin* maximizes the minimum objective value. While maximin improves the worst case, it is not Pareto optimal. Maximin is also improved with summation that breaks ties of maximin ordering. See literatures [20, 21] for the details of above criteria. The study in [13] addresses a multiple objective Asymmetric DCOP whose social welfare is based on Theil index. This social welfare also represents inequality/fairness among agents. However, a local search algorithm is employed to solve the problem, since the social welfare is non-monotonic.

Another social welfare, called *leximin*, is defined with a lexicographic order on objective vectors whose values are sorted in ascending order.

Definition 2.6. (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 2.7. (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 $\exists t, \forall t' < t, v_{t'} = v'_{t'} \wedge v_t < v'_t$.

The maximization on the leximin ordering ensures Pareto optimality. The leximin is an 'egalitarian' criterion, since it reduces the inequality on objectives.

2.4.3. Leximin Asymmetric MODCOP on preferences of agents

Leximin Asymmetric MODCOP (Leximin AMODCOP) [16] is a class of MODCOP, where each objective stands for a preference of an agent. This problem also relates to extended Asymmetric DCOPs with fairness or envy among agents [14, 13, 11, 12, 9]. Here each agent individually has its set of objective functions whose aggregated value represents the preference of the agent. On the other hand, several agents relate each other, since the subsets of their variables are contained in the scope of the same function. A Leximin AMODCOP is defined as follows [16].

Definition 2.8. (Leximin AMODCOP)
A Leximin AMODCOP is defined by (A, X, D, F), where A, X and D are similarly defined as for the DCOP in Definition 2.1. Agent $i \in A$ has its local problem defined on $X_i \subseteq X$. $\exists(i, j)$ s.t. $i \neq j, X_i \cap X_j \neq \emptyset$. F is a set of objective functions $f_i(X_i)$ for all $i \in A$. The function $f_i(X_i) : D_{i_0} \times \cdots \times D_{i_k} \to \mathbb{N}_0$ 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.

In general cases, Leximin AMODCOPs are NP-hard, similar to DCOPs.

The operations in the solution methods for DCOPs are extended for the leximin. The evaluation values are replaced by the sorted objective vectors, and the comparison on objective values is extended with the leximin operator. Also, the addition of objective values is extended as a concatenation operation of objective values. The 'addition' of sorted vectors is defined as follows [16].

Definition 2.9. (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.

In Bounded Max-Sum algorithm and our proposed method, partial solutions and related evaluation values are aggregated in a bottom up manner on a tree structure. This aggregation can be naturally extended for the leximin based on the similar operation whose correctness has been proven in [16].

Figure 1(a) shows a factor graph of the (Leximin) AMODCOP, where each agent i has a variable x_i and a function f_i. Since factor graphs directly represent n-ary functions, any asymmetric problems are well figured using this graph structure. Note that scope X_i of f_i should contain x_i. On the other hand, Figure 1(d) shows the constraint graph of the same problem. It requires directed arcs to represent the ownership of the functions. Solution methods for such constraint graphs have to handle the direction of edges. Moreover, a hyper-edge is necessary to represent an n-ray (ternary) function f_2.

With the form of asymmetric functions shown in Definition 2.2, an asymmetric ternary function for three agents in Fig. 1(a) is depicted as in Fig. 1(d). Here a_i denotes agent i. We assume that

each variable takes a value from $\{a, b\}$. The asymmetric function is also redefined via the individual functions shown in Fig. 1(e). Since agents 0 and 1 are only interested in $\{x_0, x_1\}$ and $\{x_1\}$, binary and unary functions represent the objective values of those agents.

For cyclic factor graphs, the traditional Max-Sum algorithm is inexact. Namely, an objective value is redundantly aggregated via different paths [3, 22]. A possible approach to avoid the redundant aggregation is the computation based on a spanning tree of the factor graph, similar to the Bounded Max-Sum algorithm [19]. However, this approximation is not very promising, since it eliminates several relationships between functions and variables. That may decrease the actual minimum objective value and the solution quality on leximin ordering.We therefore employ different types of algorithms.

3. Solution methods for Leximin AMODCOPs on factor graphs

As an exact solution method for Leximin AMODCOPs, we introduce a dynamic programming algorithm based on pseudo trees of factor graphs. Then, we also introduce an approximation method and a local search algorithm. Here we assume that there are communication channels between any pairs of agents.

3.1. Dynamic programming based on pseudo tree

Several solution methods employ pseudo trees [1, 2] to decompose problems on constraint graphs. On the other hand, there are a few similar studies for factor graphs [18]. We employ a solution method based on pseudo trees on factor graphs [2].

3.1.1. Pseudo trees on factor graphs

A pseudo tree on a factor graph is constructed in a preprocessing of the main optimization method. Here we employ a DFS tree for a factor graph. The DFS graph traversal is initiated from a variable node and performed for all nodes ignoring their types. Edges of the original factor graph are categorized into tree edges and back edges based on the DFS tree. Figure 1(b) shows a pseudo tree for the factor graph of Fig. 1(a). Based on the factor graph and DFS tree, several related nodes are defined for each variable/function node i as follows.

- Nbr_i: the set of i's neighborhood nodes.
- $Nbrh_i$/$Nbrl_i$: the set of i's neighborhood nodes in higher/lower depths. Here the depth is based on the DFS tree so that the root node is the highest node.
- $prnt_i$: the parent node of i.
- $Chld_i$: the set of i's child nodes.

[2]While the previous study employs cross-edge pseudo trees and a search algorithm [18], we employ DFS trees and dynamic programming methods for the sake of simplicity. The pseudo trees based on DFS trees have no cross-edges and do not need a dedicated technique in [18].

- Sep_i the set of separators: i.e., the variables related both to the subtree rooted at i and to i's ancestor nodes.
- $\overline{Sep_i}$: the set of non-separator variables. In the dynamic programming, each agent i computes a function table for separator variables in Sep_i by maximizing a function aggregated for variables in $Sep_i \cup \overline{Sep_i}$.
- $Seph^i_j$: the set of function nodes that are higher neighborhood nodes of variable node j. Here j is contained in Sep_i.

The separators and non-separators are defined for variable node i as follows.

$$Sep_i = \begin{cases} \{\ \} & \text{if } i \text{ is the root node} \\ \{i\} \cup \bigcup_{j \in Chld_i} Sep_j & \text{otherwise} \end{cases} \tag{3}$$

$$\overline{Sep_i} = \begin{cases} \{i\} & \text{if } i \text{ is the root node} \\ \{\ \} & \text{otherwise} \end{cases} \tag{4}$$

$$Seph^i_i = \begin{cases} \{\ \} & \text{if } i \text{ is the root node} \\ Nbrh_i & \text{otherwise} \end{cases} \tag{5}$$

$$Seph^i_k = \bigcup_{j \in Chld_i} Seph^j_k, \text{ where } k \in Sep_i \wedge k \neq i \wedge (i \text{ is non-root node}). \tag{6}$$

The set of separators Sep_i is empty in the root node, while other nodes aggregate their own variable and separators of child nodes (Eq. (3)). Only root node i has non-separator i (Eq. (4)). Non-root nodes set their own $Seph^i_i$ as $Nbrh_i$ (Eq. (5)). For other nodes k in separators Sep_i, node i sets $Seph^i_k$ aggregating $Seph^j_k$ of child nodes j (Eq. (6)).

For function node i, the separators and non-separators are defined as follows.

$$Sep_i = \left(Nbrh_i \cup \bigcup_{j \in Chld_i} Sep_j \right) \setminus \overline{Sep_i} \tag{7}$$

$$\overline{Sep_i} = \{l \mid l \in Nbrl_i,\ Seph'^i_l = \{\ \}\} \tag{8}$$

$$Seph'^i_k = \begin{cases} \left(\bigcup_{j \in Chld_i, k \in Sep_j} Seph^j_k \right) \setminus \{i\} & \text{if } k \in Nbrl_i \\ \bigcup_{j \in Chld_i, k \in Sep_j} Seph^j_k & \text{otherwise} \end{cases} \tag{9}$$

$$Seph^i_k = Seph'^i_k, \text{ where } k \in Sep_i. \tag{10}$$

Each node sets separators Sep_i aggregating $Nbrh_i$ and separators of child nodes. Then, non-separators $\overline{Sep_i}$ are eliminated from Sep_i (Eq. (7)). Here non-separators in $\overline{Sep_i}$ are the variable

nodes whose topmost neighborhood function node is i (Eq. (8) and (9)). For child nodes j and nodes k in separators Sep_j, node i aggregates $Seph^j_k$. Then, i is eliminated if k is i's neighborhood variable (Eq. (9)). After Sep_i is set, $Seph'^i_k$ is also used to set $Seph^i_k$ for k in Sep_i (Eq. (10)). In above equations, if function node i is the highest neighborhood node of variable node k, then k is not included in Sep_i. This computation is performed in a bottom-up manner from leaf nodes to the root node. It is possible to integrate the computation into the backtracking of the DFS traversal for the pseudo tree. Figure 1(c) illustrates separators of the pseudo tree shown in Fig. 1(b). For $i = f_1$, $\overline{Sep_{f_1}} = \{\ \}$, $Sep_{f_1} = \{x_1\}$ and $Seph^{f_1}_{x_1} = \{\ \}$. For $i = x_2$, $Sep_{x_2} = \{x_2\}$, $\overline{Sep_{x_2}} = \{\ \}$ and $Seph^{x_2}_{x_2} = \{f_2\}$. For $i = f_2$, $Seph'^{f_2}_{x_2} = \{\ \}$, $\overline{Sep_{f_2}} = \{x_2\}$, $Sep_{f_2} = \{x_0, x_1\}$, $Seph^{f_2}_{x_0} = \{\ \}$ and $Seph^{f_2}_{x_1} = \{\ \}$. For $i = x_1$, $Sep_{x_1} = \{x_0, x_1\}$, $\overline{Sep_{x_1}} = \{\ \}$, $Seph^{x_1}_{x_1} = \{f_0\}$ and $Seph^{x_1}_{x_0} = \{\ \}$. Similar computations are performed for the other nodes.

