A Distributed Algorithm for Parallel Multi-task Allocation Based on Profit Sharing Learning

SU Zhao-Pin\textsuperscript{1,2} JIANG Jian-Guo\textsuperscript{3} LIANG Chang-Yong\textsuperscript{2} ZHANG Guo-Fu\textsuperscript{1,3}

Abstract Task allocation via coalition formation is a fundamental research challenge in several application domains of multi-agent systems (MAS), such as resource allocation, disaster response management, and so on. It mainly deals with how to allocate many unresolved tasks to groups of agents in a distributed manner. In this paper, we propose a distributed parallel multi-task allocation algorithm among self-organizing and self-learning agents. To tackle the situation, we disperse agents and tasks geographically in two-dimensional cells, and then introduce profit sharing learning (PSL) for a single agent to search its tasks by continual self-learning. We also present strategies for communication and negotiation among agents to allocate real workload to every tasked agent. Finally, to evaluate the effectiveness of the proposed algorithm, we compare it with Shehory and Kraus’ distributed task allocation algorithm which were discussed by many researchers in recent years. Experimental results show that the proposed algorithm can quickly form a solving coalition for every task. Moreover, the proposed algorithm can specifically tell us the real workload of every tasked agent, and thus can provide a specific and significant reference for practical control tasks.

Key words Multi-agent systems (MAS), parallel multi-task allocation, coalition formation, profit sharing learning (PSL), negotiation


Several application domains, such as resource allocation\textsuperscript{[1–2]} and disaster response management\textsuperscript{[3–4]}, require teamwork. For example, in Fig. 1, when a disaster takes place in an area, many relief tasks need to be accomplished at once by rescue teams. Here a rescue team, which is composed of robots, persons, wrecking cars, rescue materials, can be viewed as a mobile agent. However, an agent with its insufficient resources may not complete a difficult task by itself, so it has to interact and cooperate with other agents by forming a coalition where each team is given a relief assignment.

![Fig. 1 An example of disaster response management](image)

Coalition formation is a fundamental and important form of interaction in the field of multi-agent systems (MAS). Coalitions can improve the performance of individual agents and accomplish tasks more efficiently. Thus, task allocation via forming effective coalitions is a major research challenge, and has received a considerable amount of attention. However, on the one hand, most of current researches can not tell us whether each member in its coalition has really taken on tasks or not.

On the other hand, we do not know how much workload each member should perform at least. Therefore, existing work cannot provide a specific and significant reference for practical tasks. Especially, in a number of practical scenarios, an agent may have to execute several different tasks simultaneously, so it is needed to distribute resources and capabilities among several different coalitions at the same time, and the system needs to know an agent’s real contribution to its several different tasks and judge whether there is any resource conflict or not.

Against this background, this paper is absorbed in parallel multi-task allocation via coalition formation in distributed computing environments. In such a situation, an agent may be a member of more than one coalition at the same time. Obviously, this property can improve the utilization of agents’ resources, and thus increase the efficiency of task execution. To achieve the goal, we mainly focus on how to make an agent reach an efficient and optimal outcome through its self-learning and advance the state of the art in the following ways:

1) We address the problem of parallel multi-task allocation via coalition formation in distributed computing environments, where task execution is parallel, given that an agent may join in several different coalitions at the same time.

2) We develop a novel distributed algorithm to make agents compete against each other for tasks, and give the real workload of every tasked agent in its corresponding coalitions without any resource conflict.

The remainder of this paper is organized as follows. Section 1 gives a brief description of task allocation based on coalition formation. In Section 2, we discuss the related work. In Section 3, we show the proposed algorithm, given that an agent may join in more than one coalition agilely and vigorously without any resource conflict, and in Section 4 we evaluate its performance by contrast experiments. Finally, Section 5 concludes the paper and points out some future work.

1 Problem description

We consider a grid with \(x \times y\) cells just as shown in Fig. 2, where a set of \(n\) capabilities-bounded mobile agents, \(A = \{A_1, A_2, \ldots, A_n\}\), has to cooperate to execute a finite set of stationary tasks, \(T = \{T_1, T_2, \ldots, T_m\}\).

