# RoboCupRescue 2024
# TDP Agent Simulation AIT-Rescue (Japan)

Ryoya Maeda[1], Haruki Uehara[1], Joe Fujisawa[1], Itsuki Matsunaga[1], Ryosuke Suzuki[1], Kouta Kato[1], Yuki Shimada[1], Shuntarou Fujii[1], Keisuke Ando[1], Takeshi Uchitane[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 models the task assignment problem of RRS as a layered distributed constraint optimization problem, aiming to realize cooperative rescue activities using multiple types of agents. The experimental results confirmed that the agents designed based on this model can realize cooperative rescue activities among multiple types of agents. However, we could not introduce these ideas because of communication limitations. In addition, this year we implemented a new message class for communication between agents and improved the way the Police Force clears debris. These improvements have resulted in more efficient communication between agents and more efficient debris cleaning by the Police Force, allowing for more efficient rescue operations. Consequently, we achieved better results than our agents from the previous year.

## 1 Introduction

In recent years, AIT-Rescue has been developing a method for RRS task assignment, modeling the order and deadline of task execution as a layered distributed constraint optimization problem (L-DCOP). However, conventional L-DCOP modeling only considers one type of agent and does not allow multiple types of agents to cooperate in rescue activities. Therefore, in this study, we aimed to develop a model in which multiple types of agents can cooperate with each other to perform rescue activities, taking into account the order of tasks and deadlines. Subsequently, we implemented this model, confirming that agents based on the proposed model can assign tasks considering task order and deadlines, allowing cooperative rescue activities to be realized by multiple types of agents. However, due to RRS communication limitations, we could not apply the aforementioned research.

In addition, this year, we focus on communication, unlike previously, implementing a new message class that sends only the information necessary for communication between agents. We also implemented a module that makes debris clearing by the Police Force more efficient. Using this implementation, we succeeded in surpassing our agent's score from the previous year in 8 out of 10 disaster scenarios.

In Section 2, the modeling and implementation of the RRS task-assignment problem by L-DCOP is explained, further reporting on the experiments conducted to compare the rescue results of the RRS task-assignment problem modeled using L-DCOP with those modeled using DCOP. In Section 3, the modules used in AIT-Rescue 2024 are described. In Section 4, the strategy for each agent implemented in AIT-Rescue 2024 is explained. In Section 5, the experiments conducted to evaluate the effectiveness of the implemented strategies are described. In Section 6, we summarize TDP and describe the issues to be addressed in the near future before RoboCup2024.

## 2 Scientific challenge

### 2.1 Background

In a previous study, the rescue and transport of civilians in the task allocation problem of RRS were modeled by layered DCOP (L-DCOP), and an agent was designed to allocates the rescue task to fire brigades and performs rescue activities [2]. However, only one type of agent was considered, and the agent was not fully evaluated.

The task allocation problem is important in RRS. In task allocation problems, tasks are appropriately allocated to agents, such as rescuing civilians, transporting civilians, and removing blockades. This problem can be expressed as a distributed constraint optimization problem (DCOP), and it can be solved using DCOP algorithms [1]. DCOP cannot express the time window and ordering constraints on tasks in RRS. However, must be completed within a certain amount of time and thus have time window constraints. The RRS tasks also have ordering constraints that must be executed between different types of tasks.

An extended DCOP, called the L-DCOP, is proposed to solve the task allocation problem with ordering and time window constraints [4]. L-DCOP considers the ordering constraints by expressing them as layers and solves the DCOP in each layer. L-DCOP can also consider time window constraints by ordering the task execution of all agents, following the procedures described below:

1. Create a precedence graph by layering tasks according to ordering constraints.
2. Solve the task allocation problems of each layer using a DCOP algorithm and decide on the tasks to allocate to agents.
3. Each agent generates its simple temporal network(STN).
4. Each agent selects the most efficient order of task execution from the STN.