3.1.2. Dynamic programming

Exploiting the pseudo tree on a factor graph, a dynamic programming method consisting of two phases is performed. The computation of the first phase is represented as follows.

$$g^*_i(Sep_i) = \max_{\overline{Sep_i}}{}_{leximin}\ g_i(Sep_i \cup \overline{Sep_i}) \tag{11}$$

$$g_i(Sep_i \cup \overline{Sep_i}) = \begin{cases} \bigoplus\limits_{j \in Chld_i} g^*_j(Sep_j) & \text{if } i \text{ is a variable node} \\ f_i(X_i) \oplus \bigoplus\limits_{j \in Chld_i} g^*_j(Sep_j) & \text{otherwise} \end{cases} \tag{12}$$

Note that the above expressions include the cases such that $Sep_i = \{\ \}$ (the root variable node) or $\overline{Sep_i} = \{\ \}$ (non-root variable nodes and leaf function nodes). In expression (12), for each assignment $\mathcal{A}$ of $Sep_i \cup \overline{Sep_i}$, compatible assignments $\mathcal{A}_i$ of X_i and $\mathcal{A}_{Sep_j}$ of Sep_j are aggregated. This computation is performed in a bottom-up manner. As a result, each node i has its optimal objective vectors $g^*_i(Sep_i)$ for the assignments of its separators and the subtree rooted at i.

This computation is a dynamic programming based on the following proposition [16].

Proposition 3.1. (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 $\mathbf{v} \oplus \mathbf{v}'' \prec_{leximin} \mathbf{v}' \oplus \mathbf{v}''$.

The computation of the second phase is performed in a top-down manner. The optimal assignment d^*_i of the root variable node i, that is also represented as $\mathcal{A}^*_{\overline{Sep_i}} = \{d^*_i\}$, is determined so that $g^*(\mathcal{A}^*_{Sep_i}) = g(\mathcal{A}^*_{Sep_i} \cup \mathcal{A}^*_{\overline{Sep_i}})$. Namely, $g^*(\{\}) = g(\mathcal{A}^*_{\overline{Sep_i}})$. The optimal assignments of other variable nodes are determined by their parent or ancestor node. For each child node j of i, its optimal separator Sep_j is determined by i so that $\mathcal{A}^*_{Sep_j} \subseteq \mathcal{A}^*_{Sep_i} \cup \mathcal{A}^*_{\overline{Sep_i}}$, where $g^*(\mathcal{A}^*_{Sep_i}) = g(\mathcal{A}^*_{Sep_i} \cup \mathcal{A}^*_{\overline{Sep_i}})$. Note that the above expressions also include the cases such that $Sep_i = \{\ \}$ or $\overline{Sep_i} = \{\ \}$. In the actual computation of the first phase, each agent i propagates $g^*_i(Sep_i)$ to $prnt_i$. Then, in the second phase, each agent i propagates $\mathcal{A}^*_{Sep_j}$ for each j in $Chld_i$.

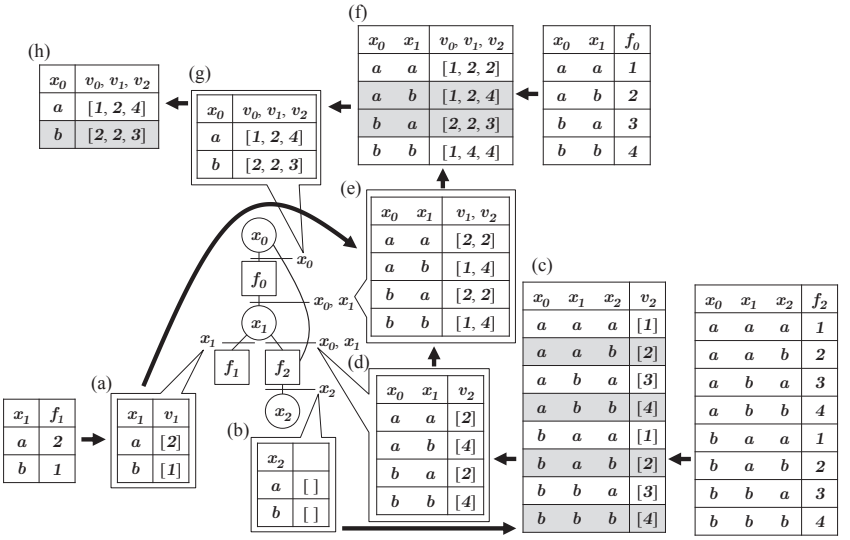


Figure 2. First phase of dynamic programming

For the problem shown in Fig. 1, the first phase of dynamic programming is depicted as Fig. 2. The computation begins from leaf nodes. Leaf node f_1 evaluates its function and propagates table (a) for its separator. This table contains objective vectors of single values for the agent of f_1. On the other hand, leaf node x_2 propagates table (b) which contains empty objective vectors. Function node f_2 generates table (c) aggregating its function and table (b) from node x_2. Then the function of table (c) is maximized eliminating assignments to variable x_2. The resulting function shown as table (d) is propagated. Variable node x_1 generates table (e) aggregating tables (a) and (d). Note that the vectors in the table are sorted vectors. Since related separators do not change in this node, table (e) is propagated. Similarly, function node f_0 generates table (f) and propagates table (g). Finally, root node x_0 generates table (h). From this table, the optimal sorted objective vector $[2, 2, 3]$ and corresponding assignment b to x_0 is selected. In the second phase, the optimal solution $(x_0, x_1, x_2) = (b, a, b)$ is selected based on tables (h), (f) and (c) in a top-down manner. When a similar computation is performed with summation operator, the optimal objective value is $4 + 1 + 4 = 9$ for $(x_0, x_1, x_2) = (b, b, b)$.

This solution method inherits most parts of the correctness and the time/space complexity from conventional methods based on dynamic programming such as DPOP [2] and Bounded Max-Sum [19]. The overhead of operations on sorted vectors for leximin can be estimated as almost $O(n)$ for a sequential comparison of values of vectors, where n is the size of sorted vector. The sorting of values can be implemented as red-black tree whose time complexity is $O(\log n)$ [16].

3.2. Approximation method fixing values of variables

In the above exact dynamic programming method, each node i computes a table of objective vectors $g_i^*(Sep_i)$ for corresponding separators Sep_i. Therefore, the solution method is not applicable for the large number of separators. In such cases, several approximation methods can be applied to eliminate several back edges and corresponding separators. However, if the relationship between a variable

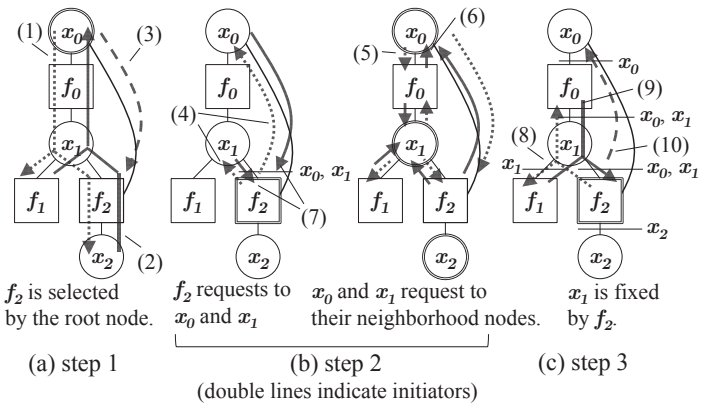


Figure 3. Flow of approximation

and a function is completely eliminated, the value of the variable is determined ignoring the actual values of other variables in the scope of the function. As a result, the actual minimum objective value cannot be well controlled. That may decrease the quality of solutions, since leximin ordering is very sensitive to the minimum objective value. Here we employ another approach that fixes several values of variables. To eliminate separators, we define a threshold value $maxnsep$ for the maximum number of separators. Based on the threshold value $maxnsep$, the approximation is iteratively performed as multiple rounds. Each round consists of the following steps. Step 1: selection of the node with the maximum number of separators (Fig. 3(a)). Step 2: selection/fixation of the variable of the largest impact in the separators (Fig. 3(b)). Step 3: notification of the fixed variable (Fig. 3(c)).

