A Composite Self-organisation Mechanism in an Agent Network

Dayong Ye[1], Minjie Zhang[1] and Quan Bai[2]

[1]University of Wollongong, Wollongong, Australia
[2]Auckland University of Technology, Auckland, New Zealand
Email: dy721@uowmail.edu.au, minjie@uow.edu.au, quan.bai@aut.ac.nz

Abstract. Self-organisation provides a suitable paradigm for developing autonomic web-based applications, e.g., e-commerce. Towards this end, in this paper, a composite self-organisation mechanism in an agent network is proposed. Based on self-organisation principles, this mechanism enables agents to dynamically adapt relations with other agents, i.e., change the underlying network structure, to achieve efficient task allocation. The proposed mechanism integrates 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.

1 Introduction

Nowadays, more and more web-based applications emerge, e.g., e-commerce. These applications have high desirability to be autonomic which are capable of self-management, because self-management applications can save labour time of human managers, are able to adapt to environmental changes and ensure their own survivability. Within this context, De Wolf and Holvoet [1] recommended that agent-based modeling is best suited to build such autonomic applications. Thus, based on De Wolf and Holvoet's recommendation, we consider that self-organising multi-agent systems are good choices for developing such autonomic web-based applications, as the self-organising applications can continuously arrange and rearrange their organisational structures autonomously to adapt to environmental changes without external control. In addition, the adaptation process should be performed in a decentralised manner, so that the autonomic applications could be robust against failures of any nodes in the environments. Self-organisation, which is defined as "*the mechanism or the process enabling the system to change its organisation without explicit external command during its execution time* (Serugendo et al. [8])", can be employed in agent networks to improve the cooperative behaviours of agents. Mathieu et al. [7] provided three principles for self-organisation agent networks design, which include (1) creation of new specific relations between agents in order to remove the middle-agents, (2) exchange of skills between agents to increase autonomy, and (3) creation of

new agents to reduce overloading. In this paper, our contribution focuses on the first principle, i.e., adaptation of existing relations between agents to achieve a better allocation of tasks in distributed environments.

Currently, research on self-organisation mechanisms in multi-agent and agent-based complex systems has produced results of significance. Some works are centralised in nature and have the potential of the single point of failure, e.g., [4]. Self-organisation mechanisms focusing on network structural adaptation (i.e., adapting relations among agents) have also been investigated by several researchers, such as [3]. However, these network structural adaptation methods assumed that only one type of relation exists in the network and the number of neighbours 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. [6] 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 maximise the utility at each step. Nevertheless, as stated by Kaelbling et al. [5], this kind of algorithm, which always takes the highest utility action, overlooked the tradeoff between exploitation and exploration and may finally converge to a sub-optimal state with a self-organisation process continuing.

Besides the disadvantages mentioned above, the common limitation of current related research is that candidate selection for self-organisation, i.e., relation adaptation, is simplified. For candidate selection, current related works have agents use only their own experience. In addition, current self-organisation 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 introduced 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 the two agents. The introduction of weighted relation into self-organisation 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 propose a composite self-organisation mechanism. This self-organisation mechanism consists of three elements, which are claimed as a three-fold contribution of this paper. (1) For candidate selection, we integrate a trust model which lets agents use not only their own experience but also other agents' opinions to select candidates. (2) For adapting multiple relations, we develop a multi-agent Q-learning algorithm which enables two agents to independently evaluate their rewards about changing relations and balances exploitation and exploration. Consequently, our mechanism could overcome the aforementioned flaws of Kota et al.'s mechanism. (3) We also introduce weighted relations into our self-organisation mechanism. The introduction of weighted re-

lations can improve the performance of our self-organisation mechanism and make the mechanism more suitable in dynamic environments.

2 The Agent Network Model

In this section, an agent network model is presented in which we will develop our self-organisation mechanism. The aim of the agent network is to allocate tasks to agents such that the communication cost among agents is minimised and the benefit obtained by completing tasks is maximised. Each task, Φ, is composed of a set of subtasks, i.e., $\Phi = \{\varphi_1, ..., \varphi_m\}$. 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 subtask has a preset deadline as well, which should be satisfied. Otherwise, the benefit will be decreased gradually with time elapse till 0. 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 requirement of the corresponding subtask, but also the token traveling path which is composed of those agents that the token has passed.