In general, task, agent and coalition can be described as follows:\textsuperscript{[5]}:
1) Each task $T_k \in T$ has an $r$-dimensional capabilities required vector, $D_k = [d_1^k, d_2^k, \cdots, d_r^k]$, $d_j^k \geq 0$, $1 \leq k \leq m$, $1 \leq j \leq r$, $r \in \mathbb{N}$. Moreover, each task $T_k$ has a two-value function $Flag(T_k)$ with $1 \leq k \leq m$, $Flag(T_k) = 0$ in the initial stage.

$$Flag(T_k) = \begin{cases} 1, & T_k \text{ has been allocated} \\ 0, & \text{otherwise} \end{cases}$$

2) Each agent $A_i \in A$ has an original vector of real non-negative $r$-dimensional capabilities, $B_i = [b_1^i, b_2^i, \cdots, b_r^i]$, $b_j^i \geq 0$, $1 \leq i \leq n$, where each capability is a property of an agent that quantifies its ability to perform a specific action. $A_i$ has a vector of real workload for $\forall T_k \in T$, $W_{i,k} = [w_1^{i,k}, \cdots, w_r^{i,k}]$, $0 \leq w_j^{i,k} \leq b_j^i$, which is a real contribution of $A_i$ to executing $T_k$. Moreover, $A_i$ can see the location of other agents and tasks only when they are in its sight, and $A_i$’s “visibility range” is often restricted by boundaries, as shown in Fig. 3. In addition, in Fig. 4, $A_i$ has a “communication range”, which is also restricted by boundaries, and any agent in its communication range is called a neighbor of $A_i$.

![Fig. 2 Parallel multi-task allocation via coalition formation](image)

A coalition $C_k$ can perform its task $T_k$ only if the vector of capabilities necessary for its fulfillment $D_k$ satisfies

$$\forall 1 \leq j \leq r, \quad d_j^k \leq b_j^{C_k}$$

In addition, the value of coalition $C_k$ with responsibility for $T_k$ is assigned by a characteristic function

$$V(C_k) = \Phi(T_k) - \Theta(C_k) - \Pi(C_k)$$

where $\Phi(T_k)$ is the guerdon paid for finishing $T_k$ and usually is a given constant number; $\Theta(C_k)$ is the total cost of all members’ contribution, namely, $\Theta(C_k) = \sum_{A_i \in C_k} \sum_{j} w_j^{i,k}$; $\Pi(C_k)$ is the total cost of communication between members, for example, the communication cost between $A_{i1}$ and $A_{i2}$ is $\pi_{12}$, which is a given constant number, satisfying $\pi_{12} = 0$, $\pi_{11} = \pi_{22}$, if $C_k = \{A_{i1}, A_{i2}, A_{i3}\}$, $\Pi(C_k) = \pi_{12} + \pi_{13} + \pi_{23}$. $V(C_k) \geq 0$ is just the gain distributed among agents in $C_k$. and each member in $C_k$ be distributed a reward $R_k = \frac{W_{i,k} \cdot V(C_k)}{D_k}$.

This given, in the process of $A_i$’s moving, if $T_k$ is in $A_i$’s visibility range, $A_i$ checks whether it can perform $T_k$ by itself. If it can, it receives a reward $R$ (see (4)) for performing $T_k$. Otherwise, it starts to communicate and negotiate with other agents in its communication range to cooperate to perform $T_k$. If $A_i$ succeeds in negotiating with its neighbors, a coalition $C_k$ is formed, and every member in $C_k$ obtains reward $R$ according to their real workload, and set $Flag(T_k) = 1$. If $A_i$ fails in any negotiation, it continues moving until another unsolved task is found, or $\forall 1 \leq k \leq m$, $Flag(T_k) = 1$.

