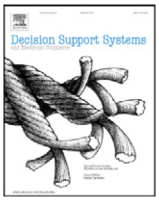
Decision Support Systems 53 (2012) 406–417

Contents lists available at SciVerse ScienceDirect

Decision Support Systems

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



Self-organization in an agent network: A mechanism and a potential application

Dayong Ye [a,*], Minjie Zhang [a], Danny Sutanto [b]

[a] School of Computer Science and Software Engineering, University of Wollongong, NSW 2500, Australia
[b] School of Electrical, Computer and Telecommunications Engineering, University of Wollongong, NSW 2500, Australia

ARTICLE INFO

Article history:
Received 24 May 2011
Received in revised form 7 February 2012
Accepted 14 February 2012
Available online 27 February 2012

Keywords:
Self-organization
Multi-agent systems
Trust
Multi-agent Q-learning

ABSTRACT

Self-organization provides a suitable paradigm for developing self-managed complex computing systems, e.g., decision support systems. Towards this end, in this paper, a composite self-organization mechanism in an agent network is proposed. To intuitively elucidate this mechanism, a task allocation environment is simulated. Based on self-organization principles, this mechanism enables agents to dynamically adapt relations with other agents, i.e., to change the underlying network structure, so as to achieve efficient task allocation. The proposed mechanism utilizes a trust model to assist agents in reasoning with whom to adapt relations and employs a multi-agent Q-learning algorithm for agents to learn how to adapt relations. Moreover, in this mechanism, it is considered that the agents are connected by weighted relations, instead of crisp relations. The proposed mechanism is evaluated through a comparison with a centralized mechanism and the *K-Adapt* mechanism in both closed and open agent networks. Experimental results demonstrate the adequate performance of the proposed mechanism in terms of the entire network profit and time consumption. Additionally, a potential application scenario of this mechanism is also given, which exhibits the potential applicability of this mechanism in some real world cases.

1. Introduction

Nowadays, more and more large and complex computing systems are emerging, e.g., decision support systems. These systems have high the desirability to be autonomic, which are capable of self-management, because self-management systems can save labor time of human managers and are able to adapt to environmental changes and ensure their own survivability [26]. Within this context, self-organization, which is defined as "*the mechanism or the process enabling the system to change its organization without explicit external command during its execution time* [34]", can be used in such systems to achieve autonomy [29,31,37,47]. For example, Smirnov et al. [37] integrated several technologies, e.g., ontology management, context management, constraint satisfaction, etc., to implement a self-organizing decision support system as a set of Web-services. On the other hand, De Wolf and Holvoet [7] recommended that agent-based modeling is best suited to build such autonomic systems. Thus, we consider that self-organizing multi-agent systems are good choices for developing such autonomic systems, as the self-organizing systems can continuously arrange and rearrange their organizational structures autonomously, without any external control, so as to adapt to environmental changes. In addition, the adaptation process should be performed in a decentralized manner, so that the autonomic systems could be robust against failures of any components in the system. With this motivation, this paper explores the area of self-organization in systems of autonomous agents, and particularly, we focus on structural adaptation in agent organized networks. Based on Kota et al.'s description [28], any self-organizing systems should have the following three properties:

1. No external control. All of the adaptation processes should be initiated internally and change only the internal state of the system.
2. Dynamic and continuous operation. The system is expected to evolve with time and thus, the self-organization process should be continuous.
3. No central control. The self-organization process should be operated through only local interactions of individual components in the system without centralized guidance.

Here, we are primarily interested in cooperative agent organized networks, which comprise cooperative autonomous problem-solving agents that receive inputs, perform tasks and return results, because such agent organized networks can clearly model the essence of complex computing systems. Moreover, the structure of an agent network is a manifestation of the relations between the agents, which in turn, determine their interactions. Therefore, structure adaptation signifies changing relations between agents, which then redirects their interactions. To make the problem clear, which we attempt to address in this paper, let us consider a simple example.

* Corresponding author. Tel.: +61 2 42214316.
E-mail addresses: dy721@uowmail.edu.au (D. Ye), minjie@uow.edu.au (M. Zhang), danny@elec.uow.edu.au (D. Sutanto).

0167-9236/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.dss.2012.02.009

Fig. 1 demonstrates an agent network. Consider that agent a_1 receives a task φ, which can only be executed by agent a_6. Then, the best option for agent a_1 is to forward task φ to its neighbor, agent a_4, which in turn forwards task φ to agent a_5 until task φ reaches agent a_6. Now, if this case occurs several times, one would expect performance to improve if a_1 adds a_6 as one of its neighbors because this will save an unnecessary overhead. Similarly, if a_1 rarely sends any request to its neighbor a_2, then removing neighbor a_2 from a_1's neighbors will reduce the computational overhead associated with taking agent a_2 into account whenever a_1 is making a decision.

Against this background, a composite self-organization mechanism in an agent network is proposed in this paper. Following self-organization principles, this mechanism is a decentralized and continuous process, which is followed by every agent to decide with whom and how to adapt its relations with others, based only on local available information. Furthermore, this mechanism involves only changing the structural relations between the agents and does not need agents to change their internal properties, e.g., resources and capacities. Likewise, this mechanism does not need new agents to be created, e.g., an overloaded agent creating a new agent and assigning part of its load to the new agent, or existing agents to be removed, e.g., agents, which are idle for a long time being removed. Therefore, this mechanism can be used as a self-management tool in some autonomic systems where creating new agents and removing existing agents are either not permitted or difficult to realize. According to this mechanism, pairs of agents are able to continuously and locally evaluate their inter-relations based on their past interactions. Every pair of agents can calculate the reward of each possible relation between them and cooperatively choose the one which can maximize the sum rewards of them. In addition, this mechanism can be operated in open agent networks as well, where new agents can join in and existing agents can leave. In comparison with current related works, which mainly focused on how to adapt crisp relations between agents, this mechanism originally integrates two notions into self-organization, i.e., trust and weighted relation. A trust model can be used by each agent to select the most valuable agents to adapt relations, and a weighted relation is used to measure how strong the relation is between two agents. Furthermore, for relation adaptation, most current related algorithms considered only one type of relations, while our relation adaptation algorithm can handle multiple types of relations. Kota et al.'s [27] algorithm can also deal with multiple types of relations. The main difference between our algorithm and theirs is that (1) our algorithm can handle weighted relations, while Kota et al.'s algorithm was designed for crisp relations only; (2) our algorithm can balance the exploration and exploitation, whereas Kota et al.'s algorithm overlooked this balance and always chose the most beneficial action at each step. This may let their algorithm converge to a suboptimal state, as stated by Kaelbling et al. [21].

In the next section, we first review related research in the self-organization area and through comparison, we point out the merits of our self-organization mechanism. Then, we develop a cooperative agent network model, which serves as an abstract platform, in Section 3, and based on this platform, our composite self-organization mechanism is devised in Section 4. By employing a general platform, i.e., task allocation in an agent network, instead of a particular existing system, we can develop a general mechanism that can be potentially applied to a wide variety of applications. After describing the self-organization mechanism, in Section 5, experimental results and analysis of the mechanism are presented. To demonstrate the potential applicability of the mechanism, in Section 6, we provide a particular scenario and exhibit how to exploit the mechanism into this scenario. Finally, this paper is concluded in Section 7.

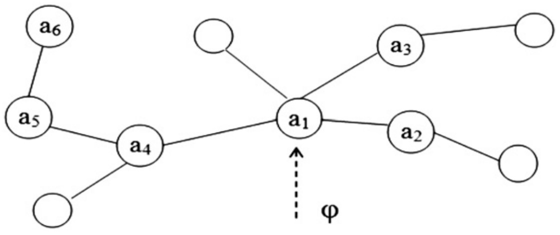


Fig. 1. A task allocation example.

2. Related work

Self-organization can be generated in multi-agent systems in several ways [3,33]. Serugendo et al. [33] divided self-organization mechanisms into five classes.

1. Direct interactions between agents using basic principles such as broadcast and localization. Typical examples of such mechanisms are those applied in the areas of self-assembly and distributed self-localization, where the formation of regular spatial patterns in mobile objects is required. For example, Mamei et al. [30] used a leader election algorithm to determine the center of gravity of the objects and propagate this information to all objects which keep moving until a specific distance from the center is reached.
2. Indirect interactions between agents, i.e., stigmergy, which means indirect coordination through traces left in the environment. Such mechanisms, based on the idea of stigmergy, have been applied to manufacturing control [24], supply network management [32], and so on.
3. Reinforcement of agent behaviors. These reinforcement self-organization mechanisms are based on the capabilities of the agents to dynamically modify their behavior according to some perceived states of the environment. For instance, Weyns et al. [6] presented a model that focuses on dynamically adapting logical relations between different behaviors, represented by roles. An agent, starting from its current state, can successively follow this model to complete its tasks.
4. Cooperation behavior of individual agents. A typical example of these mechanisms is organization self-design, which uses the primitives of agents' composition and decomposition, e.g., Kamboj and Decker [22] developed an organizational self-design mechanism in semi-dynamic environments, which enables an agent to be divided into two agents to respond to overwhelming environmental demands and enables two agents to merge into one agent if communication overhead between the two agents is too high.
5. Generic architectures or meta-models. The self-organization mechanisms, based on generic architectures or meta models of the agents organization, are instantiated and subsequently dynamically modified as needed according to the requirements of a particular application. A typical example is PROSA [4], which is based on a holonic hierarchy model. Self-organization refers to altering the holonic hierarchy following perturbations of the agent environment by using a known decision making technique.

However, most of the self-organization mechanisms, summarized in [33], are not applicable to an explicitly modeled agent network, as they are based on reactive agents interacting in unstructured ways. Some of the few mechanisms that do consider underlying agent network structures are centralized in nature, requiring some specialized agents to manage the adaptation process. In 2001, Horling et al. [17] proposed an organization structure adaptation method, which employed a central blackboard, by using self-diagnosis. Their method involves a diagnostic subsystem for detecting faults in the organization. Then, against these faults, some fixed pre-designed reorganization steps are launched. Hubner et al. [18] presented a controlled reorganization mechanism that is a top–down approach, in which a specialized group of agents perform the reorganization process for the whole organization. Bou et al. [5] developed a centralized reorganization mechanism, in which the central authority named "autonomic electronic institution" modifies the norms of the institution to achieve institutional goals. Hoogendoorn [16] presented an approach based on max flow networks to dynamically adapt organizational models to environmental fluctuation. His approach assumed two special nodes in the network, namely the *source node* with an indegree of 0 and the *sink node* with an outdegree of 0. These two special nodes make the approach centralized in nature, since if these two nodes are out of order, the organization adaptation process might be impacted. Hermoso et al. [15] developed a coordination artifact in an open multi-agent system, which is used to build and evolve a role taxonomy model over time. Their role taxonomy model, acting as an information center, is available to all the agents and helps them find valuable partners to improve their individual utilities. Hence, the approaches proposed in [5,15–18] are centralized in nature and have the potential of the single point of failure.

