Applied Soft Computing Journal 83 (2019) 105628

Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

A real-time decentralized algorithm for task scheduling in multi-agent system with continuous damage☆

Junwu Zhu *, Jieke Shi, Zhou Yang, Bin Li

School of Information Engineering, Yangzhou University, Yangzhou Jiangsu, China

HIGHLIGHTS

- A common model with continuous dynamic damage of task scheduling problems in agent rescue scenario is proposed.
- We design a heuristic algorithm based on greedy strategy to obtain the optimal dynamic scheduling strategy of agents.
- For practical application, an automatic negotiation framework is designed which realizes the real-time distributed automated negotiation.
- Using Game Description Language (GDL) as a tool, an automated negotiation algorithm is implemented.

ARTICLE INFO

Article history:
Received 2 March 2019
Received in revised form 21 May 2019
Accepted 25 May 2019
Available online 18 July 2019

Keywords:
Task scheduling
Automated negotiation
Greedy strategy
Distributed artificial intelligence
General game playing

ABSTRACT

In this paper, a common model of task scheduling problems in agent rescue scenario is proposed, in which tasks with continuous dynamic damage are introduced to capture the emerging applications of using rescue robots and other resources to enhance human disaster rescue capability. Beyond this, we mainly focus on finding the optimal task scheduling strategy. We design a heuristic algorithm based on greedy strategy to obtain the optimal dynamic scheduling strategy of agents. Compared with solving global integer programming directly, the computational time is greatly reduced. The proof of the greedy strategy's validity is also demonstrated under some specific damage functions. By comparing with the two strategies commonly used in real life, it is proved that our strategy is optimal. For practical application, we design an automatic negotiation framework, which realizes the real-time decentralized automated negotiation of agents. Then, using Game Description Language (GDL) as a tool, an automated negotiation algorithm is implemented, which enables agents to adjust the plan dispersedly. Experiments show that the algorithm is more efficient than the centralized algorithm in the case of limited communication.

1. Introduction

In large distributed artificial intelligence systems, task scheduling problem is always of great importance. Multi-agent system is also widely used in this aspect, such as the distribution of rescue tasks, emergency Scheduling and so on. The so-called task scheduling problem can be simply summarized as assigning different agents to corresponding tasks with a coherent strategy to optimize a specific goal for a given set of tasks and agents. Because of the difference of geographical distribution and agent capability, different scheduling schemes consume different resources at the completion of tasks, and decision makers can make optimal scheduling schemes with the purpose of minimizing resource consumption.

In this paper, we consider a kind of practical tasks, taking fire as the main tasks. If we do not put it out, it will continue to damage the surrounding environment. An intuitive conclusion is that the greater the fire, the greater the damage rate to the surrounding environment, we call it the sustainability of damage. In addition, in different stages of fire, the speed of fire damage is also different. One obvious conclusion is that the larger the fire, the faster the damage to the surrounding environment. Obviously, for a single task, the sooner the task is completed, the less total damage it will cause to the surrounding environment. Our goal is to find a scheduling scheme that minimizes the total damage to the system caused by all tasks.

☆ This work was supported by the National Natural Science Foundation of China under Grant 61872313, the key research projects in education informatization in Jiangsu province (20180012), in part by the Postgraduate Research and Practice Innovation Program of Jiangsu Province under Grant KYCX18_2366, and in part by the Yangzhou Science and Technology under Grant YZ2017288, YZ2018076, YZ2018209, and Yangzhou University Jiangdu Highend Equipment Engineering Technology Research Institute Open Project under Grant YDJD201707, and Jiangsu Practice Innovation Training Program for College Students under Grant 201811117029Z.

* Corresponding author.
E-mail address: jwzhu@yzu.edu.cn (J. Zhu).

https://doi.org/10.1016/j.asoc.2019.105628
1568-4946/

Although centralized methods can find the optimal dynamic scheduling, it needs a central decision agent with strong computing power. And the agent should always have complete information about each parameter of the system. Both of these requirements are difficult to meet, especially the second one. Besides, rescue scenarios inevitably have the characteristics of information dispersion, noisy communication and so on, which has limited the applicability of traditional centralized methods, resulting in the lack of flexibility and robustness of many approaches. The dynamic of damage indicates that it is difficult to produce optimal results in a single scheduling, which requires the realization of real-time dynamic scheduling of agents.

1.1. Our contribution

In this paper, a common model of task scheduling problems in agent rescue scenario is proposed, in which tasks with continuous dynamic damage are introduced to capture the emerging applications of using rescue robots and other resources to enhance human disaster rescue capability. At the technical level, we first design a heuristic algorithm based on greedy strategy, and obtain the optimal dynamic scheduling strategy of agents with the idea of centralized programming. We can prove the validity of the greedy strategy under some specific damage functions, and use the solution obtained by the greedy strategy as the benchmark for subsequent experiments.

Then, we design a real-time decentralized algorithm with good robustness for practical application. Game theory is an effective resource scheduling tool, but it usually assumes that agents cannot communicate with each other, which in many cases will have a negative impact on the system. For this reason, we design an automatic negotiation framework in multi-agent domain, which realizes the real-time decentralized automatic negotiation of agents. Then, using Game Description Language (GDL) as a tool, an automatic negotiation algorithm is implemented, which proves that the algorithm can achieve the same optimization effect as centralized deployment. Finally, the joint completion of agents with different functions, namely fire brigade, ambulance and police, is discussed, and our task scheduling strategy is extended.

1.2. Related work

Task scheduling is the key problem in multi-agent system researches. For a given set of tasks and agents, a proper strategy should be applied to assign various agents to corresponding tasks, optimizing some certain targets. The problem of finding an optimal task scheduling is proved to be NP-completed [1,2]. There are a lot of research work in this field. Some researchers assume that each agent can only work on one task and a task can only be operated by one agent at a time [3–5]. It costs agents something (time, money and so on) to complete tasks. James [6,7] introduces a kind of tasks with cost growing over time and computes the optimal rescheduling solution to complete all tasks with least cost. Many works [8,9] focus on minimizing the total cost. To our best knowledge, we are the first to introduce such tasks with continuous and dynamic damage to system.