Thus, parallel multi-task allocation via coalition formation is just simultaneously forming $m$ coalitions, $C_1, C_2, \cdots, C_m$, for solving tasks $T_1, T_2, \cdots, T_m$ by agents’ moving under the condition

$$\forall 1 \leq j \leq r, \quad \sum_{A_i \in A} b_j^i \geq \sum_{T_k \in T} d_j^k$$

The objective is to maximize the income $V_{\text{MAS}}$ of the whole system

$$V_{\text{MAS}} = \sum_{k=1}^{m} V(C_k)$$

### 2 Related work

To date, task allocation via coalition formation has been successfully and widely used in e-business[6], combinatorial optimization problems[7], multi-agent cooperation[9], and resource allocation[1-2]. Much of the existing research has focused on disjoint coalitions, where it is usually assumed that an agent that has joined a coalition is no longer available to other coalitions at any time. In this context, many centralized solutions and distributed algorithms have been proposed to form feasible coalitions.

#### 2.1 Centralized algorithms

Centralized solutions[5, 9-10] are concentrated on finding the optimal coalitions in the whole set of possible coalition structures, which is computationally complex due to the size of the set which is exponential of the number of agents.

To reduce the complexity of algorithms, Sen et al.[11] adopted the genetic algorithm and used one-dimensional
integral encoding to identify the optimal coalition structure. In addition, Yang et al.\textsuperscript{12} improved Sen and Dutta’s algorithm and designed a two-dimensional binary chromosome encoding and corresponding crossover and mutation operators to search the coalition structure space.

However, all of them suffered from two important drawbacks. On the one hand, it is assumed that each agent can exactly take part in only one coalition, producing a big waste of capabilities and limiting the scope of their applications in real-world scenarios. In fact, an agent may be involved in executing more than one task, and distributing its resources between several (not necessarily disjoint) coalitions. Although Zhang et al.\textsuperscript{13} developed a discrete particle swarm optimization based algorithm to solve the overlapping coalition formation problem in complex virtual enterprises environments. But their algorithm was centralized, and did not take into consideration the resource constraints of the computational environment, such as communication bandwidth and limited computation time.

2.2 Distributed algorithms

Shehory et al.\textsuperscript{14} firstly considered that an agent might join several different coalitions at the same time in their seminal work on coalition formation for task allocation, and developed anytime distributed algorithms for leading agents to be a member of more than one coalition. The limitation of coalitional size cannot guarantee the algorithm to search all possible coalitions, and thus certain feasible solutions may be lost. Although this algorithm was presented over ten years ago, in recent years, many researchers, such as Vig\textsuperscript{8}, Mataric\textsuperscript{15}, Thomas\textsuperscript{16}, Rahwan\textsuperscript{17}, and so on, discussed and applied Shehory and Kraus’s work. Vig\textsuperscript{8} improved the algorithm and applied it to multi-robot coalition formation to autonomously form coalitions to complete assigned missions in the multi-robot domain. Similarly, Mataric et al.\textsuperscript{15} described an empirical study to seek general guidelines for task allocation strategies in multi-robot systems, and pointed out that there is no single strategy which may produce the best performance in all cases, and the best task allocation strategy may be influenced by a function of noise in systems. Thomas\textsuperscript{16} proposed a completely distributed architecture where robots dynamically allocate their tasks, and they were involved in an incremental task allocation algorithm based on the contract-net protocol by introducing a parameter called equity coefficient to equilibrate the workload between different robots and to control the triggering of the auction process. Rahwan et al.\textsuperscript{17} presented a novel decentralized algorithm for distributing coalition value calculation among agents in the process of coalition formation, and compared with Shehory and Kraus’ work to evaluate the effectiveness of their algorithm.

Dash\textsuperscript{18} designed a task allocation mechanism in environments where sellers had finite production capacities, and introduced a novel continuous double auction protocol based on decentralized mechanism, achieving a level of efficiency that was reasonably close to the optimal solution given by centralized mechanism. Sander\textsuperscript{19} presented a distributed algorithm for task allocation in environments where agents and tasks were geographically dispersed in a two-dimensional space, and described a method which can enable agents to determine individually how to move so that they can efficiently be assigned tasks. Viguria\textsuperscript{20} presented a decentralized market-based approach based on contract net protocol to solve the initial formation problem, and tried to determine which mobile sensor should go to each position of a desired formation to minimize the objective. But one mobile sensor can only be allocated to one task.

