RoboCupRescue 2019
TDP Infrastructure AIT-Rescue (Japan)

Yuki Miyamoto[1], Taishun Kusaka[1], Yuki Okado[1], Kazunori Iwata[2], and Nobuhiro Ito[1]

[1]Department of Information Science, Aichi Institute of Technology, Japan
[2]Department of Business Administration, Aichi University, Japan

Abstract. The "task assignment problem" of RoboCupRescue Simulation (RRS) can be regarded as a Distributed Constraint Optimization Problem (DCOP). However, it is difficult to apply the DCOP algorithm to the problem on the current simulator. In this paper, we propose an extension on RRS-ADF for the difficulty. We introduce a new communication system that agents can use to communicate repeatedly within each step of this extension. Furthermore, we also describe an example that applies a DCOP algorithm, and then discuss its effectiveness. The results confirmed that our extension is effective.

1 Introduction

A typical problem in RoboCupRescue Simulation (RRS) is the task assignment problem. The task assignment problem consists of determining an approach to assign n agents to m tasks according to their purpose. This problem can be modeled as a Distributed Constraint Optimization Problem (DCOP) [4]. RMASBench [2] was proposed as an attempt to solve the task assignment problem of RRS using a DCOP and its algorithm. RMASBench is a benchmark system for the task assignment problem that introduces pseudo-agents and pseudo-communication between them on the RRS system. Research on RMASBench [3] has demonstrated that the DCOP algorithm works effectively for the task assignment problem on RRS.

Therefore, in this research, we propose an extension of RRS-ADF to use the DCOP algorithm in the task assignment problem on RRS. However, since RMASBench introduced pseudo-communication, it is difficult to use the DCOP algorithm on the current simulator. Therefore, the extension proposed in this research provides a similar pseudo-communication system on the present system. Additionally, we confirm that it operates effectively by applying the max-sum algorithm [6] which is a typical approximate solution method for the DCOP.

In Section 2, we first describe the DCOP, max-sum algorithm, and RMASBench as background information. In Section 3, we describe the pseudo-communication system that we introduce and the extension of RRS-ADF that uses this system. In Section 4, we apply the DCOP to the task assignment problem of ambulance teams and implement the max-sum algorithm for the problem on the

extension. In Section 5, we present the experiment used to confirm that our extension and the max-sum algorithm implementation work effectively. As a result of the experiment, we confirm that they work sufficiently well.

2 Background

2.1 DCOP: Distributed Constraint Optimization Problem

The DCOP is the problem of determining a combination of variable values that maximize utility when they have a constraint between a variable that corresponds to a distributed agent and other variables. The definition of the DCOP is as follows [1]:

$\mathbf{A} = \{a_1, \ldots, a_n\}$
is a set of agents, where a_i is an agent.
$\mathbf{X} = \{x_1, \ldots, x_m\}$
is a set of variables. However, $m \geqq n$, where m is the number of variables and n is the number of agents.
$\mathbf{D} = \{D_1, \ldots, D_m\}$
is a set of ranges for each variable $x_i \in X$, where D_i represents the range of corresponding variable x_i.
$\mathbf{F} = \{f_1, \ldots, f_k\}$
is a set of functions (called utility functions) that express constraints between variables. The utility function is represented by the following expression: $f_i \colon \times_{x_j \in \mathbf{x}^i} D_j \to \mathbb{R}$, where $\mathbf{x}^i$ is the set of variables whose constraint relation is represented by f_i. The utility function maps a combination of arbitrary values included in the value range for each variable of $\mathbf{x}^i$ to a real number. The value obtained by the utility function is called the "utility."
$\alpha : \mathbf{X} \to \mathbf{A}$
is a mapping function that expresses a relationship between an agent and a variable. Each agent corresponds to a distinct variable.

Objective function $\mathbf{F}_g(\mathbf{X})$ for optimization is defined as

$$\mathbf{F}_g(\mathbf{X}) = \sum_{f_i \in \mathbf{F}} f_i\left(\mathbf{x}^i\right) \tag{1}$$

by the utility functions. Then, the optimized assignment σ^* which maximizes $\mathbf{F}_g(\mathbf{X})$, is expressed as

$$\sigma^* = \arg\max_{\sigma \in \mathcal{D}} \mathbf{F}_g(\sigma) \tag{2}$$

In this case, $\mathcal{D}$ is a direct product set of all ranges in $\mathbf{D}$, which means a set of possible assignments. Note that Eq. (2) is used only when a combination that maximizes the objective function is the optimized assignment.

