

# A Negotiation Method for Task Allocation with Time Constraints in Open Grid Environments

Yan Kong, Minjie Zhang, Dayong Ye and Xudong Luo

## 1 Introduction

In recent years, more and more attention has been paid to task allocation both in research [17, 21] and applications [4, 18]. Generally speaking, task allocations are carried out mainly in two circumstances: (i) agents are cooperative, and (ii) agents are noncooperative (also known as market-based), and this paper addresses the task allocation in noncooperative grid environment. In a cooperative circumstance, agents work cooperatively to finish tasks trying to minimize the team's cost (or maximize the team's profit). In a market-based circumstance, some agents provide their resources to others for earning profits, while some agents consume others' resources to execute their tasks. The agents that provide resources are called providers or sellers, while the agents who consume others' resources are called consumers or buyers. Both providers and consumers always try to maximize their own profits while overlook others' profits. The most initial market-based task allocation is from Amazon that attempts to make profit(s) through renting its resources,<sup>1,2</sup> and a variety of both system [11, 13] and market [3, 16] structures for task allocation have been explored recently.

Some approaches have been developed for market-based task allocation [6, 8]. In the early research of market-based task allocation, auction method, such as

---

Y. Kong (✉) · M. Zhang · D. Ye  
School of Computer Science and Software Engineering, University of Wollongong,  
Wollongong 2522, Australia  
e-mail: yk573@uowmail.edu.au

X. Luo  
Institute of Logic and Cognition, Sun Yat-sen University,  
Guangzhou 510275, Guangdong, China

<sup>1</sup> <http://aws.amazon.com/ec2/>.

<sup>2</sup> <http://aws.amazon.com/ebs/>.

the Vickrey-Clarke-Groves auction [5], was introduced, in which all the resource consumers submit their needed resource information and bids to an auctioneer. However, auction suffers from the single point of failure problem. If the single point of failure happens, the entire system will stop working. Hence, it is undesirable in any system with a goal of high availability or reliability, for example, business practices, software applications, or other industrial systems [9]. Moreover it is also difficult to choose an agent that runs the auction and will be trusted by other agents due to the selfishness of agents [2].

To address the issue, many researchers tried to introduce negotiation-based methods to solve task allocation problems. For example, Jennings et al. introduced the prospects and challenges of negotiation in task allocation in [14]. Fatima et al. introduced negotiation-based method when allocate tasks and resources in [10]. Gatti et al. [12] proposed a negotiation-based method for task allocation with uncertain negotiation deadlines. Later on, An et al. [1] proposed another negotiation-based method that dealt with both the deadline and reserve price. In [2], An et al. further considered resource competition, deadline, reserve price and cost for negotiation.

With the development of grid systems and applications in broad domains and the nature of Internet, most grid environments have become open and dynamic, in which grid consumers and providers can enter and leave at any time. The problem of task allocation in such a grid environment is how to assign a set of resources to a set of tasks in the circumstances that both resources and tasks may change unpredictably as time progresses [17].

To address the challenge of dynamics and openness of grid environments, this paper presents a negotiation-based task allocation method. Due to the dynamics and openness, it is hard to apply a central controller. Hence, in this method, there is no central controller. Each agent only has a local view and consumers find potential resources through peep-to-peer (p2p) neighborhood relationships. The resources in our method include loosely coupled resources in grids, such as the grid computing, the computer storage, or even some virtual resources, for example, the electric data library of some university. The resources can be used for only one task at a moment, and it can be used to another task as soon as it is released from the current task. Each agent can contain multiple types of resources in our method. Because a p2p network is decentralized in nature, the proposed method can meet the decentralization requirement in general grid environments. After finding the potential resources, consumers begin to negotiate with the resource providers. Because of the dynamics and openness of the environment, it is difficult for agents to decide the optimal contract prices, so that agents are allowed to negotiate with more than one opponents, and thus the decommitment and penalty are necessary and considered for negotiation.

We evaluate our method through comparing it with the two other methods of task allocation (i.e., the method proposed in [2] and the method proposed in [7]). The experimental results show that our method outperforms the two state-of-art methods in terms of success rate of task allocation and the total profit agents gained in the grid environment in some specific circumstances.

The rest of this paper is organized as follows. Section 2 describes the problems and introduces the procedure of our task allocation method. Section 3 presents the

negotiation method. Section 4 presents the experimental results and analysis. Section 5 discusses the related work. Finally, Sect. 6 concludes the paper and points out the future work.

## 2 Problem Description and the Procedure of the Task Allocation

The problem that this paper will solve is how to allocate tasks under time constraints in dynamic, open grid environments. A task in this paper is specified by required resources, the generation time, the deadline (i.e., the latest start time) of the task, and the maximum reward that the task's owner can gain when the task is allocated successfully (we assume that a task will be completed as long as it is allocated successfully). Formally, a task can be defined as follows:

**Definition 1** A task, denoted as  $\tau_k$ , is 5-tuple  $(R_k, t_g, t_{ls}, \bar{r}, t_d)$ , where  $R_k$  is the resource set required by  $\tau_k$ , i.e., all the resources required by  $\tau_k$ ,  $t_g$  is the generation time of  $\tau_k$ ,  $t_{ls}$  is the deadline (i.e., the latest start time) of  $\tau_k$ ,  $\bar{r}$  represents the maximum reward that  $\tau_k$ 's owner can gain after  $\tau_k$  is completed successfully, and  $t_d$  is the duration time of  $\tau_k$ , i.e., the time needed to complete  $\tau_k$ .

Each node in the grid environment is an agent, and each agent has its own unique *ID*, resource set, and neighborhood. When an agent needs other agents' resources to execute its own tasks, it is called a consumer; when an agent provides its own resources to other agents, it is called a provider. An agent can be either a consumer or a provider or even both. Formally, we have:

**Definition 2** Agent  $a_i$  is 3-tuple  $(ID_i, R_i, Set_i)$ , where  $ID_i$  (a non-negative integer) is the unique identifier of  $a_i$ ,  $R_i$  is the resource set that  $a_i$  owns, i.e., all the resources owned by  $a_i$ , and  $Set_i = \{a_{i1}, \dots, a_{ik}\}$  is  $a_i$ 's neighbor set where  $k$  (a positive integer) is the number of  $a_j$ 's neighbors.