The task scheduling method can be divided into centralized methods and decentralized methods. In a centralized system, an agent has a higher status and is assumed to have full information of the system. This central agent computes the optimal or near-optimal decisions, maximizing the effectiveness of multi-agent systems. The centralized decision process is always modeled as integer programming [10]. Although these methods may find the optimal solution, requirements for full information are hardly satisfied [11], raising need for decentralized or decentralized scheduling method. In the distributed system, each agent need to make its own decision, which makes the system more flexible and robust. Agents can cooperate with each other to maximize the efficiency of the system. Negotiation is an effective way for distributed agents to cooperate with each other [12–15]. However, an agent can always only participate in one negotiation. If an agent wants to participate another one, he need to go off-line and re-program to adjust new negotiation protocol [16,17]. In other words, agents are not domain or protocol-independent. However, a agent often needs to complete different works with different teammates in various negotiation scenarios.

General Game Playing is a relatively new topic. Although earlier work has been done, it really started to draw widespread attention in the AI community after the introduction of GDL [18] and the organization of the annual AAAI GGP competition since 2005. Zhang [19] propose to use GGP to model automated negotiations, which shows the possibility to design protocol-independent agents for better task scheduling. Standing upon the shoulders of former researchers, we propose our model and method in the following sections.

2. Definitions of common model

In this section, firstly we define some notations and describe the task environment with continuous and dynamic damage. The optimized objective is pointed out. Then, we take a simple example to illustrate our model.

2.1. Task scheduling problem

The task scheduling problem can be concluded that for a given set of tasks and agents, a proper strategy should be applied to assign various agents to corresponding tasks, optimizing some certain targets. In this model, $R = \{1, 2, \ldots, N\}$ is the set of agents. It means that there are N agents available to be allocated. $T = \{1, 2, \ldots, M\}$ is the set of tasks.

We make some assumptions about the task and agent sets:

- The number of agents N is constant.
- There will be no more new tasks added to the system.

For a agent $i \in R$ in the task, c_i represents the agent i can complete in a unit time. For a task $j \in T$, we define that $w_j^t \geq 0$ is the remaining workload of task j at time t. Time in this model is discrete to be each single time point. A interval like $[t, t+1)$ is used to describe a period of time. C_j^t is the number of agents assigned to task j at time t. A agent can only be allocated to one task at a time point however a task can be operated by many agents at a time. The scheduling strategy at time t can be represented by a vector $C^t = (C_1^t, C_2^t, \ldots, C_M^t)$. To achieve better working efficiency, scheduling plans at different time points may be highly different. It means that agents can be reallocated frequently in the whole working process.

In the real world, different agents and tasks have different geographical distribution. It takes some time for a agent to come from one place to another. This time may be determined by the distance between the departure and destination, road conditions and the speed of agents themselves. We use the notation $TT(a, b)$ to represent the time required to transfer from task a to task b. $TT(a, b)$ is also called *transfer time*.

2.2. Continuous damage of tasks

The remaining workload of a task only relies on the initial setting and scheduling results at each time point. The changing of

workload can be represented by the following recursion formula:

$$w_j^{t+1} = w_j^t - \sum_{i=1}^{C_j^t} c_i \tag{1}$$

According to this formula, the agent scheduling strategy at time t will not change the remaining workload instantly. In the real world situation, agents need time to operate on tasks. In our model, agents work on tasks at time period $[t, t+1)$, updating remaining workload of tasks at time point $t+1$. We use a vector $\boldsymbol{W}^t = (w_1^t, w_2^t, \ldots, w_M^t)$ to represent remaining workload of all tasks at time point t.

One of the main innovations of this model is that tasks with continuous and dynamic damage are firstly introduced. We define a function $H : w \to R$ to quantity such damage. At some time point t, $H(w_j^t)$ represents the damage task j causes to the system during time period $[t, t+1)$ as w_j^t is the remaining workload of j at time t.

Agents cannot complete tasks when transferring from one to another. Transferring on road can be viewed as a waste of resources. In any period of time, the total amount of tasks completed by agents can be determined. When the amount of remaining workload is greater than the total working capacity of all agents, each agent must be assigned to a task, which means during this period, total workload completed is $\sum_{i=1}^{N} c_i$. When the amount of remaining workload is no more than the total working capacity of all agents, total workload completed is equal to remaining workload. Actually, this situation only happens at the final scheduling. If the initial workload is known, the time when all tasks are finished can be easily calculated:

$$t_d = \frac{\sum_{j=1}^{M} w_j^0}{\sum_{i=1}^{N} c_i} \tag{2}$$

As stated above, C_j^t is the number of agents assigned to task j at time t. A vector $\boldsymbol{C}^t = (C_1^t, C_2^t, \ldots, C_M^t)$ is used to describe the scheduling of all tasks at time t. It is expected that the amount of all $\boldsymbol{C}^t$ is a combination number. It can be thought of as putting M boards among N orderly arraigned items. The scheduling space is extremely large even at one time point. It is a quite challenging thing to compute the global optimal solution.

The target of our model is that when the completing time t_d is determined, choosing the proper rescheduling plan at each time point to minimize the total damage suffered by the system. The total damage can be represented as below.

$$\min \sum_{t=0}^{t_d} \sum_{j=1}^{M} H(w_j^t) \tag{3}$$

2.3. An illustrative example

In this section, we give a simple example to instantiate the above symbols and models. There are two tasks $T = \{1, 2\}$ and four agents $R = \{1, 2, 3, 4\}$ in the system. When $t = 0$, the initial workload of two tasks are $w_1^0 = 6$ and $w_2^0 = 3$, that is, $W^0 = (6, 3)$. The damage function of task 1 is $H_1(w) = w^2$ and that of task 2 is $H_2(w) = w^3$. For each agent, $c_i = 1$. Obviously, the finishing time t_d can be easily calculated that $t_d = \frac{6+3}{4} = 3$. After the third scheduling, all tasks will be completed. We list three possible scheduling plans.