Generally speaking, however, all works cited above can only tell the system that a task may be allocated to a coalition, and moreover, it is not clear that whether every member in this coalition has really taken on given tasks, or how much workload every member should at least perform. Therefore, their algorithms cannot provide a specific and significant reference which is always very important for some practical control tasks, such as disaster response management, resource allocation, and so on.

To address these shortcomings, we do a thorough literature review of existing algorithms, and evaluate them theoretically and empirically. Based on our findings, we develop a fast and efficient algorithm for the problem proposed in Section 1. Like Shehory and Kraus’ algorithm\textsuperscript{14}, our algorithm mainly focuses on how to encourage an agent to join several different coalitions and execute its tasks without any resource conflict. Therefore, when it comes to evaluating performance, we also compare our algorithm with Shehory and Kraus’ work just as Vig\textsuperscript{8}, Mataric\textsuperscript{15}, Thomas\textsuperscript{16}, Rahwan\textsuperscript{17}, and so on.

3 Distributed algorithm for parallel multi-task allocation based on PSL

From Section 1, since randomicity and complexity of task allocation and sensory limitation of agents, agents cannot identify and sense all states in their moving. Moreover, the model of state transitional probabilities of each agent environment cannot be obtained either. These characteristics make the problem not agree with Markov decision process. Therefore, profit sharing learning is a suitable approach to solve the problem. Thus, we first introduce profit sharing learning, then illustrate how agents move towards tasks by learning, and how they form coalitions.

3.1 Profit sharing learning

Profit sharing learning (PSL) was proposed to utilize effective reinforcement rules to seek optimal solutions in uncertain and dynamic environments, which has been proved to be an excellent reinforcement learning approach in literature\textsuperscript{21–24}.

In PSL, the weight of each rule is reinforced according to its distance from the goal. In Fig. 5, at time $t$, an agent enters state $s_t$ and selects action $a_t$ from its action set, which contains all its available actions, and continues this cycle until it receives a reward $R$ at time $t_R$. At this point, the episode consists of the state-action pair (SAP) $\{(s_1, a_1), (s_2, a_2), \cdots, (s_t, a_t)\}$, and then each SAP $(s_t, a_t)$ is assigned some credit according to the credit assignment function $f(R, t)$, which denotes an assignment value for the SAP which is fired at time $t$. For example, the last SAP $(s_{t_R}, a_{t_R})$ is assigned credit $R$, the penultimate $(s_{t_R-1}, a_{t_R-1})$ is assigned credit $f(R, t_R - 1)$, and so on. The weight of each SAP in the episode is modified by

$$w(s_t, a_t) \leftarrow w(s_t, a_t) + f(R_{s_t}, t)$$  \hspace{1cm} (7)

where $R_{s_t}$ denotes the credit given at time $t$ after reaching the goal. $f(R_{s_t}, t)$ is commonly denoted as

$$f(R_{s_t}, t) = R_{s_t} \cdot e^{-\beta t}$$  \hspace{1cm} (8)

where $\beta \in (0, 1)$ is a discount rate, and moreover, Hasegawa
et al. [23] have pointed out that if \( f(R_{tn,i}, t) \) satisfies
\[
\forall t = 1, 2, \ldots, t_{rn}, \sum_{i=1}^{t-1} f(R_{tn,i}, i) < f(R_{tn}, t)
\]
then the agent may at least find an approximately optimal solution within finite loops.

3.3 The rules of forming a coalition

Assuming \( A_i \)'s initial state is \( s_i^0 = (x_i^0, y_i^0) \), the details that \( A_i \) moves towards tasks can be described as follows:

1) At the first iteration, each agent \( A_i \) randomly selects an action \( a_i^1 \) from \( \text{Action} = \{ \text{right, left, down, up} \} \) until it finds a task \( T_k \) at time \( t_0 \). \( A_i \) obtains a reward \( R_i \) after performing \( T_k \) according to (4), then it updates \( \omega(s_i^0, a_i^1) \) for every \( (s_i^0, a_i^1) \) in its episode according to (7).

