Agent_NAITO-Rescue

Yuki Miyamoto[1], Shunki Takami[1], Akira Hasegawa[1], Nobuhiro Ito[1], Kazunori Iwata[2]

[1]Department of Information Science, Aichi Institute of Technology, Japan
[2]Department of Business Administration, Aichi University, Japan
rescue-2017@maslab.aitech.ac.jp

Abstract

We consider efficient task allocation under the RoboCupRescue Simulation environment [1] in which agents cannot always communicate effectively. In this paper, we describe the modules developed in this task. These modules form groups of agents who execute the same task using near field communication, and enable the groups to adjust the number of agents executing the same task. We reduce the redundancy in the task execution process and improve the score over that achieved last year.

1 Introduction

In the disaster environment handled by the RoboCupRescue Simulation, we cannot perform efficient task allocation when intensive control is hampered by insufficient communication. To allocate tasks in such cases, we must introduce leaders that can collect as much information as possible and perform task allocation. Thus, we attempt to solve this problem by forming groups of agents (agent group) using unimpeded near-field communication and electing a leader in each agent group. We implement the formation of agent groups and task allocation using a Target Detector module.

In addition, we implement an A* algorithm that can evaluate whether a road is passable based on information obtained through communication with other agents. This algorithm is contained in a Path Planning module. About other modules, we implemented them with referring to RCRS-ADF's sample modules [5].

In Section 2, we describe the Target Detector module and the Path Planning module. In Section 3, we describe the evaluation of the agents using these modules.

As a result of the evaluation experiment described in Section 3, the score achieved by the FireBrigades using the modules from last year can be improved. However, efficient task allocation is not possible when the fires are spreading widely.

2 Modules

2.1 Target Detector

This module forms agent groups and allocates tasks by leaders [4]. Using near-field communication, agents that are in proximity form groups. From this, we can allocate tasks regardless of the communication environment changing with a simulation situation. However, to convey information as broadly as possible, agents sometimes use

long-distance communication, even if the communication environment is unstable. In the current RCRS-ADF specification, an agent cannot pass a user-defined values such as evaluation values to other agents. Therefore, this module only uses general information about the agent, task and command.

In addition, we must independently implement the formation of agent groups and elect the leader of each group using a Clustering module. However, task allocation in this module involves forming agent groups, which cannot be implemented in the current RCRS-ADF specifications. Therefore, we form agent groups using the Target Detector module.

We illustrate the transition of an agent's active state in this module in Figure 1. Here, we define agents that are not the leader of an agent group as followers. The transition of each agent's active state can be roughly divided into two processes: (i) forming agent groups and electing the leader, and (ii) task allocation and adjusting the agent group. We describe each process in turn. It is necessary for this module to calculate the importance of each tasks and the required number of agents. As an example, we describe the application of our approach to the FireBrigades task.

2.1.1 Forming agent groups and electing leaders

This module forms agent groups according to the task and ensures that multiple agents efficiently execute the same task. In each agent group, the agent with the lowest ID value becomes the leader, who then selects which task the agent group will perform. When the same task is selected by more than one agent group, the leaders communicate each other to ensure more efficient task execution. The integration of agent groups is performed in the following procedure. The number given at the end of each step indicates the corresponding point in Figure 1.

0. Let all agents be the leader of an agent group with 0 followers.
1. Each leader conveys information about itself and its task to all other leaders within a short distance using near-field communication. →Ⓘ
2. Leader B adds leader A to its agent group when leader B receives task information concerning the same task. Leader A also receives information from leader B and performs similar processing. →①
3. Leaders A and B elect the next leader of an agent group formed in step 2. If leader A's ID value is lower than that of leader B, leader A is selected as the next leader. Leader B becomes one of followers in the group. →①
4. Agent B, who is no longer a leader, commands the followers of its agent group before step 2 to leave the group while keeping the task. →Ⓘ
5. The follower who is commanded to leave the agent group while keeping the task in step 4 forms a new agent group only consisting of itself and continues the task. The followers become leaders of an agent group with 0 followers. →Ⓘ
6. The new leaders in step 5 have same task, and thus they continue to integrate their groups again, until the number of agents in the integrated group reaches the number to complete the task. →Ⓘ①