Self-organization mechanisms focusing on network structural adaptation (namely adapting relations among agents), which are used to improve team formation or task allocation, have also been researched [1,9,10,12,27].Gaston and desJardins [9] developed two network structural adaptation strategies for dynamic team formation. Their first strategy was a structure-based approach, where an agent prefers to select another agent to form a connection, which has more neighbors. Their second strategy was a performance-based approach, where an agent prefers to form a connection with the agent who has better performance. The two strategies are suitable in different situations. Glinton et al. [10] empirically analyzed the drawback of the structure-based strategy proposed in [9], and then designed a new network adaptation strategy, which limits the maximum number of links an agent can have. Abdallah and Lesser [1] did further research in self-organization of agent networks and creatively used reinforcement learning to adapt the network structure. Their method enables agents to not only adapt the underlying network structure during the learning process but also use information from learning to guide the adaptation process. Griffiths and Luck [12] presented a tag-based mechanism for supporting cooperation in the presence of cheaters by enabling individual agents to change their neighborhoods. Griffiths and Luck's mechanism is very suitable in some special dynamic environments where trust or reputation among agents is difficult to build up. However, these network structural adaptation methods, proposed in [1,9,10,12], assumed that only one type of relation exists in the network and the number of neighbors possessed by an agent has no effect on its local load. These assumptions are impractical in some cases where multiple relations exist among agents in a network and agents have to expend resources to manage their relations with other agents. To overcome this disadvantage, Kota et al. [27] devised a network structural adaptation mechanism, which took multiple relations and relation management load into account. The relation adaptation algorithm adopted in their mechanism lets agents take actions, which can maximize the utility at each step. Nevertheless, as stated by Kaelbling et al. [21], this kind of algorithms, which always takes the highest utility action, overlooks the tradeoff between exploitation and exploration and may finally converge to a sub-optimal state. Furthermore, Kota et al.'s mechanism is somewhat biased from the agent, which initializes the relation adaptation process, towards the agent, which is requested to adapt a relation, as the initializing agent evaluates most attributes of the reward of changing a relation. This bias might cause agents to be a little subjective when making decisions.

Besides the disadvantages mentioned above, the common limitation of current related research is that candidate selection for self-organization, i.e., relation adaptation, is simplified. Intuitively, before adapting relations, agents should first decide with whom to adapt a relation and this course is known as candidate selection. For candidate selection, agents in current related work use only their own experience.

In addition, current self-organization mechanisms consider only crisp relations between agents, which might be another limitation. Here, crisp relation means that between two agents there is either a relation or no relation. To overcome this limitation, weighted relation is used in this paper, which means that between two agents, there is a relation strength, ranged in [0, 1], to indicate how strong the relation is between two agents. The introduction of weighted relation is reasonable, because, in the real world, the relation change between two persons usually occurs gradually rather than suddenly. Thus, weighted relations should be more flexible and more suitable in agent networks than crisp relations.

Against this background, in this paper, we first model a task-solving agent network and then propose a composite self-organization mechanism. This self-organization mechanism consists of three elements, which are claimed as a three-fold contribution of this paper. The first one is that, for candidate selection, we integrate a trust model to enable agents to use not only their own experience but also other agents' opinions to select candidates, which can make agents select the most valuable candidates to adapt relations. The second one is that, for adapting multiple relations, we develop a multi-agent Q-learning algorithm. This algorithm enables two agents to independently evaluate their rewards about adapting relations and balances exploitation and exploration. Consequently, our mechanism could overcome the aforementioned flaws of Kota et al.'s mechanism [27]. The last one is that, in contrast with current related approaches, which consider only crisp relations, we introduce weighted relations into our self-organization mechanism. The introduction of weighted relations can improve the performance of our self-organization mechanism (shown in Section 5) and make the mechanism more suitable in dynamic environments.

3. The agent network model

There are several existing frameworks for modeling agent organizations or networks, such as [42] and [35]. The focus of these frameworks is on how to automatically build an agent organization or an agent network based on users' requirements. These frameworks [35,42] did not consider self-organization of the established agent organizations or agent networks. In this paper, our focus is on self-organization, which requires that agent networks can autonomously and dynamically adapt their structures at run-time when the environment changes. Hence, these frameworks cannot be utilized in this paper. In this section, an agent network model is proposed which concentrates on how to self-adapt an existing agent network structure to achieve efficient task allocation. The agent network model describes how the network is organized and the features of the network. Those contents regarding how to operate the network will be depicted in the next section, i.e., how to choose candidates for relation adaptation and how to adapt relations with these candidates.

The agent network model proposed in this paper is based on the model devised by Kota et al. [27], i.e., defining multiple relations in an agent network. Our agent network model, on the other hand, extends Kota et al.'s model in two aspects. The first aspect is that instead of crisp relations used by Kota et al., we define weighted relations, which connect agents in the network. The second aspect is that we employ and extend Semi-Markov Decision Process (SMDP) to model the multi-agent environment, while Kota et al.'s model did not include such a mathematical framework. We use SMDP to model the agent network, because SMDP is useful for studying various optimization problems, which can be solved by reinforcement learning [39].Thus, we can then conveniently develop a multi-agent Q-learning algorithm in this model.

3.1. Weighted relations of agents

In our model, an agent network comprises a set of collaborative agents, i.e., $A=\{a_1, \ldots, a_n\}$, situated in a distributed task allocation environment. In the agent network, instead of crisp relations, two weighted relations are defined, which are *peer-to-peer* relation and *subordinate–superior* relation, respectively. The formal definitions of these two relations and *relation strength* are given as follows.

Definition 1. peer-to-peer

A peer-to-peer relation, denoted as "$\tilde{}\mu$" ($\sim^{\mu}\ A\times A$), is a Compatible Relation, which is reflexive and symmetric, such that $\forall a_i \in A : a_i \sim^{\mu_{ii}} a_i$ and $\forall a_i, a_j \in A : a_i \sim^{\mu_{ij}} a_j \Rightarrow a_j \sim^{\mu_{ji}} a_i$, where $\mu_{ij} = \mu_{ji}$.

Definition 2. subordinate–superior

A subordinate–superior relation, written as "$\prec^{\mu}$" ($\prec^{\mu}\ A\times A$), is a Strict Partial Ordering Relation, which is irreflexive, asymmetric and transitive, such that $\forall a_i \in A : \ddot{A}(a_i \prec^{\mu_{ii}} a_i)$, $\forall a_i, a_j \in A : a_i \prec^{\mu_{ij}} a_j \Rightarrow \ddot{A}(a_j \prec^{\mu_{ji}} a_i)$ and $\forall a_i, a_j, a_k \in A : a_i \prec^{\mu_{ij}} a_j \wedge a_j \prec^{\mu_{ji}} a_k \Rightarrow a_i \prec a_k$.

Definition 3. relation strength

Relation strength, denoted as μ_{ij}, indicates how strong the relation is between agents a_i and a_j. μij is ranged in [0, 1], where a higher value means a stronger relation and 0 demonstrates no relation between two agents.

Relation strength affects the task allocation process, as agents usually prefer to allocate tasks to those agents which have high *relation strength* with them. Initially, the *relation strength* between any two neighboring agents is set to 0.5 by default, but it might be adapted during the succeeding task allocation process.

According to the above definitions, there are three neighborhood types, which are *peer*, *subordinate* and *superior* neighbors. Here is an example to explain the relations defined above. If agent a_i is a *peer* of a_j, written as $a_i \sim^{\mu_{ij}} a_j$, a_j is also a *peer* of a_i, written as $a_j \sim^{\mu_{ji}} a_i$, where $\mu_{ij} = \mu_{ji}$. If agent a_i is a *subordinate* of a_j, i.e., a_j is a *superior* of a_i, written as $a_i \sim^{\mu_{ij}} a_j$, and a_j is a *subordinate* of a_k, so that a_i is a *subordinate* of a_k as well. Since a_i is not a direct linked neighbor of a_k, there is no *relation strength* between them. For convenience, we stipulate that relation $\succ$ is the reverse relation of $\prec$, namely $a_i \prec^{\mu_{ij}} a_j \Leftrightarrow a_j \prec^{\mu_{ji}} a_i$, where $\mu_{ij} = \mu_{ji}$. As the agent network model operates in a cooperative environment, it is assumed that the relation is mutual between the concerned agents and both the concerned agents respect the relation type and *relation strength* between them. To clearly understand the meanings of relations and *relation strength*, here is a real-world example. In the human real-world, the *peer-to-peer* relation can be seen as the friendship relation, while the *subordinate–superior* relation can be considered the employee–employer relation. The *relation strength* indicates how strong the relation is between two persons. For instance, if two persons are friends, they could be general friends, good friends or very good friends. This difference can be demonstrated by using relation strength. Thus, our agent network model could be applied to model a social network. Although in a social network, more than two types of relations may exist, our model can be easily extended by defining more relations.

3.2. SMDP model for the agent network

The standard Semi-Markov Decision Process (SMDP) [40] provides a basic approach for modeling an agent's decision if the agent's actions may change at different states. In this paper, however, we need to model an open agent network where a mutative number of agents exist. We, thus, extend the standard SMDP in order to model a multi-agent environment, so that the open agent network is modeled as a tuple $\langle n(t), S, Act_1, \ldots, _{n(t)}, T, R_1, \ldots, _{n(t)}\rangle$, where $n(t)$ is the number of agents in the open agent network at time t, S is the state space and its specific value in time t is written as $s(t)$ and $s(t) \in S$, $Act_i(s(t))$ is the set of actions available to agent a_i at a given state $s(t)$. The transition function T is defined as $T: S \times Act_1 \times \ldots \times Act_n \rightarrow S$, which describes the resulting state $s(t+1) \in S$ when each agent takes one of their available actions at states(t). R_i is the reward function for agent a_i, $R_i : S \times Act_1 \times, \ldots, \times Act_{n(t)} \rightarrow \mathcal{R}_i$ and $\mathcal{R}_i$ is the immediate reward obtained by agent a_i when each agent takes one of their available actions at state $s(t)$. The details of reward will be depicted in Section 4.2. We first provide formal definitions of *State*, *Action* and *Action Set*.

Definition 4. State

A specific state at time t is defined as a tuple $s(t) = \langle Neig_1(t), \ldots, Neig_n(t)\rangle$, $s(t) \in S$. The element, $Neig_i(t)$, is the neighbor set of agent a_i at time t. $Neig_i(t)$ can be further divided into three subsets, $Neig_i^{-}(t)$, $Neig_i^{\prec}(t)$ and $Neig_i^{\succ}(t)$. $Neig_i^{-}(t)$ contains the peers of a_i, $Neig_i^{\prec}(t)$ consists of the direct superiors of a_i, and $Neig_i^{\succ}(t)$ comprises the direct subordinates of a_i.