In Step 1, each node i reports the number of separators in Sep_i and its identifier. In actual computation, the computation is initiated by the root node in a top-down manner (Fig. 3(1)). The information of the number of separators is then aggregated in a bottom-up manner (Fig. 3(2)). Based on the aggregated information, an agent j who has the maximum number is selected to eliminate one of its separators. If $|Sep_j|$ is less than or equal to the threshold value $maxnsep$, the iteration of rounds is terminated. Otherwise, the root node notifies j so that j eliminates a separator (Fig. 3(3)).

In Step 2, node j eliminates a separator by fixing its value. First, for each separator x_k in Sep_j, node j requests the variable node of x_k to evaluate the impact of variable x_k (Fig. 3(4)). Then, for each neighborhood (function) node of f_l in Nbr_k, each variable node k of separator x_k requests function node of f_l to evaluate its impact (Fig. 3(5)). Each function node of f_l then returns the information of $f_l^{\perp}(x_k) = \min_{X_i \setminus \{x_k\}} f_i(X_i)$ to variable node of separator x_k (Fig. 3(6)). $f_l^{\perp}(x_k)$ represents the lower bound of f_l for x_k. Then, for each f_l, the boundaries are aggregated into sorted vectors so that $h_k^{\perp}(x_k) = \bigoplus_{f_l \in Nbr_k} f_l^{\perp}(x_k)$. Then, lower bound $h_k^{\perp\perp} = \min_{leximin_{x_k}} h_k^{\perp}(x_k)$ and upper bound $h_k^{\perp\top} = \max_{leximin_{x_k}} h_k^{\perp}(x_k)$ of $h_k^{\perp}(x_k)$ are computed. Variable node of x_k returns $h_k^{\perp\perp}$ and $h_k^{\perp\top}$ to node j (Fig. 3(7)). Now, node j determines the separator $x_{\hat{k}}$ to be fixed so that $\hat{k} = \text{argmin}_{leximin_k} h_k^{\perp\perp}$.

Note that the length of sorted objective vectors $h_k^{\perp\perp}$ can be different. In such cases, ∞ is employed as a padding value. As a result, a longer vector that affects more functions is selected in the case of a tie. We infer from the above expression that $x_{\hat{k}}$ is a 'risky' variable, since it's choice may be restricted

to yield lower objective values in future computations of the approximation. Therefore, we prefer to fix this variable in advance. The value of $x_{\hat{k}}$ is fixed to $d_{\hat{k}}$ so that $h_{\hat{k}}^{\perp\top} = h_{\hat{k}}^{\perp}(d_{\hat{k}})$. Here we prefer the value corresponding to the maximum lower bound.

In Step 3, node j propagates the information of $D_{\hat{k}} = \{d_{\hat{k}}\}$ to its parent node and child nodes (Fig. 3(8)). The propagation is terminated when Sep_i or Sep_j s.t. $j \in Ch_i$ in a node i do not contain $x_{\hat{k}}$. Then, the information of termination is returned to node j (Fig. 3(9)). Then, node j notifies the root node of the termination of a round (Fig. 3(10)).

Note that the above algorithm is a base line to clarify the flow of information. We believe that there are several opportunities to optimize the message paths. This approximation method is a heuristic algorithm focusing on the worst case. Such a pessimistic approach is relatively reasonable for leximin ordering, since the minimum objective value has a major influence on the quality of the solutions. The upper bound objective value of each function, whose related variables are fixed, is calculated by maximizing its objective values for the fixed variables. However, the upper bound objective vector of an approximated solution cannot be directly calculated, since the objective values are evaluated on leximin ordering. Instead of that, the upper bound objective vector can be solved as the optimal solution of an approximated problem with the upper bound values of the functions. It also means that the technique of Bounded Max-Sum to calculate upper bound objective values is unavailable for leximin ordering.

3.3. Approximation method based on mini-buckets

In the previous subsection, for all related variable/function nodes, a consistent decision on the same variable of an eliminated edge is enforced by fixing the value of the valuable. On the other hand, if there are relatively many candidates of assignments, contradictions of decisions on the same variable may be mitigated. Here we apply the mini-buckets [23] that is an approximation method for bucket-elimination algorithm. The basic idea of mini-buckets is the approximation of function $g_i^*(Sep_i)$ using functions of smaller arities. With a limit value $maxnsep$, $g_i^*(Sep_i)$ can be approximated as $\bigoplus_{p \in P_i} g_i^{*p}(Sep_i^p) \approx g_i^*(Sep_i)$, where $|Sep_i^p| \leq maxnsep$. Each $g_i^{*p}(Sep_i^p)$ is maximized for the assignments on Sep_i^p. We call $g_i^{*p}(Sep_i^p)$ a mini-bucket. Note that the term is slightly different from the original mini-buckets due to the DPOP-like solution method on factor graphs.

In the bottom-up computation of the dynamic programming, $|P_i|$ mini-buckets $g_i^{*p}(Sep_i^p)$ of agent i are propagated to i's parent node $prnt_i$. Then the parent node $prnt_i$ aggregates $g_i^{*p}(Sep_i^p)$ and constructs its own mini-buckets $g_{prnt_i}^{*p'}(Sep_{prnt_i}^{p'})$.

To avoid double-counts of functions, We restrict the aggregation of functions and mini-buckets. A function f_i is aggregated into a single mini-bucket, when node i is a function node. Each $g_j^{*p}(Sep_j^p)$ of child agent j is aggregated into a single mini-bucket. In addition, we assume that $maxnsep$ is not less than the maximum arity for all functions, to avoid partial evaluation of the functions. We use the following heuristic to compute mini-buckets. Basically, each node i determines the number of its own mini-buckets at first. Then the vectors of the mini-buckets are computed.

Step 1: In the initial state, node i has no mini-buckets. If necessary, new mini-buckets will be generated.

Step 2: The node repeatedly checks functions f in $\{f_i\} \cup \bigcup_{j \in Chld_i} \bigcup_{p \in P_j} \{g_j^{*p}(Sep_j^p)\}$, and computes the arities of existing mini-buckets under the assumption where f is aggregated into the mini-buckets. If f can be aggregated into an own mini-bucket $g_i^{*p}(Sep_i^p)$ such that $|Sep_i^p| \leq maxnsep$, function f is related to $g_i^{*p}(Sep_i^p)$. Otherwise, a new mini-bucket is generated and f is related to the new mini-bucket. Then the arity of the corresponding mini-bucket is updated. The above computation is repeated until each function in $\{f_i\} \cup \bigcup_{j \in Chld_i} \bigcup_{p \in P_j} \{g_j^{*p}(Sep_j^p)\}$ is related to a mini-bucket.

Step 3: Based on the relationships between mini-buckets and functions, the vectors of all $g_i^{*p}(Sep_i^p)$ are computed by maximizing the aggregation of functions related to mini-bucket p. In this step, $g_i^{*p}(Sep_i^p)$ is computed for $\mathcal{A}_{Sep_i^p} \in Sep_i^p$ as follows.

$$g_i^{*p}(\mathcal{A}_{Sep_i^p}) = \max_{leximin\ \mathcal{A} \in \overline{Sep_i}} \bigoplus_{f \text{ related to mini-bucket } p} f((\mathcal{A}_{Sep_i^p} \cup \mathcal{A})_{\downarrow \text{args}(f)}), \tag{13}$$

where $\mathcal{A}_{\downarrow \text{args}(f)}$ represents the projection of $\mathcal{A}$ on the scope of f.

After the bottom-up computation, all nodes have their own mini-buckets. Actually, the computation of the mini-buckets in the root node can be omitted, since each node has to re-compute the aggregated vectors of mini-buckets to determine its approximately optimal assignment. In the top-down computation, the optimal assignment on mini-buckets is determined. The root node i determines the approximately optimal assignment $\tilde{\mathcal{A}}_i = \{\tilde{d}_i\}$ to its own non-separators $\overline{Sep_i}$ by maximizing the aggregation of approximated functions as follows. Note that $\tilde{\mathcal{A}}_{prnt_i} = \emptyset$ in the root node.

$$\tilde{\mathcal{A}}_i = \text{argmax}_{leximin\ \mathcal{A} \in \overline{Sep_i}} \bigoplus_{p \in P_i} \bigoplus_{f \text{ related to mini-bucket } p} f((\tilde{\mathcal{A}}_{prnt_i} \cup \mathcal{A})_{\downarrow \text{args}(f)}) \tag{14}$$

Then $\tilde{\mathcal{A}}_i$ is propagated to child nodes of node i. While $\tilde{\mathcal{A}}_i$ can be filtered for each separators of each child node, we commonly use $\tilde{\mathcal{A}}_i$ in the child nodes, for simplicity. When approximately optimal assignment $\tilde{\mathcal{A}}_{prnt_i}$ is propagated, agent i similarly determines its own assignment as shown in Equation (14). Note that the above re-computations are necessary to avoid contradictions on assignments among different mini-buckets, which are independently maximized for related functions in the bottom-up computation. After the top-down computation, all agents know the assignment on its own variable.

The aggregation of approximated functions $\bigoplus_{p \in P_i} g_i^{*p}(Sep_i^p)$ is an upper bound of $g_i^*(Sep_i)$. Therefore, the optimization on the mini-buckets can be considered as an optimistic approach.