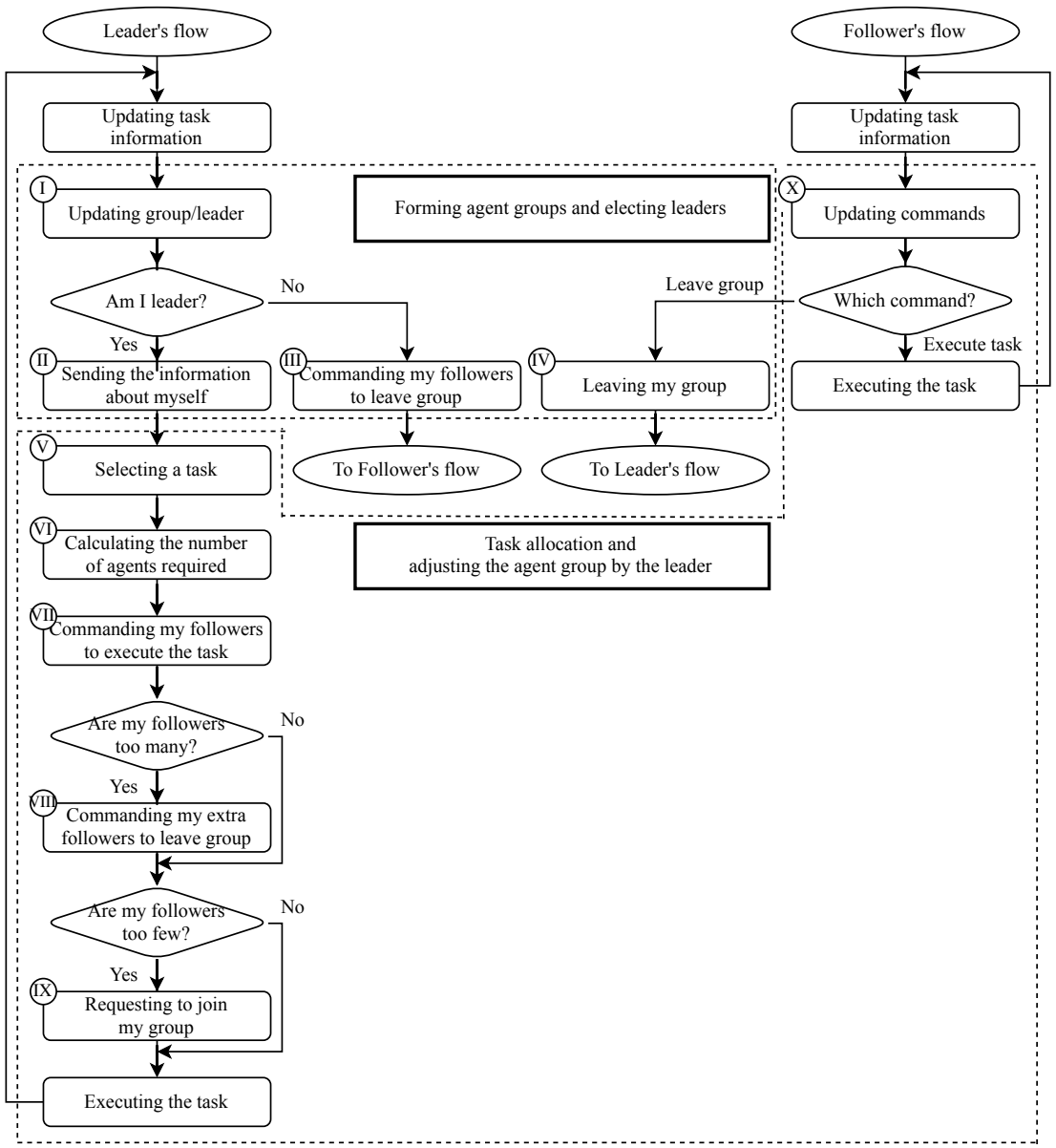

Figure 1: Transition of an agent's active state

In step 4, leader B breaks up its agent group temporarily. This is because its followers may not know the new leader (agent A) due to the restricted range of near-field communication. In addition, a follower leaves its agent group when it moves out of the range of near-field communication with its leader, because it would no longer be possible to receive commands from the group leader.

2.1.2 Task allocation and adjusting the agent group by the leader

In this module, each leader selects the task to be performed by its agent group and allocates work to its followers based on updated information concerning the task selection. At this time, each leader adjusts the number of agents in its group according to the target task. The adjustments are carried out by withdrawing agents from the group and integrating them into other agent groups. The task allocation and adjustment of the agent group by the leader is performed in the following procedure. The number after each step indicates the corresponding item in Figure 1.

1. When the leader obtains task information with higher importance than the currently selected task, the leader selects that task to be performed by its agent group. →Ⓥ
2. The leader calculates the required number of agents to complete the task selected in step 1. →Ⓥ
3. The leader commands its followers to execute the new task using near-field communication, with the number of agents obtained in step 2 as the maximum. → Ⓥ
4. The leader commands followers that are not the target of the command in step 2 to leave its agent group using near-field communication. →Ⓥ
5. The leader requests other agent groups to join its agent group when the number of agents in its group is less than the number calculated in step 2. →Ⓧ

Step 5 and the participation of other agent groups are controlled by the following procedure. Here, let agent A be the leader requesting the participation and agent B be the leader receiving the request for participation.

