RoboCupRescue 2019
TDP Agent Simulation AIT-Rescue (Japan)

Taishun Kusaka[1], Yuki Miyamoto[1], Akira Hasegawa[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. AIT-Rescue 2019 aims to introduce a task assignment strategy using a max-sum algorithm for the distributed constraint optimization problem (DCOP) algorithm. Agents are centrally assigned appropriate tasks by center agents because it is difficult for agents to distributedly assign tasks using the DCOP algorithm on the RRS system to introduce the DCOP algorithm. Thus, our new agents can behave more decentrally than our agents in RoboCup 2018. The results of some experiments confirmed that AIT-Rescue 2019 works better than our agents from last year.

1 Introduction

An important problem on RRS is the task assignment problem, which is the problem of assigning tasks to agents to minimize the costs (or maximize the benefits) of the tasks that each agent works on. The problem can be modeled as the distributed constraint optimization problem (DCOP) in multi-agent systems. The problem modeled as the DCOP is solved by the DCOP algorithm. In this paper, approximate solution methods for the DCOP are called as DCOP algorithms.

RMASBench [3] was proposed as a system to solve the task assignment problem on RRS using a DCOP algorithm and evaluate the algorithm, and to evaluate the effectiveness of the algorithm on the RRS. In RMASBench, a center agent, who can perceive all the information about a target map in a simulation, centrally assigns tasks properly with using DCOP algorithms. Research on RMASBench [6] has demonstrated that the DCOP algorithm works effectively for the RRS task assignment problem. However, for the current RRS, the center agent is restricted to information observable by the simulation system. This makes a distributed approach for task assignment on RRS difficult, even though a DCOP algorithm is essentially a type of decentralized algorithm.

Hence, we implement center agents with a similar algorithm to the designed algorithm on RMASBench and verify effectiveness of our algorithm in the current simulator system of RRS. Almost all operations of the center agents in RMASBench can be replaced with the *Target Allocator* module of RCRS-ADF.

Therefore, we design and implement the ambulance center, fire station and police office, which work centrally for task assignment using the DCOP algorithm, and conduct some experiments to compare our agent with other agents.

In Section 2 we describe DCOP, that is similar to our paper [5] for the infrastructure competition in 2019, a max-sum algorithm as a major algorithm to solve the DCOP, and the behavior of our implemented max-sum algorithm. In Section 3, we present an application the max-sum algorithm on RRS. We describe evaluations for our agents through some experiments in Section 4. As a result, we confirmed that our agent works effectively, even if it is on the current RRS.

2 Modules

In this section, we describe the modules to implemented for AIT-Rescue 2019 in detail. First, we provide definition of the DCOP and behavior of the max-sum algorithm. Second, we briefly provide an overview of the implemented agent of RoboCup 2018 and the implementation to clarify features of our agent in this year.

2.1 Overview

AIT-Rescue 2019 adopts the max-sum algorithm, that is, the DCOP algorithm, for the task assignment problem. The center agents centrally assign tasks to platoon agents in this approach. This results from the difficulty in applying the DCOP algorithm to the platoon agents.

One step of the time difference occurs between sending messages and receiving messages in the RRS system, even though the simulation time of RRS is around 300 steps. the max-sum algorithm is a type of message propagation algorithm, as mentioned in Section 2.3, and requires sending and receiving messages many times. Additionally, the scenario status gradually changes for each step because the simulated disaster space dynamically changes on RRS. Therefore, the simulated disaster space is changed steadily before the max-sum algorithm converges the assignment for each step. In the other words, it is almost impossible that DCOP algorithm behavior with such a communication system on RRS.

Therefore, our agent achieves task assignment using the DCOP algorithm by minimizing such a delay using the following procedure:

- All platoon agents send messages to each center agent.
- The center agent propagates internally pseudo-multiple messages extracted based on the received messages.
- The center agent sends assigned tasks to each agent in turn.

Thus, we implemented *Target Allocator* modules responsible for the task assignment of the center agents in AIT-Rescue 2019, as shown in Figure 1.

Additionally, the modules that we implemented in 2018 are *Target Detector* modules for the task assignment of the platoon agents, *Search* modules responsible for the searching platoon agents, *Ext Action* modules for action control of the

platoon agents, and the *Path Planning* module responsible for the fundamental algorithm to the other modules, as shown in Figure 1.