The communication between any two agents is through passing messages. In general, messages can be classified into the following five types. (i) Request message for building up neighborhood (*ReqNeighbor*): When agent  $a_i$  enters into the grid environment, in order to build up its neighborhood, it sends *ReqNeighbor* to the randomly chosen agents. (ii) Reply message for accepting *RepNeighbor*: If agent  $a_j$  receives a request message, *ReqNeighbor*, from agent  $a_i$ ,  $a_j$  will accept the request by sending back a reply message, *RepNeighbor*, to  $a_i$ . (iii) Request message for executing tasks (*ReqExecute*): When agent  $a_i$  needs other agents' resources to execute its tasks, it sends request messages (*ReqExecute*) to all of its neighbors. (iv) Reply message for executing tasks (*RepExecute*): When agent  $a_j$  receives a *ReqExecute* message from agent  $a_i$ ,  $a_j$  checks whether or not its own resources can meet the task's resource requirements. If so,  $a_j$  will send a reply message, *RepExecute*, back to  $a_i$ . (v) Heart beat message, which is used by agents to keep neighborhoods. These five types of messages can be formally defined as follows:

**Definition 3** A request message sent from agent  $a_i$  to agent  $a_j$  for building up neighborhood is 2-tuple  $(ReqNeighbor_{ij}, ID_i)$ , where  $ReqNeighbor_{ij}$  represents the message is sent from agent  $a_i$  to request building up neighborhood with  $a_j$ , and  $ID_i$  is the  $ID$  of agent  $a_i$ .

When agent  $a_j$  receives a request message  $(ReqNeighbor_{ij}, ID_i)$  from  $a_i$ , it replies  $a_i$  with a reply message defined as follows:

**Definition 4** A reply message sent from  $a_j$  to  $a_i$  to reply the message  $(Reqneighbor_{ij}, ID_i)$  is 2-tuple  $(RepNeighbor_{ji}, ID_j)$ , where  $RepNeighbor_{ji}$  represents that the message is sent by agent  $a_j$  to reply the request message sent from  $a_i$  and  $ID_j$  is the  $ID$  of agent  $a_j$ .

Due to the dynamics and openness of grid environment, in our method, there is no central controller. The agents judge whether their neighbors are still active in the environment through heart beat messages which will be defined by:

**Definition 5** A heart beat message sent from  $a_i$  to  $a_j$  is 3-tuple  $(HeartBeat_{ij}, ID_i, ID_j)$ , where  $ID_i$  and  $ID_j$  are the  $ID$  numbers of  $a_i$  and  $a_j$ , respectively.

In particular, an agent keeps sending heart beat messages to its neighbors once each time period. If an agent does not receive any heart beat message from a neighbor in the past one time period, it assumes that the neighbor has left the grid environment.

The above are the definitions of the messages for neighborhoods, while the following definitions are the messages for executing tasks.

**Definition 6** A request message sent from  $a_i$  to  $a_j$  for executing  $a_i$ 's task is 4-tuple  $(ReqExecute_{ij}, ID_i, \tau_k, HL)$ , where  $ReqExecute_{ij}$  represents that the message is sent from agent  $a_i$  to request  $a_j$  to execute task  $\tau_k$  of  $a_i$ .  $HL \geq 1$  is a hop limitation, which prevents the request message from being transmitted endlessly.

**Definition 7** A reply message sent from  $a_j$  to  $a_i$  for  $(ReqExecute_{ij}, ID_i, \tau_k, HL)$  is 4-tuple  $(RepExecute_{ji}, ID_j, t_s, \tau_k)$ , where  $RepExecute_{ji}$  represents that the message is sent from  $a_j$  to  $a_i$  to reply the message  $(ReqExecute_{ij}, ID_i, \tau_k, HL)$  from  $a_i$ ,  $t_s$  is the start time for  $a_j$  to execute  $\tau_k$  and  $t_s$  has to meet the condition that  $t_s \leq t_{ls}$ , where  $t_{ls}$  is the deadline of the task.

We assume that a new task, say  $\tau_k$ , is generated by agent  $a_i$  now and  $a_j$  is one of  $a_i$ 's neighbors.  $a_i$  sends a request message  $(ReqExecute_{ij}, ID_i, \tau_k, HL)$  to  $a_j$ . After receiving the request message,  $a_j$  will check whether its own resource set  $R_j$  can meet the resource requirement of  $\tau_k$  or not. If  $R_j \geq R_k$  ( $R_k$  is the requested resource set by  $\tau_k$ , see Definition 1),  $a_j$  will send a reply message  $(RepExecute_{ji}, ID_j, t_s, \tau_k)$  back to  $a_i$ . Otherwise,  $a_j$  will check whether  $HL = 0$ . If  $HL = 0$ ,  $a_j$  will give up this request message, while if  $HL \geq 1$ ,  $HL$  will be subtracted by 1 and then the request message,  $(ReqExecute_{im}, ID_i, \tau_k, HL - 1)$ , will be transmitted by  $a_j$  to all of  $a_j$ 's neighbors, and  $m$  is the  $ID$  number of the message's destination agent. Each time the request message is transmitted,  $HL$  will be subtracted by 1. The transmitting process will be terminated once the value of  $HL$  becomes 0 or the receiver's resource set can meet the resources requested by  $\tau_k$ .

If  $a_i$  receives a reply message ( $RepExecute_{ji}, ID_j, t_s, \tau_k$ ) for executing task from  $a_j$ ,  $a_i$  will begin to negotiate with  $a_j$ . The negotiation method will be specifically described in Sect. 3. When the negotiation succeeds, it is possible that both the parties will sign a contract (we will discuss this in Sect. 3.1), which is an agreement between  $a_i$  and  $a_j$  for executing  $\tau_k$ .

**Definition 8** A contract between  $a_i$  and  $a_j$  is 6-tuple  $con_{ij}(\tau_k, pr_{con}, t_s, t_{con}, pro_i, pro_j)$ , where  $pr_{con}$  is the price that  $a_i$  should pay to  $a_j$ ,  $t_s$  is the start time for  $a_j$  to execute  $\tau_k$ ,  $t_{con}$  is the time that the contract is signed, and  $pro_i, pro_j$  are the profits  $a_i$  and  $a_j$  gain from this contract, respectively.

In our method, any agent can negotiate with more than one opponents because in dynamic and open environment, it is hard for an agent to make a decision to get an optimal price. Hence, decommitment is necessary but should only be allowed before the task has started to be executed. The agent that initially decommits from the contract has to pay a penalty to the other party of the contract. Furthermore, the later the decommitment, the higher the penalty will be because of the pressure of the deadline of the task.