2.2 Max-Sum Algorithm

The max-sum algorithm is one of the major approximate solution methods for the DCOP. It optimizes the overall utility by propagating the utility to each constraint. The target of this algorithm is a problem that can be modeled as the factor graph. The factor graph is an undirected graph that consists of variable nodes that represent variables, factor nodes that represent utility functions between variable nodes, and edges that represent mutual relationships between variable nodes and factor nodes. An example of the factor graph is shown in Figure 1.

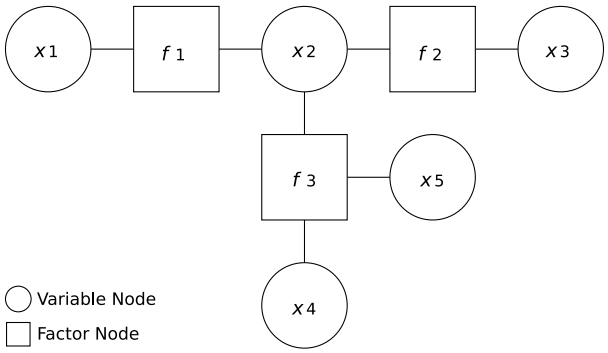

Fig. 1: Example of the factor graph

In Figure 1, the factor node has a constraint, which is a utility function, on neighboring variable nodes. Additionally, a corresponding agent calculates a utility for each variable value according to its own range and the constraints of the neighboring factor nodes in the variable node. First, initial utilities are calculated in variable nodes by corresponding agents according to the condition. Then the utilities gradually propagate to other variable nodes through neighboring factor nodes. The propagation (calculation) continues until the updating of utilities is stopped or when the designated iteration number is reached.

The following is an evaluation function [3], and is used when variable node x of the equation sends the utility on a value of the variable node to the factor node f:

$$\mu_{x \to f}(\mathbf{x}) = \sum_{g \in \mathcal{N}(x) \setminus \{f\}} \mu_{g \to x}(\mathbf{x}) \tag{3}$$

where $\mathbf{x}$ is the value calculated by an appropriate agent and included in the range of the variable node x; $\mathcal{N}(x)$ is a set of factor nodes that neighbor variable node x; and $\mu_{g \to x}$ is the utility calculated on factor node g and sent to variable node x. Eq. (3) calculates the sum of the utilities received from neighboring factor nodes except for the factor node that is the destination. The calculated value is in the range of variable node x.

The following is an evaluation function [3], and is used when factor node f sends the utility on a value to variable node x:

$$\mu_{f\to x}(\mathbf{x}) = \max_{\mathbf{Y}} \left(f(\mathbf{x}, \mathbf{Y}) + \sum_{y\in Y} \mu_{y\to f}(\mathbf{y}) \right) \tag{4}$$

where Y is the set of variable nodes that neighbor factor node f except for x, $\mathbf{Y}$ is the set of values that are calculated by appropriate agents on Y, $\mu_{y\to f}$ is the utility from variable node y to factor node f, and $\mathbf{y}$ is the value of variable node y that is included in $\mathbf{Y}$. Eq. (4) calculates the maximum value of the sum of a value of a utility function for $\mathbf{x}$ and $\mathbf{Y}$, and a sum of $\mu_{y\to f}$ for $y \in Y$ on $\mathbf{Y}$.

When the propagation is complete, the optimal assignment is calculated by

$$\sigma^* = \arg\max_{x\in\mathbf{X}} \sum_{f\in\mathcal{N}(x)} \mu_{f\to x}(\mathbf{x}) \tag{5}$$

Eq. (5) calculates an assignment that maximizes the sum of values of utility functions for each factor node.

2.3 Previous Research: RMASBench

RMASBench was proposed in 2013 to conduct DCOP research of RRS. It is a benchmark system for the task assignment method for multi-agent systems, and an evaluation system for DCOP algorithms.