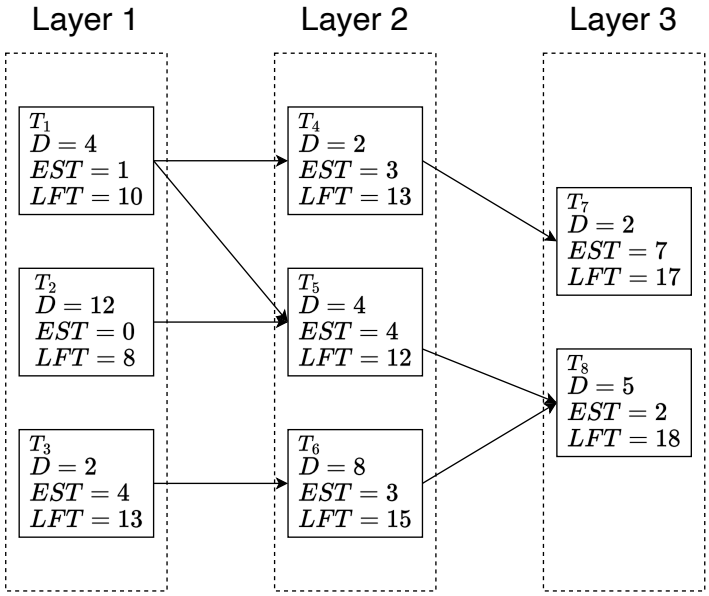


Fig. 1: Example of a precedence graph

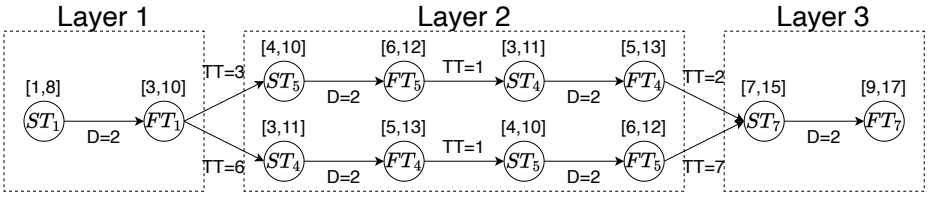


Fig. 2: An example of an STN

Fig. 1 shows an example of a precedence graph created in Procedure 1, with eight tasks: $T_1$ to $T_8$. Each node represents a task with an ID, earliest start time ($EST$), latest finish time ($LFT$), and duration ($D$). The edges show the ordering constraints.

Fig. 2 shows an example of the STN in Procedure 3. This STN is created by the agent allocated to tasks $T_1, T_4, T_5, T_7$. The nodes represent the start time ($ST$) or finish time ($FT$) of a task. The edge indicates the duration ($D$) of a task or the travel time ($TT$) between two tasks. The edge intervals indicate the ordering constraints of the tasks. An interval on $ST_i$ is $[EST, LFT - D]$ and an interval on $FT_i$ is $[EST + D, LFT]$. The order of task execution starts with the task belonging to the upper layer. In case of tasks in the same layer, the order of all the tasks should be considered.

After creating STN, the agent selects the order of task execution that satisfies the time window constraints of all tasks, aiming to complete all tasks at the earliest, following the task order. In the example in Fig. 2, there are two possible orders of task execution: $T_1, T_5, T_4, T_7$ and $T_1, T_4, T_5, T_7$. The former task execution order was chosen because the latter cannot satisfy the time window constraint of $T_5$.

### 2.2 Modeling the task allocation problem in RRS by L-DCOP

The task allocation problem in RRS was modeled using L-DCOP. In L-DCOP, the ordering constraints of tasks are represented by a precedence graph. In RRS, rescue tasks are placed in Layer 1 of the priority graph and transport tasks are placed in Layer 2. This is because there is an ordering constraint between rescue and transport tasks. The transport task cannot be executed until the rescue task is completed.

Next, the DCOP algorithm solves the task allocation problem for each layer. In RRS, rescue tasks are in Layer 1; therefore, the task allocation problem for Layer 1 allocates fire brigades to the rescue tasks. Transport tasks are in Layer 2; therefore, the task allocation problem in Layer 2 involves locating ambulance teams for transport tasks.