2) At the \( I \)-th iteration, \( 1 \leq I \leq I_{\text{max}} \), each agent \( A_i \) selects an action \( a_i^1 \) from \( \text{Action} = \{ \text{right, left, down, up} \} \) until it finds a task \( T_k \) at time \( t_I \) according to
\[
a_i^1 = \arg \max_{a_i \in \text{Action}} \omega(s_i^0, a_i)
\]
\( A_i \) obtains a reward \( R_i \) after performing \( T_k \) according to (4), then it updates \( \omega(s_i^0, a_i) \) for every \( (s_i^0, a_i) \) in its episode according to (7).

3.4 The algorithm

Having given the details relevant to our works, we now describe our distributed algorithm for parallel multi-task allocation as follows:

Step 1. Place all agents and tasks randomly in a grid with \( x \times y \) cells just as shown in Fig. 1; set \( \omega(s, a) = 0 \) and \( I = 0 \).

Step 2. Do the followings:
   1) If \( I > I_{\text{max}} \), goto Step 3, otherwise, goto Step 2.
   2) Set \( t = 0 \), and \( 1 \leq k \leq m \), set Flag\( (T_k) = 0 \); clear all state-action pairs from agent \( A_i \)'s episode.
   3) Each agent \( A_i \) should perform the following:
      a) \( A_i \), which is at state \( s_i^0 \), selects an action \( a_i^1 \) according to (10), and stores the SAP \( (s_i^0, a_i^1) \) in its episode;
      b) \( A_i \) executes \( a_i^1 \) and moves to the next state \( s_i^1 \);
      c) If \( A_i \) cannot find any unsolved task, \( t = t + 1 \), goto Step a); otherwise, an unsolved \( T_k \) is found by \( A_i \);
      d) If \( A_i \) can perform \( T_k \) by itself, set \( t_k = t \), update the weight of all SAPs in \( A_i \)'s episode according to (7), and set Flag\( (T_k) = 1 \). Then \( T_k \) is allocated and \( A_i \) stops moving. Otherwise, \( A_i \) needs to cooperate with other agents.
   4) If \( \forall 1 \leq k \leq m \), then Flag\( (T_k) = 1 \), \( I = I + 1 \), and goto Step 1.

Step 3. Output the best solution and end the algorithm.

The idea of the algorithm is that if at a given state an agent has to choose among different actions, those which were heavily chosen by preceding iterations (that is, those with a high weight of state-action pairs) are chosen with higher probability. Furthermore, high weight of state-action pairs is synonymous with more rewards given by coalitions. This causes the quantity of weight of state-action pairs to grow faster than that of state-action pairs with fewer rewards, and therefore the probability with which an agent chooses a task to perform is quickly biased towards the task with more income. The final result is that very quickly every teamed agent chooses a task which can bring more rewards to it. Obviously, the process is thus characterized by a positive feedback based reinforcement learning mechanism, which ensures that a tested agent may at least find an approximately optimal solution within finite loops [23].
3.5 Computational complexity

The efficiency of our algorithm can be judged from computations as follows:

1) In Step 1, we need to place $n$ agents and $m$ tasks in a grid with $x \times y$ cells and initialize $\omega(s, a)$ for $x \times y \times 4$ times, so the complexity of Step 1 is $O(n \times x \times y \times 4)$.

2) In Step 2, first, we should do the iteration for $I_{\text{max}}$ times; second, considering the worst case, an agent may travel at most $x \times y$ cells to find a task, and moreover, in its current cell, it may check for $9$ times in its visibility range (see Fig. 2) to find a task. Similarly, it may communicate for $3$ times with at most $9$ neighbors to form a final coalition, and in every negotiation, an agent needs to calculate its workload for each dimension capability. So, the complexity of Step 2 is $O(I_{\text{max}} \times n \times x \times y \times (9 + 9 \times 3 \times y))$, while the complexity of the algorithm in [11] is $O(n^2 \times x \times y \times (9 + 9 \times 3 \times 7))$.

