RoboCupRescue 2021
TDP Agent Simulation AIT-Rescue (Japan)

Yuki Okado[1], Toshinari Sakai[1], Akira Hasegawa[1], Hiroya Suzuki[1], Haruki Uehara[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. Our team has developed agents with a task assignment strategy using the Fast Max-Sum algorithm for distributed constraint optimization problems. The agents obtain better results than the AIT-Rescue 2020 agent, which won first prize at JapanOpen 2020. However, major changes in the rules for RoboCup 2021 have largely affected our agents. We developed new agents with a strategy suited to the new rules. This strategy allows our agents to decentralize searches and task execution. The agents achieve better results than rcrs-adf-sample in nine scenarios.

1 Introduction

The task assignment problem is important in the RoboCupRescue Simulation (RRS). The task assignment problem is the problem of deciding which task to select for each agent to minimize the total cost of the agents' work on this task or to maximize the total utility of the agents' work on the task. The task is a problem to be solved by the agents. Such a problem can be expressed as a distributed constraint optimization problem (DCOP) of a multi-agent system. The task assignment problem in the RRS expressed as a DCOP can be solved using a DCOP algorithm.

In RoboCup 2019, we modeled the task assignment problem in the RRS as a DCOP. Additionally, we developed headquarter agents that assign tasks through the Binary Max-Sum algorithm [7], which is a type of DCOP algorithm [3]. The assigning of tasks by the headquarter agents through the Binary Max-Sum algorithm scored more highly than rcrs-adf-sample agents, and the scores exceeded the results of our team at RoboCup 2018. However, the Binary Max-Sum algorithm is intended for static environments and is not designed to run in dynamic environments where the circumstances of agents change over time, such as the RRS. In addition, when applying the Binary Max-Sum algorithm or other DCOP algorithms to the current RRS, there are multiple *steps* in calculating the assignment. This is because most DCOP algorithms require multiple communications between agents to calculate the assignment once. As the assignment of each *step* is calculated on the basis of some prior disaster situation, if the situation changes appreciably between *steps*, it is conceivable that the agents' work will be inappropriate.

This year, we implemented the Fast Max-Sum algorithm [8] for Ambulance Teams in the Agent Development Framework (ADF) extension environment [4] for the DCOP. Our team adopted the old rules[1]. As a result, we realized the assignment of tasks to agents via Fast Max-Sum as a distributed control corresponding to the dynamic environment and examined the effectiveness of the assignment. Although we were aiming to use it in competitions, it is difficult to implement Fast Max-Sum at present in RRS. In addition, the competition is held according to new rules from this year. New rules use the No Fire scenario and bed capacities of Refuges have been included. The Ambulance Teams' Action commands are changed to *Load* and *Unload*, and the Fire Brigades' Action command changed to *Rescue*.

Our team therefore developed a strategy for each platoon agent to deal with the new rules. Ambulance Teams and Fire Brigades will engage in distributed searching tasks and send and receive commands between each other to rescue more civilians. Police Forces will effectively use the module developed in 2020 to clear *blockades* in a distributed manner so that Ambulance Teams and Fire Brigades can conduct their rescue operations.

In Section 2, we describe the DCOP and Fast Max-Sum. We also describe the Merged Clustering module, which has not been published in 2020. In Section 3, we explain the task assignment problem of Ambulance Teams based on the definition of the DCOP, describe the behavior of Fast Max-Sum in the ADF extended environment, and discuss the strategy adopted in competition by AIT-Rescue 2021, the winning agent of JapanOpen 2020. In Section 4, we evaluate the Fast Max-Sum agent and the AIT-Rescue 2020 agent in experiments. The experiments show that the agent being assigned tasks via Fast Max-Sum outperforms AIT-Rescue 2020. We also confirm that AIT-Rescue 2021 outperforms the rcrs-adf-sample agent in many No Fire scenarios.

2 Modules

This section defines the DCOP and Fast Max-Sum and then describes Merged Clustering, which is a module of AIT-Rescue 2021.

2.1 DCOP

The DCOP is the problem of determining a combination of variable values that maximize utility when there is a constraint between a variable that corresponds to a distributed agent and other variables. The DCOP is defined as follows [2].

- $\mathbf{A} = \{a_1, \ldots, a_l\}$
  is a set of agents, where a_i is an agent.
- $\mathbf{X} = \{x_1, \ldots, x_m\}$
  is a set of variables. However, $m \geqq l$, where m is the number of variables and l is the number of agents.