Finally, each agent creates an STN and selects the most efficient task execution order. In RRS, tasks placed in the same layer are allocated to an agent. Therefore, each agent creates its STN individually without considering the ordering constraints. The agent then selects the order of task execution in the STN, which allows it to complete the task most quickly while satisfying the time window constraints of the tasks.

### 2.3 Design of a agent

To solve the task allocation problem of rescue and transport, we designed an agent for L-DCOP, considering the task constraints and heterogeneous agents in RRS to realize task allocation. The rescue task entails rescuing buried civilians, and the transport task involves transporting injured civilians to a refuge while they are still alive.

The agent designed in the previous study allocated only fire brigades to rescue tasks using the following procedures [2]:

1. Represent all rescue and transport tasks in a priority graph.
2. Iteratively allocate rescue tasks to fire brigades, create STNs, and subsequently determine the order of task execution.
3. Each fire brigade starts rescue activities based on the task execution order.

In this study, the task of allocation transport task to ambulance teams was added. The agent in this study solved the task allocation problem and executed the rescue activity according to the following procedure:

1. Represent all rescue and transport tasks in a priority graph.
2. Iteratively allocate rescue tasks to fire brigades, create STNs, and subsequently determine the order of task execution.
3. Fire brigades send orders to the ambulance teams for task execution, as determined in Procedure 2, updating the earliest start time of the transport task based on the received order.
4. Iteratively allocate transport tasks to ambulance teams, create STNs, and subsequently determine the order of task execution.
5. Each agent starts rescue activities based on the order of task execution.

In Procedure 1, a priority graph is created to represent the order relationship between tasks. Here, rescue tasks are placed in Layer 1 and transport tasks in Layer 2 of a priority graph, according to the order relationship between rescue and transport tasks. Note that the rescue and transport tasks are mapped one-to-one and target the same civilians.

In Procedure 2, fire brigades determine the allocation of rescue tasks to the brigades and the order of task execution. For this purpose, the entire fire brigade first allocates some of the rescue tasks to fire brigades using the DCOP algorithm. The fire brigade to which the task is allocated, adds this task to its STN. The fire brigade then calculates the best order of task execution using the STNs, and maintains only that order. The above process is repeated until no more rescue tasks can be allocated to fire brigades, and the order of task execution is determined. The best order for task execution is the order that minimizes total travel time and task duration among the orders that satisfy the following conditions. If there is order of the task execution satisfies these conditions, the rescue task is not added to the STN.

- The time window constraints of all tasks can be satisfied by performing rescue activities in the order of task execution.
- The previously maintained order of task execution is not disrupted. For example, when $T_k$ is added while keeping the order of task execution $< T_i, T_j >$, the only possible orders are $< T_k, T_i, T_j >, < T_i, T_k, T_j >, < T_i, T_j, T_k >$.

The reason for not creating the STN after all task allocations have been completed in Procedure 2, is to prevent the computational effort required to create the STN from exploding with the increasing number of tasks. After the

rescue tasks have been allocated to fire brigades and the last rescue task has been added to the STN, each fire brigade decides to perform the rescue activity in the order of task execution that it has maintains.

In Procedure 3, all fire brigades send the task execution orders to the ambulance teams. An ambulance team considers the earliest start time of the transport task to be the time when the corresponding rescue task is expected to be completed, based on the order received from the fire brigades.

In Procedure 4, ambulance teams determine the allocation of transport tasks to the teams and the order of task execution. The details are the same as those in Procedure 2. However, the best order of task execution calculated using the STN is the order that minimizes total of the travel times, task duration, and waiting time until the earliest start time of the task.

In Procedure 5, each fire brigade and ambulance team start rescue activities according to the order of task execution, as determined in Procedures 2 and 5.

### 2.4 Experiment

In the experiments, we confirmed that the agents designed in this study were able to perform task allocation considering the time window and ordering constraints with the cooperation of heterogeneous agents. The experimental method simulates the rescue activities of the following two types of agents in multiple scenarios, as described below, and subsequently measures and compares the number of civilians transported and their rescue activities. Here, based on the number of civilians transported, we evaluated whether the time window constraint was fulfilled because more tasks may be completed within the time window constraint. We also evaluated whether the ordering constraint was considered based on whether the transport task could be started immediately after the completing the rescue task.