Therefore, the complexity of our algorithm for parallel multi-task allocation is $O(I_{\text{max}} \times n \times x \times y \times (9 + 9 \times 3 \times y))$ which is close to $O(n^2)$, while the complexity of the algorithm in [11] is $O(n^k \times m^k)$, where $k$ is the highest coalition size allowed.

4 Experimental results and discussion

In order to evaluate the performance of our algorithm, we compare our algorithm with Shheory and Kraus’[14] (henceforth called SK) for the reasons outlined in Section 2. The partial parameters are shown in Tables 1 and 2. The relative experimental details in our algorithm are shown in Figs. 2 ~ 4. The initial states of agents and tasks are shown in Fig. 6. We make 4 independent trails of the two algorithms.

Table 2 Demand capability vectors and reward of tasks

<table>
<thead>
<tr>
<th>$D_k$</th>
<th>$T_{\text{sk}}$</th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
<th>$T_4$</th>
<th>$T_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(T_{\text{sk}})$</td>
<td>50</td>
<td>50</td>
<td>40</td>
<td>55</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

We take agent $A_7$ for an example to show how an agent moves to tasks and negotiates with other agents.

1) In Fig. 7, when $A_7$ is in its initial state $s_0 = (1, 3)$, it observes the weight of SAP related with $\omega(s_0, \text{right}) = 8.6541$, $\omega(s_0, \text{left}) = 4.4281$, $\omega(s_0, \text{up}) = 0$, $\omega(s_0, \text{down}) = 0.3969$. Since $\omega(s_0, \text{right}) > \omega(s_0, \text{left}) > \omega(s_0, \text{down}) > \omega(s_0, \text{up})$, $A_7$ selects right as its action according to (10), executes right, and enters next state $s_1 = (1, 4)$. Since there is no unresolved task in its visibility range (see Fig. 3), it stores the SAP $(s_0, \text{right})$ in its episode. $A_7$ continues the circle until it finds task $T_2$ in state $s_0 = (2, 8)$. The episode of $A_7$ consists of $(1, 3), (1, 4), (1, 5), (1, 6), (2, 6), (2, 7), (2, 8)$.

2) $A_7$ finds that it can not perform $T_2$ by itself, so it has to cooperate with other agents. Suppose there are $A_6$ and $A_{11}$ in $A_7$’s communication range (see Fig. 4). $A_7$ negotiates with $A_6$ and $A_{11}$ to perform $T_2$.

![Fig. 7 State $s_0 = (2, 8)$ of $A_7$](image-url)

3) First, $A_7$ sends a proposal $(T_2, [0, 5])$ to $A_6$, and waits for $A_6$’s response. Then $A_6$ sends $(1, [0, 0])$ to $A_7$ to refuse the proposal according to (11). The negotiation between $A_7$ and $A_6$ is over.

4) Then, $A_7$ negotiates with $A_{11}$ and sends a proposal $(T_2, [0, 5])$ to $A_{11}$. Here, $A_{11}$ sends $(0, [0, 1])$ to accept the proposal according to (11). $A_7$ continues to negotiate with other agents in its communication range. Since there is no other agent and $A_7$ and $A_{11}$ can not perform $T_2$, $A_7$ sends a message $\text{null}$ to $A_{11}$, and continues moving towards another unresolved task.

5) Suppose $A_7$ finds task $T_4$ in its state $s_7 = (2, 9)$ as shown in Fig. 8. The episode of $A_7$ consists of $(1, 3), (1, 4), (1, 5), (1, 6), (2, 6), (2, 7), (2, 8)$.

6) Since $A_7$ can not perform $T_4$ by itself, it has to cooperate with $A_6$ and $A_{11}$ in its communication range.