3.4. Local search

Another inexact approach is based on local search methods. Here we employ a local search method from a previous study [13]. While the original method is designed for constraint networks, we adapt the method to (Leximin) AMODCOPs with factor graphs. This local search is cooperatively performed by each agent with its neighborhood agents. Since each agent i has its own variable node x_i and function node f_i, the neighborhood agents $ANbr_i$ of agent i are defined as a set of agents who have a neighborhood node of x_i or f_i. Note that we denote the neighborhood nodes of x_i and f_i as

1 Preprocessing:
2 let $Nbr^-_{x_i}$ denote $(Nbr_i$ of $x_i) \setminus \{f_i\}$. let $Nbr^-_{f_i}$ denote $(Nbr_i$ of $f_i) \setminus \{x_i\}$.
3 $ANbr_{x_i} \leftarrow \bigcup_j$(the owner agent of f_j in $Nbr^-_{x_i}$).
4 $ANbr_{f_i} \leftarrow \bigcup_j$(the owner agent of x_j in $Nbr^-_{f_i}$). $ANbr_i \leftarrow ANbr_{x_i} \cup ANbr_{f_i}$.
5 send $ANbr_i$ to j in $ANbr_i$. receive $ANbr_j$ from all j in $ANbr_i$.
6 $BANbr_i \leftarrow ANbr_i \cup \bigcup_{j \in ANbr_i} ANbr_j$.
7 Main procedure:
8 choose the initial assignment d_i^{cur} of x_i. *// locally maximize* f_i.
9 **until**(cutoff){
10 send d_i^{cur} to all agents j in $ANbr_{x_i}$. receive d_j^{cur} from all agents j in $ANbr_{f_i}$.
11 $\mathcal{A}_i^{cur} \leftarrow \{(x_i, d_i^{cur})\} \cup \bigcup_{x_j \text{ in } Nbr^-_{f_i}} (x_j, d_j^{cur})$.
12 $v_i^{cur} \leftarrow f_i(\mathcal{A}_i^{cur})$. send v_i^{cur} to agents j in $ANbr_i$. receive v_j^{cur} from agents j in $ANbr_i$.
13 $\mathbf{v}_i^{cur} \leftarrow \{v_i^{cur}\} \oplus \bigoplus_{j \text{ in } ANbr_i} \{v_j^{cur}\}$.
14 choose the new assignment d_i^{new} under $\mathcal{A}_i^{cur} \setminus \{(x_i, d_i^{cur})\}$.
15 $\mathcal{A}_i^{new} \leftarrow \{(x_i, d_i^{new})\} \cup \bigcup_{x_j \text{ in } Nbr^-_{f_i}} (x_j, d_j^{cur})$. $v_i^{new} \leftarrow f_i(\mathcal{A}_i^{new})$.
16 send d_i^{new} to all agents j in $ANbr_{x_i}$. receive d_j^{new} from all agents j in $ANbr_{f_i}$.
17 **foreach**(x_k in $Nbr^-_{f_i}$){
18 $\mathcal{A}_{i,k}^{new} \leftarrow \{(x_i, d_i^{cur})\} \cup (x_k, d_k^{new}) \cup \bigcup_{x_j \text{ in } Nbr^-_{f_i} \setminus \{x_k\}} (x_j, d_j^{cur})$. $v_{i,k}^{new} \leftarrow f_i(\mathcal{A}_{i,k}^{new})$.
19 send $v_{i,k}^{new}$ to the owner agent of x_k.
20 }
21 receive v_j^{new} from all agents j in $ANbr_{x_i}$.
22 $\mathbf{v}_i^{new} \leftarrow \{v_i^{new}\} \oplus \bigoplus_{j \text{ in } ANbr_{x_i}} \{v_j^{new}\} \oplus \bigoplus_{k \text{ in } ANbr_i \setminus ANbr_{x_i}} \{v_k^{cur}\}$.
23 **if**($\mathbf{v}_i^{cur} \prec_{leximin} \mathbf{v}_i^{new}$){ $v_i^{dif} \leftarrow \max(0, v_i^{new} - v_i^{cur})$. }**else**{ $v_i^{dif} \leftarrow 0$. }
24 send v_i^{dif} to all agents j in $BANbr_i$. receive v_j^{dif} from all agents j in $BANbr_i$.
25 **if**($v_i^{dif} = \max_{j \text{ in } BANbr_i \cup \{i\}} v_j^{dif}$){ $d_i^{cur} \leftarrow d_i^{new}$. } *// tie is broken by agent IDs.*
26 }

Figure 4. local search (procedures of node i)

Nbr_i and Nbr_i, respectively. In addition, each agent i has to know its second order neighborhood agents $BANbr_i$. $BANbr_i$ is referred in decision making among agents. Since the variable of i's neighborhood agent j affects the functions of j's neighborhood agents including i, agent i should agree with agents within two hops. The above computations are performed in a preprocessing (Fig. 4, lines 1-6).

After the initialization (Fig. 4, line 9), the local search is iteratively performed as multiple rounds (lines 10-27). Each round consists of the following steps. Step 1: notification of current assignments (lines 11 and 12). Step 2: evaluation of current assignments (lines 13 and 14). Step 3: proposal of new assignments (lines 15-17). Step 4: evaluation of new assignments (lines 18-23). Step 5: update of assignments (lines 24-26).

In Step 1, each agent i notifies the agents, whose functions relate to x_i, of the current assignment d_i^{cur} of its own variable x_i. Then, agents update the related current assignments. In Step 2, each agent i evaluates the value of its own function f_i for the current assignment. The valuation of f_i is announced to neighborhood agents. Then, agents update the current valuations. In addition, using

the valuations, a sorted vector is generated. These valuations are stored for future evaluation. In Step 3, each agent i chooses its new assignment d_i^{new} that improves the valuation of f_i under the current assignment of other variables. Agent i then announces the new assignment d_i^{new} to the agents whose functions relate to x_i. In Step 4, each agent i evaluates the value of its own function f_i assuming that an assignment d_k^{cur} in the current assignment is updated to d_k^{new} by an agent who has x_k. Agent i then returns the valuation to the agent of x_k. This process is performed for all variables in the scope of f_i. Each agent of x_k receives and stores the valuation for d_k^{new}. Then, using the valuations, x_k generates a sorted vector for the case of d_k^{new}. In Step 5, each agent i compares the sorted vectors for the cases of d_i^{cur} and d_i^{new}. If the sorted vector for d_i^{new} is preferred, the improvement d_i^{dif} of the valuation of its own function f_i is evaluated. Otherwise, d_i^{dif} is set to 0. Then, agent i notifies agents, within two hops, of the improvement d_i^{dif}. When its own improvement d_i^{dif} is the greatest value in the agents $BANbr_i$, d_i^{cur} is updated by d_i^{new}.

4. Evaluation

4.1. Settings

4.1.1. Example problems and evaluation values

We experimentally evaluated the proposed method. A class of Leximin AMODCOPs is used to generate test problems. The problems consist of n agents who have a ternary variable x_i ($|D_i| = 3$) and a function f_i of arity a. Objective values of the functions were randomly set as follows. g9_2: a rounded integer value based on a gamma distribution with $(\alpha = 9, \beta = 2)$, similar to [19]. u1-10: an integer value in $[1, 10]$ based on uniform distribution. Results were averaged over 25 instances of the problems. We evaluated the following criteria for a sorted objective vector $\mathbf{v}$. scl: a scalarized value of $\mathbf{v}$ shown below. sum: the total value of values in $\mathbf{v}$. min: the minimum value in $\mathbf{v}$. wtheil/theil: WTheil social welfare and Theil index shown below. As a normalization, each criterion (except 'theil') is divided by the corresponding criterion of the upper limit vector. The upper limit vector is defined as the vector consisting of $\max_{X_i} f_i(X_i)$ for all agent i.

4.1.2. Scalarization of sorted vectors (scl)

To visualize sorted vectors, we introduce a scalar measurement. The scalar value represents the location on a dictionary that is compatible with a lexicographic order on the leximin. Here the minimum objective value $v^\perp$ and the maximum objective value $v^\top$ are given. With these limit values, for a sorted vector $\mathbf{v}$, a scalar value $s(\mathbf{v}) = s(\mathbf{v})_{(|A|-1)}$ that represents $\mathbf{v}$'s location on the dictionary is recursively calculated as $s(\mathbf{v})_{(k)} = s(\mathbf{v})_{(k-1)} \cdot (|v^\top - v^\perp| + 1) + (v_k - v^\perp)$ and $s(\mathbf{v})_{(-1)} = 0$. Here v_k is the k^{th} objective value in sorted vector $\mathbf{v}$. Since we consider the values in $[v^\perp, v^\top]$ as the characters in $\{c_0, \cdots, c_{v^\top - v^\perp}\}$ that construct a word in the dictionary, $|v^\top - v^\perp| + 1$ is considered as the number of characters in the 'alphabet'. In the case where $|v^\top - v^\perp|$ and the number of variables are large, we can use multiple precision variables in the actual implementation. Below, we simply use 'scl' that denotes s($\mathbf{v}$).