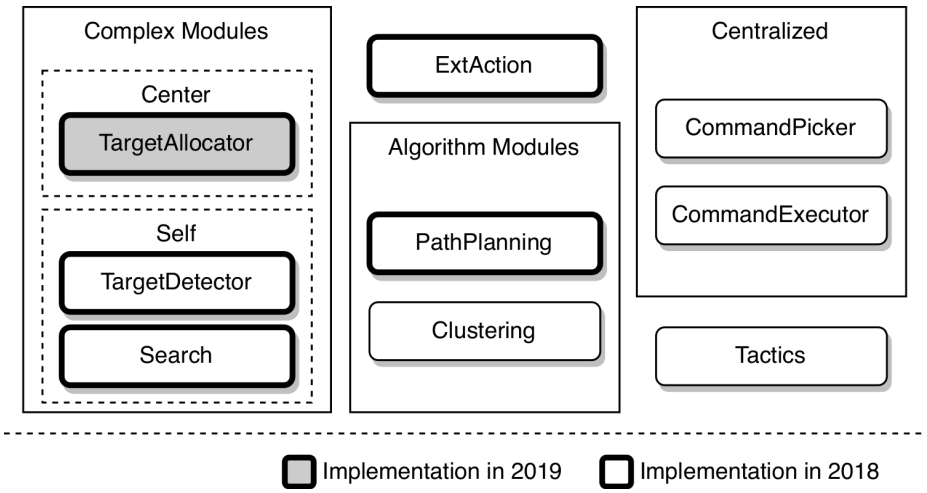

Fig. 1: RCRS-ADF modules and the range of our implementation

2.2 Distributed Constraint Optimization Problem

The distributed constraint optimization problem (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 [2]:

$\mathbf{A} = \{a_1, \ldots, a_n\}$
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 the relationship between an agent and variable. Each agent corresponds to a distinct variable.

Objective function $\mathbf{F}_g(\mathbf{X})$ for an 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, 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 the combination that maximizes the objective function is the optimized assignment.

2.3 Max-Sum Algorithm

The max-sum algorithm is a major approximate solution methods for the DCOP and a type of message propagation algorithm[8]. 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 2.

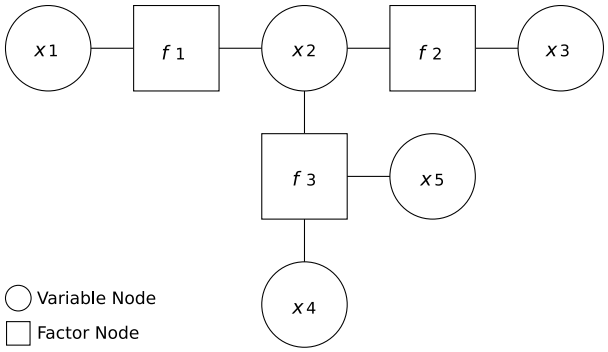

Fig. 2: Example of the factor graph

In Figure 2, the factor node has a constraint, which is a utility function, to neighboring variable nodes. Additionally, a corresponding agent calculates a utility for each variable value according to its own range and the constraints of the neighbor 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 neighbor 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 [6], and is used when the variable node x of the equation sends the utility on a value of the variable node to 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 that is calculated by an appropriate agent and included in the range of variable node x; $\mathcal{N}(x)$ is the set of factor nodes that neighbor variable node x; and $\mu_{g \to x}$ is a utility calculated on factor node g and sent to variable node x.

Eq. (3) calculates the sum of the utilities received from neighbor factor node, 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 [6], and is used when factor node f sends the utility on a value to variable node x:

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

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 the variable node y to the factor node f; and $\mathbf{y}$ is the value of the 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.

2.4 *Target Allocator*

The behavior of the max-Sum algorithm implemented on the *Target Allocator* module of AIT-Rescue 2019 is shown in Algorithm 1. First, each platoon agent sends messages of tasks observed by the agents via radio communication. When a center agent of the fire brigades, police forces, and ambulance teams receives the task messages, the center agent updates the task information held and holds the information to create a factor graph. Additionally, each platoon agent sends a message, MessageReport, to report the successful completion of a task. Therefore, the center agents can remove the completed task from the holding information.

Second, each center agent creates a factor graph, which connects all platoon agents and tasks as variable nodes and factor nodes, respectively. The center

agents repeatedly calculate new utilities by propagating calculated utilities in the previous step on the factor graph. The calculation repeats until the values of the utilities stop changing or the designated iteration number is reached. The utility functions to calculate the utilities are described in Section 3.