**Definition 9** The penalty that agent  $a_i$  pays to  $a_j$  if  $a_i$  decommits from the contract  $con_{ij}(\tau_k, pr_{con}, t_s, t_{con}, pro_i, pro_j)$  is calculated by:

$$pel_{ij} = \frac{t - t_{con}}{t_s - t_{con}} pro_j \quad (1)$$

where  $t$  is the time of the decommitment, and  $t_{con} \leq t \leq t_s$ .

Because the role of an agent can be either a consumer or a provider or even both, the profit of an agent is defined as follows:

**Definition 10** The profit agent  $a_i$  gains totally is:

$$pro_i = \sum_{i=1}^m (r_i - p_i) + \sum_{l=1}^n (p_l - c_l(R)) + \sum_{j=1}^k pel_j \quad (2)$$

where  $m$  is the number of successfully allocated tasks of  $a_i$ ,  $r_i$  is the reward  $a_i$  gains from task  $\tau_i$  and  $p_i$  is the price  $a_i$  pays to the executor of  $\tau_i$ ;  $n$  is the number of tasks that are successfully allocated to agent  $a_i$ ,  $p_l$  and  $c_l(R)$  are the price that the owner of  $\tau_l$  pays to  $a_i$  for executing  $\tau_l$  and the cost that  $a_i$  spends to execute  $\tau_l$ ; and  $k$  is the number of agents that pay penalties to  $a_i$  and  $a_i$  pays penalties to. When  $a_i$  pays penalty to  $a_j$ ,  $pel_j$  is a negative number; while if  $a_j$  pays penalty to  $a_i$ ,  $pel_j$  is a positive number.

Based on the definitions given above, the procedure of task allocation of our method include three main steps.

**Step 1:** Building up neighborhoods of newly arrived agents.

When an agent, say  $a_i$ , enters into the grid environment newly, it selects several agents from the grid randomly and sends request messages for building up neighborhood (see Definition 3) to the selected agents. After receiving the request message for

building up neighborhood, the message receiver, say  $a_j$ , sends back a reply message for building up neighborhood (see Definition 4) to  $a_i$ . Then  $a_i$  and  $a_j$  become neighbors and keep sending heart beat messages (see Definition 5) to each other, until one or both of them leave off the grid environment.

**Step 2:** Finding potential resources for consumers.

When a new task, say  $\tau_i$ , is generated, the owner agent of  $\tau_i$ , say  $a_i$ , sends request messages for executing task (see Definition 6) to all of its neighbors. We assume that  $a_j$  is one of the agents that receive the message. After receiving the request message,  $a_j$  checks whether its own resources can meet the resource requirement of  $\tau_i$ . If so,  $a_j$  sends back a reply message (see Definition 7) to  $a_i$ . Otherwise,  $a_j$  transmits the request message to all of  $a_j$ 's neighbors.

**Step 3:** Negotiation.

After  $a_i$  receives a reply message for executing task  $\tau_i$  from  $a_j$ ,  $a_i$  and  $a_j$  begin to negotiate with each other. The negotiation method will be specifically described and formulated in Sect. 3.

### 3 Negotiation Method

This section presents our negotiation method for task allocation specifically. The proposed negotiation-based task allocation method using local information (NTAL for short) extends the alternating offers protocols in [2, 19].

#### 3.1 Offer Generation and Sign Contract

When a consumer  $c$  calculates its offer price  $op_c(t, t_s)$  at time  $t$ , based on the start time  $t_s$  provided by the provider, the following factors will be taken into consideration.

- (i) The start time  $t_s$ . With the pressure of the start time, that is, with the eagerness of the consumer getting heavier and heavier, the consumer will give more concessions. Hence, the nearer  $t_s$  is, the higher  $op_c(t, t_s)$  is. Especially when  $t = t_g$ ,  $op_c(t, t_s)$  should be the lowest and its value should be  $c(R)$ . However, this is an ideal situation that hardly happens because  $t_g$  is the generation time of the task.
- (ii) The number of reply messages for executing task the consumer has received, i.e.,  $rep_{tc}$ . Taking the number of  $rep_{tc}$  into consideration is in order to consider the specific situations. The fewer reply messages for executing task ( $RepExecute$ ) the consumer has totally received, the higher  $op_c(t, t_s)$  will be. If a resource consumer  $a_i$  calculates its offer price to the provider  $a_j$ , that represents that  $a_i$  has received at least one reply message  $RepExecute$ , i.e., the  $RepExecute$  sent from  $a_j$ , thus, the minimum value of  $rep_{tc}$  is 1, once  $op_c(t, t_s)$  is calculated. When the value of  $rep_{tc}$  is 1 and the time  $t = t_{st}$ , the offer price  $op_c(t, t_s)$  is

the highest and its value is the reserve price of the provider  $rp_c(t_s)$ . In order to meet this situation, we use  $lg_2$  in the following Eq. (3).

- (iii) Reward that the consumer gains:  $r(t_s)$  is the reward that the consumer will gain if its task can start to be executed at time  $t_s$ .
- (iv) The according reserve price based on the start time  $t_s$ .  $rp_c(t_s)$ , i.e., the reserve price of the consumer during the negotiation and  $rp_c(t_s) = r(t_s)$ . The offer price  $op_c(t, t_s)$  is closely related with  $rp_c(t_s)$ , the higher the  $rp_c(t_s)$ , the higher  $op_c(t, t_s)$ , and the maximum value of  $op_c(t, t_s)$  is  $rp_c(t_s)$ .

According to the above justification,  $op_c(t, t_s)$  can be calculated as follows:

$$op_c(t, t_s) = c(R) + (rp_c(t_s) - c(R)) \frac{t - t_g}{(t_s - t_g)lg_2(rep_{tc} + 1)} \quad (3)$$

where  $t_g \leq t \leq t_s$ .