- $C^0 = (1, 3)$, $C^1 = (4, 0)$, $C^2 = (1, 0)$
- $C^0 = (2, 2)$, $C^1 = (3, 1)$, $C^2 = (1, 0)$
- $C^0 = (3, 1)$, $C^1 = (2, 2)$, $C^2 = (1, 0)$

Actually, for this simple example, all the scheduling plans can be enumerated. According to our calculation, there are 7 possible scheduling plans in total. Next, we demonstrate the total damage of the system under the above three different plans.

We assume that scheduling at time t influences the remaining workload at time $t+1$. The damage in time period $[t, t+1)$ is determined by w_j^t. For the three plans, workload at time $t = 0$ is same that $\boldsymbol{W}^0 = (6, 3)$. So the damage during $[t, t+1)$ is same either that $H_1(6)+H_2(3) = 6^2+3^3 = 36+27 = 63$. However, due to the different scheduling at the last time point, at time $t = 1$, the remaining workloads of these plans are different, just showed below.

- $\boldsymbol{W}^1 = ((6-1), (3-3)) = (5, 0)$
- $\boldsymbol{W}^1 = ((6-2), (3-2)) = (4, 1)$
- $\boldsymbol{W}^1 = ((6-3), (3-1)) = (3, 2)$

Then the damage during $[1, 2)$ is:

- $5^2 + 0^3 = 25 + 0 = 25$
- $4^2 + 1^3 = 16 + 3 = 17$
- $3^2 + 2^3 = 9 + 8 = 17$

According to C^1 and $\boldsymbol{W^1}$, $\boldsymbol{W^2}$ is:

- $\boldsymbol{W}^2 = ((5-4), (0-0)) = (1, 0)$
- $\boldsymbol{W}^2 = ((4-3), (1-1)) = (1, 0)$
- $\boldsymbol{W}^2 = ((3-2), (1-1)) = (1, 0)$

Obviously, $\boldsymbol{W^2}$ is same for the three plans. Correspondingly, damage in $[2, 3)$ is same either that $1^2 + 0^3 = 1$. Then the total damage is that:

- $63 + 25 + 1 = 89$
- $63 + 19 + 1 = 81$
- $63 + 17 + 1 = 81$

It is clear that plan 2 and plan 3 is better, because they lead to smaller total damage to the system. We use a simple example to illustrate that different agent resource scheduling plans will have different impaction on the results. However, based on the above calculation alone, we cannot strongly prove that the plan 2 or 3 is optimal. Even in this simple case, there are 7 plans. We can enumerate all the scheduling strategies to find the optimal dynamic rescheduling strategy. Through thoroughly analysis, we can tell readers that the best solution is: $\boldsymbol{C}^0 = (3, 1)$, $\boldsymbol{C}^1 = (2, 2)$, $\boldsymbol{C}^2 = (1, 0)$.

Our target is to find an optimal strategy to minimize the total damage to the system. However, as shown above, the strategy space increase exponentially with the number of agents and amount of workloads. For this problem with practical meanings, it is worth considering to design efficient algorithms. In the next section, we demonstrate the characteristics of optimal solution and design a greedy strategy to compute it with a centralized idea.

3. Centralized scheduling strategy

In this section, we propose a centralized method to compute the optimal solution. We assign a agent as the central agent or we add a virtual agent to the system. It is assumed that this agent knows the full information of the task environment, like remaining workloads of each task, current scheduling situation or the damage function of each task. According to these information, this central agent deploys a heuristic algorithm based on greedy

strategy to compute the local optimal solution at each time point to find the global optimal solution. We prove the correctness of this strategy and compare the performance with a directly computing method.

3.1. Global and local optimal solution

3.1.1. Global optimal solution

We define that the optimal solution will minimize the total damage suffered by the system during the whole progress. This can be represented as an integer programming, the formula is that:

$$\begin{aligned} \min \quad & \sum_{t=0}^{t_d}\sum_{j=1}^{M} H(w_j^t) \\ \text{s.t.} \quad & \sum_{j=1}^{M} C_j^t \le N \\ & \sum_{j=1}^{M}\sum_{i=1}^{C_j^t} c_i \le \sum_{j=1}^{M} w_j^t \\ & w_j^{t+1} = w_j^t - \sum_{i=1}^{C_j^t} c_i \\ & c_i \text{is a no-negative constant}, \forall i, \\ & C_j^t \text{ is a no-negative integer}, \forall j, \forall t \end{aligned} \tag{4}$$

The Eqs. (4) and (5) ensure that when the amount of remaining workload is greater than the total working capacity of all agents, each agent must be assigned to a task, which means during this period, total workload completed is $\sum_{i=1}^{N} c_i$. When the amount of remaining workload is no more than the total working capacity of all agents, total workload completed is equal to remaining workload. The Eq. (6) ensure the change of workload according to scheduling strategies, which should be expanded in real coding that:

$$w_j^{t+1} = w_j^0 - \sum_{i=1}^{C_j^0} c_i - \sum_{i=1}^{C_j^1} c_i - \cdots - \sum_{i=1}^{C_j^t} c_i \tag{5}$$

It is easy to calculate that the total variables in this integer programming is $t_d \cdot M$, which indicate the huge challenge we face when solving this programming.

3.1.2. Local optimal solution

The scheduling strategy $\boldsymbol{C}^t = (C_1^t, C_2^t, \ldots, C_M^t)$ at time t will influence the remaining workload at time $t+1$. And the workload $\boldsymbol{W}^{t+1}$ also determine the total damage in $[t, t+1)$. We say an scheduling strategy C_*^t is local optimal at time t if and only if this strategy lead to the minimal damage during $[t, t+1)$ among all the strategies. This characteristic is shown in Definition 1.

C_*^t is local optimal, iff $D(C_*^t) \le D(C^t), \forall C^t \in C$

According to Definition 1, the local optimal solution can be determined by computing the following integer programming.

$$\begin{aligned} \min \quad & \sum_{j=1}^{M} H(w_j^t) \\ \text{s.t.} \quad & \sum_{j=1}^{M} C_j^t \le N \\ & \sum_{j=1}^{M}\sum_{i=1}^{C_j^t} c_i \le \sum_{j=1}^{M} w_j^t \\ & C_j^t \text{ is a no-negative integer}, \forall j, \forall t \end{aligned} \tag{6}$$

The amount of variables of this integer programming is M. But as we will show in the next section, it needs to be computed for t_d times to find the optimal solution.

