A Study on Cooperative Action Selection Considering Unfairness in Decentralized Multiagent Reinforcement Learning

Toshihiro Matsui and Hiroshi Matsuo
Nagoya Institute of Technology, Gokisyo-cho, Showa-ku, Nagoya, Aichi, 466-8555, Japan

Keywords: Multiagent System, Reinforcement Learning, Distributed Constraint Optimization, Unfairness, Leximin.

Abstract: Reinforcement learning has been studied for cooperative learning and optimization methods in multiagent systems. In several frameworks of multiagent reinforcement learning, the system's whole problem is decomposed into local problems for agents. To choose an appropriate cooperative action, the agents perform an optimization method that can be performed in a distributed manner. While the conventional goal of the learning is the maximization of the total rewards among agents, in practical resource allocation problems, unfairness among agents is critical. In several recent studies of decentralized optimization methods, unfairness was considered a criterion. We address an action selection method based on leximin criteria, which reduces the unfairness among agents, in decentralized reinforcement learning. We experimentally evaluated the effects and influences of the proposed approach on classes of sensor network problems.

1 INTRODUCTION

Reinforcement learning has been studied as cooperative learning and optimization methods in multiagent systems (Hu and Wellman, 2003; Zhang and Lesser, 2011; Nguyen et al., 2014). In several frameworks of multiagent reinforcement learning, the system's entire problem is decomposed into local problems for agents. To choose an appropriate cooperative action, the agents perform an optimization method that can be performed in a distributed manner.

A class of networked distributed POMDPs was defined for a sensor network domain (Zhang and Lesser, 2011), and in the reinforcement learning for such problems, the joint actions of agents are selected using the max-sum algorithm (Farinelli et al., 2008), which is a solution method for distributed constraint optimization problems (DCOPs).

Similarly, Markovian dynamic DCOPs and their solution method were proposed (Nguyen et al., 2014), where distributed RVI Q-learning and R-learning algorithms employed DPOP (Petcu and Faltings, 2005), which is also a solution method of DCOPs, to select the joint actions of agents.

While the learning's conventional goal is the maximization of total rewards among agents, in practical resource allocation problems, unfairness among agents is crucial. In several recent studies of decentralized optimization methods including DCOPs, unfairness is deemed an important criterion (Netzer and Meisels, 2013a; Netzer and Meisels, 2013b).

As a criterion of unfairness, *leximin* has been studied (Moulin, 1988; Bouveret and Lemaître, 2009). The leximin defines the relationship between two vectors in multi-objective optimization problems. Maximization on the leximin improves unfairness among objectives. Extended classes of DCOPs applying leximin criterion and solution methods have been proposed (Matsui et al., 2014; Matsui et al., 2015).

In this study, we address an action selection method based on leximin criteria, which reduces the unfairness among agents, in decentralized reinforcement learning. We experimentally evaluated the effects and influences of the proposed approach on classes of sensor network problems.

The remainder of this paper is organized as follows. The next section shows the background of our study including reinforcement learning with a distributed setting and criteria for the cooperative actions of agents. Section 3 describes a class of sensor network problems for our motivating domain. Our proposed approach is shown in Section 4. In Section 5, our proposed methods are experimentally evaluated. Related works and discussions are addressed in Section 6, and the study is concluded in Section 7.

Matsui T. and Matsuo H.
A Study on Cooperative Action Selection Considering Unfairness in Decentralized Multiagent Reinforcement Learning.
DOI: 10.5220/0006203800880095
In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence (ICAART 2017)*, pages 88-95
ISBN: 978-989-758-219-6

2 BACKGROUND

2.1 Reinforcement Learning with Distributed Setting

Several types of reinforcement learning have been applied to multiagent systems to optimize cooperative policies among agents. In several recent studies, multiagent reinforcement learning is modeled and solved using a distributed manner (Zhang and Lesser, 2011; Nguyen et al., 2014). Basically, these approaches resemble standard settings, while the learning tables are distributed among agents. The optimal joint action is determined using a distributed optimization method.