- An agent that operates on the task allocation problem in an RRS modeled as an DCOP is an DCOP agent
- An agent that operates on the task allocation problem in RRS modeled as an L-DCOP is an L-DCOP agent

In this experiment, 12 disaster scenarios were considered. The agents and civilians were placed in a centralized or distributed layout, and three maps were used. The three maps of differing size from largest to smallest were: Eindhoven, SF, and VC. There were 10 fire brigades, 10 ambulance teams, and 50 civilians. In this experiment, only rescue and transport tasks were handled, and police forces and blockades were not implemented. In all scenarios, the simulation ended when all the civilians died.

The DCOP algorithm used by both agents in this experiment required a large number of communications for allocation decisions. Therefore, the DCOP module [3] was used to allow the agents to communicate multiple times in a single time step. In addition, the communication and perception range was the entire map area, and no communication or perception noise was generated.

| map | initial placement | | agent | |
|---|---|---|---|---|
| | agents | tasks | DCOP | L-DCOP |
| Eindhoven | centralized | centralized | 26 | 25 |
| | centralized | distributed | 21 | 11 |
| | distributed | centralized | 37 | 34 |
| | distributed | distributed | 27 | 28 |
| SF | centralized | centralized | 35 | 40 |
| | centralized | distributed | 36 | 30 |
| | distributed | centralized | 38 | 44 |
| | distributed | distributed | 33 | 36 |
| VC | centralized | centralized | 41 | 42 |
| | centralized | distributed | 37 | 38 |
| | distributed | centralized | 39 | 43 |
| | distributed | distributed | 36 | 39 |

Table 1: Number of civilians transported in each scenario

The number of successfully transported civilians in the experiment is shown in Table 1. The table shows that the L-DCOP agent transported more civilians than the DCOP agent in 8 out of 12 scenarios. However, fewer civilians were transported by the L-DCOP agent than the DCOP, especially on the map with a large area, Eindhoven. This indicates that the L-DCOP agent could not fully consider the time window constraints of the task in scenarios with large area maps.

The ratio of the number of rescued civilians to the number of civilians transported immediately after the rescue is shown in Table 2. The table shows that in most scenarios, the L-DCOP agent was able to start transport immediately after the rescue. This indicates that the L-DCOP agent considers the ordering constraints when allocating tasks.

# 3 Modules

## 3.1 Overview of this section

In this section, the MessageRequest and EfficientRectangleClearingModule that were newly developed and added this year are described. MessageRequest is a new message class that is used to send requests to other agents. EfficientRectangleClearingModule is a module that makes the debris-clearing-method by the Police Force more efficient. The other modules remain the same as those in AIT-Rescue 2023 [2].

## 3.2 MessageRequest

Agents share information with surrounding agents mainly through radio communication. In ADF-core, agents use unique message classes for communication

| map | initial placement | | agent | |
|---|---|---|---|---|
| | agents | tasks | DCOP | L-DCOP |
| Eindhoven | centralized | centralized | 0.4615 | 0.6800 |
| | centralized | distributed | 0.5714 | 0.9091 |
| | distributed | centralized | 0.3514 | 0.6176 |
| | distributed | distributed | 0.3333 | 0.6429 |
| SF | centralized | centralized | 0.1714 | 0.6250 |
| | centralized | distributed | 0.4167 | 0.8000 |
| | distributed | centralized | 0.9211 | 0.6818 |
| | distributed | distributed | 0.3636 | 0.9444 |
| VC | centralized | centralized | 0.3171 | 0.6190 |
| | centralized | distributed | 0.5405 | 0.7368 |
| | distributed | centralized | 0.4615 | 0.7442 |
| | distributed | distributed | 0.4444 | 0.8718 |

Table 2: The ratio of the number of civilians rescued to the number of civilians transported immediately after the rescue for each scenario.