7) $A_7$ sends a proposal $(T_4, [3, 2])$ to $A_6$, and waits for $A_6$’s response. Here, $A_6$ accepts the proposal and sends $(0, [3, 1])$ to $A_7$. Since $A_7$ can not perform $T_4$ with $A_6$,
it continues to negotiate with A11, and sends \( (T_4, [0, 1]) \) to A11. A11 sends its responsive message \((0, [0, 1])\). 8) After \( T \) receives \( A'_7 \)’s responsive message, it finds that it can perform \( T_4 \) with \( A_5 \) and \( A_{11} \). So, the coalition for \( T_4 \) is \( C_4 = \{A_7, A_5, A_{11}\} \), and the flag \( \text{Flag}(T_4) = 1 \).

9) The workload vectors of each member in \( C_4 \) are: \( W_{7,4} = [5, 2], W_{5,4} = [3, 1], W_{11,4} = [0, 1] \). 10) \( A_7 \) calculates the value of \( C_4 \): \( V(C_4) = \Phi(T_4) - \theta(C_4) - \Pi(C_4) = 55 - 12 - 2 \times 3 = 37 \). The member’s rewards are: \( R_7 = \frac{W_{7,4} \cdot \Phi(T_4)}{D_{7,4}} \cdot V(C_4) = (7/12) \times 57 = 21.57, R_5 = (4/12) \times 37 = 12.33, R_{11} = (1/12) \times 37 = 3.1 \).

11) \( A_7 \) updates the weight of SAPs in its episode according to 7) and 8) where \( \beta = 0.3 \).

Moreover, the SK algorithm has a limitation of coalitional size and does not consider all possible coalitions, it is possible that coalitions left in each agent’s coalition list can not perform \( T_3 \) with insufficient resources. Furthermore, SK algorithm can only tell the system that a task may be solved by a coalition, for example, it is easily known that \( \{A_8, A_{12}\} \) can perform \( T_2 \), but it is not clear that how much workload every member should perform at least for \( T_2 \).

As shown above, the whole process is a positive feedback which ensures every agent can obtain an optimal results though its learning. Fig. 9 shows the final states by agents’ learning for 200 times. In Fig. 9, each task is surrounded and performed by agents.

Table 3 shows performance comparison of our algorithm with SK’s work. As shown in the table, our algorithm can exactly form a coalition for every task, and allows an agent to join in several different coalitions without any resource conflict, such as \( A_7 \). In addition, our algorithm puts out more total income, and gives the real workload and rewards of each tasked agent.

In contrast, the SK algorithm forms coalitions for \( T_1, T_2, T_4, T_5 \), but \( T_3 \) except \( T_3 \). The reason is that \( T_3 \) has a big capability demand but owns little reward, but each agent in SK algorithm is inclined to join coalitions which have more reward. Therefore, all tasks except \( T_3 \) were allocated earlier but no any coalition is left in each agent’s coalition list, and there is no available coalition to be selected to perform \( T_3 \).

<table>
<thead>
<tr>
<th>Coalition</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
<th>( T_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_5, A_{12}</td>
<td>{A_4, A_7, A_8}</td>
<td>{A_2, A_{10}}</td>
<td>{A_1, A_7, A_{11}}</td>
<td>{A_4, A_{15}}</td>
<td></td>
</tr>
</tbody>
</table>

Income

<table>
<thead>
<tr>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
<th>( T_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>33</td>
<td>25</td>
<td>37</td>
<td>45</td>
</tr>
</tbody>
</table>

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{example.png}
\caption{Final states of agents and tasks}
\end{figure}