[1] Old rules use the Fire scenario. The Ambulance Teams' Action commands are *Rescue*, *Load* and *Unload*, and the Fire Brigades' Action Command is *Extinguish*.

- $\mathbf{D} = \{D_1, \ldots, D_m\}$
  is a set of ranges for each variable $x_i \in X$, where D_i is the range of 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 expressed as $f_i \colon \times_{x_j \in \mathbf{x}^i} D_j \rightarrow \mathbb{R}$, where $\mathbf{x}^i$ is the set of variables whose constraint relation is denoted 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."
- $\alpha : \mathbf{X} \rightarrow \mathbf{A}$
  is a mapping function that expresses the 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 by the utility functions as

$$\mathbf{F}_g\left(\mathbf{X}\right) = \sum_{f \in \mathbf{F}} f\left(\mathbf{X}^k\right). \tag{1}$$

σ is an arbitrary combination of values for each variable of $\mathbf{X}$. Then, the optimized assignment σ^*, which maximizes $\mathbf{F}_g(\mathbf{X})$, is expressed as

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

In this case, $\mathcal{D}$ is a direct product set of all ranges in $\mathbf{D}$; i.e., a set of possible assignments. Equation (2) shows that σ^* can be obtained only when the combination σ maximizes the objective function.

2.2 Fast Max-Sum

Fast Max-Sum [8] is a Max-Sum algorithm [9] that assumes that each agent cannot engage in multiple tasks at the same time, such as in the case of the RRS, and that the number of tasks increases or decreases over time. Fast Max-Sum can handle changes over time that other Max-Sum algorithms cannot handle, and it can avoid the occurrence of redundant sending utilities that occur for other Max-Sum algorithms.

The target problem of the Fast Max-Sum algorithm is a problem that can be modeled as a factor graph. A factor graph is an undirected graph that comprises 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 Fig. 1.

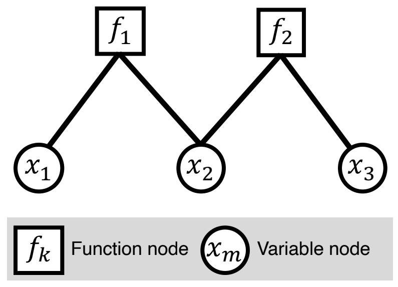


Fig. 1: Example of a factor graph

Each variable node has the utility for each range, and each factor node has the utility for each assignment that can be obtained from the range of the neighboring node. Each node then performs the following processing until the value of the utility does not change or the specified number of iterations is reached.

1. The calculation of the utility value to be sent to the neighboring node from the information currently held by each node
2. The update of the information of each node according to the utility value received from the neighboring node

The value to be sent from the variable node is obtained as

$$\mu_{x_i \to f_j}(a) = -\mu_{f_j \to x_i}(a) + \begin{cases} z_{x_i}(d_j) & \text{if } a = 1 \\ \max_{f_k \in \mathbf{N}_{x_i} \setminus \{f_j\}} z_{x_i}(d_k) & \text{if } a = 0, \end{cases} \tag{3}$$

where

$$z_{x_i}(d_j) = m_{f_j \to x_i}(1) + \sum_{f_k \in \mathbf{N}_{x_i} \setminus \{f_j\}} m_{f_k \to x_i}(0).$$

Equation (3) is used when sending a utility from an arbitrary variable node x_i to a neighboring factor node f_j. The equation calculates the utility $\mu_{x_i \to f_j}(a)$ when x_i selects d_j that is related to f_j if $a = 1$ or x_i does not select d_j if $a = 0$. Here, $\mathbf{N}_{x_i}$ represents function nodes adjoining the variable node x_i.

The value to send from the function node is obtained as

$$\mu_{f_i \to x_j}(a) = \max_{\sigma \in \left\{\sigma \middle| \sigma \in \Sigma_{f_i}, \sigma_{x_j} = a\right\}} \left(f_i(\sigma) + \sum_{x_k \in \mathbf{N}_{f_i} \setminus \{x_j\}} m_{x_k \to f_i}(\sigma_{x_k}) \right), \tag{4}$$

where

$$\Sigma_{f_i} = \prod_{x_r \in \mathbf{N}_{f_i}} \{0, 1\}.$$

Equation (4) is used when sending a utility from an arbitrary factor node f_i to a neighboring variable node x_j. The equation calculates the utility $\mu_{f_i \to x_j}(a)$

when x_j selects d_i that is related to f_i, if $a = 1$ or does not select d_j if $a = 0$. Here, $\mathbf{N}_{f_i}$ represents variable nodes neighboring the function node f_i.

