Integrating Decentralized Coordination and Reactivity in MAS for Repair-Task Allocations

Hisashi Hayashi(✉)

System Engineering Laboratory, Corporate Research and Development Center, Toshiba Corporation, 1 Komukai-Toshiba-cho, Saiwai-ku, Kawasaki 212-8582, Japan
hisashi3.hayashi@toshiba.co.jp

Abstract. Task allocation is an important research area for multi-agent systems (MASs). In a large system of systems, multiple MASs are connected through the network, and decentralized coordination among MASs is vital. In general, it takes time to coordinate task allocations. However, when a task has to be done within a short time, it is necessary to start the task execution immediately. In this paper, we present a new task-allocation algorithm that reconciles decentralized coordination and reactivity. We consider scenarios where multiple causes of future agent failures are created simultaneously and consecutively, and if they are not removed by repair actions within limited times, some agents become out of order with high probability. In this paper, we show that the combination of decentralized coordination and reactivity significantly increases (more than doubles) the average numbers of successful repairs when the time available for decision-making and repairing is short.

Keywords: Multi-agent systems · Coordination and reactivity · Decentralized task allocation · Emergency repair

1 Introduction

Task allocation is an important research area for multi-agent systems (MASs). In order to allocate tasks to agents, agents need to communicate and cooperate with one another through the network. In general, it takes time to allocate tasks to agents because of various delays such as communication, computation process, action preparation, planning, or human confirmation. However, in the case of an emergency, there is insufficient time for task allocation. In this paper, we present two new algorithms for task allocation that handle delay times. In particular, one of the algorithms reconciles decentralized coordination among agents and reactivity by local agents, which is our main contribution.

We consider large MASs where multiple unit MASs are connected through the network. We also consider the scenarios where multiple disaster events happen simultaneously and consecutively, which trigger many causes of future agent failures. If a cause of a future agent failure is not repaired within a limited

Y. Demazeau et al. (Eds.): PAAMS 2017, LNAI 10349, pp. 95–106, 2017.
DOI: 10.1007/978-3-319-59930-4_8

time, an agent will stop functioning with high probability. Some agents in unit MASs can execute repair actions. Therefore, the problem we consider is that of allocating repair tasks to unit MASs within limited times in order to prevent agent failures.

As discussed in [5,11,14], task-allocation algorithms are roughly divided into two kinds of algorithms: centralized algorithms and decentralized algorithms. In centralized algorithms, only one manager agent collects information from its child agents, computes the combination of tasks and agents, and allocates the tasks to its child agents. On the other hand, in decentralized algorithms, multiple manager agents communicate with one another to allocate tasks to other manager agents or to themselves. Many existing task-allocation algorithms are centralized algorithms. However, decentralized task-allocation algorithms are robust because the total MAS continue to function as a whole even when some managers breakdown. Auction algorithms such as the contract net protocol [15] are often used for dynamic task allocation [1,2,4,6,8,10] and it is not difficult to use them in a decentralized manner. Therefore, we modify and extend decentralized task-allocation algorithms that use the contract net protocol so that the new algorithm can allocate multiple emergency repair tasks that need to be completed within a limited time.

The rest of this paper is organized as follows. In Sect. 2, we discuss related work. In Sect. 3, we explain the MAS architecture and the problem. In Sect. 4, we define two new decentralized algorithms for repair-task allocations. In Sect. 5, we explain the simulation settings in detail. In Sect. 6, we show and analyze the simulation results. In Sect. 7, we conclude this paper.

2 Related Work

In this section, we discuss related work. In typical approaches for task allocations, meta-heuristics are used for optimizing combination of tasks and agents considering various constraints. In [9,17], multiple meta-heuristics for task allocation are compared. In [17], it is shown that a variant of tabu search is better than the other algorithms including variants of GA and ACO in terms of computation times and optimality. In [9], it is shown that a variant of PSO produces slightly better results in terms of optimality when the computation time is limited, and the other algorithms including variants of GA and fast greedy algorithms are nearly as good as PSO. However, in general, the algorithms of meta-heuristics are time-consuming and they are based on the centralized MAS architecture. Even if we use fast greedy algorithms, it is still impossible to avoid the delay times of the other computation, network delay, or human confirmation. On the other hand, we need to dynamically allocate tasks within very short times based on a decentralized MAS architecture. Therefore, to enhance reactivity, we try to avoid human confirmation and coordination among agents when the time for decision-making is short.