Here we address a standard Q-learning as a base of such a distributed version. The problem of Q-learning consists of a set of states S, a set of actions A, a function table of Q-values Q, reward function R, and an update rule:

$$Q_{t+1}(\mathbf{s}_t,\mathbf{a}_t) = (1-\alpha)Q_t + \alpha(r+\gamma\max_{\mathbf{a}} Q_t(\mathbf{s}_{t+1},\mathbf{a})). \quad (1)$$

$\mathbf{s}_t \in S$ and $\mathbf{a}_t \in A$ denote a joint state and a joint action among agents at time step t. r is the reward that is received from reward function R in an environment when agents perform joint action $\mathbf{a}_t$ at joint state $\mathbf{s}_t$. The learning is adjusted by learning rate α and discount rate γ. In each time step t, agents sense current joint state $\mathbf{s}_t$ and choose joint action $\mathbf{a}_t$ from A. After the agents perform action $\mathbf{a}_t$, they sense next state $\mathbf{s}_{t+1}$ and obtain reward r from the environment. Based on the reward and the next state, agents update the Q-values for $\mathbf{s}_t$ and $\mathbf{a}_t$. In this computation, optimal joint action $\mathbf{a}$ at joint state $\mathbf{s}_{t+1}$ is chosen and the corresponding Q-value is propagated with the discount rate. The above processing is repeated until the agents learn the environment's information.

To choose the next joint action at joint state $\mathbf{s}_t$, several heuristics are employed based on the trade-off between exploration and exploitation. With ε-greedy heuristic, agents perform random walk with probability ε and optimal joint action $\mathbf{a}^*$ such that $\mathbf{a}^* = \mathrm{argmax}_{\mathbf{a}} Q_t(\mathbf{s}_t,\mathbf{a})$ with probability $1-\varepsilon$.

In a distributed setting, each agent i has a part of states S_i and actions A_i, where $\cup_i S_i = S$ and $\cup_i A_i = A$. Note that, for $i \neq j$, $S_i \cap S_j$ and $A_i \cap A_j$ can be non-empty sets. Local problems are related by this overlap. Each agent i has a function table of Q-values Q_i for S_i and A_i. Also, partial reward function R_i is defined for S_i and A_i. An advantage of this setting is that huge global joint states and actions are approximated as sets of local joint states and actions. However, cooperation is necessary to choose an appropriate global joint action, while each agent i updates Q_i independently.

2.2 Cooperation based on Distributed Constraint Optimization

To choose an appropriate global joint action, an optimization method that resembles distributed problem solving is necessary. In the case of a ε-greedy heuristic, agents perform random walk or choose the globally optimal action. The latter case is represented as a distributed constraint optimization problem (Modi et al., 2005; Petcu and Faltings, 2005; Farinelli et al., 2008; Zivan, 2008), which is a fundamental problem in multiagent cooperation.

A distributed constraint optimization problem (DCOP) is defined by $(\mathcal{A}, X, D, F)$. Here $\mathcal{A}$ 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 $\mathcal{A}$. Variable $x_n \in X$ takes its values from its domain $D_n \in D$. Function $f_m \in F$ defines the utility values on several variables. $X_m \subset X$ defines the set of variables in the scope of f_m. $F_n \subset F$ defines a set of functions, where x_n is in their scope. f_m is defined as $f_m(x_{m0},\cdots,x_{mk}) : D_{m0} \times \cdots \times D_{mk} \rightarrow \mathbb{R}$, where $\{x_{m0},\cdots,x_{mk}\} = X_m$. $f_m(x_{m0},\cdots,x_{mk})$ is also denoted by $f_m(X_m)$. Aggregation $F(X)$ of all the objective functions is defined as $F(X) = \sum_m \text{ s.t. } {}_{f_m \in F, X_m \subseteq X} f_m(X_m)$. The goal is to find a globally optimal assignment that maximizes the value of $F(X)$.

Consider the problem for optimal joint action $\mathbf{a}^*$ such that $\mathbf{a}^* = \mathrm{argmax}_{\mathbf{a}} Q_t(\mathbf{s}_t,\mathbf{a})$. In distributed settings, since the Q-values are approximated with local Q-values, each agent i has $Q_{i,t}$ at time step t. Here variable x_n of a DCOP is defined for $A_n \subset A_i$ and D_n corresponds to A_n. Note that agent i's A_i (and S_i) can overlap with different agent j's A_j (and S_j). In this case, partial set $A_n = A_i \cap A_j$ is shared by both agents and defined as domain D_n of variable x_n. On the other hand, function f_m is defined for $Q_{i,t}$. Namely, $f_m(X_m) = Q_{i,t}(\mathbf{s}_{i,t},\mathbf{a}_i)$, where $\mathbf{s}_{i,t}$ is a vector of constant values. While X_m corresponds to A_i, X_m overlaps with the scopes of the functions in other agents.