The system structure of RMASBench is illustrated in Figure 2. The center agent can obtain current states from each agent and current situations of the disaster simulation from the kernel. Additionally, the system introduces a pseudo-communication layer (communication layer) and pseudo-agents (DCOP agent) that correspond to agents on the pseudo-communication layer, so that the pseudo-agents can communicate without restrictions imposed by the RRS system. Then, the center agent performs task assignments by making the pseudo-agents send and receive many messages repeatedly in the layer. The result of the task assignments on the pseudo-agents is communicated to the agents from the center agent directly.

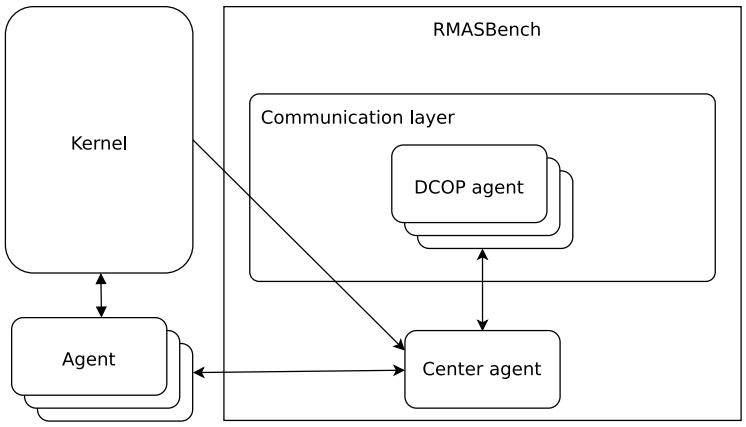

Fig. 2: System structure of RMASBench

This approach achieves the application/implementation of DCOP algorithms to the task assignment problem of RRS. However, this approach cannot solve the task assignment problem based on the local communication of the DCOP, although it can solve the problem based on global communication.

3 Extension of RRS-ADF for Task Assignment to Solve the DCOP and a Pseudo-Communication System

3.1 Necessity of Multiple Communications within a Step with the Pseudo-Communication System

The max-sum algorithm requires sending and receiving many messages repeatedly to propagate utilities among agents. By contrast, agents of RRS can send/receive messages only once within one step, which means 1 minute in real time because agents of RRS are assumed to communicate by voice. Therefore, multiple steps are required to execute the algorithm.

The DSA is one of the major approximate solution methods for DCOP similar to the max-sum algorithm. The experiments of Zhang et al. [7] demonstrated that the DSA requires approximately 60 iterations to converge to a solution. By contrast, RRS performs from 200 steps to 300 steps. This fact means that it is difficult for RRS to use the DSA effectively because the algorithm requires many steps.

To solve this problem, RMASBench enables multiple communications within each step, with a mechanism different from the simulator of RRS. In this paper, we introduce a pseudo-communication system that realizes such a communication mechanism.

3.2 Communication Condition in a Pseudo-Communication System

In a pseudo-communication system, messages sent from agent a_i are received by other agents a_j, satisfying

$$\sqrt{\left(X_{a_i} - X_{a_j}\right)^2 + \left(Y_{a_i} - Y_{a_j}\right)^2} \leqq CR, \tag{6}$$

X_{a_i}: X coordinate at which a_i is located
Y_{a_i}: Y coordinate at which a_i is located
CR: Communication radius centering on an agent

3.3 Design of the Pseudo-Communication System

It is difficult to extend the communication system of the current simulation kernel, to introduce a pseudo-communication system. Therefore, our pseudo-communication system uses a new pseudo-communication server that manages

communication among agents, and a new pseudo-communication client for current agents. In the max-sum algorithm, an agent needs to synchronize with the other agents when the agent sends/receives each message. Agents that intend to communicate with the other agents need to send/receive each message repeatedly until all agents lose the intention to continue communication.

The pseudo-communication server receives the following information from clients to confirm whether the communication condition is satisfied:

- X and Y coordinate where the agent is located; and
- intention to continue communication.

When all agents lose the intention to continue communication, the server regards the simulation step as complete and initializes the states of the agents on the server.