Let us consider an example. In Fig. 2, agent a_1 maintains a *peer-to-peer* relation with its two neighbors, a_2 and a_3, so that $Neig_i^{-}(t) = \{a_1, a_2, a_3\}$ (since *peer-to-peer* relation, ~, is reflexive, a_1 is a *peer* of a_1 itself). Agent a_1 also has two direct *subordinates*, a_5 and a_7, so that $Neig_i^{\succ}(t) = \{a_5, a_7\}$. Additionally, a_1 has a direct *superior*, a_4, so that $Neig_i^{\prec}(t) = \{a_4\}$. Therefore, $Neig_i(t) = Neig_i^{\sim}(t) \cup Neig_i^{\succ}(t) \cup Neig_i^{\prec}(t) = \{a_1, a_2, a_3, a_4, a_5, a_7\}$. The number, alongside each line, demonstrates the *relation strength* between two agents. For example, the *relation strength* between agents a_1 and a_5 is 0.8.

Definition 5. Action

An action is defined as a decision made by an agent to adapt a relation with another agent.

There are seven different atomic actions defined in our model, which are *enh_~*, *enh_≺*, *enh_≻*, *wkn_~*, *wkn_≺*, *wkn_≻* and *no_action*. The explanation of each action and its reverse action is described in Table 1. It should be noted that the meanings of actions *enh_* and *wkn_* imply not only *enhance* and *weaken* a relation but also *form* and *dissolve* a relation, respectively.

In addition, the combinations of atomic actions include *wkn_≺ + enh_~*, *wkn_≺ + enh_≻*, *wkn_~ + enh_≺*, *wkn_~ + enh_≻*, *wkn_≻ + enh_~* and *wkn_≻ + enh_≺*. The meanings of these combination actions can be easily deduced from Table 1. For example, the combination, *wkn_≺ + enh_~*, which is taken by a_i with a_j, implies that a_i first dissolves a_j from a_i's *superior* and forms a *peer* relation with a_j. It should be noticed that an agent at different states might possess different available actions.

The possible choices of actions available to agents in different situations are illustrated as follows.

1. There is no relation between agents a_i and a_j. The possible choices of actions include *enh_~*, *enh_≺*, *enh_≻* and *no_action*.
2. a_i is a *peer* of a_j, i.e., $a_i \sim a_j$. The possible actions involve *wkn_~*, *wkn_~ + enh_≺*, *wkn_~ + enh_≻* and *no_action*.
3. a_i is a *subordinate* of a_j, i.e., $a_i \prec a_j$. The possible actions include *wkn_≺*, *wkn_≺ + enh_~*, *wkn_≺ + enh_≻* and *no_action*. These actions are based on a_i's perspective, while, in a_j's view, *aj* needs to reverse these actions.
4. a_i is a *superior* of a_j, i.e., $a_j \prec a_i$. This situation is just the reverse condition of $a_i \prec a_j$.

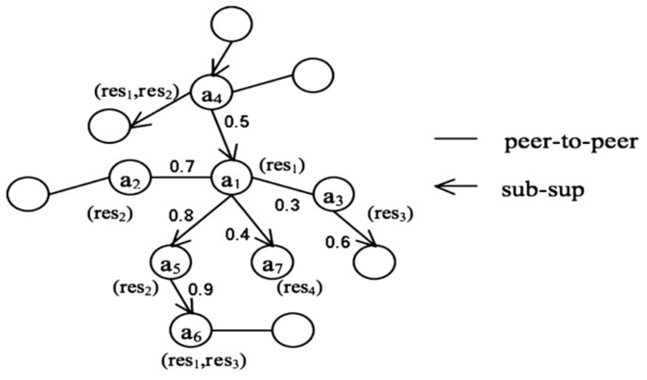


Fig. 2. An example agent network at time t.

Table 1
Explanation of each action.

Action (on a_i's view)	Explanation	Reverse Action (on a_j's view)
$a_i enh\_{\prec} a_j$	a_i enhances the *subordinate* relation with a_j, or a_i becomes a *subordinate* of a_j	$a_j enh\_{\succ} a_i$
$a_i enh\_{\sim} a_j$	a_i enhances the *peer* relation with a_j, or a_i becomes a *peer* of a_j	$a_j enh\_{\sim} a_i$
$a_i enh\_{\succ} a_j$	a_i enhances the *superior* relation with a_j, or a_i becomes a *superior* of a_j	$a_j enh\_{\prec} a_i$
$a_i wkn\_{\prec} a_j$	a_i weakens the *subordinate* relation with a_j, or a_i dissolves a *superior*, a_j	$a_j wkn\_{\succ} a_i$
$a_i wkn\_{\sim} a_j$	a_i weakens the *peer* relation with a_j, or a_i dissolves a *peer*, a_j	$a_j wkn\_{\sim} a_i$
$a_i wkn\_{\succ} a_j$	a_i weakens the *subordinate* relation with a_j, or a_i dissolves a *subordinate*, a_j	$a_j wkn\_{\prec} a_i$
$a_i no\_action\, a_j$	a_i does not take any action with a_j	$a_j no\_action\, a_i$

Definition 6. Action Set

An *action set*, $Act_i(t)$, is defined as a set of available actions for agent a_i at a given state $s(t)$, which might include some atomic actions or combinations of atomic actions.

For example, in Fig. 2, the action set of agent a_2 at time t is $Act_2(t) = \{enh\_{\sim}, enh\_{\prec}, enh\_{\succ}, wkn\_{\sim}, no\_action\}$. Agent a_2 cannot execute the actions $wkn\_{\prec}$ or $wkn\_{\succ}$ at time t, because a_2 has neither *subordinate*s nor *superior*s.

As described earlier in this section, the aim of the agent network is to assign tasks to agents such that the communication cost among agents is minimized and the benefit obtained by completing tasks is maximized. Each agent, $a \in A$, has a different set of resources, written as $Res(a)$, and different computation capacity which is used for completing tasks, written as $Comp(a)$. An agent possesses information about the resources it provides, the resources its *peer*s could provide, and the resources all of its *subordinate*s and its direct *superior* could provide. For example, in Fig. 2, the contents in parenthesis denote the resources supplied by an agent. Thus, the information about resources that agent a_1 possesses can be depicted by the following set $\{\{res_1, res_2, res_3\}, \{res_1, res_2\}, \{res_1, res_2, res_3, res_4\}\}$. The first subset demonstrates agent a_1's *peer*s resources information, including a_1 itself. The information in the second subset is about the resources that agent a_1's direct *superior* can provide. The last subset indicates the resources, which are supplied by all of a_1's *subordinate*s in total, even though a_1 might have no idea exactly which *subordinate* owns which resource.

The task environment presents a continuous dynamic stream of tasks that have to be completed. Each task, Φ, is composed of a set of subtasks, i.e., $\Phi = \{\varphi_1, \ldots, \varphi_m\}$. Each subtask, $\varphi_i \in \Phi$, requires a particular resource and a specific amount of computation capacity to fulfill. In addition, each subtask has a relevant benefit paid to the agent which successfully completes the subtask. Each task has a preset deadline as well, which should be satisfied. Otherwise, the benefit will be decreased gradually with time elapse till 0. Xu et al. [46] have shown that token-based mechanisms can collect as much information as broadcast mechanisms while using much less bandwidth. Thus, in this paper, a subtask φ_i is modeled as a token Δ_i which can be passed in the network to find a suitable agent to complete. Each token consists of not only the information about resource and computation requirements of the corresponding subtask, but also the token travel path which is composed of those agents that the token has passed. During the allocation of a subtask φ, an agent a_i always tries to execute the subtask by itself if it has adequate resources and computation capacity. Otherwise, a_i will generate a token for the subtask and pass the token to one of its *subordinate*s, which contains the expected resource. Since a_i does not know which *subordinate* has which *resource*, the token might be passed several steps in the agent network forming a delegation chain. If a_i finds no suitable *subordinate* (i.e., no *subordinate* contains the expected resource), it will try to pass the token to its *peer*s. In the case that no *peer* is capable of the subtask, a_i will pass the token back to one of its *superior*s, which will attempt to find some other *subordinate*s or *peer*s for delegation. When more than one agent is able to accept the token, a_i always passes the token to the agent, which has higher *relation strength* with a_i. Here, it should be emphasized that our focus is not on distributed task allocation. Instead, it is on the underlying agent network structure adaptation while allocating and executing tasks towards optimizing the efficiency of task completion. Thus, our work is independent of the specific task allocation algorithms that the agents may employ.

Apparently, the structure of the agent network will influence the task allocation process. In the next section, we will describe the composite self-organization mechanism used to adapt the structure of the agent network, involving an evaluation method to measure the profit of the network.

4. The composite self-organization mechanism

Before devising the self-organization mechanism, it is necessary to introduce an evaluation method to estimate the profit of the agent network. Towards this goal, we illustrate the concept of evaluation criteria, which includes cost, benefit, profit and reward of an agent and the agent network.

4.1. Network performance evaluation

The cost, benefit and profit of the network are calculated after a predefined number of time steps. The cost of the agent network, $Cost_{NET}$, consists of four attributes, i.e., communication cost, computation cost consumed by agents to complete assigned subtasks, management cost for maintaining subtasks and management cost for keeping neighborhood relations with other agents. We now depict the calculation of each of these four attributes.

4.1.1. Communication cost

Communication cost relies on how many tokens being transmitted during task allocation process and it can be calculated by using Eq. (1).

$$Cost_{NET} = \sum_{i=1}^{|A|} \left(C_{\sim} \cdot c_{i\sim} + C_{\succ} \cdot c_{i\succ} + C_{\prec} \cdot c_{i\prec} \right), \tag{1}$$

where A is the set of agents in the network, $c_{i\sim}$, $c_{i\succ}$ and $c_{i\prec}$ are the numbers of tokens agent a_i passes to its *peer*s, *subordinate*s and *superior*s, respectively, and, correspondingly, $C_{\sim}$, $C_{\succ}$ and $C_{\prec}$ are the communication cost coefficients for separately transmitting tokens to *peer*s, *subordinate*s and *superior*s. It is assumed that $C_{\succ} < C_{\sim} < C_{\prec}$, namely that passing tokens to *subordinate*s needs the lowest communication cost, while passing tokens to *superior*s requires highest communication cost. Therefore, agents always try to pass tokens to *subordinate*s first, then *peer*s, finally *superior*s, which matches the task allocation procedure described in Section 3.2.