1. Leader A conveys information about the task in a request for participation to as many other leaders as possible using near-field communication and long-distance communication. (step 5 above)
2. Leader B of another agent group receives the information sent in step 1 and updates its current information with the received information.
3. Leader B reselects a task based on importance.
4. Leader B moves to the task's target area with its followers when the task selected in step 3 is the subject of the request for participation.
5. The agent groups of leaders A and B are gathered within proximity and execute the same task.
6. The agent groups are integrated and the agents led by leader B join the group led by leader A.

2.1.3 Calculating (updating) the importance and the number of agents required for a task

Using this module for the FireBrigades task, a set of buildings that can prevent the fire from spreading further if their fire extinguished is regarded as a task candidate.

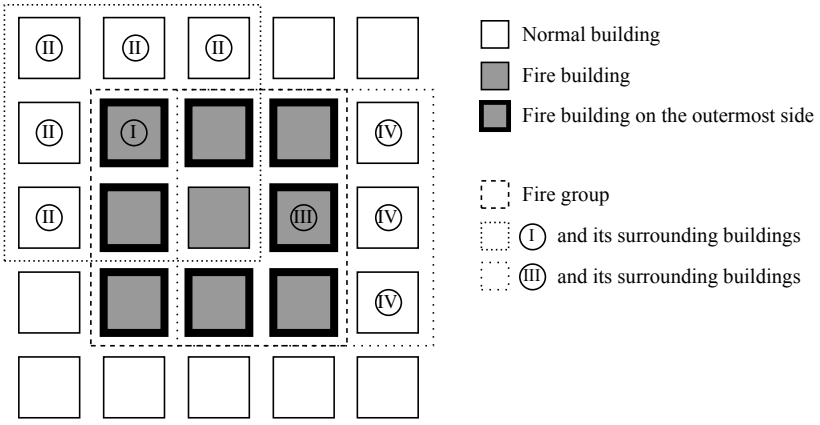

Figure 2: Forming a fire group

Only the buildings that are task candidates have importance levels. The importance levels are calculated according to the building information obtained by agents. The task candidates are selected in the following procedure. Figure 2 illustrates each step. In Figure 2, each building is represented as a square for simplicity.

1. Fire groups are formed (marked as the Fire group in Figure 2) by associating adjacent buildings on fire.
2. All buildings located on the outermost side of each fire group formed in step 1 are selected (marked as Fire building on the outermost side in Figure 2).
3. We consider each building selected in step 2 and its surrounding buildings. As examples, when we focus on the highlighted building Ⅰ, its surrounding buildings are as indicated by Ⅱ in Figure 2. Correspondingly, the buildings Ⅳ are the surrounding buildings of the highlighted building Ⅲ.
4. If the agents in the current group can terminate the fire in the building before it spreads to surrounding buildings using the agents in the current agent group, a task candidate is each element of the set of the buildings in step 2.
5. Otherwise, each task candidate is the set of the surrounding buildings in step 3 (e.g. the buildings Ⅱ or the buildings Ⅳ).

Next, we describe how to determine the importance of the task candidates. We consider the two following points: the more applicable they are, the higher the importance.

- There are a lot of surrounding buildings to which the fire could possibly spread.
- The time required to move and extinguish the fire is short.

The agents cannot judge whether a building is on fire without detailed information. Therefore, a buildings for which information has not been updated will have lower importance than those with updated information. Based on the above points, we introduce the influence on the surrounding buildings per step as the importance I:

$$I = \frac{B + \alpha \times B'}{M + E} \tag{1}$$

where B = The number of surrounding buildings that are not yet on fire and have updated information,
B' = The number of surrounding buildings that are not yet on fire and do not have updated information,
M = The time taken by the agents in moving to the building,
E = The time taken by the agents to extinguish the fire in the building,
α = A constant that reduces the influence of B' on the importance I ($0 \leq \alpha \leq 1$).

Further, if there is a request from another agent to join the task, it should be considered that the task is highly urgent. Therefore, we introduce the importance I' that depends on the number of requests. We use I' as the importance in the FireBrigades task selection.

$$I' = I \times \beta^T \tag{2}$$

where T = The number of the requests for participation,
β = A constant that increases the importance per request for participation ($\beta \geq 1$).

Next, we describe how to determine the required number of agents of a task. We consider the required number of agents for a task to be the the number that can extinguish a building before the fire spreads. Therefore, we let the required number of agents for a task be the number of agents that can completely or preliminarily extinguish the target building before its surrounding buildings catch fire.