depending on the type of agent. However, this message class is fixed for each type of source agent, and selecting the information to be sent according to message content is not possible. This implies that the agents may not be able to retrieve important information because of the bandwidth constraints imposed by the transmission of unnecessary information. Therefore, we implemented a new message class, called MessageRequest. for agents to send requests to other types of agents. MessageAmbulanceTeam and MessageFirebrigade, which are message classes in ADF-core, send the following contents as messages:

- Agent's ID
- Agent's HP
- Agent's buriedness
- Agent's damage
- Agent's position
- Agent's target task ID
- Actions to be performed by the agent

However, for debris-clearing requests directed to the Police Force or rescue requests directed to the Fire Brigade, knowing the position of the target agent and type of agent making the request, is sufficient. Therefore, the newly implemented MessageRequest sends the following content in message: This improvement reduces the message size by approximately 80% per message.

- Agent's position
- own agent type

### 3.3 EfficientRectangleClearingModule

The main method used by the Police Force to clear debris from roads is rectangular clearing. However, the conventional method requires a signficant amount

of time to clear the roads. Therefore, we developed a rectangle-clearing module for efficient clearing of the roads.

### Specification of rectangular clearing

Rectangular clearing is a method in which the police team extends a rectangle in the direction of the area they want to clear on the rectangle. Fig. 3 shows an example of rectangular clearing.

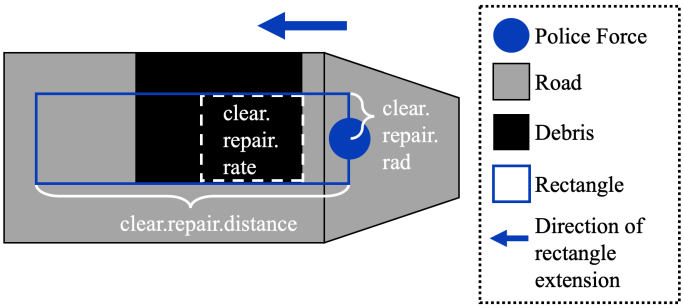


Fig. 3: Example of rectangular clearing

In Fig. 3, the Police Force clears the debris from the road by clearing a rectangle in the direction of the arrow. A rectangle is formed such that the center of the short side is located at the coordinates of the Police Force, as shown in Fig. 3.

This clearing is based on the rectangular clearing parameters set in advance by the disaster scenario. The three parameters for rectangular clearing are clear.repair.distance, clear.repair.rad, and clear.repair.rate. Each parameter is described follows:

- clear.repair.distance

  The value of clear.repair.distance is the length of the longest side of the rectangle. The larger clear.repair.distance is, the farther away the Police Force can clear the debris through rectangular clearing.

- clear.repair.rad

  This value is obtained by halving the length of the shorter side of the rectangle. The larger clear.repair.rad is, the wider is the road that can be cleared by the Police Force.

- clear.repair.rate

  This is the amount of debris that can be cleared by a single Police Force in a single clearing. As shown in Fig. 3, the rectangle contains more debris than clear.repair.rate. Therefore, the Police Force cannot clear all the debris in the rectangle in a single rectangular clearing. In addition, during the rectangular-clearing process, the Police Force clears debris from the nearest

rectangle. Therefore, debris within the rectangle that does not fit into the frame of clear.repair.rate remains after clearing as shown in Fig. 3.

### Design and Implementation

Until now, rectangle clearing has not been performed using the clear.repair.distance or the clear.repair.rate. In other words, the Police Force had to move close to the debris to clear the spot. Consequently a considerable amount of time was required to clear the debris.

In this study, we implemented a rectangular-clearing module that utilizes clear.repair.distance and clear.repair.rate. The processing steps of the implemented module are as follows.

1. Calculate the route to the destination as a series of roads
2. Form a line segment connecting the roads in the path
3. Apply rectangular cleaning or move depending on the conditions

In Step 2, a line segment is formed as shown in Fig. 4. The line segments are connected from the Police Force to the midpoint of the road edge, subsequenlty from the midpoint of the road edge to the center of the road, and finally from the midpoint of the road edge to the center of the road edge. Step 2 also determines the intersection of the line segment and debris contour. The intersection to be determined is the point closest to the starting point of the line segment.