Since this problem resembles the computation of $\max_{\mathbf{a}} Q_t(\mathbf{s}_{t+1},\mathbf{a})$ in Eq. (1), the same solution method can be applied to both computations of a joint action and learning.

Each agent locally knows the information of its own variables and the related functions in the initial state. An optimization method in a distributed manner computes the globally optimal solution. Several solution methods have been proposed for DCOPs. Here we employ a dynamic programming method that computes the exact optimal solution.

2.3 Optimization Criteria

In conventional multiagent reinforcement learning, the goal of the problem is to optimize the global sum of the rewards. On the other hand, different studies address the individuality of each agent, such as the Nash equilibrium (Hu and Wellman, 2003).

In several recent DCOP studies, fairness among agents was addressed (Netzer and Meisels, 2013a; Netzer and Meisels, 2013b; Matsui et al., 2014; Matsui et al., 2015). Since the solution methods for such classes of problems are designed as distributed algorithms, the problem of joint actions can be replaced by problems based on fairness. In particular, optimization with the leximin criterion (Moulin, 1988; Bouveret and Lemaître, 2009; Matsui et al., 2014; Matsui et al., 2015) shown below improves fairness among agents. Here we refer the definitions in (Matsui et al., 2015).

To address multiple objectives for agents, objective vectors are defined. Each value of an objective vector corresponds to a utility value for an agent. Objective vector $\mathbf{v}$ is defined as $[v_0, \cdots, v_K]$, where v_j is an objective value. Vector $\mathbf{F}(\mathcal{X})$ of objective functions is defined as $[F^0(\mathcal{X}^0), \cdots, F^K(\mathcal{X}^K)]$, where $\mathcal{X}^j$ is the subset of X on which F^j is defined. $F^j(\mathcal{X}^j)$ is an objective function for objective j. For assignment $\mathcal{X}$, vector $\mathbf{F}(\mathcal{X})$ of the functions returns an objective vector $[v_0, \cdots, v_K]$. Here $v_j = F^j(\mathcal{X}^j)$. In addition, the objective vector is sorted to employ leximin. Based on objective vector $\mathbf{v}$, in a sorted objective vector, all the values of $\mathbf{v}$ are sorted in ascending order.

Based on the sorted objective vectors, the leximin is defined as follows. Let $\mathbf{v}$ and $\mathbf{v}'$ denote the vectors of identical length $K+1$. Let $[v_0, \cdots, v_K]$ and $[v'_0, \cdots, v'_K]$ denote the sorted vectors of $\mathbf{v}$ and $\mathbf{v}'$. Also, let $\prec_{leximin}$ denote the relation of the leximin ordering. $\mathbf{v} \prec_{leximin} \mathbf{v}'$ if and only if $\exists t, \forall t' < t, v_{t'} = v'_{t'} \land v_t < v'_t$.

The maximization on leximin reduces the unfairness among the utility values among the agents by improving the worst case utilities. Also, this criterion chooses a Pareto optimal objective vector.

In several resource allocation problems, reducing unfairness among agents is an important criterion, while such joint actions may not be compatible with conventional reinforcement learning.

3 MOTIVATING DOMAIN

Considering several related works (Zhang and Lesser, 2011; Nguyen et al., 2014), we define an example problem motivated by sensor networks, as shown in

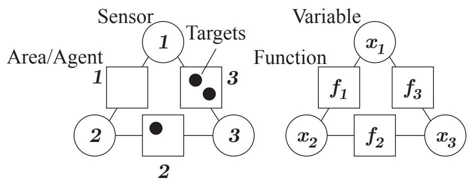

Figure 1: Sensor network problem.

Fig. 1. The system consists of sensors, areas, and targets. Each sensor adjoins a few areas, and each area adjoins a few sensors. Here we assume for simplicity that an area is related to two sensors, each of which can be simultaneously allocated to one of the adjoining areas.

While a target stays in one of the areas, multiple targets can be in the same area. The sensing event is active and detected by targets. A target randomly moves to one of the other *neighborhood* areas of sensors adjoining the current staying area when sensors are allocated to the residing area. We only consider whether an area is occupied by at least one target.

When an area is occupied by targets and allocated with sensors, a reward is given that corresponds to the combination of allocated sensors. However, if an empty area is allocated by sensors, a small negative reward is given. The goal of the problem is to improve the global reward aggregated for all the areas.