In [11,13], variants of max-sum algorithms for distributed constraint optimization (DCOP) are used for task allocation problems. Compared with

other algorithms of DCOP that utilize connectivity graphs of agents, max-sum is robust against agent failures. However, many messages are repeatedly sent between agents in DCOP, which causes delays of communication and computation.

There is some research on task allocation that is robust for agent failures. In [14], the probabilities of future agent failures are considered when allocating tasks to agents. However, the algorithm does not consider repairing. In [7], backup agents are used in the case of an emergency. However, the cost of backup agents is high when additional hardware is needed. Similarly, in [12], robust agent teams are created by preparing more agents than needed, considering future agent failure.

Our repair-task-allocation problem is closely related to the task-allocation problems of combat ships [2,3], weapon-target assignment [4,9,17], and disaster relief [1,13,16] where tasks with hard deadlines such as threat removal and civilian rescue are allocated to teams.

3 MAS Architecture and Problem Description

We consider a MAS for repair-task allocations that is composed of multiple **unit MAS**s, each of which includes **sensing agents**, **action-execution agents**, and a **manager agent**: sensing agents detect causes of future agent failures, action-execution agents fix causes of future agent failures using limited resources, and manager agents communicate with one another to allocate repair tasks to action-execution agents. In this section, we define unit MASs and the agents that belong to unit MASs. We define the functions of unit MASs as agents because each function is often deployed on different hardware and becomes out of order independently.

As shown in Fig. 1, a unit MAS is a MAS comprising 0 or more sensing agents, 0 or more action-execution agents, and 1 manager agent. When a sensing agent senses a cause of a future agent failure, it reports the information to the manager agent in the same unit MAS. When receiving the information of a cause of a future agent failure, the manager agent allocates the repair task to an action-execution agent that belongs to the same unit MAS or allocates the repair task to the manager agent of another unit MAS if there are multiple unit MASs and their manager agents are connected by the network.

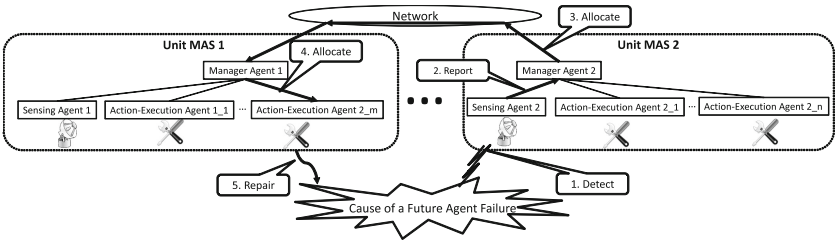

Fig. 1. MAS Architecture for repair-task allocations

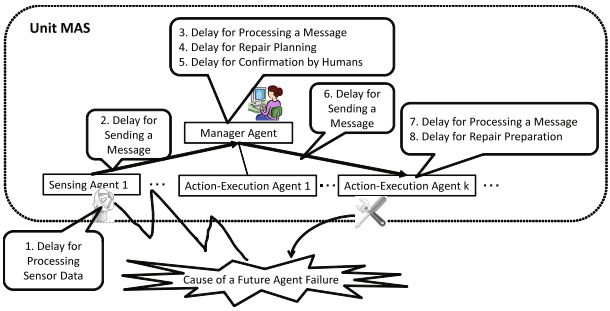

Fig. 2. Delay times

When allocated a repair task, the action-execution agent will execute a repair action consuming one resource. Execution of a repair action will succeed or fail according to the predefined probability. Unless a cause of a future agent failure is removed by a repair action, one of the agents will stop functioning according to the predefined probability.

Because we need to tackle time-critical problems, we consider delay times as illustrated in Fig. 2: time for a sensing agent to process sensor data to detect a cause of a future agent failure, time for an agent to send a message, time for an agent to receive and process a message, time for a manager agent to plan for repair, time for the human operator of a manager agent to confirm the repair plan, and time for an action-execution agent to prepare for repairing.

4 Algorithms

In this section, we introduce two new decentralized algorithms for repair-task allocations that are based on the contract net protocol [15]. These algorithms are equipped with replanning capabilities. Replanning is triggered when an action-execution agent fails to execute a repair action, which is very effective as shown in our previous study [8]. Replanning is also triggered when a manager agent with no available resource receives a repair-task allocation due to delay times.