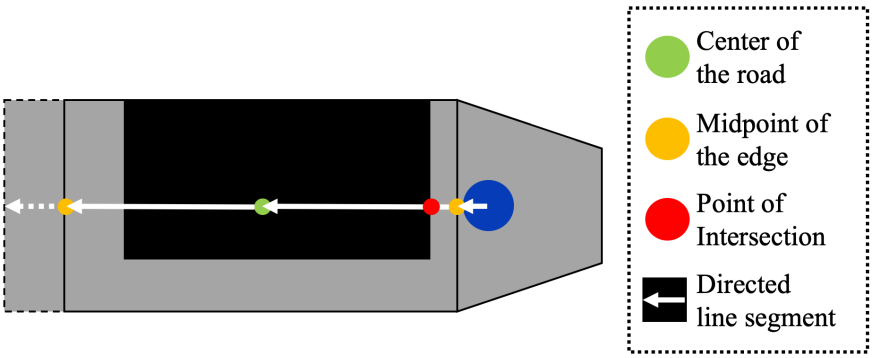


Fig. 4: Example of line segment formed

In Step 3, rectangular cleaning is performed on the spot based on the following conditions:

1. If the Police Force is located at the intersection determined in Step 2
2. If the estimated amount of debris in the rectangle exceeds the clear.repair. rate
3. Rectangular clearing is more efficient than moving in terms of the amount of debris cleared per STEP.
4. The amount of debris that can be cleared by rectangular clearing at an intersection is the same as the amount of debris that can be cleared by rectangular clearing on the spot

If any of the listed conditions are satisfied, the Police Force immediately performs rectangular clearing on the spot. If none of the conditions are fulfilled, the Police Force moves toward the intersection point determined in Step 2.

In-situ rectangular cleaning works well when Condition 2 is satisfied. Fig. 5 shows such an example. Fig. 5 shows a scene in which the Police Force can clear clear.repair.rate worth of debris by clearing a rectangle on the spot. Even if the Police Force moves to the intersection shown in Fig. 5 and clears the rectangle, the amount of debris that they can clear cannot exceed clear.repair.rate. Therefore, in-situ rectangle clearing is effective when Condition 2 is satisfied.

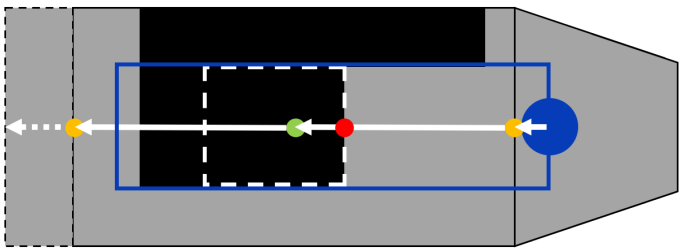

Fig. 5: The amount of debris within the rectangle exceeds clear.repair.rate

Fig. 6 shows a scene in which debris remains after rectangular clearing. In Fig. 6, Condition 3 determines whether or not to clear the remaining debris on the spot. This decision is made by comparing the amount of debris cleared per step in cases of rectangular clearing and moving. The police force can clear a large amount of debris in a shorter time by increasing the amount of debris cleared per step.

The amount of debris cleared per step is calculated by dividing the amount of debris that can be cleared through rectangular clearing at the site by the number of steps taken from the start of the rectangular clearing to the site to the next point. If clear.repair.rate of Fig. 6 is 20 and the remaining debris is 16, the amount of debris cleared per step when rectangle clearing is performed is $(20 + 16)/3 = 12$. The amount of debris cleared per step in the case of moving is $20/2 = 10$. Therefore, the amount of debris cleared per step is greater when rectangular

clearing is performed. In such cases, the police force clears the rectangle on the spot.