When a provider calculates the counter offer price  $op_p(t, t_s)$  at time  $t$ , the provider will consider the following factors:

- (i) The start time  $t_s$ . With the pressure of the start time  $t_s$ , the provider will also give more concessions. Thus, the nearer  $t_s$  is, the lower  $op_p(t, t_s)$  is.
- (ii) The number of request messages for executing task (*ReqExecute*) that the provider has received. The more request messages (*ReqExecute*) the provider has received, the higher  $op_p(t, t_s)$  is. If a resource provider  $a_j$  calculates its counter offer price to a consumer  $a_i$ , that represents that  $a_j$  has received at least one request message *ReqExecute*, i.e., the *ReqExecute* sent from  $a_i$ , thus, the minimum value of  $rep_{tp}$  is 1, once  $op_p(t, t_s)$  is calculated. When the value of  $req_{tp}$  is 1 and the time  $t = t_g$  ( $t = t_g$  is an ideal situation that hardly happens), the counter offer price  $op_p(t, t_s)$  is the highest. In order to meet this situation, we use  $lg_2$  in the following Eq. (4).
- (iii) The cost of the provider's resource to execute the consumer's task. The higher the cost of resources  $c(R)$  is, the higher  $op_p(t, t_s)$  is.

Accordingly,  $op_p(t, t_s)$  is formulated as follows:

$$op_p(t, t_s) = c(R) \left( 1 + \frac{t_s - t}{t_s - t_g} lg_2(req_{tp} + 1) \right) \quad (4)$$

where  $t_g \leq t \leq t_s$ .

From Eq. (4), we can see that when  $t = t_g$ ,  $op_p(t, t_s)$  is the highest and the value is  $c(R)(1 + lg_2(req_{tp} + 1))$ . However, this is an ideal situation that hardly happens because  $t_g$  is the generation time of the task. When  $t = t_s$ , the value of  $op_p(t, t_s)$  is the lowest and  $op_p(t, t_s) = c(R)$ .

The negotiation will terminate when at least one of the following situations occurs:

- (i) the task of agent  $a_i$  starts to be executed by another provider;
- (ii) the negotiation opponent terminates the negotiation;
- (iii) the negotiation succeeds; and
- (iv) the deadline of the task arrives.

We assume that  $a_i$  and  $a_j$  are the both parties of a negotiation, and the current total profits of  $a_i$  and  $a_j$  are  $pro_i$  and  $pro_j$ , respectively. When the negotiation succeeds, the total profits of  $a_i$  and  $a_j$  will become  $pro'_i$  and  $pro'_j$ , respectively, if  $a_i$  and  $a_j$  will sign a contract. The condition that  $a_i$  and  $a_j$  will sign a contract is:  $pro'_i > pro_i$  and  $pro'_j > pro_j$ .

The negotiation strategies of consumers and providers are detailed in Algorithms 1 and 2, respectively. In the both algorithms, we assume that  $a_i$  is the consumer and  $a_j$  is the provider. Let  $pro_i$  and  $pro_j$  are the current total profits of  $a_i$  and  $a_j$ , respectively. When the negotiation succeeds, if  $a_i$  and  $a_j$  can sign a contract, the total profits of  $a_i$  and  $a_j$  are  $pro'_i$  and  $pro'_j$ , respectively. Hence, if  $a_i$  and  $a_j$  will sign a contract,  $pro'_i - pro_i$  is the profit that  $a_i$  will gain from this contract, and  $pro'_j - pro_j$  is the profit that  $a_j$  will gain from this contract.  $t$  is the synchronization time.

---

**Algorithm 1:** Consumer's strategy
 

---

```

After receiving a reply message (RepExecuteji,
IDj,  $t_s$ ,  $\tau_k$ ) from  $a_j$ ,  $a_i$  calculates  $op_i(t, t_s)$  by
Eq. (3) and sends  $op_i(t, t_s)$  to  $a_j$ ;
1 While  $t < t_s$  and the task has not started to be
2   executed by any other agent (because any agent
3   is allowed to negotiate with more than one
4   opponents, it is possible that  $a_i$  has signed a
5   contract for  $\tau_k$  with another provider, and there
6   is a start time in that contract) do
7   if  $a_i$  receives  $op_j(t, t_s)$  from  $a_j$  then
8      $a_i$  calculates  $op_i(t, t_s)$  by Eq. (3);
9     if  $op_j(t, t_s) \leq op_i(t, t_s)$  then
10       negotiation succeeds; break;
11     else
12        $a_i$  calculates  $op_i(t, t_s)$  based on the
13       synchronization time again by
14       Eq. (3) and sends  $op_i(t, t_s)$  to  $a_j$ ;
15     end
16 end while
17 if negotiation succeeds then
18    $a_i$  calculates  $pro'_i$  by Eq. (2);
19   if  $pro'_i > pro_i$  and  $a_j$  is also willing to
20   sign a contract then
21      $a_i$  and  $a_j$  sign a contract
22      $con_{ij}(\tau_k, pr_{con}, t_s, t_{con}, pro'_i - pro_i, pro'_j - pro_j)$ 
23 end if

```

---

In Algorithm 1, when  $t < t_s$ , and the task does not start to be executed by any other provider (lines 1–6), if the consumer  $a_i$  receives the counter offer,  $op_j(t, t_s)$ , from  $a_j$  (line 7), it calculates the maximum value of  $op_i(t, t_s)$  that  $a_i$  can accept at that moment (line 8). If  $a_i$  will accept the counter offer price of  $a_j$  (line 9), the negotiation succeeds (line 10), then both  $a_i$  and  $a_j$  calculate their new profits if they will sign a contract, i.e.,  $pro'_i$  and  $pro'_j$  respectively (line 18). If  $pro'_i - pro_i > 0$  and  $pro'_j - pro_j > 0$ ,  $a_i$  and  $a_j$  will sign a contract (lines 19–22). Otherwise,  $a_i$  will



calculate the new offer price  $op_i(t, t_s)$  in the new round of the negotiation, and sends the offer price to  $a_j$  (lines 12–14).