We modify the contract net protocol so that we can handle multiple causes of future agent failures that are detected nearly simultaneously when repairing is delayed because of communication, computation process, action preparation, planning, or human confirmation. In these algorithms, when a manager agent M tries to allocate a repair-task R to a manager agent of another unit MAS, M tries to allocate R to a manager agent that M has not allocated a task for a certain period of time because even when multiple new causes of future action failures are found at nearly the same time, only a few agents are likely to be selected for repair-task allocations during the delay times, which triggers unnecessary replanning.

The second algorithm is an extension of the first algorithm. In the second algorithm, we combine decentralized coordination and reactivity. This is the

main contribution of this paper. The idea is that in the case of an emergency, the manager agent that has the information of a cause of a future agent failure allocates the task to itself without communicating with the other manager agents and avoids confirmation by the human operator, which saves much time and enhances reactivity. We expect this new algorithm to become more effective when the time available for decision-making and repairing is short.

Algorithm 1 (Decentralized Coordination). *The sensing agents, the manager agent and the action-execution agents in each unit MAS work as follows if they are alive:*

- *Algorithm of Sensing Agents*
  1. *When a sensing agent detects a new cause of a future agent failure, it reports the information to the manager agent in the same unit MAS if the manager agent is alive.*
- *Algorithm of Manager Agents*
  1. *When the manager agent M of a unit MAS U receives the information of a new cause of a future agent failure C from a sensing agent of U, M asks each alive manager agent M2, if it exists, whether the unit MAS U2 of M2 can be in charge of the repair task R of C and how quickly an action-execution agent of U2 can start the repair action of R.*
  2. *If there exists a unit MAS that can be in charge of the repair task R of C, then the manager agent M performs the following procedure:*
     - ***If there exists a unit MAS that can be in charge of the repair task R of C and has not been in charge of any repair task for a certain period*[1] *of time, then from those manager agents, M selects the manager agent M3 of the unit MAS U3 such that an action-execution agent of U3 can start the repair action of R the quickest and allocates R to M3.***
     - *Otherwise, M selects*[2] *the manager agent M4 of the unit MAS U4 such that an action-execution agent of U4 can start the repair action of R the quickest and allocates R to M4.*
  3. *When the manager agent M5 receives the allocation of a repair task R, M5 performs the following procedure:*
     - *If there exists an action-execution agent E5 in the same unit MAS such that E5 is alive, the number of resources of E5 is more than 0 and E5 is not reserved for another cause of a future agent failure, then M5 performs the following procedure:*
       (a) *The manager agent M5 selects and reserves the action-execution agent E5 for R.*
       (b) *The manager agent M5 calculates the plan P for R.*
       (c) *The human operator of M5 confirms the plan P for R.*

[1] We set the period of time to be 1 min in our experiments.
[2] M4 has been in charge of a repair task for a certain period of time in this case.

(d) *When it becomes possible for the reserved action-execution agent E5 to start executing the repair action A for the reserved repair task R, if E5 is alive, the manager agent M5 orders E5 to execute A and erases the reservation information.*

(e) *When the manager agent M5 receives the result of action execution of A for the repair task R from the action-execution agent E5 in the same unit MAS, if the result is a failure, M5 asks each alive manager agent and reallocates R to one of the manager agents in the same way.*

- ***Otherwise***[3]***, M5 asks each alive manager agent and reallocates R to one of the manager agents in the same way.***

– *Algorithm of Action-Execution Agents*

1. *When receiving an execution order of the repair action A, from the manager agent M in the same unit MAS, the action-execution agent E executes A, decrements 1 resource whether the result of A is a success or a failure, and reports the result to M.*

Algorithm 2 (Decentralized Coordination + Reactivity). *The sensing agents, the manager agent and the action-execution agents in each unit MAS work as follows if they are alive:*

– *Algorithm of Sensing Agents*
- *Same as the algorithm of sensing agents in Algorithm 1*

