# Leximin Asymmetric Multiple Objective DCOP on Factor Graph

Toshihiro Matsui[1], Marius Silaghi[2], Tenda Okimoto[3], Katsutoshi Hirayama[3], Makoto Yokoo[4], and Hiroshi Matsuo[1]

[1] Nagoya Institute of Technology, Gokiso-cho Showa-ku Nagoya 466-8555, Japan
{matsui.t, matsuo}@nitech.ac.jp
[2] Florida Institute of Technology, Melbourne FL 32901, United States of America
msilaghi@fit.edu
[3] Kobe University, 5-1-1 Fukaeminami-machi Higashinada-ku Kobe 658-0022, Japan
{tenda, hirayama}@maritime.kobe-u.ac.jp
[4] Kyushu University, 744 Motooka Nishi-ku Fukuoka 819-0395, Japan
yokoo@is.kyushu-u.ac.jp

**Abstract.** Leximin AMODCOP has been proposed as a class of Multiple Objective Distributed Constraint Optimization Problems, where multiple objectives for individual agents are optimized based on the leximin operator. This problem also relates to Asymmetric DCOPs with the criteria of fairness among agents, which is an important requirement in practical resource allocation tasks. 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 inexact algorithms.

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

## 1 Introduction

Multiple Objective Distributed Constraint Optimization Problems (MODCOPs) [1, 7] have been studied as an extension to DCOPs [8, 12, 2, 16]. With MODCOPs, agents cooperatively solve multiple objective problems. As a class of MODCOPs, Leximin AMODCOP, where multiple objectives for individual agents are optimized based on the leximin operator, has been proposed [15]. This problem also relates to Asymmetric DCOPs with a criteria of fairness among agents [10, 11, 3]. The fairness among agents is an important requirement in practical resource allocation tasks [9, 5, 10, 11]. For example, in a smart grid, autonomous consumers should share power resource without unfairness on their preferences considering rlationship among them. Leximin is a well-known egalitarian social welfare that represents the fairness/unfairness among agents. Since maximization based on leximin ordering improves equality among agents, the Leximin AMODCOP is considered as a fundamental class of DCOPs.

The previous study [15] 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 [3, 9]. 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.

In this paper, we evaluate several solution methods to Leximin AMODCOPs on factor graphs. A dynamic programming method on factor graphs is investigated as an exact/approximation solution method in conjunction with other inexact algorithms also applied to the factor graphs.

## 2 Preliminary

### 2.1 DCOP

In the following, we present preliminaries of our study. Several definitions and notations are inherited from the previous literatures [15, 6].

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

**Definition 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 $f_m(x_{m0}, \cdots, x_{mk}) : D_{m0} \times \cdots \times D_{mk} \rightarrow \mathbb{N}_0$, 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: $F(X) = \sum_m$ **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 [2] 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 [2] 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 [2] 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 [13], 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 [12].

### 2.3 Multiple Objective DCOP for Preferences of Agents

**Multiple Objective DCOPs** Multiple objective DCOP [1] (MODCOP) is a generalization of the DCOP framework. 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 the *objective vector*.

**Definition 2 (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 [14, 4]. Multiple objective problems generally have a set of Pareto optimal solutions that form a Pareto front.

**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 [14] and scalarization methods [4]. 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 [14, 4] for the details of above criteria. The study in [9] 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 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* $\exists t, \forall t' < t, v_{t'} = v'_{t'} \wedge v_t < v'_t$*.*

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

**Leximin Asymmetric MODCOP on preferences of agents** Leximin Asymmetric MODCOP (Leximin AMODCOP) [15] 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 [5, 9–11, 3]. 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 [15].

**Definition 5 (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 1. Agent* $i \in A$ *has its local problem defined on* $X_i \subseteq X$*.* $\exists (i, j)$ *s.t.* $i \neq$

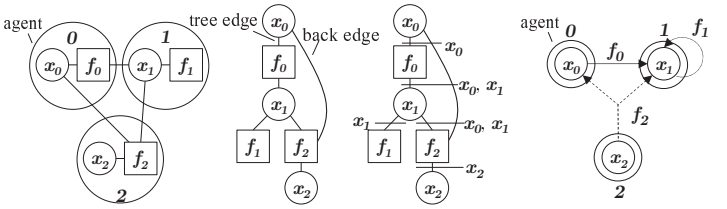


**Fig. 1.** AMODCOP on factor graph

$j, X_i \cap X_j \neq \emptyset$. *$F$ is a set of objective functions $f_i(X_i)$ for all $i \in A$. The function $f_i(X_i) : D_{i_0} \times \cdots \times D_{i_k} \to \mathbb{R}$ 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.*

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. Also, the addition of objective values is extended as a concatenation operation of objective values. The 'addition' of sorted vectors is defined as follows [15].