Finally, according to the assignment, each center agent sends messages of assigned tasks to the platoon agents.

However, each platoon agent receives the assigned task after two steps from sending the messages because of the delay of one step described above. Additionally, if there is more than one center agent in each type, then the center agent with the minimum *Entity ID* works according to the algorithm.

Algorithm 1 Behavior of the max-sum algorithm on the center agents

Receive task information messages from each platoon agent.
Update the task information held according to the received messages.
Create a factor graph according to the task information held.
for 0 **to** arbitrary iteration number **do**
  Calculate and update the utilities of each value in each nodes.
  Propagate pairs of values and the corresponding utility from each node to its neighbor nodes.
  Decide an assigning task by selecting the value with the maximum utility in each factor node.
  Send messages of assigned task information to each platoon agent.
end for

2.5 Other Modules

AIT-Rescue 2019 uses the same modules as the 2018 version, except those described above, although we will fix them of 2018 to exactly behave strategies that described below before RoboCup 2019. We explain them briefly in this section.

The RCRS-ADF and the RRS-OACIS introduced in 2018 made it easier to conduct a comparison experiment for each module. Therefore, we designed AIT-Rescue 2018 based on the best combination of modules found through many experiments with RCRS-ADF and RRS-OACIS.

In RoboCup 2019, we used the same modules as AIT-Rescue 2018, except for modules mentioned above. The major modules are as follows[4]:

Target Detector This module prioritizes conditions of disaster scenarios that are tasks, and selects a target according to the prioritized conditions in descending order. For example, fire brigades select a target based on the prioritized convex hulls constructed by vertices of burning buildings. Police forces select a destination in each cluster that each police force belongs in, which is an area on a map. Ambulance teams select a target based on the prioritized buried depth and physical strength of observed civilians.

Search This module searches all buildings except for refuges, first, in a cluster that a agents belongs in on a map, and second, in the map, as search-candidates that are referring to buildings. From the candidates, the closest building that *Path Planning* module calculated is selected as a target.

Ext Action First, with respect to a move action, this module attempts to arrive at an observable location of the target even if the path to the target is blockaded. Although fire brigades essentially calculate the shortest path considering blockages, the module simply calculates the shortest path without considering blockages. Second, with respect to disaster relief actions, the module prioritizes some conditions using more detailed disaster scenarios than the *Target Detector* module, and selects an action according to the prioritized conditions in descending order.

Path Planning This module calculates the shortest path using the Dijkstra algorithm[1]. The *Ext Action* module uses this module to calculate a more detailed shortest path than the sample module of RCRS-ADF.

3 Strategies

In this section, we describe an approach to model the task assignment problem on RRS to the DCOP and how to adapt the model to max-sum algorithm. Some parts of the utility functions change in accordance with the types of agent. Because the task assignment problem in RRS differs depending on the type of agents. Therefore, we explain the difference between these task assignment problems for each type of agent.

3.1 Task Assignment Problem in RRS

We model the task assignment problem in RRS as the DCOP, in accordance with the definitions described in Section 2.2 as follows:

$\mathbf{A} = \{a_1, \ldots, a_n\}$
represents the set of all agents that exist in a simulation.

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

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

- multiple disaster relief tasks for various disasters 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 RRS-ADF handles that assigning disaster relief 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 correspond to the combination referring to variables, of value $\mathbf{x}_i$ of variable $x_i \in \mathbf{X}$. We evaluate disaster-relief tasks according to the cost of the tasks, and the penalty based on

the lack of the necessary number of assigned agents to the tasks. The cost indicates the number of necessary steps to move to the task and the penalty indicates a value that differs according to the task assignment for each type of agent.

$\alpha\colon \mathbf{X} \rightarrow \mathbf{A}$
represents the function that determines the agent $a_i \in \mathbf{A}$ that manages the variable $x_i \in \mathbf{X}$. Because the RRS agent cannot perform more than one task simultaneously, the variable managed by the agent $a_i \in \mathbf{A}$ is always limited to one variable. Therefore, this function is always bijection.