4.1.2. Computation cost

Computation cost is measured as the amount of computation executed by the agents for completing the assigned tasks as in Eq. (2),

$$Load_{NET} = \sum_{i=1}^{|A|} \sum_{j=1}^{|T_i^E|} Comp_{ij}, \tag{2}$$

where $Comp_{ij}$ is the amount of computation cost agent a_i consumes for executing a subtask, which is represented by token Δ_j, and T_i^E is the set of subtasks, which have been executed by agent a_i.

4.1.3. Management cost

Management cost consists of two types of costs.

The first management cost is management computation cost, which is for maintaining subtasks. For each agent, there are two waiting lists. One stores the subtasks owned by the agent, say a_i, which have not yet been finished, written as $T_i^{W_1}$, and another list records the subtasks assigned by other agents to a_i, written as $T_i^{W_2}$. Therefore, the management computation load for agent a_i can be calculated by using Eq. (3),

$$load_i^1 = M \cdot \left(\sum_{j=1}^{\left|T_i^{W_1}\right|} t_{ij}^{(1)} \right) + \sum_{j=1}^{\left|T_i^{W_2}\right|} t_{ij}^{(2)}), \tag{3}$$

where $t_{ij}^{(1)}$ demonstrates the number of time steps during which agent a_i maintains one of its own subtasks in $T_i^{W_1}$ before the subtask is completed, while $t_{ij}^{(2)}$ indicates the number of time steps during which agent a_i keeps one of the assigned subtasks in $T_i^{W_2}$ before the subtask is finished. M is the management computation coefficient.

The second management cost for an individual agent is the management relation load generated during keeping relations with other agents, which can be calculated by utilizing Eq. (4),

$$load_i^2 = M_{\sim} \cdot \sum_{j=1}^{\left|Neig_i^{\sim}\right|} t_{ij}^{\sim} + M_{\succ} \cdot \sum_{j=1}^{\left|Neigh_i^{\succ}\right|} t_{ij}^{\succ} + M_{\prec} \cdot \sum_{j=1}^{\left|Neigh_i^{\prec}\right|} t_{ij}^{\prec}, \tag{4}$$

where $Neig_i^{\sim}$, $Neig_i^{\succ}$ and $Neig_i^{\prec}$ have been defined in Definition 3, which are composed of a_i's *peers*, *subordinates* and *superiors*, respectively. $t_{ij}^{\sim}$, $t_{ij}^{\succ}$ and $t_{ij}^{\prec}$ are the number of time steps, which demonstrate the time length of agent a_i keeping the different relations with agent a_j. $M_{\sim}$, $M_{\succ}$ and $M_{\prec}$ are management relation coefficients for the relations *peer*, *subordinate* and *superior*, respectively. It is supposed that $M_{\succ} < M_{\sim} < M_{\prec}$, namely that an agent maintaining the *superior–subordinate* relation needs the lowest management load, while maintaining the *subordinate–superior* relation is the most expensive.

The benefit of each agent depends on how many subtasks that are completed by the agent. As depicted in Section 3.2, each task Φ contains several subtasks, φ_1, φ_2, …, represented as tokens Δ_1, Δ_2…, and when a subtask φ_i is successfully completed, the agent, which executes this subtask, can obtain the relevant benefit.

Therefore, the benefit of the network is the sum of the benefits that all the agents obtain in the network as in Eq. (5),

$$Benefit_{NET} = \sum_{i=1}^{|A|} benefit(a_i), \tag{5}$$

where $benefit(a_i)$ is the benefit that agent a_i obtained for successfully completing subtasks.

The reward of each agent, $\mathcal{R}_i$, is based on how much load could be reduced on agent a_i and how much load could be decreased on the intermediate agents, and how much potential benefit might be obtained by agent a_i in the future. Here, an intermediate agent is an agent, which resides on a token path. For example, agent a_i has no relation with agent a_j, but a_i received many subtasks from a_j before. a_i, then, makes the decision with regard to forming a relation with a_j. If a_i would like to form the *subordinate–superior* relation with a_j, i.e., performing the action *form*_$\prec$ (for a_j, performing the reverse action *form*_$\succ$, recall Table 1). The load reduction on a_i is $-M_{\prec} \cdot \hat{t}_{ij}^{\prec}$, as a_i would maintain a new relationship with a_j, where $\hat{t}_{ij}^{\prec}$ is the expected time length that a_i will keep this relation with a_j and the negative sign denotes this value represents an increase in the load. However, other intermediate agents between a_i and a_j are affected by a_i's decision and would obtain rewards, because they do not need to pass tokens between agents a_i and a_j any more. For one of those intermediate agents, say a_k, the reduced load is $C \cdot c_k$, where c_k is the number of tokens that are passed by a_k and received by a_i. The rewards benefited by those intermediate agents are accounted as a_i's rewards. For a_j, it could save management computation load, i.e., $load_j^1$, since it can directly pass tokens to a_i without waiting for intermediate agents to pass tokens, but a_j also needs to maintain a new relationship with a_i. The load reduction on a_j, therefore, is $M \cdot \sum_{k=1}^{m} |\Delta_k.path| - M_{\succ} \cdot \hat{t}_{ij}^{\succ}$, where m is the number of tokens passed from a_j to a_i, and $|\Delta_k.path|$ means the length of the path Δ_k has passed. Since we assume that a token is passed between two agents during one time step, $|\Delta.path|$ actually demonstrates the time steps of a token passing.

The potential benefit, which will be obtained by a_i, is evaluated on the basis of the benefit a_i gained for completing the subtasks assigned by a_j, i.e., $\delta_{\prec} \sum_{\varphi.owner=a_j} \varphi.benefit$, where $\delta_{\prec}$ is a potential benefit coefficient. Analogically, for a_j, the potential benefit is $\delta_{\succ} \sum_{\varphi.owner=a_i} \varphi.benefit$. We suppose that $\delta_{\prec} > 1$, $\delta_{\sim} = 1$ and $\delta_{\succ} < 1$, namely that the action *form*_$\prec$, with a_i as the *subordinate* and a_j as the *superior*, can make a_i potentially receive more subtasks from a_j and then get more benefits. Therefore, the reward of an agent can be calculated by adding load reduction with potential benefit. The reward of the network, $\mathcal{R}$, is then calculated as the sum of rewards achieved by all the agents in the network after reorganization of the network as in Eq. (6).

$$\mathcal{R} = \sum_{i=1}^{|A|} \mathcal{R}_i \tag{6}$$

Finally, the profit of the entire network can be calculated by using Eq. (7).

$$Profit_{NET} = Benefit_{NET} - Cost_{NET} - Load_{NET} - \sum_{i=1}^{|A|} \left(load_i^1 + load_i^2 \right) \tag{7}$$

4.2. Self-organization mechanism design

In this paper, the aim of our self-organization mechanism is to improve the efficiency of task allocation in the agent network via changing the network structure, i.e., adapting the relations among agents. As described in Section 2, when an agent, a_i, wants to adapt relations with other agents, there are two problems, which have to be faced by a_i. The first problem is determining with whom a_i should adapt relations, and the second one is determining how to adapt relations with those selected agents. For the first problem, the selection process of each agent is based not only on its own former task allocation process but also on the integrated trust model. Through the trust model, an agent can get opinions from other agents about candidate selection. For example, when a_i is considering a_j as one of its candidates for adapting relations, a_i will ask other agents, say a_k and a_l, for their opinions regarding whether a_j is worthy to be a candidate. For the second problem, i.e., how to adapt relations, a multi-agent Q-learning approach is employed. The reason for choosing the Q-learning approach is that it provides a simple and suitable methodology for representing our mechanism in terms of actions and rewards. The self-organization mechanism is now illustrated in the following subsections.

4.2.1. Candidate selection

To assist each agent to select the most valuable candidates to adapt relations, we present a trust model based on Dezert-Smarandache theory (DSmT) [36]. Many researchers have made prominent efforts on trust and reputation models, such as probabilistic theory-based trust models (Teacy et al. [41]), certified reputation model (Huynh et al. [19]), and evidential trust models (Wang and Sun [43]). In addition, a thorough survey of trust and reputation systems for online service provision was also given by Josang et al. [20]. Briefly, trust models are usually developed for identifying whether an agent is trustworthy or not. In this paper, to the best of our knowledge, we originally use a trust model for candidate selection in an agent network for relation adaptation and there is not an existing quantitative comparison between different trust models for agent candidate selection. Thus, the criteria, based on which we opt for a trust model, are that whether the trust model is suitable for candidate selection, whether it is easy to understand by both the readers and us and whether it is easy to implement. The emphasis of this paper is not developing a new and efficient trust model. Instead, the trust model is used only as a complementary part of our self-organization mechanism. Through our investigation, Dezert-Smarandache theory is more suitable for our requirements than other trust models, because the theory has good expressiveness and low computational complexity for trust representation, and is easy to implement.

We now introduce the key concepts of Dezert-Smarandache theory [36]. Let T mean that the given agent considers a given correspondent to be trustworthy and $\Theta=\{T, \neg T\}$ be the general framework of discernment. The hyper-power set of Θ is represented as $H^{\Theta}=\{\emptyset, \{T\},\{\neg T\},\{T\cap\neg T\}, \Theta\}$. There is a general basic belief assignment which is a function $m:H^{\Theta}\rightarrow[0, 1]$ where

$$\begin{cases} m(\theta)=0 \\ \sum_{B\ H^{\Theta}} m(B)=1. \end{cases} \tag{8}$$

Thus, $m(\{T\})+m(\{\neg T\})+m(\{\Theta\})+m(\{T\cap\neg T\})=1$, as $m(\emptyset)=0$.

The trust model is defined as a tuple, $T_j^i=\langle m_j^i(\{T\}), m_j^i(\{\neg T\}), m_j^i(\{\Theta\}), m_j^i(\{T\cap\neg T\})\rangle$, where i and j represent two different agents a_i and a_j, separately. Each element in T_j^i is described as follows,

1. $m_j^i(\{T\})$ means the degree of a_i trusting a_j;
2. $m_j^i(\{\neg T\})$ indicates the degree of a_i distrusting a_j;
3. $m_j^i(\{\Theta\})$ demonstrates the degree of uncertainty about the trustworthiness a_i to a_j. This case happens when a_i lacks evidence regarding a_j. If a_i has no evidence at all, $m_j^i(\{\Theta\})=1$; otherwise, $m_j^i(\{\Theta\})<1$;
4. $m_j^i(\{T\cap\neg T\})$ depicts the degree of contradiction with regard to the trustworthiness a_i to a_j. This case is caused by the situation, for example, that a_i trusts a_j but other agents, which provide a_i their opinions, distrust a_j. Thereby, a_i gets into contradiction.