**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.*

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 [15].

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 $f_2$.

For cyclic factor graphs, the traditional Max-Sum algorithm is inexact. Namely, an objective value is redundantly aggregated via different paths [2, 17]. 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 [13]. 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 [8, 12] to decompose problems on constraint graphs. On the other hand, there are a few similar studies for factor graphs [6]. We employ a solution method based on pseudo trees on factor graphs [5].

**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 depth.
- $prnt_i$: the parent node of $i$.
- $Chld_i$: the set of $i$'s child nodes.
- $Sep_i$ 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 that $i$ has to consider in addition to the separators.
- $Seph^i_j$: 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.

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

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

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

[5] While the previous study employs cross-edge pseudo trees and a search algorithm [6], 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 [6].

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

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 thier own $Seph_i^i$ as $Nbrh_i$ (Eq. (5)). For other nodes $k$ in separators $Sep_i$, node $i$ sets $Seph_k^i$ aggregating $Seph_k^j$ of child nodes $j$ (Eq. (6)).

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

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

$$\overline{Sep_i} = \{l \mid l \in Nbrl_i,\ Seph_l'^i = \{\ \}\} \quad (8)$$

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

$$Seph_k^i = Seph_k'^i, \text{ where } k \in Sep_i. \quad (10)$$

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 $Seph_k^j$. Then, $i$ is eliminated if $k$ is $i$'s neighborhood variable (Eq. (9)). After $Sep_i$ is set, $Seph_k'^i$ is also used to set $Seph_k^i$ 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 $Seph_{x_1}^{f_1} = \{\ \}$. For $i = x_2$, $Sep_{x_2} = \{x_2\}$, $\overline{Sep_{x_2}} = \{\ \}$ and $Seph_{x_2}^{x_2} = \{f_2\}$. For $i = f_2$, $Seph_{x_2}'^{f_2} = \{\ \}$, $\overline{Sep_{f_2}} = \{x_2\}$, $Sep_{f_2} = \{x_0, x_1\}$, $Seph_{x_0}^{f_2} = \{\ \}$ and $Seph_{x_1}^{f_2} = \{\ \}$. For $i = x_1$, $Sep_{x_1} = \{x_0, x_1\}$, $\overline{Sep_{x_1}} = \{\ \}$, $Seph_{x_1}^{x_1} = \{f_0\}$ and $Seph_{x_0}^{x_1} = \{\ \}$. Similar computations are performed for the other nodes.

**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.

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

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

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 $i$.

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 $Chld_i$.

This solution method inherits most parts of the correctness and the time/space complexity from conventional methods based on dynamic programming such as DPOP [12] and Bounded Max-Sum [13]. 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 $O(\log n)$ [15].

### 3.2 Approximation method

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 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. 2(a)).
- (Step 2) selection/fixation of the variable of the largest impact in the separators (Fig. 2(b)).
- (Step 3) notification of the fixed variable (Fig. 2(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

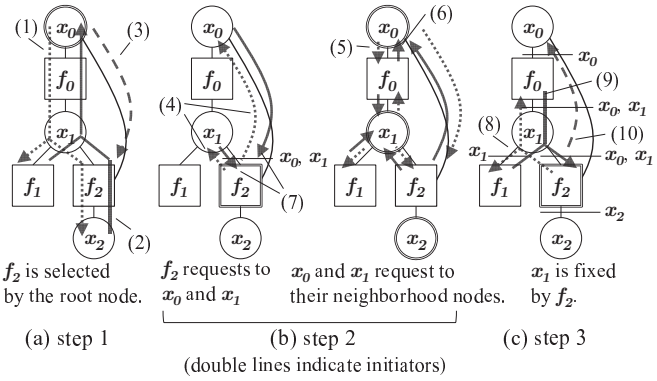


**Fig. 2.** Flow of approximation

(Fig. 2(1)). The information of the number of separators is then aggregated in a bottom-up manner (Fig. 2(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. 2(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. 2(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. 2(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. 2(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. 2(7)). Now, node $j$ determine 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, $\infty$ 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. 2(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. 2(9)). Then, node $j$ notifies the root node of the termination of a round (Fig. 2(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

Preprocessing:
let $Nbr^-_{x_i}$ denote ($Nbr_i$ of $x_i$) $\setminus \{f_i\}$. let $Nbr^-_{f_i}$ denote ($Nbr_i$ of $f_i$) $\setminus \{x_i\}$.
$ANbr_{x_i} \leftarrow \bigcup_j$(the owner agent of $f_j$ in $Nbr^-_{x_i}$).
$ANbr_{f_i} \leftarrow \bigcup_j$(the owner agent of $x_j$ in $Nbr^-_{f_i}$). $ANbr_i \leftarrow ANbr_{x_i} \cup ANbr_{f_i}$.
send $ANbr_i$ to $j$ in $ANbr_i$. receive $ANbr_j$ from all $j$ in $ANbr_i$.
$BANbr_i \leftarrow ANbr_i \cup \bigcup_{j \in ANbr_i} ANbr_j$.