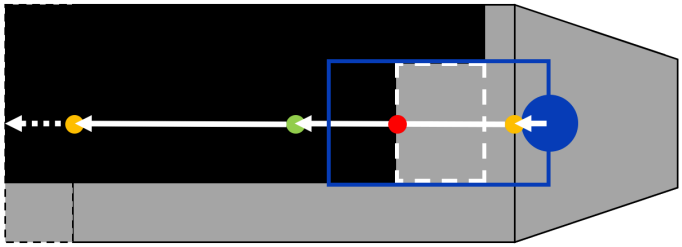

Fig. 6: Scenes of debris remnants in rectangular clearing

In-situ rectangular cleaning works well when Condition 4 is satisfied. Fig. 7 shows such an example. In Fig. 7, the estimated amount of debris that can be cleared by rectangular clearing at the intersection and on the spot are the same. In such cases, there is no need to move to the intersection; therefore, in situ rectangular clearing is effective.

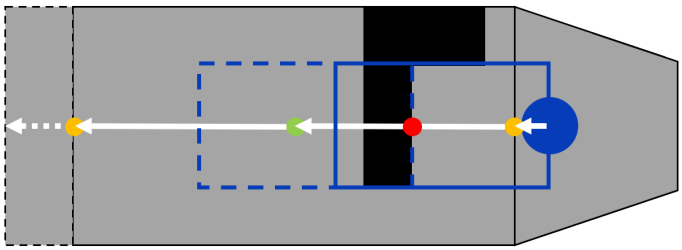

Fig. 7: Scenes where the amount of debris in the rectangle does not change before and after the move

## 4 Strategies

### 4.1 Ambulance Team

The Ambulance Team must transport as many citizens as possible to the refuge. Starting with the 2021 RoboCup, a bed set was added to the refuge. Ambulance Teams must transport civilians to the refuge with available beds. Therefore, the choice of the refuge is important. The optimal refuge is then selected using Eq. 1.

$maxEval$ represents the value of each refuge, considering both travel time to the refuge and the number of available beds. The highest value is selected as the target refuge. $stepsCounts$ denotes the number of steps take to move from the current location to the target refuge. $agentToRefuge$ is the straight-line distance between the current location and the target refuge. $agentAvgMove$ is a parameter representings the average speed of the agent. The average speed of the agent in the experiment was determined to be approximately 40,000. Therefore,

the value of $agentAvgMove$ was set to 40,000. In addition, $numAvailableBeds$ represents the number of available beds in the target refuge.

$$maxEval = \frac{1}{stepsCounts} + numAvailableBeds \tag{1}$$

$$stepsCounts = \frac{agentToRefuge}{agentAvgMove} \tag{2}$$

### 4.2 Fire Brigade

The Fire Brigade is responsible for rescuing buried civilians and preparing the for transport by ambulance teams. It is also responsible for rescuing other agents and preparing them for action.

In addition, to prioritize the lives that can be saved as much as possible, it is important to select rescue targets considering the civilian's survival time, rescue time, and distance to the refuge. Therefore, inactive agents should be called to action to rescue as many citizens as possible.

The Fire Brigade rescues agents and citizens using the priorities listed in Table 3. Agents should prioritize is given to agents to increase the efficiency in rescuing citizens. If there are multiple targets with the same priority, the target candidate with the smallest degree of burying possibility is rescued first.

Table 3: Priority of conditions used by the Fire Brigade

| priority | target |
|---|---|
| 1 | Disaster relief agents in the cluster in charge |
| 2 | Citizens in the cluster |
| 3 | Citizens outside the cluster |
| 4 | Disaster relief agents obtained from messages |
| 5 | Citizens acquired from the message |

### 4.3 Police

In some disaster scenarios, densely populated agents may be initially deployed. In the RRS, the debris prevents the agents from moving, making it impossible to rapidly start rescue activities. Therefore, the debris should be cleared by the Police Force, with priority given to areas with a high density of agents.

The Module can be used to prioritize the initial location of the agent in the case of dense debris so that the agent can clear the debris in the dense area. This Module detects areas with dense disaster relief agents during pre-computation. To ensure that the Ambulance Team can quickly transport injured civilians to refuges, the highest priority is placed on clearing the debris around the refuges within the area they are in charge of. Table 4 lists the priorities of the Police Force for clearing the debris. In case of the same priority, the one with the closest linear distance from the current position is prioritized.