When sending utility to one node, the traditional Max-Sum algorithm calculates the utility for every combination of values taken by variables. Fast Max-Sum focuses on specific tasks. This property reduces the number of sent utilities and the number of utility combinations.

The value $\sigma^*_{x_i}$ finally selected by the variable node x_i is determined using $z_{x_i}(\cdot)$ of Eq. (3) as

$$\sigma^*_{x_i} = \arg\max_{d_j \in D_i} z_{x_i}(d_j)\,. \tag{5}$$

In addition, the traditional Max-Sum algorithm recreates factor nodes when the number of tasks increases or decreases. In Fast Max-Sum, factor nodes can be added or deleted. The assignment result and utility of the previous step are retained. Furthermore, it is not necessary for the variable node x_i to send utility to the function node f_j if it holds that

$$z^n_{x_i}(f_j) = z^{n-1}_{x_i}(f_j) \wedge \max_{f_k \in \mathbf{N}_{x_i} \setminus \{f_j\}} z^n_{x_i}(f_k) = \max_{f_k \in \mathbf{N}_{x_i} \setminus \{f_j\}} z^{n-1}_{x_i}(f_k)\,, \tag{6}$$

where the function $z_{x_i}(\cdot)$ is used by x_i to calculate utility and the n th sending is represented as $z^n_{x_i}(\cdot)$.

2.3 Merged Clustering

In the past, our team could not make an efficient search after the search in the assigned cluster was completed. The Merged Clustering module assigns a cluster to each agent. In addition, this module expands the cluster assigned to an agent by combining multiple clusters. Thereby, an agent that has completed all tasks in its assigned cluster can search for new tasks in the new combined cluster.

The Merged Clustering module assigns clusters and expands the assigned clusters through the following procedure.

1. A map is divided into the same number of clusters as the number of agents using k-means++ [1],
2. Clusters are assigned to agents on a one-to-one basis using the Hungarian method [6],
3. The assigned clusters are expanded by combining n other clusters with the assigned clusters[2].

Let k be the index for the result of the Merged Clustering module, calculated as

$$k = (\text{The number of agents}) \times (n-1) + i, \tag{7}$$

[2] Clusters are compared in terms of the center distance, and n clusters are joined, starting with the closest one.

where i is an index for the result of the k-means ++ calculation and n is the number of clusters to be combined.

Note that if $n = 1$, Eq. (7) is used to calculate the index of the one-to-one assignment of clusters to agents.

3 Strategies

We first describe how to form a factor graph for the task assignment problems of Ambulance Teams with the old rules. We then explain how the DCOP (e.g., variable nodes, factor nodes, and utility functions) is applied to the RRS (e.g., agents, tasks, and RRS scores) when using Fast Max-Sum. In addition, we describe modifications of Fast Max-Sum for implementation in the RRS environment. We finally discuss the strategy of each platoon agent used by AIT-Rescue 2021.

3.1 Factor Graph for the RRS

In terms of the definitions of agents, variables, and utility functions for the DCOP, an agent must form or update a factor graph, which is the environment in which Fast Max-Sum operates, in each *step*.

As mentioned in section 2.2, a factor graph is a graph comprising variable nodes, factor nodes, and edges. A variable node represents a variable whereas a factor node represents a utility function. An edge represents the dependency between a variable and utility function and connects a variable node and factor node. A utility function represented as a factor node on the factor graph affects multiple variables. The utility function is defined for a single task, as described in section 3.2. Therefore, each task is represented by a factor node.

However, a Civilian cannot manage its factor node, and the factor node is managed by the Ambulance Teams instead. Each Ambulance Team grasps the distance between each Ambulance Team and each task using the information received through communication. Then, when the Ambulance Team determines that it is the Ambulance Team closest to a certain task, it manages the factor node that expresses that task.