Main procedure:
choose the initial assignment $d_i^{cur}$ of $x_i$. *// locally maximize* $f_i$.
**until**(cutoff){
send $d_i^{cur}$ to all agents $j$ in $ANbr_{x_i}$. receive $d_j^{cur}$ from all agents $j$ in $ANbr_{f_i}$.
$\mathcal{A}_i^{cur} \leftarrow \{(x_i, d_i^{cur})\} \cup \bigcup_{x_j \text{ in } Nbr^-_{f_i}} (x_j, d_j^{cur})$.
$v_i^{cur} \leftarrow f_i(\mathcal{A}_i^{cur})$. send $v_i^{cur}$ to agents $j$ in $ANbr_i$. receive $v_j^{cur}$ from agents $j$ in $ANbr_i$.
$\mathbf{v}_i^{cur} \leftarrow \{v_i^{cur}\} \oplus \bigoplus_{j \text{ in } ANbr_i} \{v_j^{cur}\}$.
choose the new assignment $d_i^{new}$ under $\mathcal{A}_i^{cur} \setminus \{(x_i, d_i^{cur})\}$.
$\mathcal{A}_i^{new} \leftarrow \{(x_i, d_i^{new})\} \cup \bigcup_{x_j \text{ in } Nbr^-_{f_i}} (x_j, d_j^{cur})$. $v_i^{new} \leftarrow f_i(\mathcal{A}_i^{new})$.
send $d_i^{new}$ to all agents $j$ in $ANbr_{x_i}$. receive $d_j^{new}$ from all agents $j$ in $ANbr_{f_i}$.
**foreach**($x_k$ in $Nbr^-_{f_i}$){
$\mathcal{A}_{i,k}^{new} \leftarrow \{(x_i, d_i^{cur})\} \cup (x_k, d_k^{new}) \cup \bigcup_{x_j \text{ in } Nbr^-_{f_i} \setminus \{x_k\}} (x_j, d_j^{cur})$. $v_{i,k}^{new} \leftarrow f_i(\mathcal{A}_{i,k}^{new})$.
send $v_{i,k}^{new}$ to the owner agent of $x_k$.
}
receive $v_j^{new}$ from all agents $j$ in $ANbr_{x_i}$.
$\mathbf{v}_i^{new} \leftarrow \{v_i^{new}\} \oplus \bigoplus_{j \text{ in } ANbr_{x_i}} \{v_j^{new}\} \oplus \bigoplus_{k \text{ in } ANbr_i \setminus ANbr_{x_i}} \{v_k^{cur}\}$.
**if**($\mathbf{v}_i^{cur} \prec_{leximin} \mathbf{v}_i^{new}$){ $v_i^{dif} \leftarrow \max(0, v_i^{new} - v_i^{cur})$. }**else**{ $v_i^{dif} \leftarrow 0$. }
send $v_i^{dif}$ to all agents $j$ in $BANbr_i$. receive $v_j^{dif}$ from all agents $j$ in $BANbr_i$.
**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.*
}

**Fig. 3.** local search (procedures of node $i$)

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 Local search

Another inexact approach is based on local search methods. Here we employ a local search method from a previous study [9]. 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 $Nbr_i$ 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. 3, lines 1-6).

After the initialization (Fig. 3, 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

**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 [13]. 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 $i$.

**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 $k^{\text{th}}$ 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($\mathbf{v}$).

**Social welfare based on Theil Index (wtheil/theil)** In a previous study [9], 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.

**Bounded Max-Sum algorithm** As addressed in Subsection 2.3, 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

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 [5]. 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 a improved version of '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

**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.

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. Therefore, we did not compared exact methods and approximate methods.