Thus, initially, trust between any two agents is $T_j^i=\langle m_j^i(\{T\})=0, m_j^i(\{\neg T\})=0, m_j^i(\{\Theta\})=1, m_j^i(\{T\cap\neg T\})=0\rangle$. Trust is, then, acquired through task allocation between agents. Suppose that a_i is evaluating the trust of a_j and a_j has completed x subtasks for a_i. a_i uses the notion, QoS (Quality of Service), to express how it is satisfied with a_j's service of each subtask. Therefore, for x subtasks, there are x QoS values. QoS is in the range of [0, 1], and its calculation depends on the time spent to complete a subtask. For example, if a_j can finish a subtask on time for a_i, a_i's estimation on this subtask is $QoS=1$; otherwise, the value of QoS will be decreased over time. Based on QoS, a_i's trust evaluation to a_j can be obtained through Eq. (9).

$$\begin{cases} m_j^i(\{T\})=\sum_{QoS>\Omega} QoS \\ m_j^i(\{\neg T\})=\sum_{QoS<\omega} QoS \\ m_j^i(\{\Theta\})=\sum_{\omega\leq QoS\leq\Omega} QoS \end{cases} \tag{9}$$

where ω and Ω, $0\leq\omega\leq\Omega\leq 1$, are lower and upper thresholds, respectively. After calculation, the elements in T_j^i are then normalized to satisfy $\sum_{B\ H^{\Theta}} m_j^i(B)=1$.

In addition, a_i might ask one of its neighboring agents, say a_k, for trust evaluation to a_j, and then combines a_k's opinion with a_i's own view to a_j. This is trust evaluation combination which can be computed as

$$T_j^i=T_j^i\oplus T_j^k, \tag{10}$$

where
$m_j^i(B_1)=m_j^i(B_2)\oplus m_j^k(B_3)=\sum_{(B_1,B_2,B_3\ H^{\Theta})\wedge(B_2\cap B_3=B_1)} m_j^i(B_2)m_j^k(B_3)$.

Furthermore, there might be another case. a_i asks a_k's opinion of a_j, but a_k may have no idea about a_j. a_k then inquires one of its neighbors, a_l's, opinion to a_j. This is trust transitivity which can be calculated as

$$T_j^k=T_l^k\otimes T_j^l, \tag{11}$$

where

$$m_j^k(\{T\})=\left(m_l^k(\{T\})\right)+\beta m_l^k(\{T\cap\neg T\})\cdot m_j^l(\{T\})$$

$$m_j^k(\{\neg T\})=\left(m_l^k(\{\neg T\})\right)+\beta m_l^k(\{T\cap\neg T\})\cdot m_j^l(\{\neg T\})$$

$$m_j^k(\{T\cap\neg T\})=\left(m_l^k(\{T\cap\neg T\})\right)+\beta m_l^k(\{T\cap\neg T\})\cdot m_j^l(\{T\cap\neg T\})$$

$$m_k^j(\{\Theta\})=1-m_j^k(\{T\})-m_j^k(\{\neg T\})-m_j^k(\{T\cap\neg T\}),$$

and β is a constant which is in the range (0, 1).

We are now ready to describe the candidates selection process which is shown in Algorithm 1.

Algorithm 1: candidates selection according to *ai*.

```
1 $Candidates_i \leftarrow \emptyset$, $TempCands_i \leftarrow \emptyset$;
2 After predefined time steps, through Equation 9, $a_i$
   evaluates the trust of its neighbors and those agents
   that completed $a_i$'s subtasks but not $a_i$'s neighbors;
3 $TempCands_i \leftarrow$ all the agents mentioned in Line 2;
4 for each $a_j \in TempCands_i$ do
5     if $m_j^i(\{T\}) > \rho_1$ and
          $a_j \notin Neig_i^{\sim} \vee Neig_i^{\succ} \vee Neig_i^{\prec}$ then
6         $a_i$ inquires other agent's opinions;
7     else if $m_j^i(\{\neg T\}) < \rho_2$ and
          $a_j \in Neig_i^{\sim} \vee Neig_i^{\succ} \vee Neig_i^{\prec}$ then
8         $a_i$ inquires other agent's opinions;
9     end if
11 end for
12 $a_i$ syncretizes opinions via Equations 10 and 11;
13 for each $a_j \in TempCands_i$ do
14     if $m_j^i(\{T\}) > \rho_1$ and
          $a_j \notin Neig_i^{\sim} \vee Neig_i^{\succ} \vee Neig_i^{\prec}$ then
15         $Candidates_i \leftarrow a_j$;
16     else if $m_j^i(\{\neg T\}) < \rho_2$ and
          $a_j \in Neig_i^{\sim} \vee Neig_i^{\succ} \vee Neig_i^{\prec}$ then
17         $Candidates_i \leftarrow a_j$;
18     end if
19 end for
```

Firstly, in Line 2, after a period, a_i should have some subtasks completed by other agents. a_i, then, evaluates the trust of those agents, which completed a_i's subtasks by using Eq. (9). Meanwhile, a_i also evaluates the trust of its neighboring agents, as a_i might want to weaken relations with those neighbors that cannot complete a_i's subtasks on time. In Lines 4–11, a_i inquires other agents' opinions about the potential candidates in $TempCands_i$. Here, ρ_1 and ρ_2 are two thresholds ranged in (0, 1). The inquiry process begins with a_i initially contacting a neighbor. If this neighbor cannot provide an answer to a_i, it then gives a_i a referral which designates another agent. The process terminates in success when a trust value tuple, i.e., T_y^x, is received and in failure when the *depthLimit* is reached or when the inquiry arrives at an agent, which neither gives an answer nor a referral. Here, *depthLimit* restricts the length of referral chains. Kautz et al. [25] have proved that shorter referral chains are more likely to be fruitful and accurate, so we set *depthLimit* as 2 in this paper. Line 5 indicates that a_j is not a neighbor of a_i but a_j could complete a_i's subtasks on time, while Line 6 demonstrates that a_j is a neighbor of a_i but a_j has a bad task completion record. After inquiry, a_i synthesizes other agents' opinions and selects candidates again (Lines 12–19). After candidate selection, a_i will start the relation adaptation with those candidates, which is introduced in the next subsection.

4.2.2. Relation adaptation

Before describing our relation adaptation algorithm, we first consider a simple scenario with two agents, a_i and a_j, and three available actions for each agent. The reward matrix of the two agents is displayed in Table 2.

Each cell $(r_i^{x,y}, r_j^{x,y})$ in Table 2 represents the reward received by the row agent (a_i) and the column agent (a_j), respectively, if the row agent a_i plays action x and the column agent a_j plays action y. Algorithm 2 demonstrates our relation adaptation approach in pseudocode form.

Algorithm 2: relation adaptation according to *ai*.

```
1 for each $a_j \in Candidates$ do
2     $Act_i \leftarrow available\_actions(a_i, a_j)$;
3     $Act_j \leftarrow available\_actions(a_j, a_i)$;
4     for each $x \in Act_i, y \in Act_j$ do
5         Initialise $Q_{ix}$ and $Q_{jy}$ arbitrarily;
6         for $k = 0$ to a predefined integer do;
7             calculate $\pi_{ix}(k)$ and $\pi_{jy}(k)$;
8             $Q_{ix}(k+1) = Q_{ix}(k) +$
                  $\pi_{ix}(k)\alpha(\sum_y r_i^{x,y}\pi_{jy}(k) - Q_{ix}(k))$;
9             $Q_{jy}(k+1) = Q_{jy}(k) +$
                  $\pi_{jy}(k)\alpha(\sum_x r_j^{x,y}\pi_{ix}(k) - Q_{ix}(k))$;
10        end for
11    end for
12    $\langle x_{opti}, y_{opti}\rangle \leftarrow argMax_{match(x,y)}(Q_{ix} + Q_{jy})$;
13    $a_i, a_j$ take actions $x_{opti}$ and $y_{opti}$, respectively;
14    $\mu_{ij} \leftarrow \mu_{ij} + (N_j^i/\rho_3 - 1)$;
15    if $\mu_{ij} > 1$ then $\mu_{ij} \leftarrow 1$;
16    if $\mu_{ij} < 0$ then $\mu_{ij} \leftarrow 0$;
17    $\mu_{ji} \leftarrow \mu_{ij}$;
18 end for
```

After selecting candidates, a_i and a_j, firstly, estimate which actions are available at the current state (Lines 2 and 3) as described in Section 3.2. Then, a_i and a_j learn the Q-value of each available action, separately (Lines 4–11). In Line 5, the Q-value of each action is initialized arbitrarily, while, in Line 7, π_{ix} indicates the probability regarding agent a_i taking the action x. To calculate π_{ix}, we employ the -greedy exploration method devised by Gomes and Kowalczyk [11] shown in Eq. (12), where 0< <1 is a small positive number and n is the number of available actions of a_i. The reason for choosing the Q-greedy exploration method is that it has been justified through many applications, for example Galstyan et al. [8] applied the method to develop a decentralized resource allocation mechanism.

$$\pi_{ix} = \{\,(1-\epsilon) + (\epsilon/n),\ \text{if}\ Q_{ix}\ \text{isthehighest}\epsilon/n,\ \text{otherwise} \qquad (12)$$

Furthermore, in Line 8, Q_{ix} is the Q-value of action x taken by agent a_i. In Lines 8 and 9, $0<\alpha<1$ is the learning rate.

When finishing learning Q-values, a_i and a_j (Line 12) cooperate to find the optimal actions for both of them, where *match*(x, y) is a function which is used to test whether the actions x and y taken by a_i and a_j, respectively, are matched. An action is only matched by its reverse action described in Table 1. Therefore, a_i and a_j have to cooperate to find the actions, which can be matched together and maximize the sum of their Q-values. Finally, a_i and a_j adjust their *relation strength* (Lines 14–17). The adjustment depends on how many subtasks a_j completed for a_i in the last time steps, where the more subtasks completed by a_j the higher *relation strength* value achieved. In Line 14, N_j^i means the number of a_i's subtasks completed by a_j, and ρ_3 is a threshold which is an integer.

Table 2
Reward matrix of a_i and a_j.