3.4 Implementation of a Task Assignment Module for the DCOP with Our Pseudo-Communication System

We implemented a task assignment module (*DCOP Target Detector*) using the pseudo-communication system. This module was implemented by inheriting the task assignment module (*Target Detector*) of RRS-ADF. *DCOP Target Detector* internally manages a pseudo-communication client and assigns a task to its agent based on the communications of a DCOP algorithm. Figure 3 shows the program structure and data flow in the RRS simulator with our extension.

In this approach, the pseudo-communication server is implemented as an independent program with the kernel. Additionally, the server receives information required for communication from agents. However, when we used this system for competitions, we assumed that it was appropriate to implement it as an extension to the communication component of the kernel from the viewpoint of fairness.

In *DCOP Target Detector*, the following procedure is executed:

1. Initialize state (according to the user definition).
2. Receive all messages simultaneously.
3. Execute task assignment based on the DCOP algorithm (according to the user definition).
4. Send all messages simultaneously.
5. Return to step 2 and repeat to step 4, if necessary

In step 3, the module returns the task that is assigned to the agent, and whether a repeat of the procedure is required. In step 5, the module repeats in the case in which a repeat is required in step 3 or until the number of repeats reaches the defined limit. To execute this procedure, *DCOP Target Detector* provides the new methods shown in Table 1. The user then has to define the method shown in Table 2.

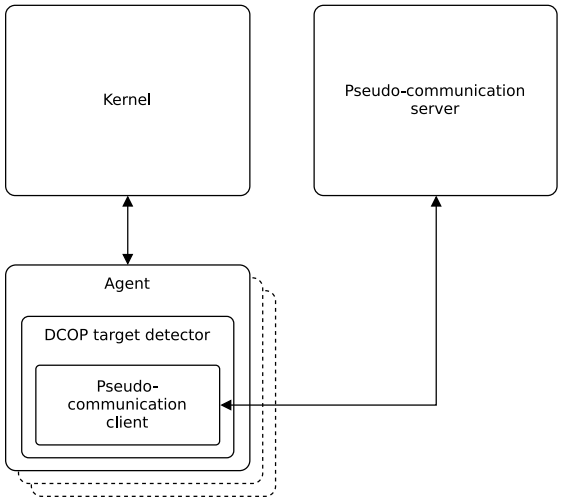

Fig. 3: Part of the program structure and the data flow

Table 1: Methods provided by *DCOP Target Detector*

Definition	Explanation
void send(CommunicationMessage)	Add a message to a blanket sending list
List<CommunicationMessage> receive()	Receive blanket received messages

Table 2: User-defined methods in *DCOP Target Detector*

Definition	Explanation
void initialize()	Initialize the state at the beginning of the procedure
Pair<EntityID, Boolean> improveAssignment()	Execute task assignment

4 Applying the DCOP to RRS and Implementing the DCOP Algorithm

4.1 Modeling the Task Assignment Problem of Ambulance Teams as a DCOP

The task assignment problem is considered as an approach to determine an optimal task with an evaluation function for each agent from a set of tasks recognized by agents. When the definitions of the DCOP described in Subsection 2.1 are applied to the task assignment problem, each element in the definitions has the following meaning:

$\mathbf{A} = \{a_1, \ldots, a_n\}$
denotes the set of all ambulance agents existing in a simulation.

$\mathbf{X} = \{x_1, \ldots, x_n\}$
denotes the set of variables x_i that represent the task selected by agent $a_i \in \mathbf{A}$. A task that is eventually the value of $x_i \in \mathbf{X}$ means a task that a_i will take.

$\mathbf{D} = \{D_1, \ldots, D_n\}$
denotes the set of a task set D_i that agent $a_i \in \mathbf{A}$ can select. $D_i \in \mathbf{D}$ for an ambulance team is composed of the following two elements:

- multiple rescue tasks for multiple civilians that a_i recognizes; and
- a single situation search task that a_i performs to recognize a simulation situation.

The situation search tasks are gathered into only one task because of RRS-ADF handles that assigning civilian rescue tasks and situation search tasks to agents as different problems.