– *Algorithm of Manager Agents*

1. *When the manager agent M of a unit MAS U receives the information of a new cause of a future agent failure C from a sensing agent of U, M performs the following procedure:*
   - ***If there is time***[4] ***to communicate with other manager agents to allocate the repair task R of C before an agent fails owing to C****, then the manager agent M allocates R to one of the manager agents in the same way as step 1 and step 2 of the algorithm of manager agents in Algorithm 1.*
   - ***Otherwise***[5]***, M allocates the repair task R to itself****.*
2. *When a manager agent M5 receives the allocation of a repair task R, M5 tries to do the repair task R in the same way as step 3 of the algorithm of manager agents in Algorithm 1 except that step 3c is modified as follows:*
   - *The human operator of M5 confirms plan P for R* ***if***[6] ***the repair action A of R can be completed before an agent fails after the human confirmation****.*

– *Algorithm of Action-Execution Agents*
- *Same as the algorithm of action-execution agents in Algorithm 1*

[3] In this case, M5 has received repair-task allocations beyond its currently available resources because multiple manager agents in different unit MASs tried to allocate a task to M5 at nearly the same time during the delay times.

[4] The threshold is calculated based on Table 5 in our experiments.

[5] In this case, M does not have the time to communicate with other manager agents.

[6] This step is skipped when there is insufficient time for human confirmation. The threshold is calculated based on Table 5 in our experiments.

5 Simulation Settings

In this section, we explain the details of simulation settings to compare and evaluate the two new algorithms defined in the previous section. In the following, we set typical values of unit MASs, considering our target applications.

5.1 The Number of Agents and Resources in Unit MASs

As shown in Table 1, we use 5 kinds of unit MASs: UMAS 1, ..., and UMAS 5, which are typical unit MASs of our target application. The numbers of UMAS 1, ..., and UMAS 5 are 1, 2, 2, 4, and 8, respectively. The total number of these unit MASs is 17 ($=1+2+2+4+8$). Each unit MAS has exactly one manager agent and one sensing agent. Because high-performance unit MASs are costly in general, considering the balance, we use a smaller number of high-performance unit MASs and a larger number of low-performance unit MASs. We introduce the performance of each unit MAS in the next subsection.

The numbers of action-execution agents in UMAS 1, ..., and UMAS 5 are 0, 0, 4, 2, and 1, respectively. The total number of action-execution agents is 24 ($=2*4+4*2+8*1$). An action-execution agent cannot execute more than one repair action in parallel but multiple action-execution agents can execute repair actions at the same time. The numbers of initial resources that each action-execution agent in UMAS 3, ..., and UMAS 5 has are 6, 4, and 8, respectively. The total number of initial resources is 144 ($=2*4*6+4*2*4+8*1*8$).

UMAS 1 and UMAS 2 do not have action-execution agents, which means that the causes of agent failures found by the sensing agent of UMAS 1 or UMAS 2 need to be repaired by the action execution agents of UMAS 3, ..., or UMAS 6.

Table 1. The number of unit MASs, Agents, and Resources

Type of unit MAS	# of Unit MASs	# of Manager agents	# of Sensing agents	# of Action-execution agents	# of Resources of each action-execution agent
UMAS 1	1	1	1	0	-
UMAS 2	2	1	1	0	-
UMAS 3	2	1	1	4	6
UMAS 4	4	1	1	2	4
UMAS 5	8	1	1	1	8

5.2 Performances of Sensing Agents and Action-Execution Agents

Table 2 shows the times for sensing agents to start detecting causes of future agent failures before the expected times of agent breakdown. The probability of detecting causes of future agent failures is 90%. The sooner the sensing agent detects a cause of a future agent failure, the higher the performance is, which means that performance of the sensing agent in UMAS 1 is the best.

Table 3 shows the times for action-execution agents to start repairing and removing causes of future agent failures before the expected time of agent breakdown. The sooner the action-execution agent can start repairing, the higher the performance is, which means that performance of the action-execution agent in UMAS 3 is the best. The success probability of repairing is 80%.

Table 4 shows the times for action-execution agents to repair and remove causes of future agent failures when they start repairing x seconds before the expected time of agent's breakdown. We assume a situation where a cause of agent failure approaches the target agent at constant speed and the action-execution agent sends the resource for a repair to the cause of future agent failure at constant speed. Among the three causes of future agent failures, cause 3 approaches the target agent at the fastest speed.