2.2 Path Planning

This module performs path planning using the A* algorithm [3], which is one of many path planning algorithms [2]. Here, we describe the evaluation value used in the A* algorithm. We consider the two following points for the evaluation value: the more applicable they are, the more appropriate the path.

- The distance to the destination is short.
- The path is not impassable (according to Blockades).

In most cases, an agent moves to target points beyond its perceivable range. For this reason, the agent cannot judge whether the path determined by the path planning algorithm is passable or not. Therefore, the agent judges whether the path is feasible based on road information received by communication with other agents. However, the credibility of the road information received by communication decreases with the passage of time. Based on the above points, we introduce the evaluation formula $COST(n, m)$

to the general evaluation formula $f(m)$ of the A* algorithm. We use $COST(n, m)$ as the cost for passing through from n to m in consideration of the information that it is impassable instead of usual cost L.

$$f(m) = g(n) + COST(n, m) + h(m) \tag{3}$$

$$COST(n, m) = (1 + \gamma \times \frac{S - P}{S}) \times L \tag{4}$$

where $g(n)$ = The total $COST(i, j)$ of the path from the start node to n,
$h(m)$ = The Euclidean distance from m to the destination,
L = The Euclidean distance from n to m,
S = The effective time of the road information that the path from n to m is impassable,
P = The elapsed time since the agent received the information ($0 \leq P \leq S$),
γ = A constant that makes L longer when the path from m to n is impassable ($\gamma \geq 0$).

3 Preliminary Results

3.1 Evaluation experiment

We measured the score using three maps used in the RoboCup 2016 Rescue Simulation Agent Simulation League to evaluate the agents performance using the proposed modules. However, we removed the PoliceForces, AmbulanceTeams and Blockades from the maps, because only the FireBrigades are currently fully developed. We used the score achieved by our team's agents last year [6] for comparison. The average scores over three repeated tasks are presented in Table 1.

Table 1: Scores

Team	**Map**		
	Eindhoven1	**VC1**	**Mexico1**
2017	87.269	13.737	51.698
2016	76.531	6.126	38.486

3.2 Considerations

Table 1 shows that the proposed modules achieved higher score than the agents used in last year's competition. There are two possible reasons for this increase in the score. The first point is that tasks no longer involve more agents than necessary, because the number of agents working on the same task is adjusted by forming agent groups. The second point is that it is possible to suppress the spread of fire when the agents identify

a fire in its early stages. This is because it is possible to predict the spread of fire and execute a preliminary extinguishing in the appropriate building.

However, the agents fail to suppress the spread of fire if they cannot identify it in the early stages. The agents cannot completely suppress the spread of fire in fire groups that grow significantly, so the agents need to partially suppress the fire. However, as the fire group becomes larger, the number of task candidates increases. Therefore, it is impossible to suppress the spread of fire even partially, because the task targets are greatly dispersed. To solve this problem, it is necessary for local agents to form an agent group irrespective of the task, and the leader should allocate appropriate tasks to each follower in consideration of the efficient combination of the target tasks.

4 Conclusion

In this paper, we describe the Target Detector module and Path Planning modules. We conducted an evaluation of the agents using these modules. As a result, we found that a higher score can be achieved than with the agents used last year. However, we also identified several problems in the Target Detector module. Therefore, we will use this module after solving the problems in the Agent Simulation Competition.

References

[1] Robocup Rescue Simulation. http://roborescue.sourceforge.net/web/.

[2] E. W. Dijkstra. A note on two problems in connexion with graphs. *NUMERISCHE MATHEMATIK*, 1(1):269–271, 1959.

[3] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968.

[4] Nobuhiro Ito, Yoshiki Asai, Tetsuya Esaki, and Koichi Wada. A cooperative model of rescue agents by a group forming algorithm. *T.SICE*, 41(12):964–973, 2005.

[5] Kazuo Takanayagi, Takuma Kawakami, Shunki Takami, Yusuke Kitagawa, Shivashish Jaishy, Nobuhiro Ito, and Kazunori Iwata. Proposal and Implementation of new framework to RoboCup Rescue Simulation NAITO-Rescue 2015 (Japan). *RoboCup 2015: Robot World Cup XIX*, 2015.

[6] Kazuo Takanayagi, Shunki Takami, Yuki Miyamoto, Masahiro Yamamoto, Yoshiyuki Kozuka, Nobuhiro Ito, and Kazunori Iwata. Contraction hierarchy algorithm that considers fault incidences under disaster environment NAITO-Rescue 2016 (Japan). *RoboCup 2016: Robot World Cup XX*, 2016.