3.2. A centralized strategy

At some time point t, $H(w_j^t)$ represents the damage task j cause to the system during time period $[t, t+1)$ as w_j^i is the remaining workload of j at time t. We think that there is a positive correlation between $H(w_j^t)$ and the remaining workload w_j^t. Namely, the damage of larger tasks must grow faster than smaller tasks.

The optimal scheduling for every time unit t, that is, the minimization of the damage $H(w_j^t)$, can eventually lead to the global optimal solution, when $\frac{\partial}{\partial t} H(w_j^t) \ge 0$.

The scheduling scheme obtained by the greedy strategy is $C[0, t_d] = \{C^0, C^1, \ldots, C^{t_d}\}$. Assume there exists a better solution $C_*[0, t_d] = \{C_*^0, C_*^1, \ldots, C_*^{t_d}\}$, which differs from the greedy solution. This means that at some time t_0 the better solution must assign agents differently than the greedy solution. The scheduling scheme of our greedy algorithm at time t_0 is $C^{t_0} = (C_1^{t_0}, C_2^{t_0}, \ldots, C_M^{t_0})$, the better scheduling is $C_*^t = (C_{1*}^{t_0}, C_{2*}^{t_0}, \ldots, C_{M*}^{t_0})$. By definition of the greedy algorithm, we know $\sum_{j=1}^{M} H(w_j^{t_0}) \le \sum_{j=1}^{M} H_*(w_j^{t_0})$. Thus we prove by induction that $\sum_{t=0}^{t_d}\sum_{j=1}^{M} H(w_j^{t_0}) \le \sum_{t=0}^{t_d}\sum_{j=1}^{M} H(w_j^t)$.

If $\sum_{j=1}^{M} H(w_j^{t_0}) < \sum_{j=1}^{M} H_*(w_j^{t_0})$, then there must exist $\sum_{t=0}^{t_d}\sum_{j=1}^{M} H(w_j^t) < \sum_{t=0}^{t_d}\sum_{j=1}^{M} H_*(w_j^t)$. If $\sum_{j=1}^{M} H(w_j^{t_0}) = \sum_{j=1}^{M} H_*(w_j^{t_0})$, there also exists $\sum_{t=0}^{t_d}\sum_{j=1}^{M} H(w_j^t) < \sum_{t=0}^{t_d}\sum_{j=1}^{M} H_*(w_j^t)$.

So, according to the greedy strategy, we can conclude $\sum_{j=1}^{M} H(w_j^{t_0+1}) \le \sum_{j=1}^{M} H_*(w_j^{t_0+1})$ at the next unit time t_0+1.

The remaining workload of the two scheduling schemes at time t_0 are respectively $w_j^{t_0}$ and $w_{j*}^{t_0}$. Due to a positive correlation between $H(w_j^t)$ and w_j^t, we know that $w_j^{t_0} \le w_{j*}^{t_0}$. We denote $w_{j*}^{t_0}$ by $w_{j*}^{t_0} = w_j^{t_0} + a_j, \forall j \in T, a_j \ge 0$. Thus $H(w_j^{t_0+1}) = H(w_j^t - \sum_{i=1}^{C_j^t} c_i)$ and $H(w_{j*}^{t_0+1}) = H(w_j^t - \sum_{i=1}^{C_j^t} c_i + a_j)$.

β_i is the slope between $w_j^{t_0+1}$ and $w_{j*}^{t_0+1}$, so we hold that:

$$H(w_{j*}^{t_0+1}) - H(w_j^{t_0+1}) = \beta_i \times a_j$$

$\frac{\partial}{\partial t} H(w_j^t) \ge 0$, so $\beta_i > 0$, we can conclude $\beta_i \times a_j > 0$. We can get

$$\sum_{j=1}^{M} H(w_j^{t_0+1}) \le \sum_{j=1}^{M} H_*(w_j^{t_0+1})$$

This means $\sum_{t=0}^{t_d}\sum_{j=1}^{M} H(w_j^t) \le \sum_{t=0}^{t_d}\sum_{j=1}^{M} H_*(w_j^t)$. This is a contradiction to the fact that $C_*[0, t_d]$ is a better solution, therefore the optimal scheduling for every time unit t can eventually lead to the global optimal solution.

3.3. Effectiveness against with other method

We compare the optimal solution result with other methods to demonstrate the effectiveness of our approach. There is no former work on scheduling strategies of such tasks we described. We introduce two common strategies often used in real world:

- **Average strategy**: allocating agents to each task averagely. If there are 10 agents and 5 tasks, then each task is assigned to 2 agents. If $\frac{N}{M}$ is not an integer, the remainder agents are allocated with roulette.

Table 1
Task parameter settings.

	Task 1	Task 2	Task 3	Task 4	Task 5
Damage function	w^2	$\frac{1}{3}w^3$	$\frac{1}{4}w^4$	w^2	$12w$
Initial workload	9	10	8	11	12

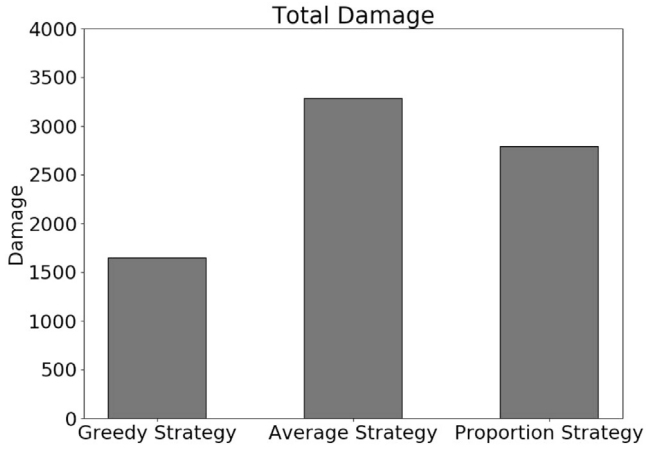

Fig. 1. Shows the total damage comparison among three methods.