$a_i a_j$	$enh\_\prec$	$enh\_\sim$	$enh\_\succ$
$enh\_\succ$	$r_i^{1,1}, r_j^{1,1}$	$r_i^{1,2}, r_j^{1,2}$	$r_i^{1,3}, r_j^{1,3}$
$enh\_\sim$	$r_i^{2,1}, r_j^{2,1}$	$r_i^{2,2}, r_j^{2,2}$	$r_i^{2,3}, r_j^{2,3}$
$enh\_\prec$	$r_i^{3,1}, r_j^{3,1}$	$r_i^{3,2}, r_j^{3,2}$	$r_i^{3,3}, r_j^{3,3}$

Having illustrated the fundamental part of our self-organization mechanism, we now discuss how this mechanism deals with an open agent network. When a new agent a_j joins the network, it has no historical interactions with the existing agents in the network. Thereby, a_j cannot use Algorithm 1 to select candidates to establish relations. Generally, a_j prefers to connect to an existing agent, a_i, which has more uncompleted subtasks compared with those agents which have less uncompleted subtasks, in order to achieve benefits by completing those subtasks. At the same time, a_j does not like to connect to a_i, which already maintains many neighbors, because the more neighbors a_i has, the less opportunity for a_j to get subtasks from a_i. In addition, too many neighbors will bring a heavy workload on a_i for keeping relations with those neighbors. Thus, the candidate selection of a_j is based on the ratio between each existing agent's amount of uncompleted subtasks and the number of neighbors (usually known as degree), recorded as *TDR* (Task–Degree Ratio). If the *TDR* of a_i is larger than a predefined threshold, ρ_4, a_j builds a *subordinate–superior* relation with a_i, as a_j being the *subordinate* and $\mu_{ij} = \mu_{ji} = 0.5$. In contrast, when an existing agent leaves, the other agents can easily reorganize by using our self-organization mechanism.

5. Experiment and analysis

In this section, the effectiveness of our self-organization mechanism is demonstrated through experimental evaluation. We first describe the experimental setup and thereafter present the experimental results and analysis.

5.1. Experimental setup

To objectively exhibit the effectiveness of our self-organization mechanism, named *Learn-Adapt*, we compare our mechanism with three other mechanisms, i.e., *Central*, *K-Adapt* [27] and *Learn-Adapt without Trust*. As the aim of our work is to extend Kota et al.'s [27] work, i.e., *K-Adapt*, we intuitively choose *K-Adapt* as a standard for comparison. On the other hand, *K-Adapt* is the first self-organization mechanism which considered multiple relations in an agent network and other related works considered only one type of relations. Thus, Kota et al.'s work represents the most efficient work in our research domain. If our mechanism could (hopefully) outperform Kota et al.'s work to some extent, our contribution could be demonstrated. It is difficult to use other existing approaches for comparison because they were designed in a different environment (existence of only one type of relations) from ours.

1. *Central*: this is an ideal centralized task allocation mechanism in which there is an external omniscient central manager that maintains information about all the agents and tasks in the network. The central manager is able to interact with all the agents in the network without cost. This method is neither practical nor robust, but it can be used as an upper bound of the performance of an organization in our experiment. By comparing with Central mechanism, we can have an intuitional view of how the decentralized mechanisms can do and the performance gap between the decentralized mechanisms and the best mechanism, i.e., Central mechanism. Obviously, it is hoped that the performance of a good decentralized mechanism can meet or approach to the performance of Central mechanism.
2. *K-Adapt*: this mechanism was proposed by Kota et al. [27], which utilized a meta-reasoning approach to adapt the relation between two agents. Their approach always chooses the actions that can bring the most utilities. The aim of this paper is to improve and extend K-Adapt in some aspects, namely introducing a trust model, extending crisp relations to weighted relations, and developing a (hopefully) more efficient relation adaptation algorithm. Thus, we selected *K-*

Adapt as a standard to evaluate our mechanism. In that case, we can see whether our mechanism can really improve somewhat compared with *K-Adapt*.

3. *Learn-Adapt without Trust*: this mechanism is the same as *Learn-Adapt* but without the trust model. Thus, the candidate's selection is based only on an agent's own interaction history with other agents, i.e., that a_i wants to adapt relations with those agents who complete many or few subtasks of a_i. The purpose of including this mechanism is to reveal the significance of the trust model.

Since the *Central* and *K-Adapt* mechanisms do not cover weighted relations, in these two mechanisms, we simply set $\mu_{ij} = \mu_{ji} = 1$, if a_i and a_j have a relation; otherwise $\mu_{ij} = \mu_{ji} = 0$.

In this experiment, the agent organized network is generated by using the Small World network [44], in which most neighbors of an agent are connected to each other. Nonetheless, the approach presented by Watts and Strogatz [44] deals with only one relation between agents in the Small World network. We, thus, modify the approach to accommodate multiple relations by randomly changing the relation between two neighboring agents. Moreover, in order to control the number of resources an agent can have, a parameter called *Resource Probability* (*RP*) is employed, such that an agent is assigned a resource with probability *RP*. Hence, with the increase of *RP*, agents could possess more resources. For simplicity, tasks are created by randomly generating required resource types and the amount of resource of each resource type. In addition, the task arrival and task required time are distributed according to Poisson and exponential distributions, separately, and each task is randomly distributed to one of the agents in the system. Finally, the evaluated criteria consist of $Profit_{NET}$ (Eq. (7)), obtained by each approach in a percentage format with the maximum network profit gained by the *central* approach, and time consumed by these mechanisms. The experimental setting of our work is similar as that of Kota et al.'s work, since the essence of our work is the same as Kota et al.'s work, i.e., reasonably adapting multiple relations in an agent network. Our experimental setting, however, includes some new parameters which are exclusive in our work, e.g., learning rate, action selection distribution probability, etc., as several novel contents, i.e., the trust model, weighted relations and a new relation adaptation algorithm, are integrated in our mechanism. For clarity, the values of parameters that are used in this experiment and their meanings are listed in Table 3. These values were chosen experimentally to provide the best results.

5.2. Experimental results: closed networks

Fig. 3(a) and (b) demonstrate the percentage profits obtained by *K-Adapt*, *Learn-Adapt without Trust* and *Learn-Adapt* with different resource probabilities (*RP*), compared with the maximum profit, which is gained by *Central*. The x-axis represents the number of simulation runs and each run consists of 50,000 time steps. It can be seen that *Learn-Adapt* performs consistently better than both *K-Adapt* and *Learn-Adapt without Trust* in all situations. In Fig. 3(a) (*RP* = 0.1), with more simulation runs, the difference between *Learn-Adapt* and *K-Adapt* is gradually increasing. This is because when each agent has very few resources, agents have to allocate tasks to others. Thus, an

Table 3
Parameters setting.

Parameters	Values	Explanations
n	200	The number of agents
deg	6	The average number of neighbors
RP	0.1 and 0.6	Resource Probability
m	50,000	The number of time steps
α	0.2	Learning rate
	0.4	Action selection distribution probability
k	100	Learning rounds
ρ_1, ρ_2, ρ_4	0.7, 0.5, 1	Thresholds for choosing agents to adapt
ρ_3	5	Threshold for adapting *relation strength*

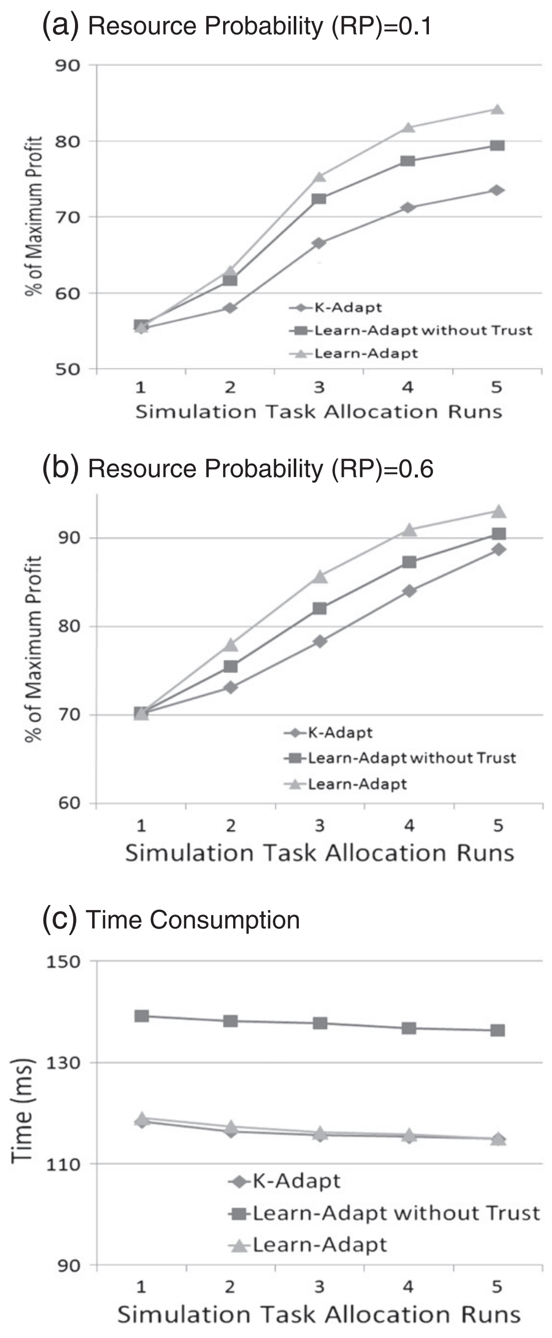


Fig. 3. Performance of three mechanisms in a closed agent network.

effective network structure could reduce agents' communication cost and management cost, and could further raise the profit of the entire network. In this case, a smarter mechanism could bring better performance through generating a more effective network structure. With the increase of resource probability (Fig. 3(b)), *K-Adapt*, *Learn-Adapt without Trust* and *Learn-Adapt* could achieve better performance. This can be explained by the fact that with higher resource probability, each agent would have more resources and, thus, could fulfill more tasks by itself. It should also be noted that the difference between *Learn-Adapt* and *K-Adapt* narrows as the resource probability rises. This is due to the fact that when agents become more homogeneous, a smarter method cannot correspondingly bring more profit for the network. Furthermore, Fig. 3(c) indicates the time consumption of *K-Adapt*, *Learn-Adapt without Trust* and *Learn-Adapt*. It can be seen that the time consumption of *Learn-Adapt* costs as little as *K-Adapt*, while *Learn-Adapt without Trust* consumes more time. This can be ascribed