$\mathbf{F} = \{f_1, \ldots, f_k\}$
denotes the set of utility functions f_j that corresponds to the combination of value $\mathbf{x}_i$ of variable $x_i \in \mathbf{X}$. In the evaluation of civilian rescue tasks, the final civilian survival number is the most important. Therefore, it is necessary to minimize the cost of the tasks after assigning the number of agents required to keep a civilian alive in each task. For the reasons stated above, objective function $\mathbf{F}_g(\mathbf{X})$ for civilian rescue tasks can be defined as

$$\mathbf{F}_g(\mathbf{X}) = \sum_{x_i \in \mathbf{X}} C\left(\alpha\left(x_i\right), \mathbf{x}_i\right) + \sum_{d \in \bigcup_{i=1}^{n} D_i} P\left(d, \left|\{\mathbf{x}_i | \mathbf{x}_i = d \wedge x_i \in \mathbf{X}\}\right|\right) \quad (7)$$

using the equations

$$C(a, d) = \begin{cases} \frac{\sqrt{(X_a - X_d)^2 + (Y_a - Y_d)^2}}{\tau} & (\text{if } d \text{ is a civilian rescue task}) \\ 0 & (\text{if } d \text{ is a situation search task}), \end{cases} \quad (8)$$

which is a function that calculates the length of time required by agent a to start to execute task d;

$$P(d, n) = \begin{cases} \rho\left\{1 - \left(\frac{min(REQ(d), n)}{REQ(d)}\right)^2\right\} & (\text{if } d \text{ is a civilian rescue task}) \\ 0 & (\text{if } d \text{ is a situation search task}), \end{cases} \quad (9)$$

which is a function that determines the penalties arising from an insufficient number of agents when n agents are assigned to task d; and

$$REQ(d) = \frac{BD_d \times DT_d}{HP_d} + 1 \quad (10)$$

which is a function that estimates the number of agents required to rescue the civilian in task d by considering the civilian's current state, where

$\mathbf{x}$: value of variable x
BD_d: the buried depth of the civilian in task d
DT_d: physical strength that the civilian in task d loses per step
HP_d: physical strength of the remaining civilians in task d
τ: constant representing an estimated value of the movable distance per step $(0 < \tau)$
ρ: constant representing a penalty $(0 \leqq \rho)$

α: $\mathbf{X} \rightarrow \mathbf{A}$
denotes the function that finds agent $a_i \in \mathbf{A}$ that manages variable $x_i \in \mathbf{X}$. Because the RRS agent cannot perform more than one task simultaneously, the variable managed by agent $a_i \in \mathbf{A}$ is always limited to one variable. Therefore, this function is always bijective.

We then minimize the value obtained from Eq. (7). To obtain the same result as objective function $\mathbf{F}_g(\mathbf{X})$, fitting utility function $f_j \in \mathbf{F}$ for the max-sum algorithm is defined in Subsection 4.2.

4.2 Applying the Max-Sum Algorithm

In this section, we describe the method of forming a factor graph and its utility functions to execute the max-sum algorithm.

Method for Forming a Factor Graph
As described in Subsection 2.2, the factor graph consists of variable nodes that represent variables, factor nodes that represent utility functions between variable nodes, and edges that connect a variable node to a factor node. The factor graph has a feature such that a utility function affects multiple variables represented as factor nodes. Such a utility function is needed for each task, such as function $P(d, n)$ shown in Eq. (9). However, tasks cannot manage the factor nodes. The factor nodes are managed by the closest agent to the task's location instead of the task to solve such a situation. The nearest agent is determined by each agent that received information from the other agents. When an agent for a factor node and an agent for a variable node can communicate with each other according to the restriction of the communication range, an edge is created between the factor node and variable node. As an example of constructing a factor graph, Figure 5 shows the factor graph constructed by an agent and a task that has the relationship shown in Figure 4. The agent for variable node x_3 is the closest to the location of the task, and factor node f_1 for the task is managed by the same agent as x_3.