In our model, an agent network comprises a set of collaborative agents, i.e., $A = \{a_1, ..., a_n\}$, 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. A *peer-to-peer* relation, denoted as "$\overset{\mu}{\sim}$" ($\overset{\mu}{\sim} \subseteq A \times A$), is a *Compatible Relation*, which is reflexive and symmetric, such that $\forall a_i \in A : a_i \overset{\mu_{ii}}{\sim} a_i$ and $\forall a_i, a_j \in A : a_i \overset{\mu_{ij}}{\sim} a_j \Rightarrow a_j \overset{\mu_{ji}}{\sim} a_i$, where $\mu_{ij} = \mu_{ji}$. A *subordinate-superior* relation, written as "$\overset{\mu}{\prec}$" ($\overset{\mu}{\prec} \subseteq A \times A$), is a *Strict Partial Ordering Relation*, which is irreflexive, asymmetric and transitive, such that $\forall a_i \in A : \neg(a_i \overset{\mu_{ii}}{\prec} a_i)$, $\forall a_i, a_j \in A : a_i \overset{\mu_{ij}}{\prec} a_j \Rightarrow \neg(a_j \overset{\mu_{ji}}{\prec} a_i)$ and $\forall a_i, a_j, a_k \in A : a_i \overset{\mu_{ij}}{\prec} a_j \wedge a_j \overset{\mu_{jk}}{\prec} a_k \Rightarrow a_i \prec a_k$. The notation, μ_{ij}, is *relation strength*, which indicates how strong the relation is between agents a_i and a_j. μ_{ij} is ranged from $[0, 1]$, where 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 neighbouring agents is set to 0.5 by default, but it might be adapted during the succeeding task allocation process.

In the agent network, each agent is of the form $a_i = \langle Res_i, Comp_i \rangle$, where Res_i is the set of resources possessed by a_i and $Comp_i$ is the computation capacity of a_i for completing tasks. Moreover, the knowledge of each agent is modeled as a tuple $\langle Neig_i(t), Act_i(t), Tokens_i(t) \rangle$. The first element, $Neig_i(t)$, is the neighbour set of agent a_i at time t. $Neig_i(t)$ can be further divided into three subsets, $Neig_i^{\sim}(t)$, $Neig_i^{\prec}(t)$ and $Neig_i^{\succ}(t)$. $Neig_i^{\sim}(t)$ contains the *peer*s of a_i, $Neig_i^{\prec}(t)$ consists of the direct *superior*s of a_i, and $Neig_i^{\succ}(t)$ comprises of the direct *subordinate*s of a_i. The second element in the agent knowledge tuple, $Act_i(t)$, is the action set of agent a_i at time t. An action set, $Act_i(t)$, is defined

as a set of available actions for agent a_i at time t, while an *action* is defined as a decision made by an agent to adapt the relation with another agent. There are seven different atomic actions defined in our model, which are $enh\_\sim$, $enh\_\prec$, $enh\_\succ$, $wkn\_\sim$, $wkn\_\prec$, $wkn\_\succ$ and *no_action*. It should be noted that the meanings of actions $enh\_$ and $wkn\_$ imply not only *enhance* and *weaken* but also *form* and *dissolve*, respectively. The atomic actions can also be combined together. The meanings of combination actions can be easily deduced from the meanings of atomic actions.

It should be noticed that an agent at different time steps 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 $enh\_\sim$, $enh\_\prec$, $enh\_\succ$ and *no_action*. (2) a_i is a *peer* of a_j, i.e., $a_i \sim a_j$. The possible actions involve $wkn\_\sim$, $wkn\_\sim +enh\_\prec$, $wkn\_\sim +enh\_\succ$ and *no_action*. (3) a_i is a *subordinate* of a_j, i.e., $a_i \prec a_j$. The possible actions include $wkn\_\prec$, $wkn\_\prec +enh\_\sim$, $wkn\_\prec +enh\_\succ$ and *no_action*. These actions are based on a_i's perspective, while, in a_j's view, a_j needs to reverse these actions. (4) a_i is a *superior* of a_j, i.e., $a_i \succ a_j$. This situation is the reverse condition of $a_i \prec a_j$.