Additionally, the extended environment of Miyamoto et al. [4] uses voice communication. Scenarios limit this voice communication range. Owing to this limitation, the messages required by the algorithm cannot be sent/received between all Ambulance Teams. Therefore, the problem cannot be represented using a complete bipartite graph, and it is necessary to limit the range of variables according to the communicable range. For these reasons, the edges of the graph are only connected

– between a variable node that represents the task selected by the Ambulance Team with any *ID*:n and factor nodes managed by the Ambulance Team with *ID*:n

– between a variable node managed by the Ambulance Team with *ID*:n and the factor nodes managed by other Ambulance Teams that can communicate with *ID*:n

Each agent thus needs to identify other agents that can communicate with each other at the beginning of each *step*.

Figure 2 is an example of the relationships between agents and tasks. The Ambulance Team with *ID*:1 and *ID*:2 can communicate with each other. However, the Ambulance Team with *ID*:3 cannot communicate with either *ID*:1 or *ID*:2. At this time, the factor graph looks like Fig. 3. x_1 represents the task selected by the Ambulance Team of *ID*:1, x_2 represents the task selected by the Ambulance Team of *ID*:2, and x_3 represents the task selected by the Ambulance Team of *ID*:3. The Ambulance Team with *ID*:2 that manages the variable node x_2 is closest to the location of the task, and the factor node f_1 is managed by the Ambulance Team with *ID*:2.

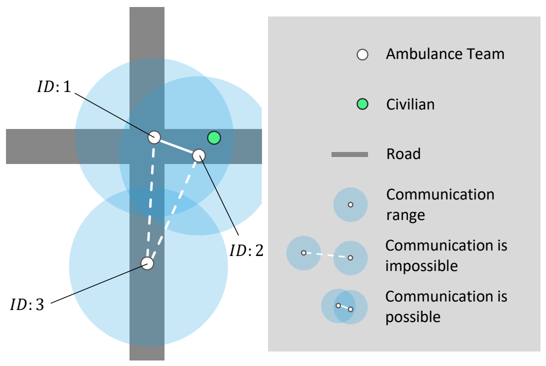
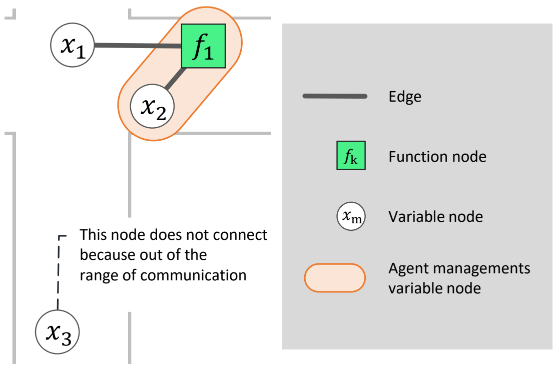


Fig. 2: Example of agents and tasks



x_1
f_1
x_2
x_3
This node does not connect because out of the range of communication
Edge
f_k
Function node
x_m
Variable node
Agent managements variable node

Fig. 3: Factor graph corresponding to Fig. 2

3.2 Task Assignment Problem in the RRS

We model the Ambulance Teams according to the rules used until 2019 by associating the task assignment problem in the RRS as the DCOP, adopting the definitions given in Section 2.1, as follows.

$\mathbf{A} = \{a_1, \dots, a_l\}$ represents the set of all Ambulance Teams in the simulation.

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

$\mathbf{D} = \{D_1, \dots, D_l\}$ denotes the set of task sets D_i that Ambulance Team $a_i \in \mathbf{A}$ can select. $D_i \in \mathbf{D}$ for an Ambulance Team comprises two elements:
- multiple Civilian rescue tasks that a_i recognizes and
- a single situation search task that a_i performs to recognize a simulation situation.

We implement Fast Max-Sum only in the TargetDetector module. Therefore, if Ambulance Team a_i wants to detect a simulation situation, it does not use TargetDetector but the Search module. In other words, Fast Max-Sum does not select a search target in the TargetDetector module. Hence, there is a single situation search task.

$\mathbf{F} = \{f_1, \dots, f_k\}$ denotes the set of utility functions f_j that correspond to the combination referring to variables, of the value $\mathbf{x}_i$ of the variable $x_i \in \mathbf{X}$. We evaluate Civilian rescue tasks according to the cost of the tasks and the penalty based on the lack of the necessary number of Ambulance Teams assigned to the tasks. The cost indicates the number of *steps* required to move to the task.

$\alpha\colon \mathbf{X} \rightarrow \mathbf{A}$ represents the function that determines the Ambulance Team $a_i \in \mathbf{A}$ that manages the variable $x_i \in \mathbf{X}$. Because no Ambulance Team can perform more than one task simultaneously, only one variable can be managed by the Ambulance Team $a_i \in \mathbf{A}$. Therefore, this function is always bijective.