Table 4: Priority of conditions used by our Police Force

| priority | target |
|---|---|
| 1 | Refuge in the cluster |
| 2 | Streets and buildings where agents are densely populated in their initial positions |
| 3 | Debris in the vicinity of an agent who is stuck in debris within the perceived range |
| 4 | The building where the ambulance team is initially located in the cluster |
| 5 | Priority roads in the responsible cluster |
| 6 | Buildings in the cluster |
| 7 | Entity in the responsible cluster |

## 5 Preliminary Results

Comparative experiments were conducted using our agent from the previous year to examine the effectiveness of AIT-Rescue 2024, considering scenarios from the final round of RoboCup 2023. The experimental results are shown in Table 5.

Table 5: AIT-Rescue 2023 and AIT-Rescue 2024 Experimental results

| **Scenario** | **Team** | |
|---|---|---|
| | **AIT-Rescue 2023** | **AIT-Rescue 2024** |
| sydney1 | 61.878 | 63.335 |
| mexico2 | 20.653 | 25.206 |
| ny2 | 31.648 | 33.976 |
| eindhoven2 | 19.023 | 18.929 |
| montreal1 | 19.253 | 19.515 |
| paris1 | 218.200 | 232.258 |
| kobe2 | 51.862 | 54.620 |
| vc2 | 65.638 | 64.436 |
| bordeaux2 | 6.303 | 6.303 |
| sf1 | 56.826 | 57.042 |

The experimental results showed that our improved agent achieved a higher score than the agent from the previous year in 8 out of 10 scenarios. Additionally, the scores substantially improved for mexico2 and ny2, which are large map sizes. Therefore, we believe that the debris-clearing methods of the Police Force worked well in situations where there was a lot of agent movement.

Thus, we conclude that the modules and strategies described in this TDP are effective for RRS.

## 6 Conclusions

In this study, we modeled the RRS task assignment problem as an L-DCOP, subsequently conducting experiments to confirm that the L-DCOP agent can perform cooperative rescue activities in coordination with multiple types of agents

considering the task constraints. Indeed, we confirmed that the L-DCOP agents can perform task assignments considering the task constraints. We also confirmed that the L-DCOP agent can perform cooperative rescue activities by multiple types of agents. However, this algorithm cannot be used in competition yet because it requires an extended ADF environment [3].

In addition, we implemented a new message class that sends only the necessary information in communications between agents. This improvement resulted in smaller message sizes in communications between agents. We also implemented a module that makes debris cleaning by the police force more efficient. With this improvement, the Police Force can clear debris more efficiently. These improvements also reduced the agents' scores from the previous year in 8 out of 10 disaster scenarios.

We will develop a mechanism whereby each agent will aggregate information and assign tasks to specific agents prior to RoboCup 2024. In the current AIT-Rescue, each agent decides which task to perform and operates accordingly. However, there are cases in which rescue activities can be performed more efficiently by consolidating the information possessed by each agent into a single location and assigning tasks based on this information. Therefore, we plan to create and implement a module that aggregates the information possessed by each agent into a rescue team and a command center, and assigns tasks to them.

## Acknowledgements

This work was supported by JSPS KAKENHI Grant Number JP17K00317, JP21K12039, and JP23K11225; and the Kayamori Foundation of Information Science Advancement.

## References

1. 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#FiorettoP016`
2. Haruki, U., Hiroya, S., Joe, F., Itsuki, M., Ryoya, M., Kaito, I., Ai, O., Keitaro, S., Yuki, S., Yuya, S., Ryosuke, S., Keita, H., Keisuke, A., Takeshi, U., Kazunori, I., Nobuhiro, I.: RoboCupRescue 2023 TDP Agent Simulation AIT-Rescue (Japan) (2023)
3. Miyamoto, Y., Kusaka, T., Okado, Y., Iwata, K., Ito, N.: RoboCupRescue 2019 TDP Infrastructure AIT-Rescue (Japan) (2019)
4. Suslova, E., Fazli, P.: Multi-Robot Task Allocation with Time Window and Ordering Constraints. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) pp. 6909–6916 (October 2020)