---

**Algorithm 2:** Provider's strategy
 

---

```

1 While  $t < t_s$  and the task has not started to be
2   executed by any other agent do
3   if  $a_j$  receives  $op_i(t, t_s)$  from  $a_i$ , then
4    $a_j$  calculates  $op_j(t, t_s)$  by Eq. (4);
5   if  $op_i(t, t_s) \geq op_j(t, t_s)$  then
6   | negotiation succeeds; break;
7   else
8   |  $a_j$  calculates  $op_j(t, t_s)$  based on the
9   | synchronization time again by
10  | Eq. (4) and sends  $op_j(t, t_s)$  to  $a_i$ ,
11  | after receiving the new offer price from  $a_i$ ;
12  end
13 end while
14 if negotiation succeeds then
15 |  $a_i$  calculates  $pro'_i$  by Eq. (2);
16 | if  $pro'_i > pro_j$  and  $a_i$  is also willing to sign
17 | a contract then
18 |  $a_i$  and  $a_j$  sign a contract
19 |  $con_{ij}(\tau_k, pr_{con}, t_s, t_{con}, pro'_i - pro_i, pro'_j - pro_j)$ ;
20 end if

```

---

In Algorithm 2, when  $t < t_s$ , and the task does not start to be executed by any other provider (lines 1–6), if the provider  $a_j$  receives the offer price,  $op_i(t, t_s)$ , from  $a_i$  (line 3), it calculates the minimum price  $op_j(t, t_s)$  that it can accept at that moment (line 4). If it will accept the offer price of  $a_i$  (line 5), the negotiation succeeds (line 6), then both  $a_i$  and  $a_j$  calculate their new profits if they will sign a contract, i.e.,  $pro'_i$  and  $pro'_j$ , respectively (line 15). If  $pro'_i - pro_i > 0$  and  $pro'_j - pro_j > 0$ ,  $a_i$  and  $a_j$  will sign a contract (lines 16–19). Otherwise,  $a_j$  will calculate the new minimum counter offer price  $op_j(t, t_s)$  after receiving the offer price from  $a_i$  in the next round of the negotiation, and will send the new counter offer price to  $a_i$  (lines 8–11).

## 4 Experiment

In this section, we evaluate the performance of our method.

An et al. proposed a Negotiation-based Task Allocation method (NTA for short) in [2] for task allocation. The NTA assumes that each consumer knows all the providers' information, i.e., each consumer has a global view. Besides, only consumers are allowed to enter into or leave off the environments freely, but providers are not. A provider and a consumer negotiate with each other under the pressure of the deadline of a task.

In addition to the NTA, we also compare our method with the Distributed Greedy Task Allocation (DGA) method [7]. In DGA, the tasks are distributed, nodes are

connected as a social network (i.e., each node has only a local view and can only communicate with its limited neighbors), and only consumers can enter into or leave off the environments freely. This method assumes that each resource provider can accept the coming tasks of its neighbors as long as it can meet the resource requirements of the tasks, but can select only one task one time according to the ratio values of the tasks (the ratio value is the value of the task's reward divided by the resource quantity requested by the task). A task is considered to be successfully allocated only when it is selected by the provider. The purpose of comparing our method with DGA is to evaluate the advantage of negotiation when allocating tasks.

One of the main purpose of the task allocation methods in grid systems is to successfully allocate as many tasks as possible [10]. Thus, we experimentally study the success rate of task allocation. Besides the success rate, the total profit, i.e., the sum of all agents' profits, is also one of the main performance measures [2]. Hence, we also report the total profit of all consumers and providers involved in the grid environment of our method against two other methods developed in [2, 7]. The evaluation metrics include:

- (i) The success rate which is calculated by:

$$r_{suc} = \frac{N_{suc}}{N_{\tau}} \quad (5)$$

where  $N_{suc}$  is the number of tasks that are successfully allocated and  $N_{\tau}$  is the number of all tasks involved in the environment.

- (ii) The total profit, which is the sum of the profits of all the agents involved in the grid environment. The total profit is calculated by:

$$P_{tot} = \sum_{i=1}^n pro_i \quad (6)$$

where  $n$  is the total number of agents involved in the grid environment, and  $pro_i$  is the profit gained by agent  $a_i$ . Because the total profit of a method could be significantly different in different settings, we will experimentally study the ratios of total profit between NTAL and DGA, and the ratios of total profit between NTAL and NTA.

## 4.1 Experimental Settings

In order to do a comparison experiment with NTA [2], the parameters used in the experiment are inspired by [2] and listed in Table 1. The parameters setting in the experiments is similar to those in [2] (see Table 1). However, we do not set a crisp value to each parameter, rather we set a range and thus each parameter actually takes different values randomly in the range from time to time during the course of the experiment to reflect the nature of dynamics of grid systems.

**Table 1** Parameters setting

Variables	Meanings	Values
$N_{ra}$	Number of resource types per agent	[0, 10]
$c(R)$	Cost to complete a task	[100, 150]
$\bar{r}$	Maximum reward of a task	[250, 500]
$\bar{r}/c(R)$	The ratio between $\bar{r}$ and cost of a task	[1.7, 5]

Each task needs 1–3 types of resources and each agent owns 0 to 10 types of resources. The cost to complete a task is in the range of [100, 150], and each task's maximum reward is in the range of [250, 500], so the ratio between  $\bar{r}$  and *cost* of a task, i.e.,  $\bar{r}/c(R)$ , is from 1.7 to 5, which is reasonable in practical markets. Besides, there are another two important variables used in the experiment, i.e.,  $\psi(r)$  and  $flex(\tau)$ .

(i)  $\psi(r)$  reflects the resource competition and is defined as follows:

$$\psi(r) = N_\tau / N_p \quad (7)$$

where  $N_t$  and  $N_p$  are the numbers of tasks and resource providers, respectively. Because both the numbers of tasks and providers are dynamically changing (i.e., both  $N_\tau$  and  $N_p$  vary from time to time during the course of the experiment), actually,  $\psi(r)$  varies during the course of the experiment as well. Moreover, we have to state that the values of resource competition  $\psi(r)$  in the following experiments are the ratios between the maximum numbers of the resource providers and the maximum numbers of consumers. Hence, in the following experiments, if the value of  $\psi(r)$  is given, it does not represent that  $\psi(r)$  is fixed, contrary, it varies from time to time.

(ii)  $flex(\tau)$  reflects the allocation flexibility of a task and is defined as follows:

$$flex(\tau_k) = t_{ls} - t_g \quad (8)$$

where  $t_{ls}$  and  $t_g$  are the deadline and the generation time of  $\tau_k$ , respectively.

At the beginning of the experiments, we generate 100 agents and 600 tasks. The 600 tasks are distributed to the 100 agents randomly. Each agent has at least one type of resource, and so, each agent can be a provider. Moreover, each agent is also a consumer if at least one task is distributed to it. These agents and tasks are not in the grid environment at the beginning but enter into or leave off the environment randomly after the beginning of the experiment. Each agent selects 5 other agents randomly as its neighbors when it newly arrives in the environment. The experiment is conducted according to the following two different scenarios.