4.1.3. Social welfare based on Theil Index (wtheil/theil)

In a previous study [13], a social welfare based on Theil Index has been employed. Originally, Theil index is a criterion of unfairness defined as $T = \frac{1}{N}\sum_N^i \left(\frac{x_i}{\overline{x}} \ln \frac{x_i}{\overline{x}}\right)$. Here $\overline{x}$ denotes the average value for all x_i. T takes zero if all x_i are equal. The social welfare is defined as $WTheil = \overline{x}e^{-T}$ so that the average (summation) is integrated to the fairness. We compared the results with Theil Index and WTheil.

4.1.4. Bounded Max-Sum algorithm

As addressed in Subsection 2.4.3, the Bounded Max-Sum algorithm can be adapted to leximin optimization problems. We evaluated such a Bounded Max-Sum (Bounded Max-Leximin) algorithm. While there are opportunities to modify the impact values of edges for minimum spanning trees, we found that other types of impact values were not very effective. Therefore, we simply employed the spanning trees of the original algorithm.

4.2. Results

4.2.1. Different optimzation criteria

First, we compared different criteria of optimization. In this experiment, we employed exact algorithms based on dynamic programming, except the case of WTheil as shown below. The aggregation and maximization operators of the solution method were replaced by other operators similar to the previous study [14]. Those operators are correctly applied to the dynamic programming based on pseudo trees. Table 1 shows the results of the comparison. Here 'ptmaxleximin' denotes the proposed method based on pseudo trees. Compared methods maximize the summation ('ptmaxsum') and the minimum value ('ptmaximin'), respectively. Additionally, 'ptmaximinsum' is an improved version of

Table 1. Comparison with different optimization criteria ($n = 15$, $a = 3$)

prb.	opt. criteria	scl	sum	min	wtheil	theil
g9_2	maxwtheil	0.563	0.815	0.637	**0.799**	0.031
	ptmaximin	0.698	0.735	**0.752**	0.730	0.017
	ptmaximinsum	**0.699**	0.769	**0.752**	0.763	0.019
	ptmaxsum	0.513	**0.818**	0.596	0.797	0.037
	ptmaxlexmin	**0.699**	0.759	**0.752**	0.755	**0.016**
u1-10	maxwtheil	0.636	**0.888**	0.668	**0.879**	0.010
	ptmaximin	0.688	0.840	**0.722**	0.832	0.010
	ptmaximinsum	0.691	0.882	**0.722**	0.874	0.009
	ptmaxsum	0.599	**0.888**	0.632	0.878	0.013
	ptmaxlexmin	**0.692**	0.875	**0.722**	0.869	**0.008**

* Problems were solved by exact algorithms.
* scl, sum, min and wtheil are ratio values to the upper limit vector.
* To be maximized: scl, sum, min, wtheil. To be minimized: theil.

Table 2. Size of pseudo tree (a=3)

n	depth	#leafs	avg. #branches	max. $\lvert Sep_i \rvert$	max. $\prod_{k \in Sep_i} \lvert D_k \rvert$
10	16	3	1.15	6	1558
20	30	7	1.18	11	447460
30	42	11	1.20	15	462162351
40	55	14	1.20	19	1.165E+11
50	65	18	1.21	25	1.156E+13

* Each factor graph consists of n variable nodes and n function nodes.

'ptmaximin' that maximizes the summation when two minimum values are the same. Moreover, we also evaluated an exact solution method that maximizes WTheil ('maxwtheil'). Since WTheil cannot be decomposed into dynamic programming, we employed a centralized solver based on tree search. Due to the time/space complexity of the solution methods, we evaluated the case of $n = 15$ and $a = 3$.

The results in Table 1 shows that 'ptmaxleximin' always maximizes sorted vectors on leximin ordering ('scl'). Similarly, 'ptmaxsum' and 'maxwtheil' always maximize summation ('sum') and wtheil, respectively. 'ptleximin', 'ptmaximin' and 'ptmaximinsum' maximize the minimum value ('min'). While 'ptmaximinsum' relatively increases 'scl' in average, Theil index ('theil') of 'ptleximin' is less than that of 'ptmaximinsum'. Therefore, it is considered that 'ptleximin' improves fairness among agents. Table 2 shows the size of pseudo trees in the case of $a = 3$. Due to the size of $|Sep_i|$, even in the case of $n = 20$, the exact solution method is not applicable in practical time. Therefore, we did not compared exact methods and approximate methods.

4.2.2. Approximation methods and local search method

Next, we evaluated approximate methods and local search methods. Figure 5 shows the results in the case of g9_2 and $a = 3$. Here we evaluated the following methods. bms: the original Bounded Max-Sum algorithm. bmleximin: a Bounded Max-Sum algorithm whose values and operators are replaced for leximin. lsleximin100/1000: the local search method shown in Subsection 3.4, where the cutoff round is 100 or 1000. ptmaxleximin1/4/8: the approximation method fixing values of variables shown in Subsection 3.2, where the maximum size $maxnsep$ of $|Sep_i|$ is 1, 4 or 8. ptmaxleximin8_ub: the upper bound of 'ptmaxleximin8' that is addressed in Subsection 3.2.

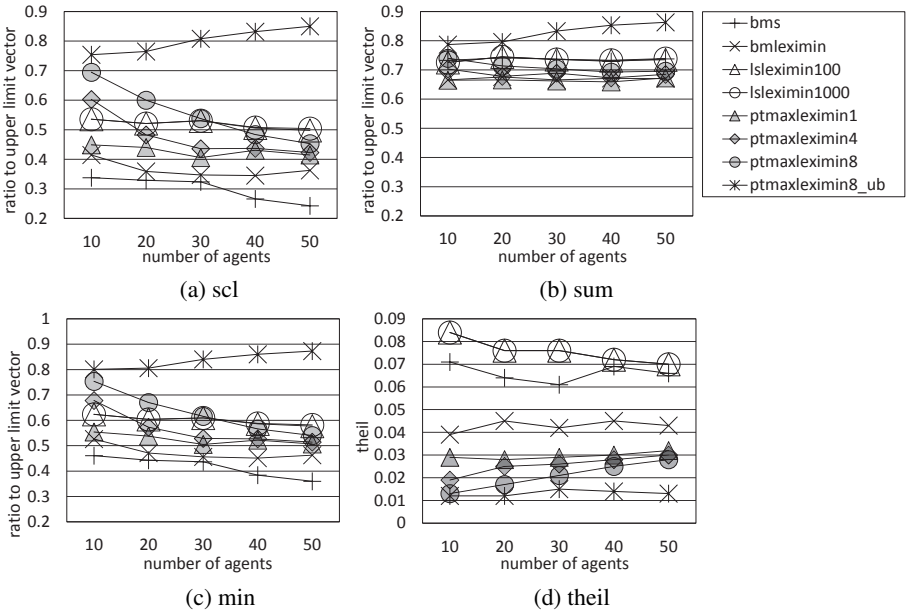
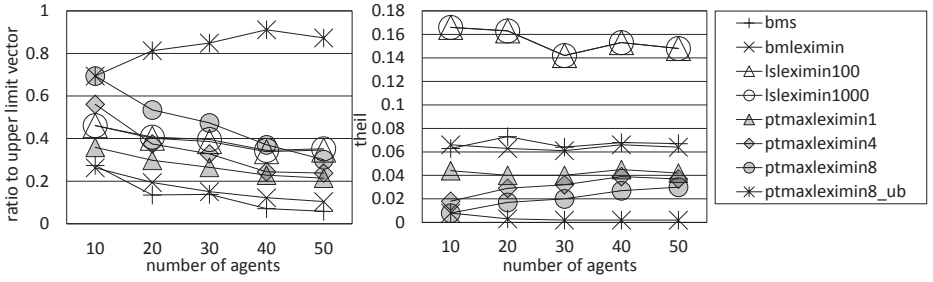


Figure 5. g9_2, a=3

While we also evaluated a local search which employs WTheil, the results resemble that of 'lslexiimin'. It is considered that the both criteria resemble and only work as a threshold in the local search. Therefore, we show the results of 'lsleximin'. Figure 5 (a) shows the result of 'scl'. The values of 'bms' and 'bmleximin' are relatively low, since those algorithms eliminate edges of factor graphs. As a result, actual values of several variables are ignored by other nodes. That decreases the minimum objective value and 'scl'. However, the results of 'bmleximin' are slightly better than that of 'bms'. When the maximum size of $|Sep_i|$ is sufficient, 'ptmaxleximin' is better than other methods. On the other hand, with the number of fixed variables, the quality of solutions decreases. The local search method outperforms 'ptmaxleximin' around thirty agents. Also, the local search method is better than Bounded Max-Sum/Leximin methods. Figures 5 (b) and (c) show the results of 'sum' and 'min'. The results show that 'min' mainly affects the quality of 'scl'. Figure 5 (d) shows the results of Theil index. Even if 'ptmaxleximin' loses the best quality on leximin ordering, it still holds relatively low unfairness. Figure 6 shows the cases of u1-10 and $a = 3$. While the results resemble the cases of g9_2 and $a = 3$, 'ptmaxleximin' is slightly better. It is considered that relatively uniform objective values mitigate the influence of the approximation.