The final number of Civilian survivors is of paramount importance in an Ambulance Team task. It is therefore necessary to allocate the number of agents required to keep a Civilian alive and minimize the cost of rescue of that Civilian. The objective function $\mathbf{F}_g(\mathbf{X})$ for the Civilian rescue task that the Ambulance Team is working on is given by equations (8), (9), (10), and (11). The objective function in equation (8) is then minimized. The function in equation (9) gives the time required for Ambulance Team a to start working on task d. The function in equation (10) gives the penalty for there being an insufficient number of agents when an Ambulance Team is assigned n to task d. The function in equation (11) estimates the number of Ambulance Teams required to save the Civilian of task d considering the current state of the Civilian.

$$\mathbf{F}_g\left(\mathbf{X}\right) = \sum_{x \in \mathbf{X}} C\left(\alpha\left(x\right), \mathbf{x}\right) + \sum_{d \in \bigcup_{i=1}^{l} D_i} P\left(d, \left|\left\{\mathbf{x} \middle| \mathbf{x} = d \wedge x \in \mathbf{X}\right\}\right|\right) \tag{8}$$

$$C\left(a, d\right) = \begin{cases} \frac{\sqrt{(X_a - X_d)^2 + (Y_a - Y_d)^2}}{\tau} & \text{(If } d \text{ is a Civilian rescue task)} \\ 0 & \text{(If } d \text{ is a situation search task)} \end{cases} \tag{9}$$

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

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

$\mathbf{x}$: Current value of variable x
X_a: X coordinate of a
Y_a: Y coordinate of a
BD_d: Buriedness of d
DT_d: HP that the Civilian in task d loses per *step*
HP_d: Remaining HP of d
τ: Constant that represents an estimate of the distance that can be moved in each *step* $(0 < \tau)$
ρ: Constant that represents a penalty $(0 \leqq \rho)$

When Fast Max-Sum is applied to the task assignment problem in the RRS modeled as the DCOP as described above, the utility function f_{FMS} used at factor node f_j representing a certain task d_j is

$$f_{FMS} = \sum_{x \in N_{f_j}} C(\alpha(x), d_j) + P(d_j, |\{\mathbf{x}|\mathbf{x} = \mathbf{d_j} \wedge \mathbf{x} \in \mathbf{N_{f_j}}\}|), \tag{12}$$

where N_{f_j} represents a variable that depends on f_j.

3.3 Unnecessary utility

Fast Max-Sum calculates assignments multiple times in 1 *step.* Then, when the number of assignment calculations reaches a certain number, the final assignment is the assignment of that step. Each node sends utilities for each assignment calculation in 1 *step.* Then, Each node calculates the next utilities and assignment based on the received utility. Our team sends utility by voice communication. Sending utility from all nodes is synchronized because in the ADF extended environment [4], voice communication is synchronized between agents.

Normally, in Fast Max-Sum, when sending utility, each node sends utility all at once. However, in Fast Max-Sum implemented by our team, when utility is sent an odd number of times, it is sent only from a factor node. In the case of utility being sent an even number of times, it is sent only from a variable node. This makes it possible to reduce the number of sending times without affecting the assignment results. Figure 4 and Table 1 are example of utility sending impremented by our team. Figure 4 shows the factor graph and the direction of the utility sent according to the number of assignments. Table 1 shows the sending nodes according to the number of assignments in the graph of Figure 4.

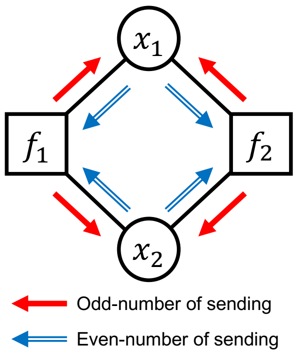


Fig. 4: Direction of messages

Number of calculate	Sender nodes
1	x_1,x_2
2	f_1,f_2
3	x_1,x_2
4	f_1,f_2
⋮	⋮

Table 1: Sender nodes when even or odd

Thus, the utility sends in a single assign calculation is only the utilities sent by the variable node either the utility sent by the factor node.