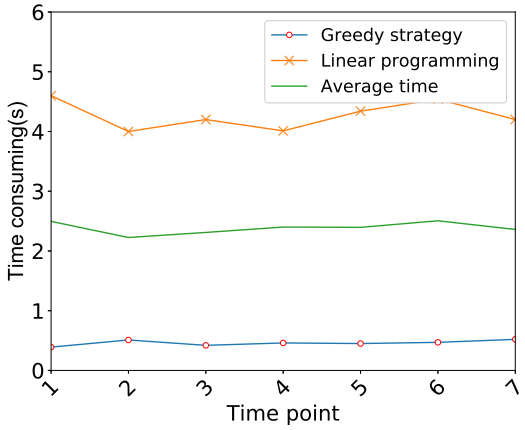

Fig. 2. Shows the time comparison among two methods.

- **Proportion strategy**: allocating agents to each task according to the proportions of workload. The central agent calculates proportions of workload of each task then allocate agents according to it with roulette.

We compare the two methods with our optimal solution in a task environment. We assume that there are 8 agents available. The environment setting is illustrated in Table 1:

The Fig. 1 shows the total damage comparison among three methods. The first rank is the total damage of our optimal solution, which is showed to be the least. The highest one is the average method and the third one is the proportion method.

Beyond this, we compare the execution time of the algorithm. Using the task settings in Table 1, we use linear programming to calculate the global optimal solution directly. In Fig. 2, the blue curve represents the running time of the greedy strategy. The orange one is to calculate the global optimal solution directly. The result of direct calculation is at each beginning, so the average time of each time point is calculated to compare the two strategies more intuitively.

4. A real-time decentralized scheduling algorithm

In the centralized method, the solution only depends on the central agent who computes everything. However, other agents only obey the orders mechanically, which does not reflect the intelligence of agents or the multi-agent system. Moreover, at the end of the previous section, we explained the drawbacks of such centralized approaches and the enormous difficulty we need to face when using then in real world. Therefore, we hope to design a decentralized task scheduling method, allowing agents to adjust scheduling plans in a decentralized way.

4.1. A general automated negotiation framework

We model the automated negotiation in agent task scheduling scenarios with GDL. A kind of automated negotiation framework is proposed. The main idea of *General Game Playing* is to design domain-independent game players, who can participate in any games described with GDL. We also hope to design such general framework about negotiations, which allow us to design domain and protocol-independent agent to join various negotiation scenarios without going off and re-programming. Based on GGP, using GDL to describe the agent task scheduling negotiations allows us to take good use of lots of current GGP techniques, which can accelerate development in this area. According to some GDL and automated negotiation researches, we initially define a framework at agent task scheduling automated negotiation as follows:

- $Ag = \{a_1, a_2, \ldots, a_n\}$ is the set of agents who participate in a negotiation. The negotiation host is not included in Ag.
- $Ac = (Ac_1, Ac_2, \ldots, Ac_n)$ is a tuple in which Ac_i represents the action set of negotiation agent $i \in Ag$.
- W is a non-empty set of states. $w_0 \in W$ is the initial state of a negotiation. $T \subset W$ is the set of terminal states $w \in W$.
- $L = (L_1, L_2, \ldots, L_n)$ is a tuple where each $L_i : W \backslash T \rightarrow 2^{A_i}$ is the legal function for a_i. It determines the actions a_i can take at some state.
- A so-called state update function is defined as $u : W \times Ac_i \rightarrow W$ to represent at what condition, a state will change to another.
- Agr is the set of all possible negotiation agreements. The function $Q : T \rightarrow Agr$ maps each terminal state to an agreement.

Such definition of negotiation is unified GGP , which allows us to use GDL to describe automated negotiation scenarios in agent task scheduling. In the next section, against the tasks raised in this paper, we design a negotiation protocol and strategy, then represent them with GDL.

Now we propose a decentralized task scheduling algorithm. We view each task as agents and call them "task agent". We assign one of them as the central agent. Although there is still a so-called central agent existing in this method, it just aims to maintain the negotiation and do little computation. It can be replaced by other external agents. This decentralized method can be described as follow:

Step 1. For all $j \in Ag$, agent j proposes two values: $u_j(1)$ and $u_j(-1)$. The central agent combines all those values as two sequences $U(1)$ and $U(-1)$. If $\min U(-1) > \max U(1)$, the algorithm terminates.

Step 2. Assuming that $u_i(-1)$ is the minimal in $U(-1)$, the corresponding agent i picks agent $j \neq i$ whose $u_j(1)$ is maximal in $U(1)$. If the condition $u_i(-1) < u_j(1)$ is satisfied, agent i will transfer a agent of itself to agent j then both agent i and j is deleted from $U(1)$ and $U(-1)$. If the condition is not satisfied, algorithm returns to Step 1.

This procedure can be represented in pseudo-code as below:

Algorithm 1: A real-time decentralized scheduling algorithm

```
repeat
    U(1) = U(−1) = ∅;
    for each i ∈ Ag do
        propose(u_i(−1), u_i(1));
        U(1) = U(1) ∪ u_i(1);
        U(−1) = U(−1) ∪ u_i(−1);
        repeat
            i = arg min U(−1);
            delete i from U(1) and U(−1);
            j = arg max U(1);
            if u_i(−1) < u_j(1) then
                agent i gives a task to agent j;
                delete j from U(1) and U(−1);
            else
                Algorithm Stops
        until true
until true
```

We assume that at some time t the amount of agents at task j is C_j^t. Then the damage caused in $[t+1, t+2)$ is $H_j(w_j^t - C_j^t)$. If we add one more agent to task j at time t, then the damage changes into $H_j(w_j^t - C_j^t - 1)$. We define $u_j(1) = H_j(w_j^t - C_j^t) - H_j(w_j^t - C_j^t - 1)$. Similarly, $u_j(-1)$ is defined as $H_j(w_j^t - C_j^t + 1) - H_j(w_j^t - C_j^t)$. There are some special cases like $C_j^t = 0$, $C_j^t = w_j^t$ or $C_j^t = N$. In these cases, $u_j(1)$ or $u_j(-1)$ is assigned as -1.