(a) scl (b) theil

Figure 6. u1-10, a=3

While we presented baseline approximation/inexact algorithms for the sake of simplicity, we evaluated the total number of synchronized message cycles and the total number of messages. Note that the current evaluation is not in the main scope of this study. Tables 3 and 4 show the results of 'lslexiimin' and 'ptmaxleximin', respectively. While the approximation method requires relatively large number of cycles, the total number of messages is less than that of the simple local search, where agents basically multicast messages to their neighborhood agents.

Table 3. Total number of cycles/messages (g9_2, n=50, a=3, lsleximin)

cutoff	#converg.	#cyc.	#cyc. in converg.			#msg.	#msg. in converg.		
round			min.	ave.	max.		min.	ave	max.
100	4	457	121	254	436	119903	31590	66249	113990
1000	6	3886	121	372	676	1017571	31590	98736	181640

Table 4. Total number of cycles/messages (g9_2, n=50 (100 nodes), a=3, ptmaxleximin)

lmt.	#cyc.					#msg.				
$\|Sep_i\|$	step1	step2	step3	DP	total	step1	step2	step3	DP	total
1	4302	129	1386	192	6009	6638	3184	2075	297	12195
8	2479	73	936	192	3680	3828	2729	1422	297	8276

* DP includes an extra top-down initiation phase

4.2.3. Approximation method based on mini-buckets

Then, we evaluated the approximation method based on mini-buckets. Figure 7 shows the comparison on approximation methods in the case of g9_2 and $a = 3$. Here the following approximation methods are evaluated. ptmaxleximin8: the approximation method fixing values of variables shown in Subsection 3.2, where the maximum size $maxnsep$ of $|Sep_i|$ is 8. ptmaxleximin_mb3/6: the approximation method based on mini-buckets shown in Subsection 3.3, with $maxnsep = 3$ and 6.

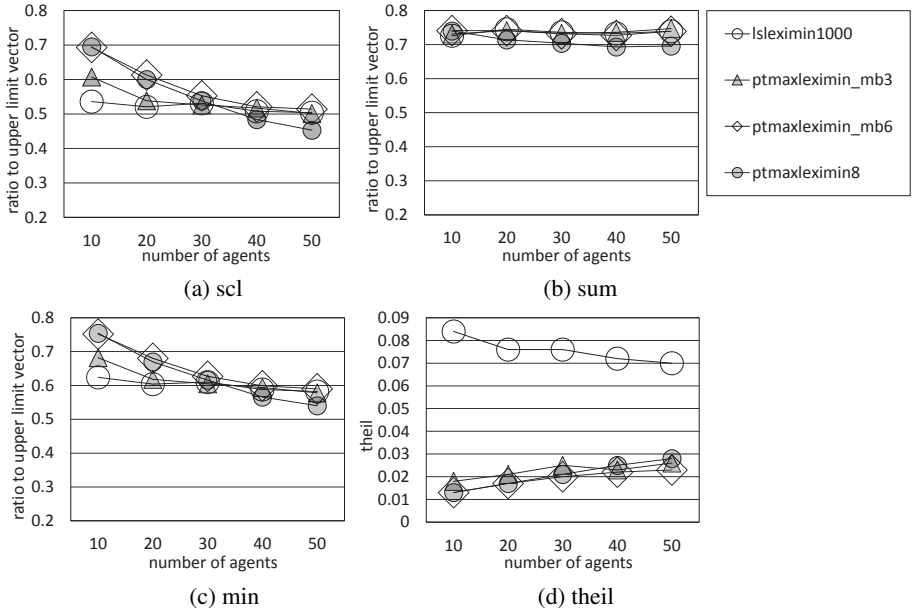
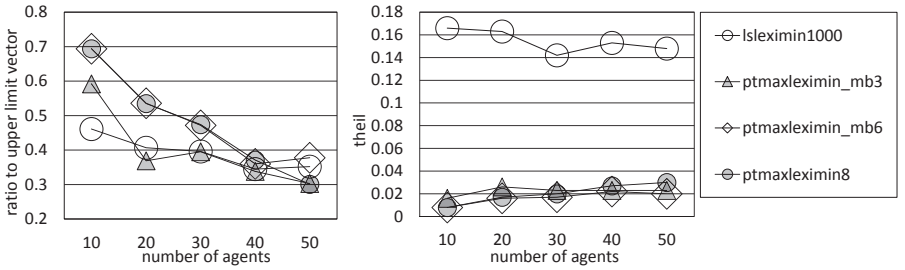


Figure 7. g9_2, a=3, mini-buckets

The results show that, in the cases of larger number of agents, 'scl' and 'min' of mini-buckets methods are relatively better in average. In such cases, 'ptmaxleximin8' fixes the values of large number of variables to eliminate back edges. Since this approach is greedy, the opportunities of better assignments for several functions are also decreased. On the other hand, mini-buckets preserve various combinations of assignments. Therefore, if the sizes of mini-buckets are relatively large, there are the



(a) scl (b) theil

Figure 8. u1-10, a=3, mini-buckets

opportunities to choose better assignments. In particular, 'ptmaxleximin_mb6' is also better in the case of smaller number of agents. Figure 8 shows the comparison on approximation methods in the case of u1-10 and $a = 3$. While the results resemble the case of g9_2 and $a = 3$, 'scl' of mini-buckets methods relatively decreased. It is considered that uniform objective values mitigate the influence of 'ptmaxleximin8' that fixes values of valuables, as addressed in the previous subsection.

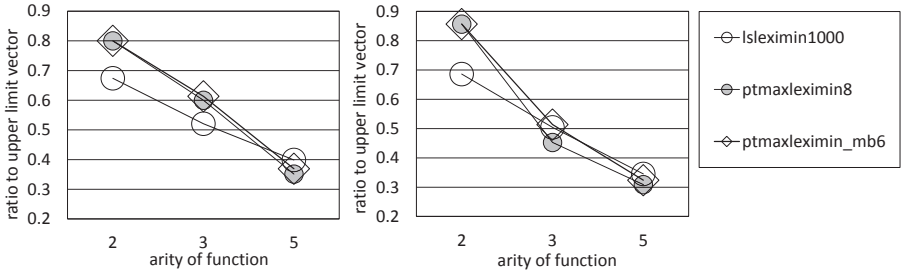
Table 5. Size of function tables ($a = 3$)

n	alg.	max. num. of asg. on reduced separtors	total num. of asg. on all tables	max. num. of tables
40	ptmaxleximin8	6561	77342	
	ptmaxleximin_mb3	27	7886	11.6
	ptmaxleximin_mb6	729	68486	5.7
50	ptmaxleximin8	6561	90578	
	ptmaxleximin_mb3	27	12219	15.0
	ptmaxleximin_mb6	729	100225	7.5

Table 5 shows the results on size of function tables. The maximum number of assignments on reduced separators in third column depends on the $maxnsep$ (8 for 'ptmaxleximin8' and 3/6 for 'ptmaxleximin_mb3/6'). The total number of assignments on all tables in fourth column represents the total size of tables for all function/variable nodes. In the case of 'ptmaxleximin8', each node has a single table. When a value of a variable is fixed to reduce the size of a separator, it also reduces the size of other separators. Therefore, the total size of tables is highly affected by the tables of large size, which are bounded with $maxnsep$. On the other hand, in the cases of 'ptmaxleximin_mb3/6', the size is affected by $maxnsep$, and the number of mini-buckets. Note that several function/variable nodes have multiple mini-buckets of $maxnsep$. The maximum number of tables in fifth column represents the maximum number of mini-buckets for all nodes. There is a negative correlation between $maxnsep$ and the number of mini-buckets.

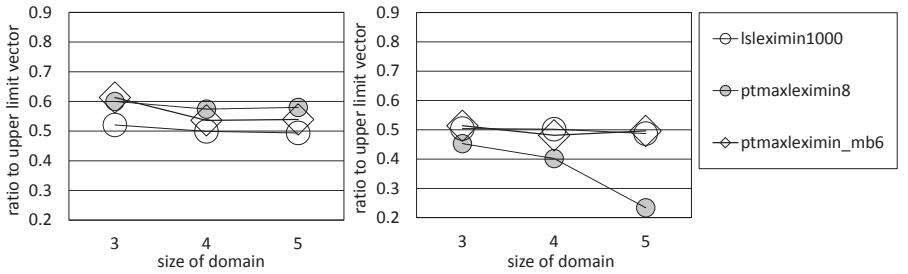
Figure 9 shows the results for different arities. Basically, the quality of solutions decreases with arities. The results for different sizes of domains are shown in Fig. 10. In the case of $n = 20$, the

results of the proposed methods are similar. On the other hand, in the case of larger size problems such as $n = 50$, 'ptmaxlesimin8', which fixes the values of selected variables, was affected by the size of domain.