Next, we evaluated approximate methods and local search methods. Figures 4-7 show 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 3.3, where the cutoff round is 100 or 1000. ptmaxleximin1/4/8: the approximation method shown in Subsection 3.2, where the maximum size of $|Sep_i|$ is 1, 4 or 8. ptmaxleximin8_ub: the upper bound of 'ptmaxleximin8' that is addressed in Subsection 3.2. While we also evaluated a local search which employs WTheil, the results resemble that of 'lsleximin'. 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 4 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 and 6 show the results of 'sum' and 'min'. The results show that 'min' mainly affects the quality of 'scl'. Figure 7 shows the results of Theil index. Even if 'ptmaxleximin' loses the best quality on leximin ordering, it still holds relatively low unfairness.

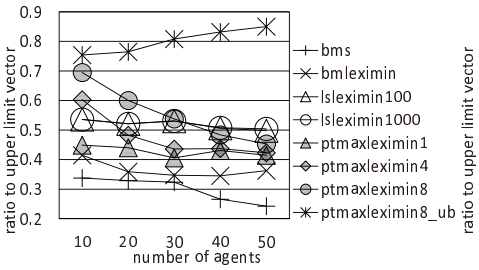


**Fig. 4.** scl (g9_2, a=3)

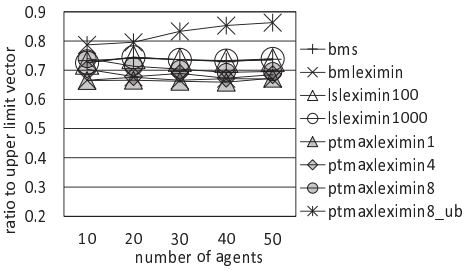


**Fig. 5.** sum (g9_2, a=3)

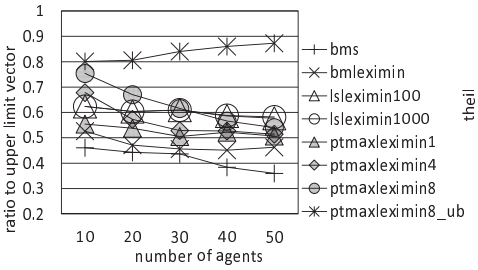


**Fig. 6.** min (g9_2, a=3)

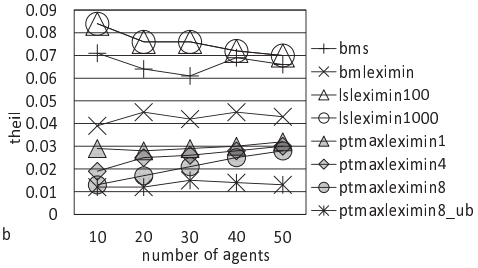


**Fig. 7.** theil (g9_2, a=3)

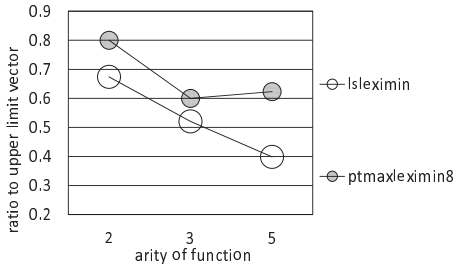


**Fig. 8.** scl (g9_2, n=20)

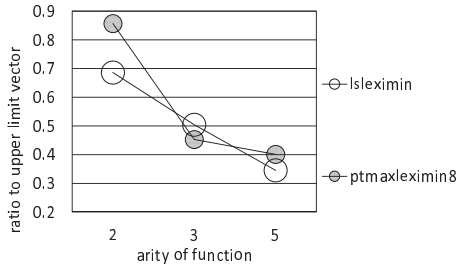


**Fig. 9.** scl (g9_2, n=50)

Figures 8 and 9 show the results for different arities. Basically, the quality of solutions decreases with arities. On the other hand, the influence on 'ptmaxleximin' is not monotonic in the case of $n = 20$. It is considered that the heuristic of approximation is affected both of arity and the number of nodes. Figures 10 and 11 show 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.

While we presented base line approximation 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 'lsleximin' and 'ptmaxleximin', respectively. While the approximation method requires relatively large number of cycles, the total number of messages is less than that of local search, where agents basically multicast messages to their neighborhood agents.