We implement that the variable node determines the assignment when the 100th utility is calculated. At this time, the utility is calculated using the utility sent from the factor node for the 99th time. This is the same before the 100th time. When all nodes send utility at once, the even-numbered utility sent by the factor node is not used to calculate the even-numbered utility at the variable node. This is also the case when the factor node and odd numbers and even numbers are reversed. Therefore, each node does not need to calculate/sending all at once. That is to say, it is sufficient for the variable node and the factor node to send utility with different timings.

We begin the utility calculate/sending from the factor node. The variable node uses only the messages sent by the neighboring factor node when calculating utility. Therefore, when the utility calculate/sending begins from the variable node, it becomes difficult to consider the surroundings. Meanwhile, the factor node uses the message and utility function from the neighboring variable node when calculating utility. By calculating/sending utility from the factor node, the assignment can be calculated with consideration of the surrounding environment.

3.4 Ambulance Teams

An Ambulance Team is responsible for transporting as many injured Civilians as possible. Each Ambulance Team searches for collapsed buildings in the cluster assigned to it. There will be no search uncollapse buildings because there are no Civilians. The clusters are created using the Merged Clustering module. The Hungarian algorithm [6] assigns Ambulance Teams to each cluster on a one-to-one basis such that the distances between the initial positions of the Ambulance Teams and clusters are minimized. Owing to a change in rules, Ambulance Teams can no longer *Rescue* Civilians, and cooperation with the Fire Brigade, which can *Rescue* Civilians, is thus important. To this end, the Fire Brigade is linked to Ambulance Teams through radio communication.

Calling a Fire Brigade for rescue When an Ambulance Team finds a Civilian buried in a collapsed building, the Ambulance Team sends information of the target Civilian to the Fire Brigade via radio communication and requests them to rescue the Civilian. At the same time, the Ambulance Team calculates the time that it will take to complete the *Rescue* of the Civilian and plan to return to the building at that time. In this way, rapid transport is made possible.

Search and transport A voice-based search [5] described in the 2020 TDP is adopted to efficiently search for Civilians in buildings. Additionally, because capacities have been newly set for Refuges, it is necessary to determine to which Refuges Civilians should be transported. However, information on the number of beds available at a Refuge can only be obtained by the agents near the Refuge or the Ambulance Center and cannot be shared with remote agents through communication. It is therefore difficult to obtain the latest information on Refuges, and a Civilian is thus transported to the nearest Refuge.

3.5 Fire Brigade

The Fire Brigade aims to *Rescue* many disaster rescue agents so that they can conduct their own tasks and as many Civilians as possible such that they survive at the end of the simulation. This is because the survival rate of Civilians has a greater effect than the *HP* of Civilians on the score. Owing to a change in rules for 2021, the Fire Brigade can only *Rescue*. Therefore, if the target is a Civilians, it is critical to cooperate with an Ambulance Team that can transport the Civilian. When the rescue is complete, a transport request is sent to an Ambulance Team via radio communication.

Rescue activities that prioritize the survival of disaster rescue agents and Civilians As soon as the Fire Brigade spots a Civilian or disaster rescue agent in need of rescue, it will rescue them if it can. Because the Fire Brigades cannot identify the target through the exterior walls of buildings and *Blockades* interfere with the Fire Brigades' move. If the Fire Brigade finds multiple candidates, it determines the target to *Rescue* according to the priorities given in Table 2.

The rescue target having the highest priority is a disaster rescue agent in the cluster assigned to the Fire Brigade. The aim of this prioritization is to increase the number of active disaster rescue agents in the initial stage of the disaster and thus increase the rescue efficiency. When a buried Civilian is found, the Fire Brigade determines whether it can rescue the Civilian according to the *steps* calculated by

$$(\text{time to reach} + \text{buriedness}) * \text{damage}. \tag{13}$$

Table 2: Priority of the rescue target for our Fire Brigades

Priority	Rescue target
Highest	Agent in the cluster
High	Civilian in the cluster
Medium	Agent outside the cluster
Low	Civilian outside the cluster

If there are no more targets to rescue, the Fire Brigade will search for more targets in the same way as an Ambulance Team does.

Calling the Ambulance Teams for transport A Fire Brigade cannot transport Civilians and therefore sends information of the target Civilians to an Ambulance Team, with a request to transport the Civilians, after the buried Civilians are rescued. The Ambulance Team that receives this message transports the targeted Civilians.