(a) n=20 (b) n=50

Figure 9. g9_2, a=2-5, scl



(a) n=20 (b) n=50

Figure 10. g9_2, a=3, different sizes of domains, scl

4.2.4. Structured problem domain

Finally, we applied the proposed methods to a structured problem domain. This class of problem is based on a DCOP representation of the Coalition Structure Generation problem [15]. In this problem, an agent decides whether it belongs to one of the groups. If the agent does not belong to any groups, a utility value is given. Otherwise, the agent obtains utility values based on its selection and the relationships among several neighborhood agents. In this case, the problem is defined as a DCOP with additional variables, under the same selection of groups. The additional variables represent states of agents in a group.

We defined an asymmetric DCOP, where utility values are aggregated for each agent. In addition, to emphasize the aim of our experiment, we simplified the original problem so that a cooperation with a neighborhood agent yields a constant utility value. Namely, additional variables in the original definition were integrated with the selection of groups. The definition of the problem 'csg' is as follows.

A problem is based on a DCOP consisting of n variables/agents, n unary functions and r binary functions. A variable takes a value in a set of group names G or a special value '$alone'$. An agent i has a variable x_i that represents a group to which agent i belongs. Depending on x_i, utility values that relate to x_i are defined as follows. For x_j of another agent j, $f^g_{i,j}(x_i, x_j) = grpw_{i,j}$ if $x_i \neq$ '$alone' \wedge x_i = x_j$. Otherwise, $f^g_{i,j}(x_i, x_j) = 0$. $f^a_i(x_i) = alnw_i$ if $x_i =$ '$alone'$. Otherwise, $f^a_i(x_i) = 0$. With these functions, the local problem of agent i is defined as $f_i(X_i) = f^a_i(x_i) + \sum_{j \in Nbr_i} f^g_{i,j}(x_i, x_j)$, where agent i aggregates functions among i and its neighborhood agents in Nbr_i. When $alnw_i$ is a sufficiently large value, agent i prefers '$alone'$. Otherwise, the agent will choose cooperation.

We set the number of groups $|G|$ and the maximum size of arity a for $f_i(X_i)$ to 3 and 4, respectively. Each $grpw_{i,j}$ was randomly set to 1 or 2 with uniform distribution. In addition, while each $alnw_i$ of a half of agents was set to 1, a different value of $alnw_i$ was chosen for all other agents. With the above setting, we evaluated two contrastive cases, where $alnw_i = 1$ or 2, and $alnw_i = 1$ or 8.

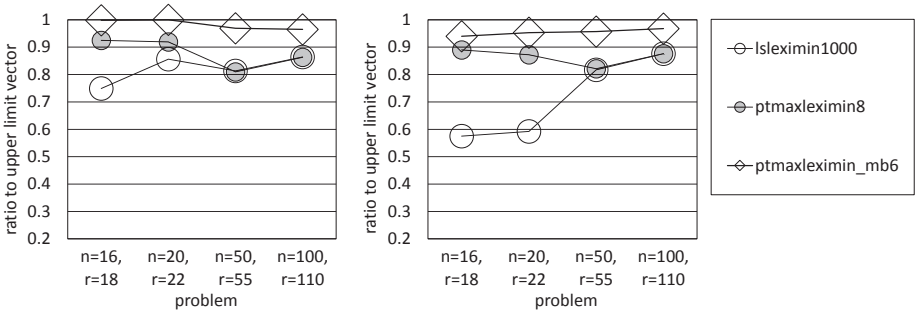
Table 6. Comparison with different optimization criteria (csg, $n = 12$, $r = 16$, $2 \leq a \leq 4$)

prb.	opt. criteria	scl	sum	min	wtheil	theil
$alnw =$	maxwtheil	0.806	0.901	0.790	**0.89**	0.040
1 or 2	ptmaxmin	0.894	0.822	**0.927**	0.814	0.039
$grpw =$	ptmaxminsum	0.907	0.886	**0.927**	0.880	0.036
1 or 2	ptmaxsum	0.775	**0.902**	0.757	0.889	0.042
	ptmaxlexmin	**0.913**	0.884	**0.927**	0.879	**0.034**
$alnw =$	maxwtheil	0.635	0.772	0.640	**0.732**	0.097
1 or 8	ptmaxmin	0.867	0.642	**0.903**	0.642	0.045
$grpw =$	ptmaxminsum	0.870	0.724	**0.903**	0.699	0.079
1 or 2	ptmaxsum	0.366	**0.798**	0.370	0.695	0.185
	ptmaxlexmin	**0.879**	0.695	**0.903**	0.695	**0.044**

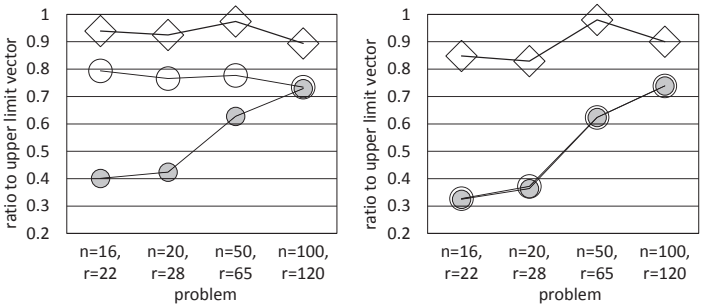
* Problems were solved by exact algorithms.
* scl, sum, min and wtheil are ratio values to the upper limit vector.
* To be maximized: scl, sum, min, wtheil. To be minimized: theil.

Table 6 shows the results of exact solution methods in the case of $n = 12$, $r = 16$. The results resemble ones shown in Table. 1. In the case of $alnw_i = 1$ or 8, the difference between 'ptmaxsum' and 'ptmaxlexmin' were significant in 'scl' and 'min'.

Figure 11 shows the results of approximation methods for lexmin criterion. 'ptmaxleximin_mb6', which employs mini-buckets approach, was relatively better in all cases. On the other hand, 'ptmaxleximin8', which fixes values of selected variables, was ineffective for relatively small and dense problems as shown in Figs. 11 (c) and (d). While local search 'lslexmin1000' was relatively effective in cooperative cases $alnw_i = 1$ or 2, its solution quality was less effective in selfish cases $alnw_i = 1$ or 8. These results reveal that the greedy heuristic and the local search were influenced by the structures and densities of the problems. The drawbacks of the both methods were emphasized in the cases of small numbers of agents. A possible reason is that the sparseness of the problems



(a) $alnw = 1$ or 2, relatively sparse (b) $alnw = 1$ or 8, relatively sparse



(c) $alnw = 1$ or 2, relatively dense (d) $alnw = 1$ or 8, relatively dense

Figure 11. csg, $2 \leq a \leq 4$, $grpw = 1$ or 2, scl

is correlated to the number of agents in these settings. The results show that the greedy heuristic is affected by the relatively high density due to its approximation strategy. On the other hand, the local search is affected by relatively *deep* local optimal solutions in the case of $alnw_i = 1$ or 8.

5. Related works and discussions

Pseudo trees on factor graphs of DCOPs have been proposed in [18]. While the previous study addresses the conventional DCOPs on factor graphs, we employ the factor graphs to eliminate the directions of edges in the cases of Asymmetric DCOPs on constraint graphs [16, 11, 9]. As a result, the obtained solution methods do not directly handle the direction of edges that was necessary in the previous studies. In addition, the factor graph represents n-ary functions. In Subsection 3.3, we proposed a mini-bucket method for pseudo trees on factor graphs. That approach can also be applied to the conventional DCOPs by replacing objective vectors and operators.

The Max-Sum algorithm is a logarithmic representation of the Max-Product algorithm, which is derived from Belief Propagation on factor graphs. Belief Propagation have been applied to graphical models such as Bayesian networks and Markov Random Fields. The Max-Sum algorithm has been applied to DCOPs in [3]. Since this algorithm is inexact solution method except in the case where a factor graph is a tree, cyclic graphs are modified to trees in the previous studies [19, 18]. We employed a pseudo tree approach which is based on a tree decomposition.

Asymmetric DCOPs on factor graphs have been presented in [10]. In that work, several solution methods based on the Max-Sum algorithm have been proposed for the original Asymmetric DCOPs, where the summation of utilities among agents is optimized. On the other hand, we address the different classes of problems with leximin criterion.

Theil based social welfare WTheil has been proposed in [13]. However, that social welfare cannot be decomposed into dynamic programming. We evaluated exact algorithms that optimize leximin and WTheil. In our experiment, the result shows that leximin is better than WTheil on the criteria of leximin, maximin and Theil Index.

This study is based on the previous work of Leximin AMODCOP [16]. The correctness of our proposed method based on a dynamic programming is similar to the solution method on constraint graphs in the previous study as shown in Proposition 3.1. In the previous study, an efficient method that compresses objective vectors using run-length encoding. While we did not address the compression technique for simplicity, similar methods can be employed in our proposed methods. We experimentally evaluated several approximate/inexact optimization methods on leximin ordering. The results reveal that a difficulty in such incomplete methods is the preservation of the minimum objective value, which directly affects the leximin. The incomplete methods that well handle the minimum objective value can be an important issue, even if the optimization on maximin captures neither inequality nor Pareto optimality.