**Scenario 1: examination of the impact of the deadlines of tasks.**

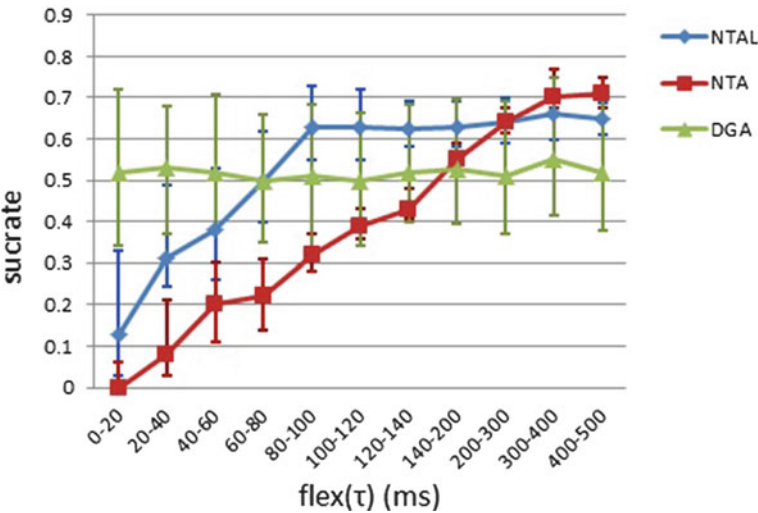
The purpose of this Scenario is to test the impact of the deadlines of tasks on our task allocation method. The parameters used in Scenario 1 are listed in Table 2.

**Table 2** Parameters setting for Scenario 1

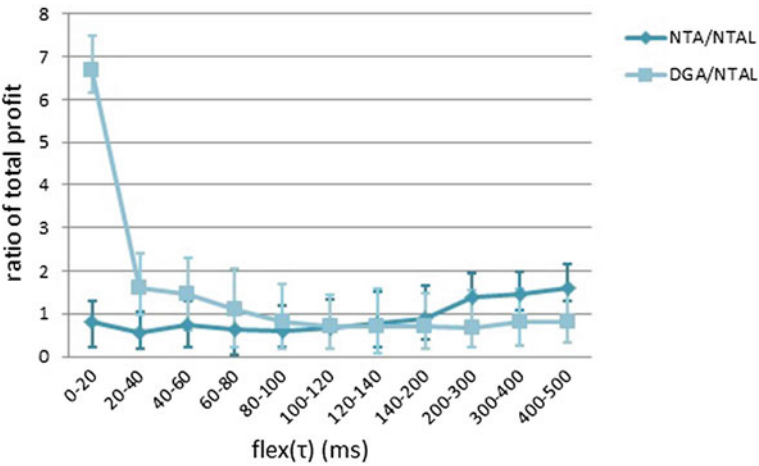
Variables	Meanings	Values
$N_{\tau}$	Number of tasks	[0, 100]
$N_p$	Number of resource providers	[0, 100]
$\psi(r)$	Resource competition	1
$N_{ave}$	The average required resource types	5

To control the maximum numbers of providers and tasks that enter into the grid environment, we set both the maximum providers and tasks as 100, that is, both the numbers of providers and consumers are in the range of [0, 100] during the course of the experiment. This does not mean that the numbers of providers and tasks of the experiment are randomly chosen from the range of [0, 100] at the start of the experiment, but rather it means that the numbers of providers and tasks change from time to time between 0 and 100 during the course of the experiment because providers and tasks can leave or enter at any time. Similarly, the number of tasks changes from time to time in the range of [0, 100]. We set the parameters like this because the grid environment is dynamic and open. Based on this, we change the deadlines (i.e., latest start times) of tasks and get different success rates and total profits accordingly, which are shown in Figs. 1 and 2, respectively.

In fact, even though the purpose of Scenario 1 is to test the impact of different deadlines of tasks on success rate and total profit, the resource competition is not fixed, it varies during the course the the experiment according to the parameter settings above. However, the resource competition varies in a small range and this



**Fig. 1** Deadline of task and the success rate



**Fig. 2** Deadline of task and the total profit ratio

cannot be avoided due to the dynamics and openness of the environment. We also conduct an experiment to test the impact of resource competition when it varies in bigger ranges in Scenario 2.

**Scenario 2: examination of the impact of resource competition.**

The aim of this Scenario is to test the impact of different resource competitions on our task allocation method. The parameters used in Scenario 2 are listed in Table 3.

The allocation flexibilities of tasks (i.e.,  $flex(\tau)$ ) are in the range of [400, 500]. We make the number of providers in-between [0, 100], and we change the maximum number of tasks involved in the environment from 20 to 600. Hence, we get different resource competitions from 0.2 to 6, according to Eq. (7). Based on different resource competitions, we obtained the corresponding success rates, which are shown in Fig. 3. Furthermore, we also obtained the ratios of total profits between NTAL and NTA, and the ratios of total profits between NTAL and DGA, which are shown in Fig. 4.

**Table 3** Parameters setting for Scenario 2

Variables	Meanings	Values
$t_{ls}$	Deadlines of tasks	[400, 500]
$t_g$	Generation times of tasks	0
$flex(\tau)$	Allocation flexibility	[400, 500]
$N_{ave}$	Average required resource types	5