4.2. Effectiveness against with centralized strategy

In this section, we show the application of our centralized strategy in *RoboCup Rescue Simulation*(RCRS). RCRS is a simulation of disaster response scenarios in large cities (for more information, see http://www.robocurescue.org). This is a complex environment in which the agent team must assign and execute tasks using incomplete information in real time in an uncertain environment. Therefore, it provides an ideal platform for evaluating the efficiency of our control strategy. Fig. 5 gives an example of the maps we used in our RCRS experiments. More specifically, in order to clearly identify the effect of using our algorithm, we consider a limited version of RCRS, which does not contain blocked roads, so the only problem is to coordinate rescue agents to rescue injured civilians. Therefore, the overall goal is to coordinate the actions of emergency services so that they can save as many casualties as possible in the shortest possible time. We are now discussing the design of our centralized strategy experiment to evaluate our performance.

Specifically, we run on three standard RCRS maps, Kobe, Berlin and Foligno, each containing 100 civilians and 20 rescue agents. Each simulation lasts 300 time steps, and each experiment runs 20 times. The performance of the experiment is evaluated by the scores obtained at the end of the simulation. (See Fig. 3.)

We did two batches of experiments. Firstly, the communication range of the agent is not limited. We use these experimental results to compare the effectiveness of our discentralized algorithm to centralized algorithm. Fig. 4 shows the average score of

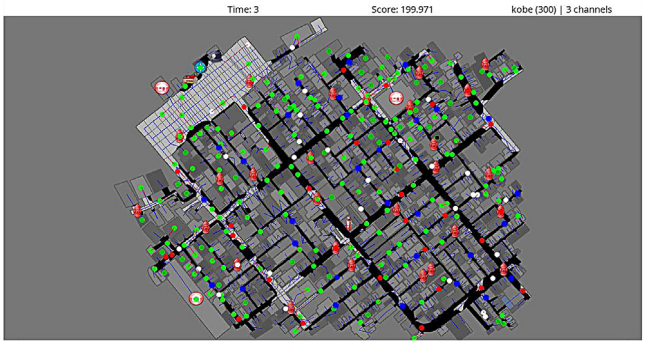

Fig. 3. Example of a map used in RoboCup Rescue Simulation.

100% compared with the centralized method in the three experiments. Although the score between decentralized algorithm and centralized heuristic algorithm is significant in statistics, the solution of decentralized generation is less than 8% of the centralized method. Considering comprehensively, these results show that the decentralized method is a good approximate solution in RCRS scenarios.

More specifically, both algorithms perform better in Kobe and Foligno scenarios than in Berlin scenarios, we think it is because Berlin is a larger and unreasonably structured map than Kobe or Foligno, it is difficult to complete rescue missions there quickly. Our algorithm will lead to the aggregation of agents, which means that agents will deal with the same tasks after negotiation and transfer to new tasks together after dealing with the current tasks. This often results in serious resource overscheduling to tasks with very close deadlines, which has a significant negative impact on the quality of the generated solutions. Excess scheduling is at the cost of other tasks not being completed. It is likely that the deadlines for other tasks have passed before they can be effectively handled, which means a serious waste of resources. The centralized heuristic algorithm is set to share the same world view in the current complete information. Therefore, centralized algorithm can avoid this defect. That is to say, because it is a centralized algorithm, it can assign the most minimum number of agents to complete most tasks, thus releasing resources allocated to other tasks is about to expire. And if the task cannot be completed before the deadline, it will not try. In the case of ideal computing state and complete communication, this heuristic algorithm can lead to better completion of all tasks by agents between tasks.

However, under the circumstance of limited communication range, serious resource scheduling errors may occur in both algorithms. Specifically, since a single agent shares their world view with all other agents, the scope of communication limits the ability to share information with each other. However, the behavior and performance of the decentralized algorithm remain relatively stable, and only when the communication distance is severely limited will there be oscillation. Like a centralized heuristic approach, this is the result of agents sharing information only after they enter the communication range with each other. However, since our decentralized algorithm only calculates the new optimal response strategy based on the communication strategy, it avoids other costly misscheduling, and thus produces better performance when the communication scope is limited. (See Figs. 6 and 7.)

4.3. Implementation with GDL

$Ag = T$, because the negotiation participants are task agents.

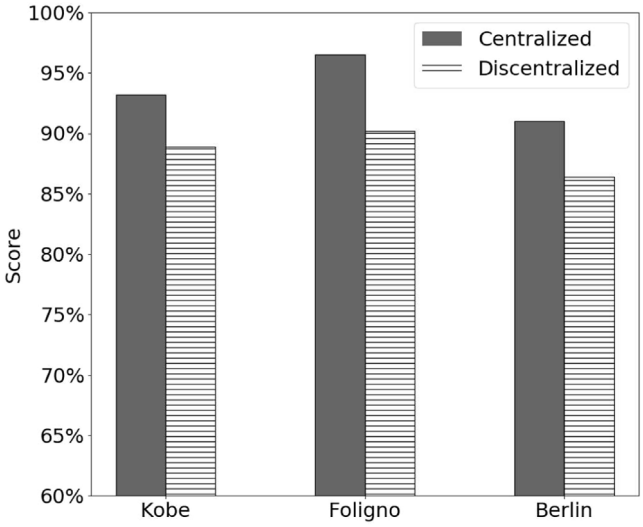

Fig. 4. Comparing the algorithms across three maps under unrestricted communication.