Table 2. Probability and time to start detecting a cause before an agent failure

Type of unit MAS	Cause type of future agent failure		
	Cause 1	Cause 2	Cause 3
UMAS 1	90%, 43.2 s	90%, 120.0 s	90%, 42.4 s
UMAS 2	90%, 43.2 s	90%, 60.0 s	90%, 21.2 s
UMAS 3	90%, 43.2 s	90%, 24.0 s	90%, 8.5 s
UMAS 4	90%, 43.2 s	90%, 14.4 s	90%, 5.1 s
UMAS 5	90%, 18 s	90%, 6 s	90%, 2.1 s

Table 3. Success probability and time to start repairing before an agent failure

Type of unit MAS	Cause type of future agent failure		
	Cause 1	Cause 2	Cause 3
UMAS 3	80%, 36.0 s	80%, 12.0 s	80%, 4.2 s
UMAS 4	80%, 18.0 s	80%, 6.0 s	80%, 2.1 s
UMAS 5	80%, 10.8 s	80%, 3.6 s	80%, 1.3 s

Table 4. Repair time when starting the repair x seconds before an agent failure

Type of unit MAS	Cause type of future agent failure		
	Cause 1	Cause 2	Cause 3
UMAS 3	x/2.5 s	x/4.5 s	x/9.5 s
UMAS 4	x/2.5 s	x/4.5 s	x/9.5 s
UMAS 5	x/2.5 s	x/4.5 s	x/9.5 s

5.3 Delay Times

In our simulation scenarios, we take various delay times into consideration because they affect the simulation results when the time is limited. Table 5 shows

Table 5. Delay times

Time for processing sensor data	Time for sending a message	Time for processing a message	Time for repair planning	Time for human confirmation	Time for repair preparation
1 s	1 s	1 s	5 s	0, 10, 12, 13, 14, 15, 20, 30 s	5 s

the delay times. When a sensing agent detects a cause of future agent failure, it takes 1 s for processing sensor data. When an agent sends a message to another agent, it takes 1 s. When an agent receives and processes a message, it takes 1 s. When a manager agent calculates a plan for a repair task, it takes 5 s. When a human operator of a manager agent confirms a plan for a repair task, it takes 0, 10, 12, 13, 14, 15, 20, or 30 s. (We change the times for human conformation to see how the length of each delay affects the simulation results.) When an action-execution agent prepares for the execution of a repair action, it takes 5 s.

5.4 Occurrence Patterns of Disasters and Agent Failures

Table 6 summarizes the occurrence patterns of disaster events and causes of future agent failures. In our simulation scenarios, when a disaster event occurs, a cause of a future agent failure is created every second. The total number of causes of future agent failures created by a disaster event is 10. Disaster events repeatedly happen up to 10 times, and the interval between disaster events is 1 h. Note that the total number of causes of future agent failures created by 10 disaster events is 100 (=10 * 10) whereas the total number of initial resources is 144. It seems that the resources are sufficient for repairing. However, when an action-execution agent or its manager agent fails, its resource becomes unavailable. When a cause of a future agent failure is not removed, an agent becomes out of order with the probability of 90%. The proportions of cause 1, cause 2, and cause 3 are 60%, 30%, and 10%, respectively. The times from occurrence of cause 1, cause 2, and cause 3 to agent failures are 1800, 600, and 212 s, respectively.

Table 6. Occurrence patterns of disasters and causes of future agent failures

# of Disaster events	Occurrence interval of disaster events	# of Causes of future agent failures created by a disaster event	Occurrence interval of causes of future agent failures	Prob. of agent failures when not repaired	Proportions of causes of future agent failures and times from occurrence of a cause to an agent failure
1, 2, 3, 4, 5, 6, 7, 8, 9, 10	1 h	10	1 s	90%	Cause 1: 60%, 1800 s, Cause 2: 30%, 600 s Cause 3: 10%, 212 s

6 Simulation Results

This section shows the simulation results. We conducted simulations 1000 times using different random seeds for each algorithm and for each simulation setting.