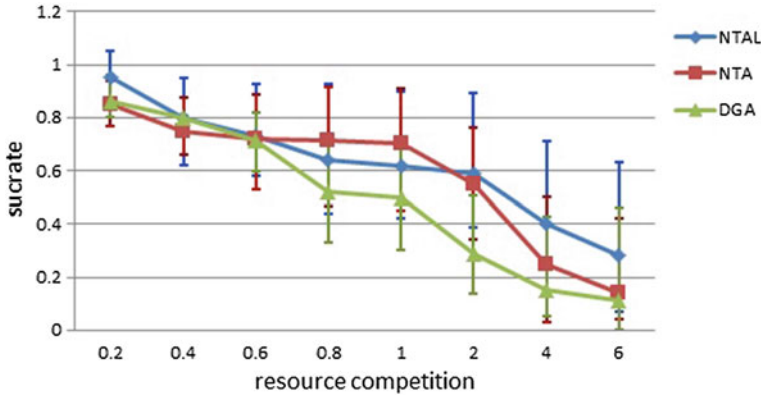


Fig. 3 Resource competition and success rate

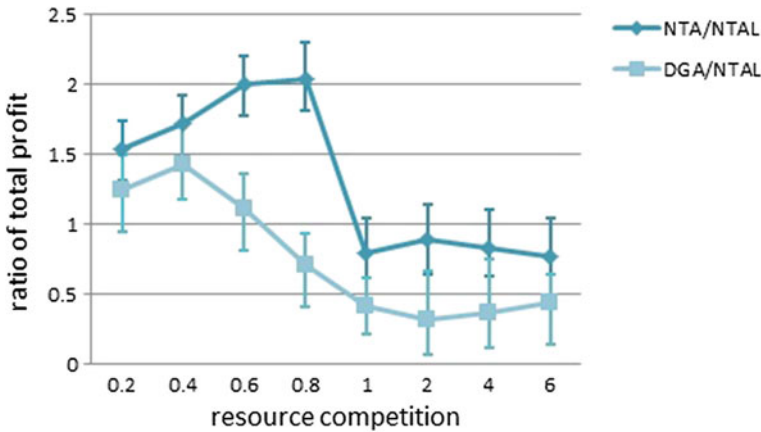


Fig. 4 Resource competition and total profit ratio

## 4.2 Results of Experiments

### Results of Scenario 1

In Scenario 1, in the case that the allocation flexibilities of tasks change during the course of the experiment, the corresponding success rates are shown in Fig. 1. Figure 2 illustrates the ratios of total profit between NTAL and NTA, and the ratios of total profit between NTAL and DGA, based on different allocation flexibilities.

From Fig. 1, we can see that the success rate of NTAL reaches the peak value earlier than that of NTA. This result can be explained from the view point of negotiation strategies of the providers of both NTAL and NTA. In fact, in NTAL, the offer price of provider is closely related to time  $t$ . The provider gives more concessions as time progresses, which can be seen from Eq.(4) in Sect. 3.1. However, in NTA,

only the consumer gives concessions while the provider gives its counter offer price just based on the resource competition. Hence, NTAL reaches the peak success rate quicker than NTA. For the same reason, when the allocation flexibilities are lower than 200 ms (milliseconds), the success rates of NTAL are higher than that of NTA. This demonstrates that NTAL works better in the circumstances that tasks are urgent, i.e., tasks need to be started in short times (low allocation flexibilities). Even though the peak value of success rate of NTAL is lower than that of NTA, it is reasonable because in NTA, any consumer knows all the providers' information, while the consumer only has a local view in NTAL. The peak value of success rate is limited by the local view in NTAL comparing with the global view in NTA.

As we can see from Fig. 1, the success rates of DGA are stable in the different ranges of allocation flexibility. Because there is no negotiation process in DGA and only greedy algorithm is used. Hence, the success rate is less related to the allocation flexibility compared with NTAL and NTA. However, when allocation flexibilities are higher than 140 ms, the success rates of DGA are lower than the peak values of both NTAL and NTA, and this result demonstrates the advantage of our negotiation-based method in task allocation.

In Fig. 2, when the allocation flexibilities are lower than 100 ms, the ratios of total profit between NTAL and NTA are higher than 1. This is reasonable because the corresponding success rates of NTAL are higher than the results of NTA when allocation flexibilities are lower than 100 ms. The ratios of total profit between NTAL and DGA become higher than 1 when the allocation flexibilities are higher than 100 ms, which is reasonable because NTAL needs time to negotiate.

### **Results of Scenario 2**

In Scenario 2, Fig. 3 displays the success rates based on different resource competitions. The ratios of total profit between NTAL and NTA, and the ratios of total profit between NTAL and DGA are shown in Fig. 4.

Figure 3 presents that when the resource competition is either lower than 0.6 or higher than 2, the success rates of NTAL are higher than those of NTA. This is because when the resource competition is lower than 0.6, the limitation of the local view of the consumer in NTAL is not apparent due to the low resource competition. Contrarily, when the resource competition is higher than 2, the advantage of the global view in NTA is not apparent compared with the local view due to the high resource competition. Hence, when the resource competition is either lower than 0.6 or higher than 2, the p2p neighborhood used in NTAL can work as well as the global information in NTA, or even better.

As it can be seen from Fig. 3, when the values of resource competition are higher than 0.6, the success rates of DGA are lower than those of NTAL and NTA. This also demonstrates the advantage of our negotiation-based method for task allocation.

In Fig. 4, it can be seen that the ratios of total profit between NTAL and NTA, and the ratios of total profit between NTAL and DGA vary with the different values of resource competition. When resource competitions are higher than 1, the total profit ratios between NTAL and NTA are higher than 1, which means the total profit of NTAL is higher than that of NTA even though NTA has global views while NTAL uses only the local views. This can be explained from two view points. First, whether

an agent has global view or not does not affect the total profit much because of the high resource competitions. Second, the corresponding success rates of NTAL are higher than those of NTA. Hence, in the situations of high resource competitions, NTAL still works better than NTA.

## 5 Related Work

In recent years, some task allocation methods have been developed by the consideration of various uncertain factors during task allocation. In 2008, Gatti et al. proposed a task allocation method by considering the different deadlines of tasks during the negotiation in market-based grid environments [12]. In 2009, An et al. developed a negotiation-based method for task allocation by taking two uncertain factors (i.e., deadline and reserve price) into consideration during negotiation [1]. However, in practice, negotiation for task allocation in most open environments needs to consider more than two uncertain factors. In 2010, An et al. further extended their previous work by considering more uncertainty factors [2], such as resource competition, deadline, reserve price and cost under the assumption of a global view of each resource consumer (i.e., each consumer has complete information of all providers).

This paper is mostly related to the method proposed by An et al. [2]. However, their method assumes that resource providers are not allowed to enter into or leave off the grid environment freely, and it also assumes that each consumer knows all the providers' information (i.e., each consumer has a global view). The two assumptions do not always hold, especially applying them to an open grid environment, due to no central controllers in open, dynamic grid environments. Besides, if each consumer knows all the providers' information, it means that there is a central controller, which can easily introduce the single point of failure problem. Moreover, the central controller will limit the scalability of grid systems. Our NTAL does not assume the global view of any agent. Additionally, in NTAL, both consumers and providers can enter into or leave off grid environments freely at any time. Hence, the environment in this paper is completely dynamic, open for consumers and providers.

In 2009, Klos et al. proposed an Agent-based Computational Economics model (ACE for short) [15], which is also for market-based task allocation in dynamic environments. ACE focuses on the adaptation of agents, due to the dynamics of the environments and the competitions among agents. Agents adapt themselves based on the trusts on their partners. However, our method introduces the concepts of negotiation and penalty. The negotiation can make the transactions between agents more flexible, and meanwhile the penalty prevents the distrust between agents.