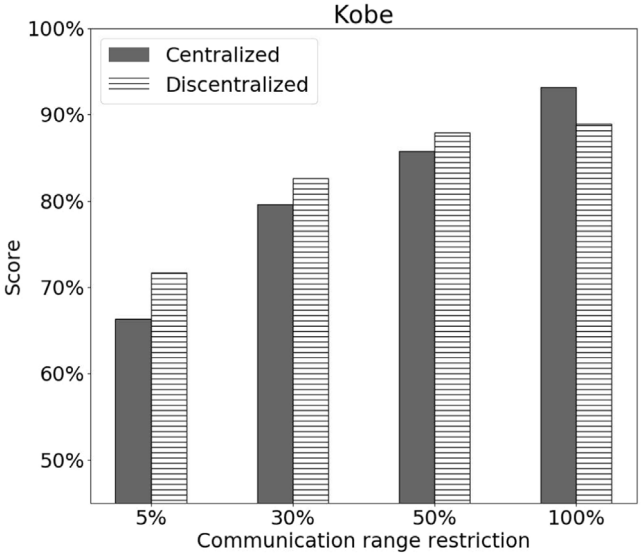

Fig. 5. Comparing the algorithms across Kobe map under restricted communication.

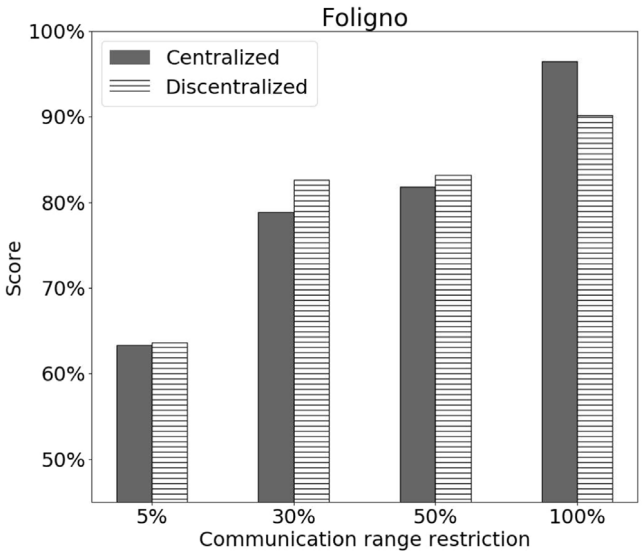

Fig. 6. Comparing the algorithms across Foligno map under restricted communication.

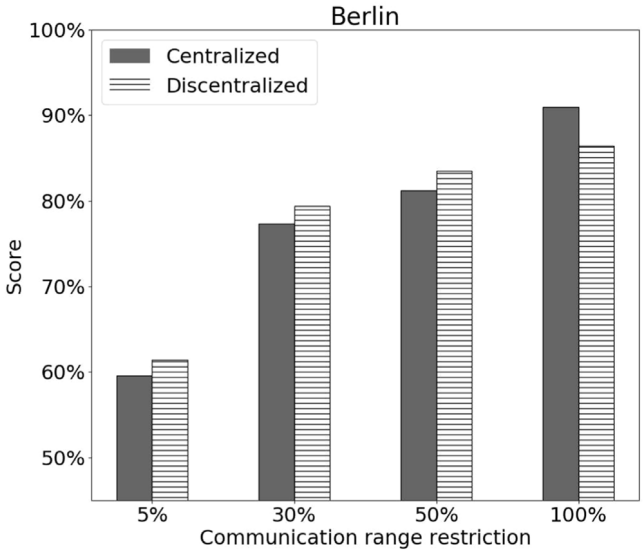

Fig. 7. Comparing the algorithms across Berlin map under restricted communication.

$Ac_i = \{propose(u_i(1), u_i(-1)), give(j), noop\}$. $propose(u_i(1), u_i(-1))$ means to propose $u_i(1)$ and $u_i(-1)$ to host and other agents. $give(j)$ means to give a agent to agent j. $noop$ means no-operation.

$w \in W$. The state w can be represented as a tuple (I, K) consisting of a sequence I and a integer K. The sequence I consists of 0, -1 and 1, whose length is M, for example $I = (0, 1, -1, 0, \ldots, 0)$. The meaning of 0 at the first position is that till the current negotiation, task agent 1 do not give a agent neither accept a agent. Besides numbers on second and third order, other numbers are all 0. Then 1 and -1 on the second and third position mean that task agent 3 give a agent to task agent 2. According to the prove of greedy strategy, we can go towards the final global optimal solution via find local optimal solution at each time point step by step. As showed in Algorithm 1, negotiations at each time can be viewed as a serious of sub-negotiations (Line x y). The sub-negotiation can be divided into propose state and adjust state. The second parameter K of $w = (I, K)$ aims to distinguish the two states. In a sub-negotiation, one agent need to act twice. The total number of steps in a sub-negotiation is a constant $2M$. We initialize $K = 2M$. Every time a agent takes actions, K reduces by 1. When $K > M$, it is propose state. When $M \geq K > 0$, it is adjust state. When $K = 0$, the sub-negotiation terminates.

As stated above, once a agent takes actions, K reduces by 1. In the propose state, this feature can be represented as $u(I, K), i.propose(u_i(1), u_i(-1)) = (I, K - 1)$. However, in the adjust state, if an agent i takes action $give(j)$, giving a agent of his own to task agent j, in the sequence I, $I[i]$ reduces by 1 and $I[j]$ increases by 1. It can be represented as $u(((\ldots, 0, \ldots, 0, \ldots), K), i.give(j)) = ((\ldots, -1, \ldots, 1, \ldots), K - 1)$. In this formula, two 0s at the left side are on the ith and jth place. $i.give(j)$ means that task agent i delivers a agent to task agent j.

We use GDL to implement this negotiation protocol on General Game Playing platform. We did not implement the agents negotiation strategy by coding, deployed the strategy by human operation. It comes to the fact that after a serious sub-negotiation we can achieve a local optimal solution. By doing so repeatedly, the global optimal solution will be reached.

It is of little meaning to compare the running time of our negotiation method and the centralized method in Section 4. Because the negotiation method requires a branch of communication among agents, which is time-consuming. However, the actual computational burden of the negotiation method will be shared by each agent participating in the task, so it is more practical in the real world.

5. Conclusion and future work

In this paper, we are the first to introduce a novel and practical kind of task into agent resource scheduling field, which we call tasks with continuous and dynamic damage. Moreover we design a centralized multi-period greedy strategy, to compute the global optimal strategy. The prove of the greedy strategy's correctness is also demonstrated. Then we propose an decentralized scheduling algorithm for agents to adjust schedules themselves.