Figure 3 shows the simulation results after 10 disaster events when changing the times for human confirmation in the x-axis. Recall that 100 causes of agent

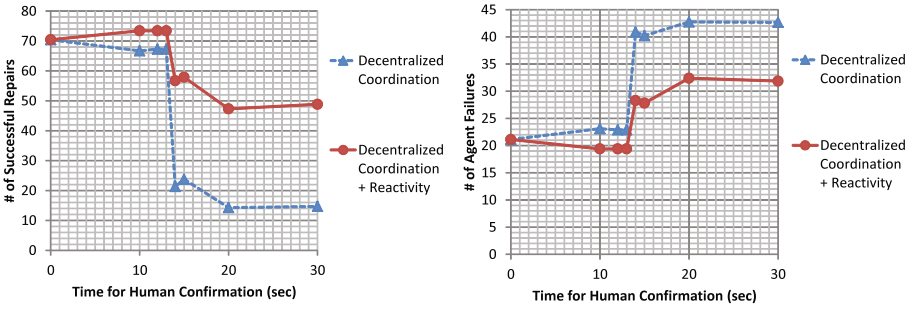

Fig. 3. Simulation results when changing the times for human confirmation

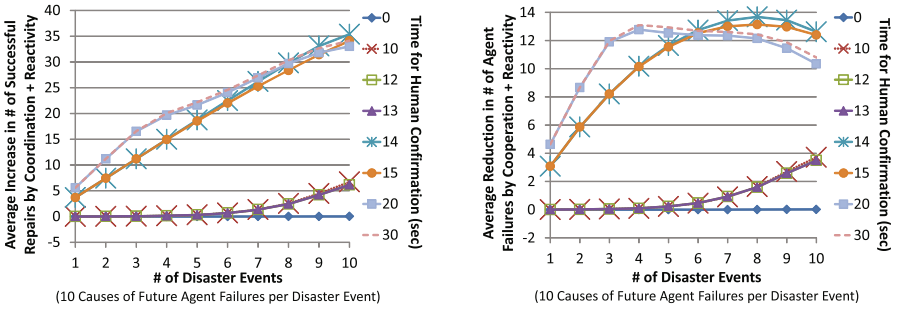

Fig. 4. Simulation results when changing # of disaster events

failures are created by 10 disaster events. In the graph of Fig. 3(a), the average number of successful repairs is shown in the y-axis. In the graph of Fig. 3(b), the average number of agent failures is shown in the y-axis.

In the two graphs of Fig. 3, we can see that Algorithm 2 (Decentralized Coordination + Reactivity) always produces better results than Algorithm 1 (Decentralized Coordination). The differences of these two algorithms are slight when the times for human confirmation are between 0 and 13 s. Algorithm 2 suddenly produces better results than Algorithm 1 when the time for human confirmation changes from 13 s to 14 s. The simulation results do not change much afterwards. This means that reactivity is very effective when the delay times are long compared with the remaining times before the agents become out of order. In other words, reactivity becomes very effective when there is insufficient time for coordination among unit MASs and for human confirmation.

Figure 4 shows the simulation results when changing the number of disaster events in the x-axis for different times for human confirmation. In the graph of Fig. 4(a), the average increase in the number of successful repairs by combining decentralized coordination with reactivity is shown in the y-axis. In the graph

of Fig. 4(b), the average reduction in the number of agent failures by combining decentralized coordination with reactivity is shown in the y-axis.

We can confirm that reactivity is very effective when the delay times are long even for different numbers of disasters. When the times for human confirmation are between 14 and 30 s, the average increase in the number of successful repairs linearly increases as the number of disaster events increases in the graph of Fig. 4(a). However, the average reduction in the number of agent failures does not increase beyond 14 in the graph of Fig. 4(b). This is because the number of agent failures increases as the number of disaster events increases, and repairing a cause of an agent failure does not reduce the number of agent failures when the agent is already out of order.

7 Conclusions

In this paper, we presented two new algorithms (Algorithms 1 and 2) for decentralized repair-task allocation. These two algorithms were developed by modifying the contract net protocol so that we can dynamically handle multiple causes of future agent failures even when repairing is delayed because of communication, computation process, action preparation, planning, or human confirmation.

Although we improved decentralized repair-task allocation algorithms considering delay times, this was insufficient when a repair task has to be completed within a short time. In this case, we do not have time for coordination among agents and human confirmation. Therefore, in Algorithm 2, we combined decentralized coordination and reactivity, which is the main contribution of this paper. The idea of this algorithm is to skip coordination among agents and human confirmation when there is insufficient time.