From the above definition, an objective function of the task assignment in RRS is defined as

$$\mathbf{F}_g\left(\mathbf{X}\right) = \sum_{x \in \mathbf{X}} C\left(\alpha\left(x\right), \mathbf{x}\right) + \sum_{d \in \bigcup_{i=1}^{n} D_i} P\left(d, |\{\mathbf{x} | \mathbf{x} = d \wedge x \in \mathbf{X}\}|\right) \tag{6}$$

by way of

$$C\left(a, d\right) = \begin{cases} \frac{\sqrt{(X_a - X_d)^2 + (Y_a - Y_d)^2}}{\tau} & \text{if } d \text{ is a disaster relief task} \\ 0 & \text{if } d \text{ is a situation search task} \end{cases} \tag{7}$$

$P\left(d, n\right)$: function that calculates the penalty of n agents assigned to d
$\mathbf{x}$: The value of the variable x
X_a: X coordinate at which a_i its located
Y_a: Y coordinate at which a_i its located
τ: constant that represents the estimated value of the movable distance per step $(0 < \tau)$

Furthermore, we apply the task assignment of RRS modeled as the DCOP to the max-sum algorithm. The max-sum algorithm uses utility functions related to agents and factor nodes. Additionally, the utility function related to a single agent is used on the variable node. Hence, objective function $\mathbf{F}_g(\mathbf{X})$ shown in Eq. (6) is divided according to the relationship among the agents and, used as a utility function. The following shows the utility function $f_{MSV}(x_i)$ that is used for the variable node x_i to calculate the cost required for the agent to work on the tasks:

$$f_{MSV}\left(x_i\right) = C\left(\alpha\left(x_i\right), \mathbf{x}_i\right) \tag{8}$$

and

$$f_{MSF}\left(d_j\right) = P\left(d_j, |\{\mathbf{x} | \mathbf{x} = d_j \wedge x \in \mathcal{N}\left(d_j\right)\}|\right) \tag{9}$$

shows the utility function $f_{MSF}(d_j)$ that calculates penalties from task d_j and the set of neighbor variable nodes used on the factor node related to task d_j. Because utility function $f_{MSF}(d_j)$ calculates penalties according to the number of agents assigned to task d_j, it is considered as cardinality-based potential [7].

3.2 Ambulance Center

Regarding task assignments for the ambulance center, an agent is an ambulance team and a disaster relief task is a civilian rescue task. Information about the tasks is shared between the ambulance center and other agents using *Message Civilian*. Then, a task reported by *Message Report* described in Section 2.4 is regarded as already completed task, and is not assigned an agent.

We define

$$P(d,n) = \begin{cases} \rho\left\{1 - \left(\frac{min(REQ(d),n)}{REQ(d)}\right)^2\right\} & \text{if } d \text{ is a civilian rescue task} \\ 0 & \text{if } d \text{ is a situation search task} \end{cases} \tag{10}$$

as the utility function that calculates a penalty with

$$REQ(d) = \frac{BD_d \times DT_d}{HP_d} + 1 \tag{11}$$

BD_d: 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
ρ: constant that represents a penalty $(0 \leqq \rho)$

3.3 Fire Station

Regarding task assignments for the fire station, an agent is a fire brigade and a disaster relief task is a task to extinguish the fire in a burning building. Information about the buildings is shared between the fire station and other agents using *Message Building*. Then, the burning building neighbor to one or more non-burning buildings is regarded as a task. The task reported by *Message Report* is not assigned to an agent.

We define

$$P(d,n) = \begin{cases} \rho\left\{1 - \left(\frac{min(REQ(d),n)}{REQ(d)}\right)^2\right\} & \text{if } d \text{ is a building extinguish task} \\ 0 & \text{if } d \text{ is a situation search task} \end{cases} \tag{12}$$

as the utility function that calculates a penalty with

$$REQ(d) = \frac{FR_d^2 \times VL_d}{\upsilon} \tag{13}$$

FR_d: burnup of a building in task d
VL_d: volume of a building in task d
υ: constant that represents the estimated volume that an agent can extinguish per step ($0 \leqq \upsilon$)
ρ: constant that represents a penalty ($0 \leqq \rho$)

3.4 Police Office

Regarding task assignments for the police office, an agent is a police force and a disaster relief task is a task to clear all blockages on a road. Information about roads is shared between the police office and other agents using *Message Road*. Then, the road that has a blockage around any police force is regarded as a task. Additionally, the road that another agent has requested to clear from other agents is also regarded as a task. The task reported by *Message Report* is handled similarly to those of other agents.