Due to limited abilities of authors, there are many shortage in this work. On one hand, the workload of a task will grow naturally. For example, a fire in a forest will burn down nearby woods. The burned woods can be considered as damage. But the fire spreads as well. The scale of fire will increase, which means the workload grows over time. On the other hand, in this paper, the transfer time of rescheduling is not considered. In some situations it can be ignored, however, is can be non-negligible in other occasions. Both natural growing of workload and non-zero transfer time can make the centralized method invalid. In the future work, we will take these two parameters into consideration to modify the model.

Growth of workload, damage to system and transfer time, these three aspects are closely related to the scheduling plans. Moreover, they affect each other making it hard to deploy the centralized method. In this paper, the damage function is assumed to be a-prior and static. But in the real world, this assumption is hardly satisfied. In forest fires, a strong wind or a heavy rain can have huge impacts on parameters. Therefore, to deal with the problem of incomplete information and dynamic random change of information, we design a decentralized automated scheduling method, allowing agents to communicate with each other and allocate resources in decentralized manners [20–22].

In this paper, it is an initial work of our research. In the real-world situations, compared with centralized method, automated negotiations are more practical to deploy. Inspired by Jonge [19], we think that using GDL to describe negotiation protocol means we can apply many well-developed GGP technologies to task scheduling researches, contributing to the possibilities of general automated negotiations in multi-agent field. As an initial research, there are a lot to improve. In the future work, we need to investigate in different negotiation scenarios widely to construct a general framework. Efficient protocol need to be designed to facilitate process and reduce communication consuming. GDL is used to describe some simple automated negotiations. For more strong expression, expanded GDL like SGL [23] to implement protocols and strategies is a research direction as well.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.asoc.2019.105628.

References

[1] B. An, V. Lesser, Characterizing contract-based multiagent resource allocation in networks, IEEE Trans. Syst. Man Cybern. B 40 (3) (2010) 575–586.
[2] J. Shi, Z. Yang, J. Zhu, An auction-based rescue task allocation approach for heterogeneous multi-robot system, Multimedia Tools Appl. (2018) 1–10.
[3] H.L. Choi, L. Brunet, J.P. How, Consensus-based decentralized auctions for robust task allocation, IEEE Trans. Robot. 25 (4) (2009) 912–926.
[4] A. Kleiner, A. Farinelli, S. Ramchurn, et al., Rmasbench: benchmarking dynamic multi-agent coordination in urban search and rescue, in: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems, International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 1195–1196.
[5] Y. Zhang, L.E. Parker, Task allocation with executable coalitions in multirobot tasks, in: 2012 IEEE International Conference on Robotics and Automation, IEEE, 2012, pp. 3307–3314.
[6] J. Parker, M. Gini, Tasks with cost growing over time and agent reallocation delays, in: Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems, International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 381–388.
[7] H. Lu, B. Li, J. Zhu, et al., Wound intensity correction and segmentation with convolutional neural networks, Concurr. Comput.: Prac. Exper. 29 (6) (2017) e3927.
[8] S. Amador, S. Okamoto, R. Zivan, Dynamic multi-agent task allocation with spatial and temporal constraints, in: Twenty-Eighth AAAI Conference on Artificial Intelligence, 2014.
[9] L. Ren, Y. Yu, Z. Cao, et al., An optimal task allocation approach for large-scale multiple robotic systems with hierarchical framework and resource constraints, IEEE Syst. J. (99) (2017) 1–4.
[10] H.W. Kuhn, The Hungarian method for the assignment problem, Nav. Res. Logist. Q. 2 (1–2) (1955) 83–97.
[11] G.A. Korsah, A. Stentz, M.B. Dias, A comprehensive taxonomy for multi-robot task allocation, Int. J. Robot. Res. 32 (12) (2013) 1495–1512.
[12] L. Jin, S. Li, Distributed task allocation of multiple robots: A control perspective, IEEE Trans. Syst. Man Cybern. Syst. 48 (5) (2016) 693–701.
[13] X. Zheng, S. Koenig, K-swaps: Cooperative negotiation for solving task-allocation problems, in: Twenty-First International Joint Conference on Artificial Intelligence, 2009.
[14] R. Stranders, L. Tran-Thanh, F.M.D. Fave, et al., DCOPs and bandits: Exploration and exploitation in decentralised coordination, in: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1, International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 289–296.
[15] S.D. Ramchurn, A. Farinelli, K.S. Macarthur, et al., Decentralized coordination in robocup rescue, Comput. J. 53 (9) (2010) 1447–1461.
[16] G. Wang, T.N. Wong, X. Wang, An ontology based approach to organize multi-agent assisted supply chain negotiations, Comput. Ind. Eng. 65 (1) (2013) 2–15.
[17] H. Lu, J. Zhu, B. Li, et al., A fast level set model for intensity inhomogeneity correction in eHealth analysis system, in: 2015 Third International Conference on Advanced Cloud and Big Data, IEEE, 2015, pp. 180–183.
[18] M. Genesereth, N. Love, B. Pell, General game playing: Overview of the AAAI competition, AI Mag. 26 (2) (2005) 62.
[19] D. De Jonge, D. Zhang, Using GDL to represent domain knowledge for automated negotiations, in: International Conference on Autonomous Agents and Multiagent Systems, Springer, Cham, 2016, pp. 134–153.
[20] L. Teng, J. Zhu, B. Li, et al., A voting aggregation algorithm for optimal social satisfaction, Mob. Netw. Appl. 23 (2) (2018) 344–351.
[21] J. Zhu, H. Song, Y. Jiang, et al., On cloud resources consumption shifting scheme for two different geographic areas, IEEE Syst. J. 11 (4) (2016) 2708–2717.
[22] L. Jin, S. Li, H.M. La, et al., Dynamic task allocation in multi-robot coordination for moving target tracking: A distributed approach, Automatica 100 (2019) 75–81.
[23] D. Zhang, M. Thielscher, A logic for reasoning about game strategies, in: Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.