This system is represented as a problem of multiagent reinforcement learning. To represent cooperative action, agent i corresponds to area i. In actual settings, such an agent will be operated by a sensor node that adjoins the area. States S_i of agent i correspond to the states of sensors adjoining area i. Each pair of states $(s^a_k, s^o_k) \in S_i$ is defined by the states of sensor k. s^a_k represents the area to which sensor k is allocated. s^o_k represents whether sensor k detects that an area is occupied by the targets. Similarly, actions A_i are based on the areas to which the sensors are allocated. Each action $a_k \in A_i$ represents the area to which sensor k is allocated. Q_i is defined for S_i and A_i. The values of reward function R_i are defined for area i, as shown above.

The DCOP for a joint action at time step t is represented as follows. Variable x_k is defined for a_k of sensor k. Function $f_i(X_i)$ is defined for $Q_{i,t}(\mathbf{s}_{i,t}, \mathbf{a})$, where $x_k \in X_i$ corresponds to $a_k \in A_i$.

In the example shown in Fig. 1, the sensor network consists of three sensors and three areas. The agent of area 1 has states S_1, actions A_1, and Q-values Q_1. Since the states and actions are related to sensors 1 and 2, the local problem partially overlaps with those of other agents. In a DCOP, a variable corresponds to the actions of a sensor, and a function corresponds to a part of a table of Q-values.

4 COOPERATIVE ACTION CONSIDERING UNFAIRNESS

4.1 Applying Leximin Optimization

In this study, we apply a leximin operator for action selection based on the unfairness among agents. In a sensor network, when an occupied area is observed by multiple sensors, a higher reward is given for more information. On the other hand, such a concentration decreases the number of observed areas. Namely, several occupied areas can be ignored even if smaller rewards are given.

Since the global sum of utilities among agents does not consider individual utilities, other criteria are required in this situation. A comparison of unfairness based on leximin is expected to handle this case. On the other hand, this action selection might be incompatible with the original reinforcement learning. As the first study, we experimentally evaluate the effect and influence of such action selection.

For multiple objective problems, each objective for agent *i* of area *i* corresponds to function f_i. Therefore, a sorted objective vector consists of the values of f_i for all agents, but the conventional case employs the value's total summation.

4.2 Solution Method

To solve the DCOPs for action selection, we employ a dynamic programming approach that can be performed in a distributed manner (Matsui and Matsuo, 2014; Matsui et al., 2015). Since the solution method's basic framework is the same for both criteria, we first briefly sketch the solution method for the summation. A DCOP is represented as a factor graph, which is a bipartite graph that consists of variable nodes, function nodes, and edges. For the factor graph, a pseudo-tree, which is a tree-like graph structure defining the (partial) order of nodes, is generated (Fig. 2). Here we assume that the root node is one of variable nodes. If the topology of the original sensor network is static, a single pseudo-tree can be reused for each problem solving. The problem is decomposed based on pseudo-trees. As a result, each node has a partial problem consisting of the subtree rooted at the node. The partial problem of the subtree is only related to its higher nodes with cut edges between the subtree and higher nodes. The variables that correspond to the cut edges are called separators.

Based on these relations, each node performs part of the dynamic programming that consists of two phases. The first phase is performed in a bottom-up manner. Each node aggregates function tables from its

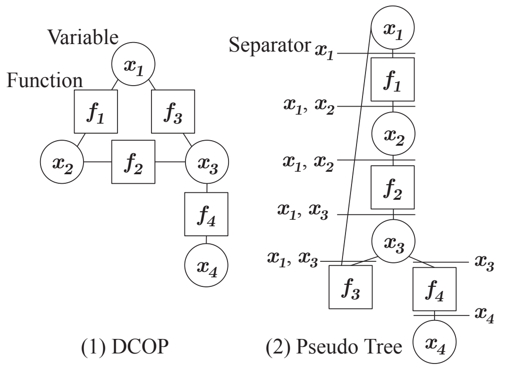

Figure 2: Solution method.

child nodes, if they exist, and generates a function table by adding the function values for each assignment to the related variables. In addition, a function node also aggregates its function with the function table and maximizes the aggregated function table for non-separator variables. As a result, a smaller function table for just the separators is generated and sent to the parent node.