The last element in the agent knowledge tuple, $Tokens_i(t)$, stores not only the tokens agent a_i currently holds at time t but also the previous tokens incoming and outgoing through a_i. With time elapse, old tokens will be automatically deleted from the set $Tokens_i(t)$. Furthermore, 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, although the agent might have no idea exactly which *subordinate* owns which resource. 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., that 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 passes the token to the agent which has higher *relation strength* with a_i.

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

3 The composite Self-organisation Mechanism

Before devising the self-organisation 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.

3.1 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 neighbourhood relations with other agents. Due to the page limitation, the detailed calculation of each attribute of $Cost_{Net}$ cannot be presented.

The benefit of the network, $Benefit_{NET}$, is the sum of benefits obtained by all the agents in the network. The benefit of each agent depends on how many subtasks are completed by that agent. As depicted in Section 2, each task Φ contains several subtasks, $\varphi_1, \varphi_2, ...$, represented as tokens Δ_1, Δ_2.... When a subtask φ_i is successfully completed by an agent, that agent would obtain the corresponding benefit.

Finally, the profit of the entire network, $Profit_{NET}$, is:

$$Profit_{NET} = Benefit_{NET} - Cost_{NET} \tag{1}$$

3.2 Self-organisation Mechanism Design

As described in Section 1, 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 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-organisation mechanism is now illustrated in the following subsections.

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) [9]. Other trust models may also be available for our problem, but, through our investigation, Dezert-Smarandache theory is more suitable for our requirements, 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. 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

$$\begin{cases} m(\emptyset) = 0, \\ \sum_{B \subseteq H^{\Theta}} m(B) = 1. \end{cases} \tag{2}$$

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, who 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. For example, if a_j completed many tasks for a_i on time, then the value of $m_j^i(\{T\})$ may increase. In addition, a_i might ask one of its neighbouring 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

$$T_j^i = T_j^i \oplus T_j^k, \tag{3}$$

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

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

$$T_j^k = T_l^k \otimes T_j^l, \tag{4}$$

where $m_j^k(\{T\}) = (m_l^k(\{T\})) + \beta m_l^k(\{T \cap \neg T\}) \cdot m_j^l(\{T\})$
$m_j^k(\{\neg T\}) = (m_l^k(\{\neg T\})) + \beta m_l^k(\{T \cap \neg T\}) \cdot m_j^l(\{\neg T\})$
$m_j^k(\{T \cap \neg T\}) = (m_l^k(\{T \cap \neg T\})) + \beta m_l^k(\{T \cap \neg T\}) \cdot m_j^l(\{T \cap \neg T\})$
$m_j^k(\{\Theta\}) = 1 - m_j^k(\{T\}) - m_j^k(\{\neg T\}) - m_j^k(\{T \cap \neg T\})$,
and β is a constant which is in the range $(0, 1)$.

The candidate selection process can be simply summarised as follows. After a period, each agent, say a_i, first evaluates the trust of those agents ($TempCands_i$) which completed many or few tasks for it during the last period. Then, a_i asks the opinions of other agents against those agents in $TempCands_i$ and, afterwards, adjusts the trust values based on those opinions. Finally, a_i filters the agents in $TempCands_i$ and makes a new group of agents called $Candidates$ with which a_i will adapt relations.

Relation Adaptation Before describing our relation adaptation algorithm, we 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 1.

Each cell $(r_i^{x,y}, r_j^{x,y})$ in Table 1 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. The reward of each agent, r_i, is

Table 1. Reward Matrix of a_i and a_j

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

based on how much load could be reduced on agent a_i and 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, written as $\Delta.path$. For example, agent a_i has no relation with agent a_j, but a_j completed many subtasks for a_i during former task allocation processes. a_i, then, might want to establish a relation with a_j. If a_i would like to form the *superior-subordinate* relation with a_j, i.e. performing the action $enh\_\succ$ (for a_j, performing the reverse action $enh\_\prec$), the management cost on both a_i and a_j will rise because both a_i and a_j have to maintain a new neighbour. Nevertheless, those agents, which are in the $\Delta.path$, could save communication cost (which are considered as a_j's reward), since they do not need to pass tokens between a_i and a_j any more. Here, Δ refers to the tokens that are held by a_j and sent by a_i. In addition, a_i could save management cost for maintaining subtasks, as a_i can directly pass tokens to a_j without waiting for intermediate agents to pass tokens and, hence, a_i's subtasks could be allocated in less time steps. The potential benefit which would be obtained by a_j is evaluated on the basis of the benefit a_j gained for completing the subtasks assigned by a_i, while the potential benefit of a_i is calculated in an analytical way. We suppose that the action $enh\_\prec$, with a_i as the *superior* and a_j as the *subordinate*, can make a_j potentially receive more subtasks from a_i and then get more benefits. **Algorithm 1** demonstrates our relation adaptation approach in pseudocode form.