While we addressed fairness among agents in a class of asymmetric and multiple objective DCOPs, similar approaches may be applied to solve the problems in multiagent planning [24, 25, 26]. In several phases of the multiagent planning, agents cooperatively solve subproblems to allocate goals and tasks, or to resolve consistencies/conflicts among individual plans. Several studies have addressed DCOPs and multiagent planning [27, 28]. In the situation where the fairness among individual plans of selfish agents is important, such problems may be defined using several representations similar to Asymmetric DCOPs and various criteria of fairness.

6. Conclusions

We proposed the solution methods for the leximin multiple objective optimization problems with preferences of agents and employing factor graphs. We presented a complete solution method based on a dynamic programming method on factor graphs. Then several approximation methods and a local search method were also presented to address the issue of combinational explosion in the complete solution method. The experimental results show the influences brought by the approximation/inexact methods on the leximin social welfare and factor graphs.

In the case of exact solution methods, the optimization based on the leximin criterion has significantly improved the minimum values and the equality on both random and structured problems. Approximation methods and a local search method were firstly evaluated on random problems. The solution quality of approximation on leximin criterion decreased with the loss of information of problems, while the local search was relatively stable. In addition, as we addressed in Section 5, these results revealed that the solution quality was mainly affected by the minimum objective value, in comparison to the summation value and the inequality. It can be considered that the approximation of 'maximin' should be more focused.

In the case of structured problems, the solution quality of the approximation methods and the local search were different with the settings. The results show that the heuristic, which eliminates assignments to selected variables, may be too greedy for some problem structures. Similarly, several patterns of local minimum solutions may affect the local search. On the other hand, the approximation method with a mini-buckets approach was relatively better than other methods. It can be considered that the mini-buckets approach also leaves a diversity of solutions in the case of the leximin criterion. We believe that these results reveal several important issues for distributed optimization problems with equality/inequality criteria.

The proposed approach will be applied to several distributed resource allocation problems such as distributed sensor networks, where sensor resources are cooperatively allocated to targets. In such a case, a sensor agent will perform as a representative of a target within its observation range, and will cooperate with other sensor agents to allocate resources for targets considering the fairness of observation quality among targets. Other possible tasks are exclusive resource allocation problems such as wireless communication channels/slots between autonomous radio stations, which serve as access points of their neighborhood areas. The fair resource allocation on these applications is obviously important. Since the proposed approximation methods are influenced by the size of separators and variables' domains, dedicated designs of DCOPs and additional methods will be necessary for practical problems. On the other hand, the proposed approach is promising for distributed resource allocation systems on tree-like networks.

Our future research directions will include improvements of solution quality including the minimum objective value, detailed evaluations on different criteria of fairness, and application of the proposed method to practical problems instances.

Acknowledgments: This work was supported in part by KAKENHI Grant-in-Aid for Scientific Research (C), 25330257 and 16K00301.

References

[1] Modi PJ, Shen W, Tambe M, Yokoo M. Adopt: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence, 2005;161(1-2):149–180. https://doi.org/10.1016/j.artint.2004.09.003.

[2] Petcu A, Faltings B. A Scalable Method for Multiagent Constraint Optimization. In: 19th International Joint Conference on Artificial Intelligence. 2005 pp. 266–271. http://dl.acm.org/citation.cfm?id=1642293.1642336.

[3] Farinelli A, Rogers A, Petcu A, Jennings NR. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: 7th International Joint Conference on Autonomous Agents and Multiagent Systems. 2008 pp. 639–646. ISBN: 978-0-9817381-1-6.

[4] Zhang W, Wang G, Xing Z, Wittenburg L. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. Artificial Intelligence, 2005;161(1-2):55–87. https://doi.org/10.1016/j.artint.2004.10.004.

[5] Miller S, Ramchurn SD, Rogers A. Optimal decentralised dispatch of embedded generation in the smart grid. In: 11th International Conference on Autonomous Agents and Multiagent Systems, volume 1. 2012 pp. 281–288. ISBN: 0-9817381-1-7, 978-0-9817381-1-6.

[6] Ramchurn SD, Farinelli A, Macarthur KS, Jennings NR. Decentralized Coordination in RoboCup Rescue. 2010;53(9):1447–1461. https://doi.org/10.1093/comjnl/bxq022.

[7] Delle Fave FM, Stranders R, Rogers A, Jennings NR. Bounded decentralised coordination over multiple objectives. In: 10th International Conference on Autonomous Agents and Multiagent Systems, volume 1. 2011 pp. 371–378. ISBN: 0-9826571-5-3, 978-0-9826571-5-7.

[8] Matsui T, Silaghi M, Hirayama K, Yokoo M, Matsuo H. Distributed Search Method with Bounded Cost Vectors on Multiple Objective DCOPs. In: Principles and Practice of Multi-Agent Systems - 15th International Conference. 2012 pp. 137–152. https://doi.org/10.1007/978-3-642-32729-2_10.

[9] Grinshpoun T, Grubshtein A, Zivan R, Netzer A, Meisels A. Asymmetric Distributed Constraint Optimization Problems. Journal of Artificial Intelligence Research, 2013;47(1):613–647. http://dl.acm.org/citation.cfm?id=2566972.2566988.

[10] Zivan R, Parash T, Naveh Y. Applying Max-sum to Asymmetric Distributed Constraint Optimization. In: 24th International Joint Conference on Artificial Intelligence. 2015 pp. 432–438. ISBN: 978-1-57735-738-4.

[11] Netzer A, Meisels A. Distributed Envy Minimization for Resource Allocation. In: 5th International Conference on Agents and Artificial Intelligence, volume 1. 2013 pp. 15–24.

[12] Netzer A, Meisels A. Distributed Local Search for Minimizing Envy. In: 2013 IEEE/WIC/ACM International Conference on Intelligent Agent Technology. 2013 pp. 53–58. doi:10.1109/WI-IAT.2013.90.

[13] Netzer A, Meisels A. SOCIAL DCOP - Social Choice in Distributed Constraints Optimization. In: 5th International Symposium on Intelligent Distributed Computing. 2011 pp. 35–47. https://doi.org/10.1007/978-3-642-24013-3_5.

[14] Matsui T, Matsuo H. 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. 2012 pp. 25–32. doi:10.1109/WI-IAT.2012.22.

[15] Ueda S, Iwasaki A, Yokoo M, Silaghi MC, Hirayama K, Matsui T. Coalition Structure Generation Based on Distributed Constraint Optimization. In: 24th AAAI Conference on Artificial Intelligence. 2010 pp. 197–203. http://dblp.uni-trier.de/db/conf/aaai/aaai2010.html#UedaIYSHM10.

[16] Matsui T, Silaghi M, Hirayama K, Yokoo M, Matsuo H. Leximin Multiple Objective Optimization for Preferences of Agents. In: 17th International Conference on Principles and Practice of Multi-Agent Systems. 2014 pp. 423–438. https://doi.org/10.1007/978-3-319-13191-7_34.

[17] Matsui T, Silaghi M, Okimoto T, Hirayama K, Yokoo M, Matsuo H. Leximin Asymmetric Multiple Objective DCOP on Factor Graph. In: 18th International Conference on Principles and Practice of Multi-Agent Systems. 2015 pp. 134–151. https://doi.org/10.1007/978-3-319-25524-8_9.

[18] Matsui T, Matsuo H. Complete Distributed Search Algorithm for Cyclic Factor Graphs. In: 6th International Conference on Agents and Artificial Intelligence. 2014 pp. 184–192.

[19] Rogers A, Farinelli A, Stranders R, Jennings NR. Bounded Approximate Decentralised Coordination via the Max-Sum Algorithm. Artificial Intelligence, 2011;175(2):730–759. doi:10.1016/j.artint.2010.11.001.

[20] Sen AK. Choice, Welfare and Measurement. Harvard University Press, 1997.

[21] Marler RT, Arora JS. Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization, 2004;26:369–395. https://doi.org/10.1007/s00158-003-0368-6.

[22] Zivan R, Peled H. Max/Min-sum Distributed Constraint Optimization Through Value Propagation on an Alternating DAG. In: 11th International Conference on Autonomous Agents and Multiagent Systems. 2012 pp. 265–272. ISBN:0-9817381-1-7, 978-0-9817381-1-6.

[23] Dechter R. Mini-buckets: A General Scheme for Generating Approximations in Automated Reasoning. In: 15th International Joint Conference on Artifical Intelligence, volume 2. 1997, pp. 1297–1302. ISBN: 1-555860-480-4.

[24] de Weerdt M, Clement B. Introduction to Planning in Multiagent Systems. Multiagent and Grid Systems - Planning in multiagent systems. 2009;5(4): 345–355. http://dl.acm.org/citation.cfm?id=1735317.1735318.

[25] Weiss G (ed.). Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, 1999.

[26] Weiss G (ed.). Multiagent Systems, Second Edition. MIT Press, 2013. ISBN: 9780262018890.

[27] Cox JS, Durfee EH, Bartold T. A Distributed Framework for Solving the Multiagent Plan Coordination Problem. In: rth International Joint Conference on Autonomous Agents and Multiagent Systems. 2005 pp. 821–827. ISBN: 1-59593-093-0.

[28] Nissim R, Brafman RI, Domshlak C. A General, Fully Distributed Multi-agent Planning Algorithm. In: 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1. 2010 pp. 1323–1330. ISBN: 978-0-9826571-1-9.