The second phase is performed in a top-down manner. In the root (variable) node, the optimal assignment for the variable is determined from the aggregated function table. Then the root agent sends its optimal assignment to its child nodes. The child nodes determine the optimal assignment to the related non-separator variables, if necessary, and propagate the optimal assignment, including those of higher nodes, to its child node. The computational and space complexity of the solution method is exponential with the number of separators for each partial problem. However, we prefer the exact solution method to choose one optimal solution.

By replacing the objective values and the addition/maximization operators to the sorted objective vectors and extended operators, the solution method solves the maximization problem on leximin criterion. The addition operator is replaced by a couple of concatenate and resorting operators for objective vectors. The maximization is based on leximin. Since we use real values as Q-values, the opportunities for tiebreaks in leximin comparisons will be less than the case of integer values. However, the leximin operator continues to work.

4.3 Selection of Action in Learning Equations

Since action selection based on leximin criterion is different from the original one, it may be incompatible with reinforcement learning. In particular, the selection of an optimal joint action in Eq. (1) might

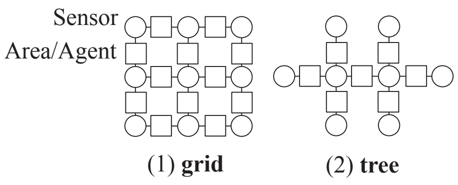

Figure 3: Topologies of problems.

be corrupted, but the actual action selection can be considered an exploration strategy.

To remove such influence from the learning equation, discount rate γ can be set to zero. Another approach is to employ the original optimization criterion for the learning equation. In this case, how the exploration strategy based on leximin affects the original method will be evaluated.

5 EVALUATION

5.1 Problem Settings

We experimentally evaluated the effects and the influences of our proposed approach and evaluated the following topologies of sensor network problems.

- **grid**: We placed sensors at the grid vertices and the areas at the edges (Fig. 3 (1)). Since this topology easily increases the number of separators, we limit the size to 3×3 sensors.
- **tree**: A small tree is designed based on the capacity of the sensor resources. Similar to the grids, sensors and areas are placed at the tree's vertices and edges. When two sensors are allocated to an area in the center of this network, at least one sensor can be allocated to all of the other areas ((Fig. 3 (2)).
- **tree-like**: In addition to a randomly generated tree, a few edges/areas are contained to compose cycles.

To reduce the size of the state and action spaces, we limited the problems so that an area adjoins two sensors. The number of targets was set to the number of areas based on the average occupancy ratio for the areas.

We set the rewards for the areas as follows.

- **0-2**: When an occupied area is sensed by one sensor, no reward is given, and in the case of two sensors a reward of 2 is given.
- **1-2**: When an occupied area is sensed by one and two sensors, rewards of 1 and 2 are given.

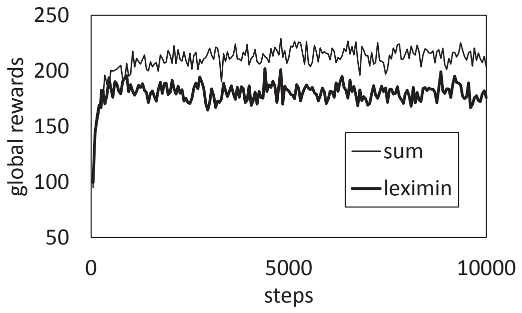

Figure 4: Global reward (**grid**, **0-2**, $\gamma = 0$).

Table 1: Global reward and allocated sensors to occupied areas in last quarter steps (**grid**, **0-2**, $\gamma = 0$).

alg.	rwd.	num. of rwd.		num. of alc. sns.			rate. of
		≤ 0	2	0	1	2	occ.
sum	10722	24636	5364	5997	2332	5364	0.46
leximin	9006	25494	4506	6546	2918	4506	0.47

- **2-2_0-2**: The areas are categorized into two types. In most occupied areas, when one is sensed by one or two sensors, a reward of 2 is given. On the other hand, in one area, the reward is defined as **0-2**.
- **2-2_1-2**: Similar to **2-2_0-2**, the areas are categorized, and the reward is defined as **1-2** in one area.

When a non-occupied area is sensed, a small negative reward -1.0^{-3} is given.

We compared the following solution methods for action selection.

- **sum**: the summation is employed for both actual actions and learning.
- **leximin**: the leximin is employed for both action selections.
- **lxmsum**: the leximin and the summation are used for the actual actions and the learning.