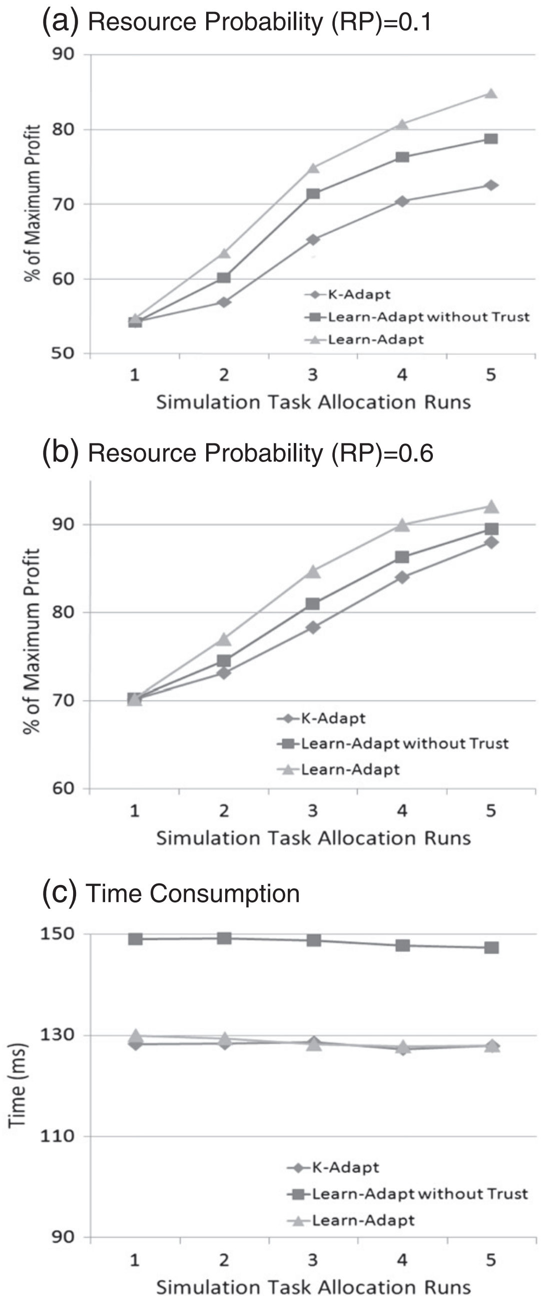


Fig. 4. Performance of three mechanisms in an open agent network.

to the fact that *Learn-Adapt without Trust* requires a time-consuming learning round in order to choose an optimal action to adapt agent relations. However, although *Learn-Adapt* also needs the time-consuming learning round, after integrating the trust model, *Learn-Adapt* curtails the time consumption markedly, as the trust model can help agents choose the most valuable candidates so as to reduce the volume of candidates set as shown in Algorithm 1.

5.3. Experimental results: open network

The experimental setup for an open agent network is analogous to the closed one. After each simulation run, 10 new agents join into the network and the resource assignments of these new agents are also based on *Resource Probability*, as described in Section 5.1. Meanwhile, 10 existing agents leave the network. If these leaving agents have uncompleted subtasks, they will transfer those uncompleted subtasks to their neighbors uniformly.

In the open agent network, we again find that *Learn-Adapt* performs better than both *K-Adapt* and *Learn-Adapt without Trust* (Fig. 4(a) and (b)), which show a similar trend as in the closed agent network. However, in Fig. 4(c), the time consumption of the three mechanisms converges relatively slowly. This is due to the fact that the agent network is open with new agents coming and existing agents leaving and hence the network is always in a dynamic state and is difficult to stabilize. In addition, time consumption in the open network is more than that in the closed network. This is because that, in the open network, at every round, new agents join into the network and these new agents have to start to build some relations with existing agents, so this process costs some extra time. Similarly, existing agents leaving may result in their neighbors reforming relations with other agents, which spends extra time as well. Contrarily, in the closed network, no agents join or leave, so the relevant time consumption can be saved.

Finally, we are also interested in whether the introduction of weighted relations could bring some performance improvement compared with the crisp relations in the open agent network. We, therefore, compare the *Learn-Adapt* mechanism with *Crisp Learn-Adapt*, which considers only crisp relations by setting $\mu_{ij}=\mu_{ji}=1$ if there is a relation between a_i and a_j otherwise $\mu_{ij}=\mu_{ji}=0$. The experimental result is shown in Fig. 5 with *Resource Probability*, $RP=0.6$. It can be seen that, by introducing weighted relations in the network, the performance of *Learn-Adapt* is improved. This can be explained by the fact that, for *Learn-Adapt*, agents could still keep some neighbors, which cannot complete enough subtasks in one simulation run, through weakening the *relation strength*, but these neighbors might do better work in the next simulation run since future tasks may require the resources, which are possessed by those incompetent neighbors. For *Crisp Learn-Adapt*, however, agents may directly cut those incompetent neighbors and lose future task allocation opportunities.

In summary, the performance of our *Learn-Adapt* is around 85%–95% of the upper bound centralized allocation method, and on average 10% better than *K-Adapt*, which demonstrates that our mechanism is better than the *K-Adapt* mechanism in some aspects. In addition, we also compared *Learn-Adapt* with its two simplified versions, i.e., *Learn-Adapt without Trust* and *Crisp Learn-Adapt* in order to demonstrate the importance of the two introduced concepts, namely *trust model* and *weighted relation*. Furthermore, we find that, by integrating the trust model, the time consumption is lowered significantly, which again reveals that the trust model is crucial for this good performance.

6. A potential application scenario

In this section, we present a potential application scenario of our self-organization mechanism, i.e., packet routing in wireless sensor networks.

Wireless sensor networks are those networks, which are connected by embedded sensors, actuators and processors in a wireless and ad hoc manner [38]. This combination of wireless and data networking will result in a new form of computational paradigm, which is more

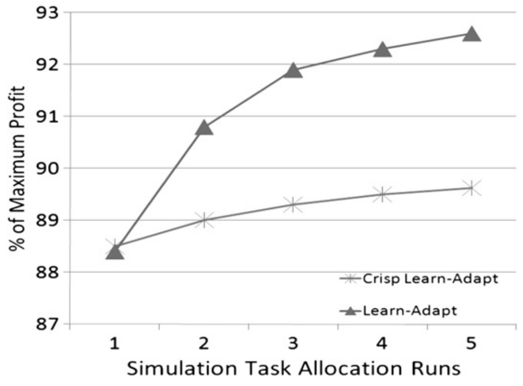


Fig. 5. Relative profit of *Learn-Adapt* and *Crisp Learn-Adapt*.

communication-centric than any other computer network. Wireless sensor networks are part of a growing collection of information technology constructs, which are moving away from the traditional desktop wired network architecture towards a more ubiquitous and universal mode of information connectivity. Wireless sensor networks can be used for combat field surveillance, intrusion detection, widespread environmental sampling and health monitoring. In this paper, a wireless sensor network refers to a group of sensors, or nodes, linked by a wireless medium, which could be achieved by using infrared devices or radio.

One of the most important issues of wireless sensor network design is packet routing protocols, which aims to deliver data to destinations effectively and efficiently. Currently, many researchers have made venerable efforts on this issue. Akkaya and Younis [2] surveyed and classified recent wireless sensor networks routing protocols. They classified the routing protocols into three categories as follows.

1. Data centric protocols, where nodes send queries to certain regions and wait for data from the sensors located in the selected regions. SPIN [14] is the first data centric protocol, which considers data negotiation between nodes in order to eliminate redundant data and save energy.
2. Hierarchical protocols. Since a single-tier network can cause the gateway to overload with an increase in sensor density, network clustering has been pursued in some routing approaches. The main aim of hierarchical routing is to efficiently maintain the energy consumption of sensor nodes by involving them in multi-hop communication within a particular cluster and by performing data aggregation and fusion in order to decrease the number of transmitted messages. LEACH [13] is one of the first hierarchical routing approaches for sensor networks. The idea proposed in LEACH is to form clusters of the sensor nodes based on the received signal strength and use local cluster heads as routers to transmit messages.
3. Location-based protocols. In most cases, location information is necessary in order to calculate the distance between two particular nodes so that the distance between two particular nodes can be estimated. Geographic Adaptive Fidelity (GAF) [45] is an energy-aware location-based routing protocol devised primarily for mobile ad hoc network, but is also applicable to sensor networks. It forms a virtual grid for the covered area and each node uses its GPS-indicated location to associate itself with a point in the virtual grid. Then, the distance between two nodes can be calculated according to their GPS coordinates.

Each of these routing protocols has both advantages and disadvantages. It is nearly impossible to develop a perfect routing protocol, which could overcome all the disadvantages of other protocols, because some attributes of a routing protocol conflict with other attributes, e.g., energy saving versus delivering successful ratio. Thus, researchers usually focus their design primarily on some specific attributes, e.g., network throughput, delivering successful ratio, communication overhead, packet delay, etc. With this motivation, we intend to employ our self-organization mechanism to enhance current routing protocols, instead of proposing a novel and specific protocol. The concept of self-organization has been introduced into wireless sensor networks for several years. For instance, Karnik and Kumar [23] introduced self-organization into wireless sensor networks for optimizing network throughput via building an optimal topology and tuning network access parameters, such as the transmission attempt rate. Nonetheless, topology of wireless sensor networks sometimes cannot be altered arbitrarily, such as environmental sampling sensor networks where the interested areas are usually fixed and thereby, the topology cannot be adapted arbitrarily.

Within this constraint, we propose a two-layer architecture to improve the routing performance, where the first layer is the wireless sensor network while the second layer is a cooperation network as shown in Fig. 6. In the first layer, sensors, represented as agents, are connected by some wireless medium as described earlier, so this layer is a physical network. In the second layer, agents are linked by weighted cooperation relations as described in Section 1, so this layer is an abstract network.

The cooperation network is formed through the agents past cooperation. For example, in Fig. 6, if many packets sent from agent 1 have to be forwarded by agent 7, agent 1 will add agent 7 as one of its cooperation neighbors and the relation could be assigned as the *subordinate–superior* relation. Then, in the future, if agent 1 has packets to be sent, it sends the packets directly to agent 7 and agent 7 then, forwards the packets to the destinations for agent 1. For the process that agent 1 sends packets to agent 7 and agent 7 forwards packets to the destinations, agents 1 and 7 can use any existing routing protocols. Thus, it can be seen that the cooperation network layer is used only to guide the packet routing process in the wireless sensor network layer in order to improve the routing efficiency, rather than becoming a concrete routing protocol which directly operates in the wireless sensor network layer. Thus, in this case, any of the current routing protocols can be employed in the wireless sensor network layer. Here, our self-organization mechanism works on the cooperation network layer, which enables each agent to keep the most useful cooperation neighbors and which, in turn, assists to achieve better routing performance. In this scenario, each packet can be modeled as a token as described in Section 2, and the agents, which can forward many tokens, are considered to be useful. We consider that this idea may be promising, as it could probably enhance many current routing protocols. Currently, we are engaged in this work by building theoretical model and we will test it within a network simulator once the theoretical work is done.

7. Conclusion

This paper introduced a composite self-organization mechanism which aims to adapt relations among agents in a network to achieve efficient task allocation. This mechanism integrates a trust model, a multi-agent Q-learning algorithm and weighted relation concept, which together resulted in good performance compared with another famous self-organization mechanism, *K-Adapt*. Since this mechanism is decentralized and continuous over time, it meets the principles of self-organization defined by Serugendo et al. [34]. In addition, this paper gave a potential application scenario of the proposed self-organization mechanism, which demonstrates that our self-organization mechanism could be applied in some real world cases.