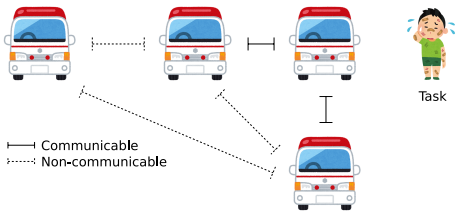

Fig. 4: Example of the communication availability between agents and the task state

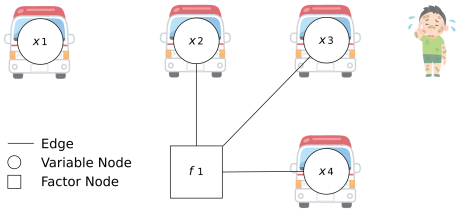

Fig. 5: Factor graph constructed from the state of Figure 4

Definition of the Utility Function
The max-sum algorithm uses utility functions related to agents and factor nodes. Additionally, the utility function related to a single agent is used as the variable node. Thus, objective function $\mathbf{F}_g(\mathbf{X})$ shown in Eq. (7) is divided according to a relationship among the agents and used as a utility function. The following equations, respectively, show utility function $f_{MSV}(x_i)$, which is used for variable node x_i to calculate the cost required for the agent to work the task, and utility function $f_{MSF}(d_j)$, which calculates penalties from the task and a set of neighboring variable nodes used on the factor node related to task d_j:

$$f_{MSV}\left(x_i\right) = C\left(\alpha\left(x_i\right), \mathbf{x}_i\right) \tag{11}$$

$$f_{MSF}\left(d_j\right) = P\left(d_j, |\{\mathbf{x}|\mathbf{x} = d_j \wedge x \in \mathcal{N}\left(d_j\right)\}|\right) \tag{12}$$

Additionally, the method [5] proposed as the cardinality-based potentials is applied to the calculation on the factor node because utility function $f_{MSF}(d_j)$ of the factor node calculates the utility based on the number of assigned agents.

5 Experiment and Consideration

5.1 Experimental Method

In this section, We confirm the effectiveness of our proposed approach. We, therefore, check whether our task assignment method with the max-sum algorithm works properly.

In this experiment, our implemented max-sum algorithm is compared with the Closest[1] and Greedy[2] methods with respect to the task assignment of RRS. These three methods were evaluated in the simulation result, that is, the "score", of RRS through some simulations. In this experiment, the VC3 and the Eindhoven3 scenarios of RoboCup 2018 were used, except entities that were not related to the ambulance team and rescue of civilians were removed from those scenarios. Furthermore, four types of communication range (distance) were applied to each scenario because the results of the simulations were strongly influenced by communication ranges.

Each scenario had the following features:

VC3

Platoon agents were densely located while the locations of refuges and civilians were dispersed.

Table 3: Simulation settings of VC3

Initial score	Refuges	Ambulance teams	Civilians
293.0	5	32	292

Eindhoven3

The locations of platoon agents were more dispersed than for VC3. The locations of refuges and civilians were also dispersed.

Table 4: Simulation settings of Eindhoven3

Initial score	Refuges	Ambulance teams	Civilians
401.0	4	15	400

5.2 Experimental Results

Tables 5 and 6 show the experimental results for each communication range in VC3 and Eindhoven3, respectively. The communication range "1/4" indicates that that communicable range for an agent was 1/4 for the entire map. The communication range "4/4" indicates that agents could communicate globally with each other in the map. The results are averages and standard deviations, which are the numbers in parentheses, of scores for 30 simulations for each algorithm and communication range. Additionally, the results of "Max-Sum" indicate the case in which agents sent and received messages 100 times in each step for task assignment using the max-sum algorithm.

[1] Task assignment with a greedy method on the distance between an agent and a task

[2] Task assignment with a greedy method on the time required for an agent to complete a task

Table 5: Experimental results on VC3

	Communication range			
Agent	1/4	2/4	3/4	4/4
Closest	63.63 (±0.87)	63.87 (±0.82)	64.30 (±1.01)	64.41 (±0.97)
Greedy	56.55 (±0.72)	53.15 (±0.74)	52.19 (±0.58)	52.35 (±0.53)
Max-Sum	62.32 (±1.36)	62.90 (±1.20)	62.85 (±1.39)	62.71 (±1.25)

Table 6: Experimental results on Eindhoven3

	Communication range			
Agent	1/4	2/4	3/4	4/4
Closest	329.18 (±1.89)	328.45 (±1.75)	328.22 (±1.73)	328.15 (±1.69)
Greedy	328.62 (±1.68)	326.43 (±1.07)	325.73 (±1.00)	325.50 (±0.88)
Max-Sum	330.11 (±1.17)	329.88 (±2.27)	330.01 (±1.79)	330.25 (±1.60)

For VC3, the score of Closest was the highest, although Max-Sum was a high score. For Eindhoven3, the results demonstrate that Max-Sum was the highest score.