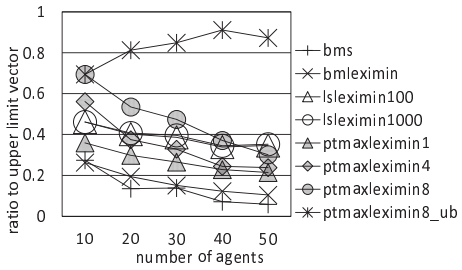


**Fig. 10.** scl (u1-10, a=3)

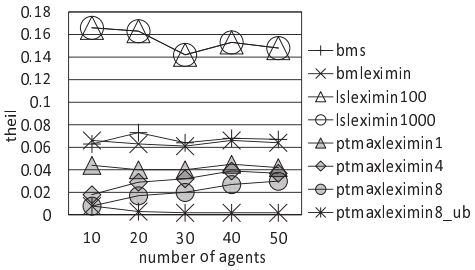


**Fig. 11.** theil (u1-10, a=3)

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

| cutoff round | #converg. | #cyc. | #cyc. in converg. | | | #msg. | #msg. in converg. | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | 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, a=3, ptmaxleximin)

| lmt. | #cyc. | | | | | #msg. | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\|Sep_i\|$ | step1 | step2 | step3 | DP | total | step1 | step2 | step3 | DP | total |
| 1 | 4303 | 129 | 1386 | 128 | 5946 | 6639 | 3184 | 2075 | 198 | 12097 |
| 8 | 2480 | 73 | 936 | 128 | 3617 | 3829 | 2729 | 1422 | 198 | 8178 |

## 5 Related Works and Discussions

While pseudo trees on factor graph have been proposed in [6], we employ the factor graphs to eliminate the directions of edges in the cases of Asymmetric DCOPs on constraint graphs [3]. As a result, the obtained solution methods do not handle the direction of edges that was necessary in the previous studies [3, 10]. In addition, the factor graph directly represents n-ary functions.

Theil based social welfare WTheil has been proposed in [9]. However, that social welfare cannot be decomposed into dynamic programming. We evaluated exact algorithms that optimize leximin and WTheil, relatively. In our experiment, the result shows that leximin is better than WTheil on the criteria of leximin, maximin and Theil Index.

## 6 Conclusions

We propose solution methods for the leximin multiple objective optimization problems with preferences of agents and employing factor graphs. A dynamic programming method on factor graphs is employed as an exact/approximation solution method in conjunction with other inexact algorithms also applied to factor graphs. The experimental results show the influences brought by the approximation method on the leximin social welfare and factor graphs.

Our future research directions will include improvement of solution quality, detailed evaluations of different criteria of fairness, and application of the proposed method to practical problems instances.

## References


1. Delle Fave, F.M., Stranders, R., Rogers, A., Jennings, N.R.: Bounded decentralised coordination over multiple objectives. In: 10th International Conference on Autonomous Agents and Multiagent Systems. vol. 1, pp. 371–378 (2011)
2. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: 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 (2008)
3. Grinshpoun, T., Grubshtein, A., Zivan, R., Netzer, A., Meisels, A.: Asymmetric distributed constraint optimization problems. Journal of Artificial Intelligence Research 47, 613–647 (2013)
4. Marler, R.T., Arora, J.S.: Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization 26, 369–395 (2004)
5. 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. vol. 2, pp. 25–32 (2012)
6. Matsui, T., Matsuo, H.: Complete distributed search algorithm for cyclic factor graphs. In: 6th International Conference on Agents and Artificial Intelligence. pp. 184–192 (2014)
7. 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. pp. 137–152 (2012)
8. Modi, P.J., Shen, W., Tambe, M., Yokoo, M.: Adopt: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence 161(1-2), 149–180 (2005)
9. Netzer, A., Meisels, A.: SOCIAL DCOP - Social Choice in Distributed Constraints Optimization. In: 5th International Symposium on Intelligent Distributed Computing. pp. 35–47 (2011)
10. Netzer, A., Meisels, A.: Distributed Envy Minimization for Resource Allocation. In: 5th International Conference on Agents and Artificial Intelligence. vol. 1, pp. 15–24 (2013)
11. Netzer, A., Meisels, A.: Distributed Local Search for Minimizing Envy. In: 2013 IEEE/WIC/ACM International Conference on Intelligent Agent Technology. pp. 53–58 (2013)
12. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: 19th International Joint Conference on Artificial Intelligence. pp. 266–271 (2005)
13. Rogers, A., Farinelli, A., Stranders, R., Jennings, N.R.: Bounded approximate decentralised coordination via the Max-Sum algorithm. Artificial Intelligence 175(2), 730–759 (2011)
14. Sen, A.K.: Choice, Welfare and Measurement. Harvard University Press (1997)
15. Toshihiro Matsui, Marius Silaghi, K.H.M.Y., Matsuo, H.: Leximin multiple objective optimization for preferences of agents. In: 17th International Conference on Principles and Practice of Multi-Agent Systems
16. Zivan, R.: Anytime local search for distributed constraint optimization. In: Twenty-Third AAAI Conference on Artificial Intelligence. pp. 393–398 (2008)
17. 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. pp. 265–272 (2012)