We compared Algorithm 2 (Decentralized Coordination + Reactivity) with Algorithm 1 (Decentralized Coordination) by means of simulation. In our severe simulation scenarios of disasters, causes of future agent failures are created simultaneously and consecutively. We conducted simulation 1000 times for each simulation setting and for each algorithm using different random seeds. We evaluated the simulation results by the average number of successful repairs and the average number of agent failures. As a result, we found the following in our simulation scenarios:

- Algorithm 2 (Decentralized Coordination + Reactivity) always produces better results than Algorithm 1 (Decentralized Coordination).
- Algorithm 2 (Decentralized Coordination + Reactivity) is effective when the delay times are long compared with the remaining times before the agents become out of order.
- There is a clear borderline of delay times such that the combination of decentralized coordination and reactivity becomes very effective.

Although these results are confirmed in a limited number of simulation scenarios, we anticipate that these results hold in general. In future work, we intend to evaluate the algorithms in more detail in our target application using many different scenarios and parameters.

References

1. Ahmed, A., Patel, A., Brown, T., Ham, M., Jang, M.-W., Agha, G.: Task assignment for a physical agent team via a dynamic forward/reverse auction mechanism. In: International Conference on Integration of Knowledge Intensive Multi-Agent Systems, pp. 311–317 (2005)
2. Beaumont, P., Chaib-draa, B.: Multiagent coordination techniques for complex environments the case of a fleet of combat ships. IEEE Trans. Syst. Man Cybern. Part C **37**(3), 373–385 (2007)
3. Brown, C., Lane, D.: Anti-air warfare co-ordination - an algorithmic approach. In: International Command and Control Research and Technology Symposium (2000)
4. Chen, J., Yang, J., Ye, G.: Auction algorithm approaches for dynamic weapon target assignment problem. In: International Conference on Computer Science and Network Technology, pp. 402–405 (2015)
5. Choi, H.-L., Brunet, L., How, J.P.: Consensus-based decentralized auctions for robust task allocation. IEEE Trans. Rob. **25**(4), 912–926 (2009)
6. Gerkey, B.P., Matarič, M.J.: Sold!: auction methods for multirobot coordination. IEEE Trans. Robot. Autom. **18**(5), 758–768 (2002)
7. Guessoum, Z., Briot, J.P., Faci, N., Marin, O.: Toward reliable multi-agent systems: an adaptive replication mechanism. Multiagent Grid Syst. **6**(1), 1–24 (2010)
8. Hayashi, H.: Comparing repair-task-allocation strategies in MAS. In: International Conference on Agents and Artificial Intelligence, vol. 1, pp. 17–27 (2017)
9. Johansson, F., Falkman, G.: Real-time allocation of firing units to hostile targets. J. Adv. Inf. Fusion **6**(2), 187–199 (2011)
10. Lagoudakis, M.G., Markakis, E., Kempe, D., Keskinocak, P., Kleywegt, A., Koenig, S., Tovey, C., Meyerson, A., Jain, S.: Auction-based multi-robot routing. In: International Conference on Robotics: Science and Systems, pp. 343–350 (2005)
11. Macarthur, K.S., Stranders, R., Ramchurn, S.D., Jennings, N.R.: A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In: AAAI Conference on Artificial Intelligence, pp. 701–706 (2011)
12. Okimoto, T., Schwind, N., Clement, M., Riberio, T., Inoue, K., Marquis, P.: How to form a task-oriented robust team. In: International Conference on Autonomous Agents and Multiagent Systems, pp. 395–403 (2015)
13. Ramchurn, S.D., Farinelli, A., Macarthur, K.S., Jennings, N.R.: Decentralized coordination in RoboCup rescue. Comput. J. **53**(9), 1447–1461 (2010)
14. Rahimzadeh, F., Khanli, L.M., Mahan, F.: High reliable and efficient task allocation in networked multi-agent systems. Auton. Agent Multi-agent Syst. **29**(6), 1023–1040 (2015)
15. Smith, R.G.: The contract net protocol: high-level communication and control in a distributed problem solver. IEEE Trans. Comput. **C–29**(12), 1104–1113 (1980)
16. Suárez, S., Quintero, C., de la Rosa, J.L.: Improving tasks allocation and coordination in a rescue scenario. In: European Control Conference, pp. 1498–1503 (2007)
17. Xin, B., Chen, J., Zhang, J., Dou, L., Peng, Z.: Efficient decision making for dynamic weapon-target assignment by virtual permutation and tabu search heuristics. IEEE Trans. Syst. Man Cybern. Part C **40**(6), 649–662 (2010)