5.3 Considerations for Experiments

The experimental results show the following from the features of the algorithms.

Closest determines task assignments only from the distance between an agent and the rescue task of a civilian; that, the scenarios in the experiment were only suitable for the Closest, although the results demonstrated a high score.

Greedy determines the task assignment with priorities calculated based on the time for solving the task. However, agents work sequentially according to the priorities only. As a result, agents sometimes cannot work decentrally. This method did not have the worst score if the scenarios included concurrent multiple tasks.

By contrast, Max-Sum could decentrally assign appropriate tasks according to the situation with utility functions for all tasks. Additionally, we confirmed that Max-Sum worked effectively from the results, even if the communication range was restricted. Therefore, we confirmed that the max-sum algorithm is an effective approach for the solution of the task assignment problem of RRS.

The above discussion has demonstrated that the three methods worked properly on RRS. Furthermore, we showed that our proposed extension of RRS-ADF is valid and effective for RRS.

6 Conclusions

In this paper, we proposed an extension of RRS-ADF to apply the DCOP algorithm to the task assignment problem of RRS. We described the necessity of multiple communications within each step for the DCOP as a reason for introducing the extension. As a result, we designed a pseudo-communication system

to realize this communication and implemented *DCOP Target Detector* for that system.

Finally, we confirmed that this extension is sufficient for applying the DCOP algorithm to RRS through designing and implementing the max-sum algorithm on RRS. We also confirmed that our max-sum algorithm implementation worked effectively using some simulations. Additionally, because the max-sum algorithm provided high scores, even in the simulation in which the situation recognition/communication range was restricted, we concluded that the algorithm is effective for the task assignment problem of RRS.

However, this extension is difficult to use in agent competitions because it is an extension to the RRS simulator. When we adopt this extension in our RRS, we need to discuss the specification and configuration in detail according to our purpose. We hope that our extension or idea is useful for the RRS community through some discussions.

References

1. Fioretto, F., Pontelli, E., Yeoh, W.: Distributed Constraint Optimization Problems and Applications: A Survey. Journal of Aritficial Intelligence Research **61**, 623–698 (2018)
2. Kleiner, A., Farinelli, A., Ramchurn, S., Shi, B., Maffioletti, F., Reffato, R.: RMAS-Bench: Benchmarking Dynamic Multi-agent Coordination in Urban Search and Rescue. In: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems. pp. 1195–1196. AAMAS '13, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2013), `http://dl.acm.org/citation.cfm?id=2484920.2485139`
3. Pujol-Gonzalez, M., Cerquides, J., Farinelli, A., Meseguer, P., Rodríguez-Aguilar, J.A.: Binary max-sum for multi-team task allocation in RoboCup Rescue. In: Optimisation in Multi-Agent Systems and Distributed Constraint Reasoning (OptMAS-DCR). Paris, France (2014)
4. Ramchurn, S.D., Farinelli, A., Macarthur, K.S., Jennings, N.R.: Decentralized Coordination in RoboCup Rescue. Comput. J. **53**, 1447–1461 (2010)
5. Tarlow, D., Givoni, I.E., Zemel, R.S.: HOP-MAP: Efficient Message Passing with High Order Potentials. In: AISTATS (2010)
6. Weiss, Y., Freeman, W.T.: On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. IEEE Trans. Information Theory **47**, 736–744 (2001)
7. Zhang, W., Wang, G., Wittenburg, L.: Distributed stochastic search for constraint satisfaction and optimization: Parallelism, phase transitions and performance. Proc. AAAI-02 Workshop on Probabilistic Approaches in Search (01 2002)