In addition, we compared cases where discount rate γ is 0 and 0.5. The probability ϵ of random walk and learning rate α were set to 0.2 and 0.2. These parameters are chosen based on preliminary experiments so that several typical cases are shown. We performed ten trials for the same problem. Each trial consisted of 10000 steps of joint actions. The results were averaged for the trials.

5.2 Results

First we evaluated the case of **grid** with a set of reward **0-2**. Here discount rate γ was set to zero. Fig. 4 shows

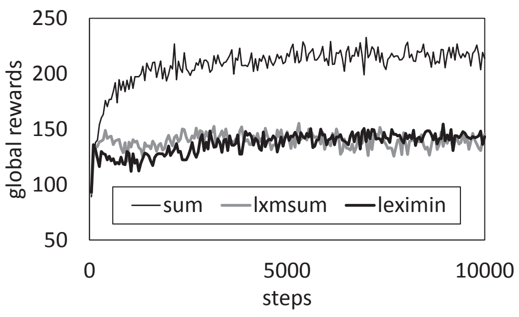

Figure 5: Global reward (**grid**, **0-2**, $\gamma = 0.5$).

Table 2: Global reward and allocated sensors to occupied areas in last quarter steps (**grid**, **0-2**, $\gamma = 0.5$).

alg.	rwd.	num. of rwd.		num. of alc. sns.			rate. of
		≤0	2	0	1	2	occ.
sum	10830	24582	5418	6209	2255	5418	0.47
lxmsum	6880	26556	3444	7101	2976	3444	0.47
leximin	7163	26415	3585	7027	4494	3585	0.50

Table 3: Global reward and allocated sensors to occupied areas in last quarter steps (**grid**, **1-2**, $\gamma = 0$).

alg.	rwd.	num. of rwd.			num. of alc. sns.			rate. of
		≤0	1	2	0	1	2	occ.
sum	15270	17976	8773	3251	4863	8773	3251	0.56
leximin	14261	16205	13322	474	3696	13322	474	0.58

the global reward among the agents for 50 steps. Table 1 shows the summation of the global rewards in the last 2500 steps (rwd.), the histograms of the rewards (num. of rwd.), the histograms of the number of allocated sensors to the occupied areas (num. of alc. sns.), and the occupancy ratio for all the areas (rate. of occ.). Since this problem contains only single types of rewards, conventional summation well works. In addition, a few areas can be covered by two sensors, but rewards are given only for two allocated sensors. In such cases, a leximin strategy is less effective.

Next we evaluated the same problem with $\gamma = 0.5$. Fig. 5 and Table 2 show the results. The result also contains the case of **lxmsum**. In this case, the reward of **leximin** decreased less than the case of $\gamma = 0$. This reveals that action selection based on leximin in learning did not improve the result. Also, **lxmsum**, which employed conventional summation for learning, was not effective in average.

The next case is **grid** with **1-2**. In this problem, sensors can cover most areas earning at least a reward of 1. Fig. 6, Table 3, Fig. 7, and Table 4 show the results. **leximin** improved the coverage of the areas by

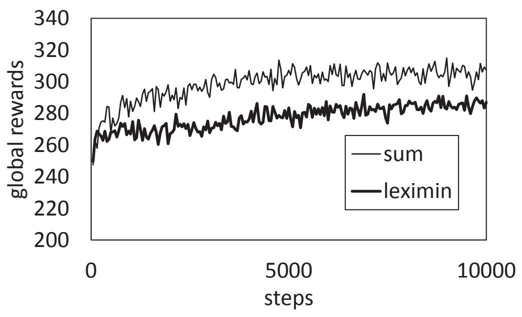

Figure 6: Global reward (**grid**, **1-2**, $\gamma = 0$).

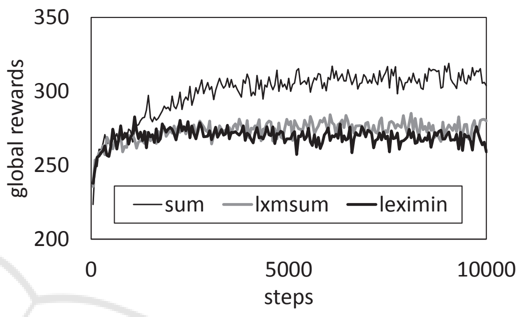

Figure 7: Global reward (**grid**, **1-2**, $\gamma = 0.5$).