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 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 initialised 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 ϵ-greedy exploration method devised by Gomes and Kowalczyk [2] shown in Equation 5, where $0 < \epsilon < 1$ is a small positive number and n is the number of available actions of a_i.

$$\pi_{ix} = \begin{cases} (1-\epsilon) + (\epsilon/n), \text{if } Q_{ix} \text{ is the highest} \\ \epsilon/n \text{ , otherwise} \end{cases} \tag{5}$$

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. In Line 12, $match(x, y)$ is a function which is used to test whether the actions x and y that are taken by a_i and a_j, respectively, are matched. An action is only matched by its reverse action described in Section 2. Therefore, a_i and a_j have to cooperate to find the actions, which can be matched together and maximise the sum of their Q-values. Finally, a_i and a_j adjust their *relation strength* (Lines 14-17). The adjustment

Algorithm 1: Relation adaptation according to a_i

```
1  for each a_j ∈ Candidates do
2      Act_i ← available_actions(a_i, a_j);
3      Act_j ← available_actions(a_i, a_j);
4      for each x ∈ Act_i, y ∈ Act_j do
5          Initialise Q_ix and Q_jy arbitrarily;
6          for k = 0 to a predefined integer do;
7              calculate π_ix(k) and π_jy(k);
8              Q_ix(k+1) = Q_ix(k)+
                 π_ix(k)α(Σ_y r_i^{x,y} π_jy(k) − Q_ix(k));
9              Q_jy(k+1) = Q_jy(k)+
                 π_jy(k)α(Σ_x r_j^{x,y} π_ix(k) − Q_ix(k));
10         end for
11     end for
12     ⟨x_opti, y_opti⟩ ← argMax_{match(x,y)}(Q_ix + Q_jy);
13     a_i, a_j take actions x_opti and y_opti, respectively;
14     μ_ij ← μ_ij + (N_j^i/ρ − 1);
15     if μ_ij > 1 then μ_ij ← 1;
16     if μ_ij < 0 then μ_ij ← 0;
17     μ_ji ← μ_ij;
18 end for
```

depends on how many subtasks that a_j completed for a_i in the last time steps, where the more subtasks are completed by a_j, the higher *relation strength* value is achieved. In Line 14, N_j^i means the number of a_i's subtasks completed by a_j and ρ is a threshold which is an integer.

Due to the page limitation, the experimental results cannot be presented here.

References

1. DeWolf, T., Holvoet, T.: Towards autonomic computing: agent-based modelling, dynamical systems analysis, and decentralised control. In: the First Intern. Workshop on Auton. Comput. Princip. and Archit. pp. 10–20. Banff, Canda (2003)
2. Gomes, E.R., Kowalczyk, R.: Dynamic analysis of multiagent q-learning with e-greedy exploration. In: ICML'09. pp. 369–376. Montreal, Canada (Jun 2009)
3. Griffiths, N., Luck, M.: Changing neighbours: Improving tag-based cooperation. In: AAMAS'10. pp. 249–256. Toronto, Canada (May 2010)
4. Hermoso, R., Billhardt, H., Ossowski, S.: Role evolution in open mas as an information source for turst. In: AAMAS'10. pp. 217–224. Canada (May 2010)
5. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research 4, 237–285 (1996)
6. Kota, R., Gibbins, N., Jennings, N.R.: Self-organising agent organisations. In: AAMAS'09. pp. 797–804. Budapest, Hungary (May 2009)
7. Mathieu, P., Routier, J.C., Secq, Y.: Principles for dynamic multi-agent organizations. In: PRIMA'02. pp. 109–122. Tokyo, Japan (Aug 2002)
8. Serugendo, G.D.M., Gleizes, M.P., K., A.: Self-organization in multi-agent systems. The Knowl. Engin. Review 20(2), 165–189 (2005)
9. Smarandache, F., Dezert, J.: Advances and Applications of DSmT for information Fusion. America Research (2004)