Since the trust model and learning algorithm are complementary parts of our mechanism, a more effective trust model and a more efficient learning algorithm could improve the performance of our mechanism. Thus, devising impactful trust models and learning algorithms is a stream of future work to refine our mechanism. Specifically, we will quantitatively compare different trust models for candidate selection to find which one is the best for our problem, and we will develop a new trust model by ourselves based on the analysis of existing trust models. Furthermore, we intend to enhance the self-organization mechanism by enabling it to handle dynamic cases, where new types of resources are introduced into the agent network and some existing resources are phased out. It is also interesting to introduce the *asymmetric relation strength* into our agent network model, e.g., an agent, say A, quite likes another agent, say B, with the degree 0.7 but B does not very like A with the degree only

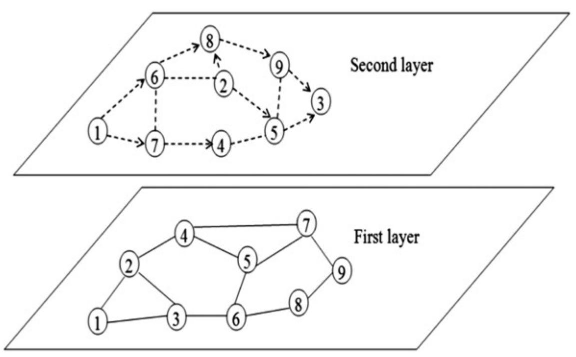


Fig. 6. A sample two-layer architecture.

0.3. We believe that introducing such *asymmetric relation strength* will make our self-organization mechanism more flexible and more suitable to real-world problems. Finally, as stated in the previous section, we will apply our self-organization mechanism to enhance packet routing in wireless sensor networks.

References

[1] S. Abdallah, V. Lesser, Multiagent reinforcement learning and self-organization in a network of agents, AAMAS'07, Honolulu, Hawai'i, 2007, pp. 172–179.
[2] K. Akkaya, M. Younis, A survey on routing protocols for wireless sensor networks, Ad Hoc Networks 3 (2005) 325–349.
[3] C. Bernon, V. Chevrier, V. Hilaire, P. Marrow, Applications of self-organising multi-agent systems: an initial framework for comparison, Informatica 30 (2006) 73–82.
[4] L. Bongaerts, Integration of scheduling and control in holonic manufacturing systems, Ph.D. thesis, Katholieke Universiteit Leuven, 1998.
[5] E. Bou, M. Lopez-Sanchez, J.A. Rodriguez-Aguilar, Selfconfiguration in autonomic electronic institutions, Coordination, Organization, Institutions and Norms in Agent Systems Workshop at ECAI'06, 2006.
[6] T.H.D. Weyns, K. Schelfthout, O. Glorieux, A role based model for adaptive agents, Fourth Symposium on Adaptive Agents and Multi-Agent Systems at AISB'04, 2004.
[7] T. DeWolf, T. Holvoet, Towards autonomic computing: agent-based modelling, dynamical systems analysis, and decentralised control, in: Proceedings of the First International Workshop on Autonomic Computing Principles and Architectures, Banff, Canda, 03, pp. 10–20.
[8] A. Galstyan, K. Czajkowski, K. Lerman, Resource allocation in the grid using reinforcement learning, AAMAS'04, New York, NY, 2004, pp. 1314–1315.
[9] M.E. Gaston, M. desJardins, Agent-organized networks for dynamic team formation, AAMAS'05, Utrecht, Netherlands, 2005, pp. 230–237.
[10] R. Glinton, K. Sycara, P. Scerri, Agent organized networks redux, AAAI'08, Chicago, 2008, pp. 83–88.
[11] E.R. Gomes, R. Kowalczyk, Dynamic analysis of multiagent q-learning with e-greedy exploration, ICML'09, Montreal, Canada, 2009, pp. 369–376.
[12] N. Griffiths, M. Luck, Changing neighbours: improving tag-based cooperation, AAMAS'10, Toronto, Canada, 2010, pp. 249–256.
[13] W. Heinzelman, A. Chandrakasan, H. Balakrishnan, Enerty efficient communication protocol for wireless sensor networks, Proc. of the Hawaii International Conference System Sciences, Hawaii, 2000, pp. 3005–3014.
[14] W. Heinzelman, J. Kulik, H. Balakrishnan, Adaptive protocols for information dissemination in wireless sensor networks, Proc. of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, Seattle, WA, 1999, pp. 174–185.
[15] R. Hermoso, H. Billhardt, S. Ossowski, Role evolution in open mas as an information source for turst, AAMAS'10, Canada, 2010, pp. 217–224.
[16] M. Hoogendoorn, Adaptation of organizational models for multi-agent systems based on max flow networks, IJCAI'07, Hyderabad, India, 2007, pp. 1321–1326.
[17] B. Horling, B. Benyo, V. Lesser, Using self-diagnosis to adapt organizational structures, AGENTS'01, Montreal, Quebec, Canada, 2001, pp. 529–536.
[18] J.F. Hubner, J.S. Sichman, O. Boissier, Using the moise+ for a cooperative framework of mas reorganisation, Proc. of the 17th Brazilian Symposium on AI, Sao Luis, Maranhao, Brazil, 2004, pp. 506–515.
[19] T.D. Huynh, N.R. Jennings, N.R. Shadbolt, Certified reputation: how an agent can trust a stranger, AAMAS'06, Hakodate, Japan, 2006, pp. 1217–1224.
[20] A. Josang, R. Ismail, C. Boyd, A survey of trust and reputation systems for online service provision, Decision Support Systems 43 (2007) 618–644.
[21] L.P. Kaelbling, M.L. Littman, A.W. Moore, Reinforcement learning: a survey, Journal of Artificial Intelligence Research 4 (1996) 237–285.
[22] S. Kamboj, K.S. Decker, Organizational self-design in semi-dynamic environments, AAMAS'07, Honolulu, Hawai'i, USA, 2007, pp. 1228–1235.
[23] A. Karnik, A. Kumar, Distributed optimal self-organization in ad hoc wireless sensor networks, IEEE/ACM Transactions On Networking 15 (2007) 1035–1045.
[24] H. Karuna, P. Valckenaers, B. Saint-Germain, P. Verstraete, C.B. Zamfirescu, H.V. Brussels, Emergent forecasting using a stigmergy approach in manufacturing coordination and control, Engineering Selforganising Systems 3464 (2005) 210–226.
[25] H. Kautz, B. Selman, A. Milewski, Agent amplified communication, AAAI'96, Portland, 1996, pp. 3–9.
[26] J.O. Kephart, D.M. Chess, The vision of autonomic computing, IEEE Computer 36 (2003) 41–50.
[27] R. Kota, N. Gibbins, N.R. Jennings, Self-organising agent organisations, AAMAS'09, Budapest, Hungary, 2009, pp. 797–804.
[28] R. Kota, N. Gibbins, N.R. Jennings, Decentralised approaches for self-adaptation in agent organisations, ACM Trans. on Autonomous and Adaptive Systems in press.
[29] G. Lampi, Self-organizing Maps in Decision Support: A Decision Support System Prototype, Master's thesis, Helsinki University of Technology, 2009.
[30] M. Mamei, M. Vasirani, F. Zambonelli, Self-organising spatial shapes in mobile particles: the tota approach, Engineering Selforganising Systems 3464 (2005) 138–153.
[31] R. Misdolea, Decision support system and customer relationship management as components of the cybernetic system enterprise, Informatica Economica 14 (2010) 201–207.
[32] A. Reitbauer, A. Battino, A. Karageorgos, N. Mehandjiev, P. Valckenaers, B. Saint-Germain, The mabe middleware: extending multi-agent systems to enable open business collaboration, 6th IFIP International Conference on Information Technology for Balanced Automation Systems in Manufacturing and Services, 2004, pp. 1–10.
[33] G.D.M. Serugendo, M.P. Gleizes, Self-organisation and emergence in MAS: an overview, Informatica 30 (2006) 45–54.
[34] G.D.M. Serugendo, M.P. Gleizes, A. Karageorgos, Self-organization in multi-agent systems, The Knowl. Engin. Review 20 (2005) 165–189.
[35] M. Sims, D. Corkill, V. Lesser, Automated organization design for multi-agent systems, JAAMAS 16 (2008) 151–185.
[36] F. Smarandache, J. Dezert, Advances and Applications of DSmT for information Fusion, America Research, 2004.
[37] A. Smirnov, T. Levashova, N. Shilov, A. Kashevnik, Hybrid technology for self-organization of resources of pervasive environment for operational decision support, International Journal on Artificial Intelligence Tools 19 (2010) 211–229.
[38] K. Sohrabi, J. Gao, V. Ailawadhi, G.J. Pottie, Protocols for selforganization of a wireless sensor network, IEEE Personal Communications 7 (2000) 16–27.
[39] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.
[40] R.S. Sutton, D. Precup, S.P. Singh, Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning, AIJ 112 (1999) 181–211.
[41] W.T.L. Teacy, J. Patel, N.R. Jennings, M. Luck, Travos: trust and reputation in the context of inaccurate information sources, JAAMAS 12 (2006) 183–198.
[42] J. Vazquez-Salceda, V. Dignum, F. Dignum, Organizing multiagent systems, JAAMAS 11 (2005) 307–360.
[43] J. Wang, H. Sun, A new evidential trust model for open communities, Computer Standards and Interfaces 31 (2009) 994–1001.
[44] D.J. Watts, S.H. Strogatz, Collective dynamics of 'small-world' networks, Nature 393 (1998) 440–442.
[45] Y. Xu, J. Heidemann, D. Estrin, Geography-informed energy conservation for ad hoc routing, Proc. of the 7th Annual ACM/IEEE International Conference on Mobile Computing and Networking, Rome, Italy, 2001, pp. 70–84.
[46] Y. Xu, P. Scerri, B. Yu, S. Okamoto, M. Lewis, K. Sycara, An integrated token-based algorithm for scalable coordination, AAMAS'05, Utrecht, 2005, pp. 407–414.
[47] L. Zhang, S. Ren, Z. Liu, Self-organization based decision support system framework, IEEE International Conference on Systems, Man, and Cybernetics, Nashville, TN, USA, 2000, pp. 609–614.

Mr Dayong Ye is a 2nd year PhD student at the University of Wollongong. His research interests focus on multi-agent systems, self-organized systems, task allocation protocols in distributed systems and agent-based modeling in complex domains.

Dr Minjie Zhang is an Associate Professor in the School of Computer Science and Software Engineering and the Director of Intelligent System Research Group in the Faculty of Informatics, at University of Wollongong, Australia. Her research interests include multi-agent systems, agent-based simulation and modeling in complex domains, agent-based grid computing, and knowledge discovery and data mining.

Dr. Danny Sutanto is currently the Professor of Power Engineering at the School of Electrical, Computer and Telecommunication Engineering at the University of Wollongong, Australia. His research interests include energy storage systems and voltage stability.