Table 4: Global reward and allocated sensors to occupied areas in last quarter steps (**grid**, **1-2**, $\gamma = 0.5$).

alg.	rwd.	num. of rwd.			num. of alc. sns.			rate. of
		≤0	1	2	0	1	2	occ.
sum	15454	18513	7516	3972	5206	7516	3972	0.56
lxmsum	13787	17899	10408	1693	4861	10408	1693	0.56
leximin	13401	17699	11192	1108	4688	11192	1108	0.56

decreasing the global rewards. In this case, **lxmsum** was slightly closer to **sum** than **leximin**. From this result, action selection based on leximin is relatively effective in simple cases, where the rewards are easily decreased to improve the worst case agents. For this kind of settings, similar results were obtained in several other topologies including **tree-like**.

As a different setting, we evaluated **tree** with **2-2_0-2**. In this problem, an area in the center of the network obtains rewards of **0_2**, while other areas obtain rewards of **2_2**. Table 5 shows that **leximin** improved the coverage of the areas. In addition, the total reward of the area in the center improved. On the other hand, for $\gamma = 0.5$, the **leximin** result is worse than **sum**, as shown in Table 6.

Similarly, we evaluated **tree** with **2-2_1-2**. Tables 7 and 8 show the results. In this case, the number of reward 1 increased, although we expected that the total reward of the area in the center would improve.

Table 5: Global reward and allocated sensors to occupied areas in last quarter steps (**tree**, **2-2_0-2**, $\gamma = 0$).

alg.	rwd.	num. of rwd.		num. of alc. sns.			rate. of
		≤0	2	0	1	2	occ.
sum	20288	7353	10147	447	8148	2246	0.62
leximin	21176	6909	10591	268	8561	2204	0.61

alg.	rwd. of cnt. area
sum	2951
leximin	3454

Table 6: Global reward and allocated sensors to occupied areas in last quarter steps (**tree**, **2-2_0-2**, $\gamma = 0.5$).

alg.	rwd.	num. of rwd.		num. of alc. sns.			rate. of
		≤0	2	0	1	2	occ.
sum	20889	7052	10448	320	8477	2214	0.63
lxmsum	19953	7520	9980	490	8121	2203	0.61
leximin	20076	7459	10041	469	8128	2231	0.62

alg.	rwd. of cnt. area
sum	3220
lxmsum	2671
leximin	2764

Table 7: Global reward and allocated sensors to occupied areas in last quarter steps (**tree**, **2-2_1-2**, $\gamma = 0$).

alg.	rwd.	num. of rwd.			num. of alc. sns.			rate. of
		≤0	1	2	0	1	2	occ.
sum	20052	7084	774	9642	420	8356	2060	0.62
leximin	20148	6648	1550	9303	237	9098	1754	0.63

alg.	rwd. of cnt. area
sum	2707
leximin	2318

Instead, the coverage of the areas improved by decreasing the rewards of the center area. The result reveals that simple action selection based on leximin does not easily adapt to relatively complex cases.

Table 9 shows the computational cost of the solution method. Here the maximum degree of sensors in **tree-like** was limited to three. Five instances were also averaged for each setting of **tree-like**. For **grid**, a pseudo tree was generated using a zigzag order from the left-top variable, while a maximum degree heuristic was employed for other graphs. The experiments were performed on a single computer with Core i7-3930K CPU @ 3.20GHz, 16GB memory, Linux 2.6.32 and g++ 4.4.7. Note that our current experimental implementation can be improved. The bottleneck of the solution methods is the optimization method for action selection. Since the time and space complexity of dynamic programming exponentially increases with the number of separators, large and dense problems will need relaxations.

Table 8: Global reward and allocated sensors to occupied areas in last quarter steps (**tree**, **2-2_1-2**, $\gamma = 0.5$).

alg.	rwd.	num. of rwd.			num. of alc. sns.			rate. of
		≤0	1	2	0	1	2	occ.
sum	20863	6754	623	10123	293	8661	2085	0.63
lxmsum	20317	6751	1174	9575	279	8864	1885	0.63
leximin	20338	6752	1151	9597	278	8836	1912	0.63

alg.	rwd. of cnt. area
sum	3126
lxmsum	2610
leximin	2616

Table 9: Computational cost.