However, from the results we can see that \( T_3 \) was not allocated based on the SK algorithm and the income of the proposed algorithm is smaller than the results of the SK algorithm except \( T_3 \). First, the reason for \( T_3 \) not to be allocated based on SK algorithm is that \( T_3 \) has a big capability demand but owns little reward, and each agent in SK algorithm is inclined to join coalitions which have more reward. Therefore, all tasks except \( T_3 \) were allocated earlier but no any coalition is left in each agent’s coalition list, and there is no any available coalition to be selected to perform \( T_3 \). Moreover, the SK algorithm has a limitation of coalitional size and does not consider all possible coalitions, it is possible that coalitions left in each agent’s coalition list can not perform \( T_3 \) with insufficient resources. Second, the reason for the income of the proposed algorithm to be smaller than the results of the SK algorithm except \( T_3 \) is that the SK algorithm is addressed in obtaining the most income for each single given task but does not consider whether the given tasks are performed really or not. Therefore, some tasks obtaining big income may make other tasks with little reward not be allocated at last. In contrast, the proposed algorithm in this paper mainly aims to allocate all given tasks successfully. The proposed algorithm is addressed in maximizing the whole income of all tasks but not the income for each single task. Therefore, although some tasks’ incomes in the proposed algorithm are slightly lower than
the SK algorithm’s, the whole income of all tasks in the proposed algorithm are much larger than the SK algorithm’s.

The reason which makes comparison of the two algorithms is that although Shehory and Kraus presented distributed algorithm to form a coalition for a task, any communication and negotiation among agents was not involved, while our algorithm designs detailed steps of communication and negotiation to determine each agent’s workload for its tasks in the process of coalition formation.

Furthermore, we also give curves of number of steps, required time and total income as shown in Figs. 10 ∼ 12, respectively. The computational environments include the Intel Core 2 Duo P7370 (2.00 GHz) and 2 GB RAM.

Intuitively, Fig. 10 shows that in our algorithm all tasks can be found and performed by teemed agents within finite loops, and the time of 50 independent trails shown in Fig. 11 is about 0.5 seconds. In addition, from Fig. 12, our algorithm can find a better coalition for each task via learning for a period of time insured by profit sharing learning.

**Fig. 10** Curves of numbers of steps

![Fig. 10 Curves of numbers of steps](image)

**Fig. 11** Curve of required time

![Fig. 11 Curve of required time](image)

**5 Conclusions and future work**

In this paper, we develop a distributed algorithm for parallel multi-task allocation in fields of multi-agent systems, and evaluate the performance of the proposed algorithm against Shehory and Kraus’ algorithm. The comparison shows that the proposed algorithm is significantly more effective and robust for mobile agent in application domains of MAS such as disaster response management. These improvements stem from the fact that the proposed algorithm can exactly find a coalition for every task and give the real workload of every tasked agent without any resource conflict, and thus provide a specific and significant reference for practical control tasks. Moreover, the proposed distributed algorithm can be used in resource allocation, disaster response management, and so on, where there are many mobile agents.

For future work, we will concentrate on decreasing communication and negotiation cost in environments with a mass of agents.

**Fig. 12** Curves of total income

![Fig. 12 Curves of total income](image)

**References**


SU Zhao-Pin Received her B.S. and Ph.D. degrees in computer science from Hefei University of Technology in 2004 and 2008, respectively. Currently, she is a lecturer with the School of Computer and Information, Hefei University of Technology. Her research interest covers autonomous agent, reinforcement learning, and immune algorithm. Corresponding author of this paper. E-mail: szp@hfut.edu.cn

Jiang Jian-Guo Received his M.S. degree in computer science from Hefei University of Technology in 1989. He is currently a professor at the School of Computer and Information, Hefei University of Technology. His research interest covers automatic control, image processing, and software engineering. E-mail: jjjiang@hfut.edu.cn

Liang Chang-Yong Received his Ph.D. degree from Harbin Institute of Technology in 2001. He is currently a professor at the School of Management, Hefei University of Technology. His research interests covers collaborative filtering and intelligent decision support system. E-mail: cyliang@163.com

Zhang Guo-Fu Received his B.S. and Ph.D. degrees in computer science from Hefei University of Technology in 2002 and 2008, respectively. He is currently a lecturer with the School of Computer and Information, Hefei University of Technology. His research interest covers evolutionary computation, intelligent agent, and multi-agent systems, especially in coalition formation. E-mail: zg@hfut.edu.cn