3.6 Police Force

The Police Force clears as many *blockades* as possible so that other agents can reach their destinations smoothly. Our Police Force thus prioritizes the *highways* [5] identified by the Passable Path Planning module [5], essential buildings such as Refuges, and the destinations of other agents as communicated. The Police Force then uses the prioritization of entities to select the *blockades* to be cleared in the most effective manner.

Each Police Force selects a road/building in its assigned cluster from among the prioritized entities. The Police Forces thus work on single tasks with no overlap among the Police Forces. When the initial positions of the Police Forces are concentrated at the same location, they likely work on the same road if it is included on the path to each task. We handle this problem by dispersing the Police Force using the Passable Path Planning module.

Dividing tasks with clustering modules Each of our Police Forces selects a road/building within its assigned cluster and clears some *blockades* while moving there. Then, if there are any immovable agents/Civilians in its perceivable range, the Police Force clears the associated blockades.

Table 3 lists the priority of each condition encountered when the Police Forces select a task. If more than one task has the highest priority, the closest task is selected. Additionally, the assigned cluster is expanded using the Merged Clustering module when there is no selectable task in the currently assigned cluster.

Table 3: Priority of conditions considered by our Police Forces

Priority	Condition
Highest	If there is any of the following: – A building (in the cluster) initially containing an agent – A Refuge (in the cluster) – An agent/Civilian prevented from moving by any *blockade* in the perceivable range – A destination road/building (in the cluster) communicated by another agent
High	*Highways* (in the cluster)
Medium	Buildings (in the cluster)
Low	Roads (in the cluster)

Using the Passable Path Planning module in the early stages of the disaster When initially outside the cluster, each of our Police Forces moves toward its assigned cluster and clears *blockades* that are obstructing movement. There will be redundancy if multiple Police Forces work on the same road as a result of being initially located in similar positions and having similar paths to their assigned clusters. Furthermore, each activity in the assigned cluster will be delayed while a Police Force clears *blockades*.

Hence, each Police Force moves to its assigned cluster according to a path generated by the Passable Path Planning module. The module derives a path that may not be the shortest in terms of the physical length but is the shortest in terms of the number of *steps*. Additionally, no Police Force considers any *highway*, for which there may be overlap with other Police Forces, outside of its cluster. The reduced redundancy comes from each Police Force moving along a different path. Moreover, each Police Force can start work in its assigned cluster quickly because it avoids *blockades* outside of this cluster.

4 Preliminary Results

In this section, we conduct experiments to examine the effectiveness of our implementation of Fast Max-Sum on TargetDetector (henceforth, Fast Max-Sum agent) and AIT-Rescue 2021 with the strategy introduced in this TDP. The conditions and results of each experiment are presented below.

4.1 Results of Fast Max-Sum

Experiments with Fast Max-Sum agents are performed in the ADF extension environment for the DCOP developed by Miyamoto et al. [4]. Therefore, each agent can communicate multiple times in one *step*. The scenario is that used in the RoboCup 2019 Final. This is the latest scenario of the old rules, which were used in international competition. However, the following changes are made to the scenario for Fast Max-Sum execution.

- There are no *blockades* or fires.
- There are no platoon agents or headquarter agents other than the Ambulance Teams.
- The score is the number of surviving Civilians.
- The number of Ambulance Teams is limited to 10.[3]

We compare our results with those of AIT-Rescue 2020, the winning team of JapanOpen 2020. However, because the execution environment of the AIT-Rescue 2020 agent is an unextended ADF environment, multiple communications in a single step are not possible. Values are thus presented for reference only. The only difference between the two agents is the TargetDetector module, with the other modules remaining the same.

These agents have not included randomness. Therefore, we will compare the results of one experiment under each scenario.

The results of the experiments are presented in Table 4.

Table 4: Fast Max-Sum experimental results

	Team	
Scenario	**Fast Max-Sum**	**AIT-Rescue 2020**
berlin2	187	184
eindhoven2	226	225
paris2	193	187
sf2	242	240
SydneyS2	253	247
vc2	151	149

The experimental results show that the number of surviving Civilians for the Fast Max-Sum agent is higher than that for the AIT-Rescue 2020 agent in six out of six scenarios. This indicates that the decentralized control of task decisions by Fast Max-Sum is an effective method for disaster rescue agents.

In future work, it will be necessary to implement Fast Max-Sum agents that support the new rules, implement other DCOP algorithms, and increase the number of agents that can run Fast Max-Sum.