Weerdt et al. proposed a Distributed Greedy Method (DGA) to allocate distributed tasks to resources in [7]. There are some aspects at which DGA is similar to our NTAL. For example, there is no central controller in DGA, which means each agent only has a local view; agents are connected as a social network which is similar to the p2p neighborhood used in our method; and consumers are dynamic. However, in DGA there is no negotiation and only greedy method is introduced. Each provider



selects one task one time only according to the ratio of the reward and the quantity of the requested resources of the task. In fact, many more factors need to be considered when allocating tasks in modern applications, not just the reward of the task and the quantity of requested resources. Our negotiation-based method NTAL is flexible by taking into more factors into consideration when allocating tasks, such as the deadline, reserve price, cost, the resource competition, and so on.

The task allocation method proposed in [20] is also decentralized in dynamic environments. However, that method focuses on the cooperation among provider agents that have to finish interdependent tasks cooperatively, because the tasks may be interdependent due to the time constraints, resource constraints or order constraints. Thus it requires high coordinations among agents in order to obtain high total profits. While our method focuses on the competitions among both provider agents and consumer agents, trying to maximize their own profits obtained from the successfully allocated tasks.

## 6 Conclusion

This paper proposed a negotiation-based task allocation method to achieve high success rates, high total profits in task allocation in decentralized, dynamic, and open grid environments. The main contributions of the proposed method are: (i) the method is based only on local views, which can make our method more applicable in open, dynamic environments. Besides, local views do not limit the scalability of the application systems; and (ii) the proposed method allows both providers and consumers to freely enter into and leave off grid environments, so it can be applied to many real open, dynamic situations.

In the future, we intend to evaluate the success rates and total profits when both the allocation flexibilities and resource competitions change in NTAL. Besides, we also intend to work on solving continuous task allocation in decentralized, dynamic, and open grid environments and to test our task allocation method in real life situations. In addition, it is also interesting to do game theoretic analysis on our method.

## References

1. An, B., Gatti, N., Lesser, V.: Bilateral bargaining with one-sided two-type uncertainty. In: *Proceedings of the International Joint Conference on Web Intelligence and Intelligent Agent Technology*, vol. 2, pp. 403–410 (2009)
2. An, B., Lesser, V., Irwin, D., Zink, M.: Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In: *Proceedings of AAMAS*, pp. 981–988 (2010)
3. AuYoung, A., Chun, B., Snoeren, A., Vahdat, A.: Resource allocation in federated distributed computing infrastructures. In: *Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-demand IT InfraStructure*, vol. 9 (2004)
4. Buyya, R., Abramson, D., Venugopal, S.: The grid economy. *Proc. IEEE* **93**(3), 698–714 (2005)

5. Cramton, P., Shoham, Y., Steinberg, R.: *Combinatorial Auctions*. MIT Press, Cambridge (2006)
6. Dash, R.K., Vytelingum, P., Rogers, A., David, E., Jennings, N.R.: Market-based task allocation mechanisms for limited-capacity suppliers. *IEEE Trans. Syst., Man Cybern., Part A: Syst. Hum.* **37**(3), 391–405 (2007)
7. de Weerd, M., Zhang, Y., Klos, T.: Distributed task allocation in social networks. In: *Proceedings of AAMAS*, p. 76 (2007)
8. Dias, M.B., Zlot, R., Kalra, N., Stentz, A.: Market-based multirobot coordination: a survey and analysis. *Proc. IEEE* **94**(7), 1257–1270 (2006)
9. Dooley, K.: *Designing Large-Scale LANs*. O'Reilly Media Inc., Sebastopol (2001)
10. Fatima, S.S., Wooldridge, M.: Adaptive task and resource allocation in multi-agent systems. In: *Proceedings of the 5th International Conference on Autonomous Agents*, pp. 537–544 (2001)
11. Fu, Y., Chase, J., Chun, B., Schwab, S., Vahdat, A.: Sharp: an architecture for secure resource peering. In: *ACM SIGOPS Operating Systems Review*, vol. 37, pp. 133–148. ACM (2003)
12. Gatti, N., Giunta, D., Marino, S.: Alternating-offers bargaining with one-sided uncertain deadlines: an efficient algorithm. *Artif. Intell.* **172**(8), 1119–1157 (2008)
13. Irwin, D., Chase, J., Grit, L., Yumerefendi, A., Becker, D., Yocum, K.G.: Sharing networked resources with brokered leases. In: *Proceedings of the USENIX Technical Conference*, pp. 199–212 (2006)
14. Jennings, N.R., Faratin, P., Lomuscio, A.R., Parsons, S., Wooldridge, M.J., Sierra, C.: Automated negotiation: prospects, methods and challenges. *Group Decis. Negot.* **10**(2), 199–215 (2001)
15. Klos, T., Nooteboom, B.: Adaptive learning in evolving task allocation networks. In: *Proceedings of AAMAS*, pp. 465–472 (2009)
16. Lai, K., Rasmusson, L., Adar, E., Zhang, L., Huberman, B.A.: Tycoon: an implementation of a distributed, market-based resource allocation system. *Multiagent Grid Syst.* **1**(3), 169–182 (2005)
17. Macarthur, K.S., Stranders, R., Ramchurn, S.D., Jennings, N.R.: A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In: *Proceedings of AAAI*, pp. 356–362 (2011)
18. Regev, O., Nisan, N.: The popcorn market. *Online markets for computational resources. Decis. Support Syst.* **28**(1), 177–189 (2000)
19. Rubinstein, A.: Perfect equilibrium in a bargaining model. *Econom.: J. Econom. Soc.* **50**(1), 97–109 (1982)
20. Theocharopoulou, C., Partsakoulakis, I., Vouros, G.A., Stergiou, K.: Overlay networks for task allocation and coordination in dynamic large-scale networks of cooperative agents. In: *Proceedings of AAMAS*, p. 55 (2007)
21. Zheng, X., Koenig, S.: Reaction functions for task allocation to cooperative agents. In: *Proceedings of AAMAS*, pp. 559–566 (2008)

Next Frontier in Agent-based Complex Automated  
Negotiation

Fujita, K.; Ito, T.; Zhang, M.; Robu, V. (Eds.)

2015, X, 151 p. 69 illus., 45 illus. in color., Hardcover

ISBN: 978-4-431-55524-7