We define

$$P(d,n) = \begin{cases} PRI(d) \times \rho \left\{1 - min(1,n)^2\right\} & \text{if } d \text{ is a blockage clear task} \\ 0 & \text{if } d \text{ is a situation search task} \end{cases} \tag{14}$$

as the utility function that calculates a penalty with

$$PRI(d) = \begin{cases} \nu & \text{if there has been a request to clear a road in } d \\ \epsilon & \text{if a road in } d \text{ is an entrance to a building} \\ \dfrac{AR_d}{\max\limits_{r \in R} AR_r} & \text{otherwise} \end{cases} \tag{15}$$

AR_d: area of the road in task d
R: set of all roads in the simulation
ν, ϵ: constants that represents coefficients on the specific tasks ($1 \leqq \epsilon \leqq \nu$)
ρ: constant that represents a penalty ($0 \leqq \rho$)

4 Preliminary Results

In this section, we present an experiment to verify the effectiveness of our centralized task assignment with the max-sum algorithm. We use two types of agents to evaluate our centralized task assignment module: one is simply the sample agent of RRS-ADF and the other is the agent who is assigned proper tasks by the DCOP algorithm on the sample agent (AIT-Rescue 2019). However, all platoon agents work only on tasks assigned by the *Target Allocator* modules. If an agent is not assigned to a task, then the agent searches its own surroundings. The constants of the utility functions defined in Section 3 are $\tau = 28000$, $\rho = 600$, $\upsilon = 500$, $\nu = 2$, and $\epsilon = 1.5$ in the experiment. The iteration number described in Section 2.4 is 100.

The maps for this experiment are Eindhoven2, Paris1, Sakae1 used in RoboCup 2018. The maps contain at least one center agent of each type. Additionally, we expand the bandwidth of communication to the maximum and disable its noise because our modules have not responded to such a situation yet.

Table 1 shows the experimental results. The results are averages and standard deviations, which are the numbers in parentheses, of scores for 20 simulations for each agent and map. As a comparison, the table also includes the results of AIT-Rescue 2018.

Table 1: Experimental results

	Scenario		
Agent	Eindhoven2	Paris1	Sakae1
AIT-Rescue 2019	66.83 (±0.86)	15.07 (±6.15)	9.99 (±0.15)
Sample	64.43 (±0.49)	11.40 (±4.32)	9.74 (±0.13)
AIT-Rescue 2018	64.64 (±0.33)	9.98 (±0.25)	9.75 (±0.16)

Table 1 shows that all the scores of AIT-Rescue 2019 are the highest for all maps. Hence, this means that our implemented module worked effectively in RRS. However, the standard deviations of the scores tends to be higher. This implies that our module properly used random search as needed. Therefore, this shows that our agents not assigned to tasks searched for new information.

5 Conclusions

We implemented the *Target Allocator* module on center agents of AIT-Rescue 2019 to assign tasks centrally to platoon agents using the max-sum algorithm. Moreover, we conducted experiments to compare our agent with other agents. As a result, we confirmed that our agent obtained higher scores than those of the other agents.

Therefore, we confirmed that centralized task assignment with the max-sum algorithm works effectively, even in the current simulation system of RRS. We plan to include the idea of corporations among heterogeneous agents for RoboCup 2019. Additionally, we would like to make our agent work even if

the communication among agents includes noise. Agents need to select a task not only using *Target Allocator* but also choose it themselves by referring to a suggestion of *Target Allocator*, to achieve our ideas.

References

1. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1), 269–271 (Dec 1959)
2. Fioretto, F., Pontelli, E., Yeoh, W.: Distributed Constraint Optimization Problems and Applications: A Survey. Journal of Aritficial Intelligence Research **61**, 623–698 (2018)
3. 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
4. Kusaka, T., Miyamoto, Y., Hasegawa, A., Takami, S., Iwata, K., Ito, N.: RoboCup 2018 Rescue Simulation League Team Description AIT-Rescue (Japan) (2017)
5. Miyamoto, Y., Kusaka, T., Okado, Y., Iwata, K., Ito, N.: RoboCup Rescue 2019 TDP Infrastructure AIT-Rescue (Japan) (to appear)
6. Pujol-Gonzalez, M., Cerquides, J., Farinelli, A., Meseguer, P., Rodr¥'¥iguez-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)
7. Tarlow, D., Givoni, I.E., Zemel, R.S.: HOP-MAP: Efficient Message Passing with High Order Potentials. In: AISTATS (2010)
8. 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)