prb.	alg.	max. sz. of separator		comp.
		variables	combinations	time [s]
grid	sum	4	72	7.37
	lxmsum	4	72	10.59
	leximin	4	72	13.77
tree	sum	1	4	0.56
	lxmsum	1	4	0.67
	leximin	1	4	0.77
tree-like	sum	4	103	8.83
15 sns.	lxmsum	4	103	13.28
20 areas	leximin	4	103	18.01
tree-like	sum	4	92	9.58
20 sns.	lxmsum	4	92	15.15
25 areas	leximin	4	92	20.63
tree-like	sum	3	23	8.62
50 sns.	lxmsum	3	23	13.80
55 areas	leximin	3	23	19.47

6 RELATED WORKS AND DISCUSSIONS

This study was motivated by several previous studies that addressed sensor network domains (Zhang and Lesser, 2011; Nguyen et al., 2014). Since those studies employed dedicated problems, we designed a relative simple one. Our setting of learning methods relatively depends on the random walk of ε-greedy heuristics because of the activities of targets. In addition, we set a number of targets so that almost half of the areas are occupied, since unfairness depends on resource capacities. Investigations of different classes of problems will be future works.

Several studies such as Nash-Q learning (Hu and Wellman, 2003) have addressed the individuality of agents. While such studies mainly focus on selfish agents, we are interested in cooperative actions by considering unfairness. As a first study, we addressed the effects and influence of action selection based on leximin that can be applied in a decentralized manner.

On the other hand, the results reveal the necessity of dedicated learning rules; in simple cases our proposed approach has some effects. Since unfairness depends on the values in Q-tables, more discussions for the case of leximin are necessary. In particular, some normalization methods, for different progresses of leaning in individual agents, are possibly important for the case of fairness.

We employed an exact solution method to select joint actions. However, for large and complex problems, approximation methods are necessary, since the time and space complexity of the exact method exponentially increases with the number of separators. Such approximation is also considered as a challenging problem.

7 CONCLUSIONS

We addressed action selection based on unfairness among agents in a decentralized reinforcement learning framework and experimentally investigated the effect and the influence of leximin criterion in action selection. Even though the proposed approach effectively worked in relatively simple settings, our results uncovered several exploration and learning issues. Our future works will analyze the relationship between the proposed cooperative action and learning rules and applications to other problem domains. Improvement of learning rules to manage information of unfairness, and scalable solution methods for joint action selection will also be important challenges.

ACKNOWLEDGEMENTS

This work was supported in part by JSPS KAKENHI Grant Number JP16K00301.

REFERENCES

Bouveret, S. and Lemaître, M. (2009). Computing leximin-optimal solutions in constraint networks. *Artificial Intelligence*, 173(2):343–364.

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

Hu, J. and Wellman, M. P. (2003). Nash q-learning for general-sum stochastic games. *J. Mach. Learn. Res.*, 4:1039–1069.

Matsui, T. and Matsuo, H. (2014). Complete distributed search algorithm for cyclic factor graphs. In *6th International Conference on Agents and Artificial Intelligence*, pages 184–192.

Matsui, T., Silaghi, M., Hirayama, K., Yokoo, M., and Matsuo, H. (2014). Leximin multiple objective optimization for preferences of agents. In *17th International Conference on Principles and Practice of Multi-Agent Systems*, pages 423–438.

Matsui, T., Silaghi, M., Okimoto, T., Hirayama, K., Yokoo, M., and Matsuo, H. (2015). Leximin asymmetric multiple objective DCOP on factor graph. In *Principles and Practice of Multi-Agent Systems - 18th International Conference*, pages 134–151.

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

Moulin, H. (1988). *Axioms of Cooperative Decision Making*. Cambridge : Cambridge University Press.

Netzer, A. and Meisels, A. (2013a). Distributed Envy Minimization for Resource Allocation. In *5th International Conference on Agents and Artificial Intelligence*, volume 1, pages 15–24.

Netzer, A. and Meisels, A. (2013b). Distributed Local Search for Minimizing Envy. In *2013 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 53–58.

Nguyen, D. T., Yeoh, W., Lau, H. C., Zilberstein, S., and Zhang, C. (2014). Decentralized multi-agent reinforcement learning in average-reward dynamic dcops. In *28th AAAI Conference on Artificial Intelligence*, pages 1447–1455.

Petcu, A. and Faltings, B. (2005). A scalable method for multiagent constraint optimization. In *19th International Joint Conference on Artificial Intelligence*, pages 266–271.

Zhang, C. and Lesser, V. (2011). Coordinated multi-agent reinforcement learning in networked distributed pomdps. In *25th AAAI Conference on Artificial Intelligence*, pages 764–770.

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