4.2 Results of Agent_AIT-Rescue 2021

To investigate the effectiveness of the AIT-Rescue 2021 agent, we conduct a comparative experiment with other agents. We compared the number of Civilians

[3] This is the same number as adopted in the experiment conducted by Ramchurn et al. [8]

in Refuge. The rcrs-adf-sample is used in the comparison because it is impossible to obtain agents from other teams that support the new rules. Excluding the test scenario, we consider 11 rcrs-server No Fire scenarios.

The experimental results are given in Tables 5.

Table 5: AIT-Rescue 2021 and rcrs-adf-sample Experimental results

	Team	
Scenario	**AIT-Rescue 2021**	**rcrs-adf-sample**
berlin	57	23
eindhoven	116	33
istanbul	119	65
joao	35	8
kobe	157	173
montreal	22	67
ny	135	102
paris	83	83
sakae	43	4
sf	35	5
vc	196	109

We see from the experimental results that the number of Civilians in Refuge of our agent are higher than or equal to those of rcrs-adf-sample in 9 out of 11 scenarios. This suggests that the modules and strategies that we have described here work effectively in disaster relief.

However, in two scenarios, the number of Civilians in Refuge is lower than that of rcrs-adf-sample. This is because the rescue priority of platoon agents is higher than that of civilians, which may increase the number of deaths.

It is therefore thought that the number of Civilians in Refuge can be improved by making it possible to judge whether the rescue of platoon agents or that of Civilians should be prioritized with the passage of time and the change in situation.

5 Conclusions

Our team has implemented TargetDetector with Fast Max-Sum for the proper task assignment of Ambulance Teams. We were thus able to achieve better results than AIT-Rescue 2020, the winning team of JapanOpen 2020, in all six scenarios that we tested. This result suggests that it is effective to use Fast Max-Sum to determine the task of an Ambulance Team. However, this agent cannot be used in competition because it requires an ADF extension to run and does not support the new rules.

We therefore implemented AIT-Rescue 2021, an agent suited to the new rules, to rescue more agents and civilians. AIT-Rescue 2021 has the following features.

– The number of rescue targets is increased as needed by expanding the area of responsibility through the Merged Clustering module.

– Cooperation between different types of agent is facilitated through communication.
– The number of agents out of action is reduced by dispersing agents and prioritizing the *Clear* of roads deemed critical.

As a result, AIT-Rescue 2021 outperformed the rcrs-adf-sample agent in 9 of the 11 scenarios that are standard in rcrs-server.

However, the rcrs-adf-sample outperformed AIT-Rescue 2021 agent in 2 scenarios. This was because the rescue of platoon agents was prioritized over that of the civilians in the case of AIT-Rescue 2021. We therefore need to make changes to determine the priority target flexibly.

Additionally, AIT-Rescue 2021 does not consider the bed capacities of refuges in determining the destination. It is therefore difficult to obtain results for scenarios where bed capacities are low. We need to implement a method of determining the refuge destination with consideration of the bed capacity of the refuge.

References

1. Arthur, D., Vassilvitskii, S.: k-means++: the advantages of careful seeding. In: SODA '07 (2007)
2. Fioretto, F., Pontelli, E., Yeoh, W.: Distributed constraint optimization problems and applications: A survey. CoRR **abs/1602.06347** (2016), `http://dblp.uni-trier.de/db/journals/corr/corr1602.html#FiorettoPO16`
3. Kusaka, T., Miyamoto, Y., Hasegawa, A., Iwata, K., Ito, N.: RoboCupRescue 2019 TDP Agent Simulation AIT-Rescue (Japan) (2019)
4. Miyamoto, Y., Kusaka, T., Okado, Y., Iwata, K., Ito, N.: RoboCupRescue 2019 TDP Infrastructure AIT-Rescue (Japan) (2019)
5. Miyamoto, Y., Kusaka, T., Okado, Y., Sakai, T., Hasegawa, A., Uehara, H., Ito, N., Iwata, K.: RoboCup Japanopen 2020 ONLINE Rescue Simulation AIT-Rescue (Japan) (2020)
6. Munkres, J.: Algorithms for the Assignment and Transportation Problems. Journal of the Society for Industrial and Applied Mathematics **5**(1), 32–38 (1957)
7. 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) (2014)
8. Ramchurn, S., Farinelli, A., Macarthur, K., Jennings, N.: Decentralized coordination in robocup rescue. Comput. J. **53**, 1447–1461 (10 2010). https://doi.org/10.1093/comjnl/bxq022
9. 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)