Modeling and Evaluation of Trusts in Multi-Agent Systems

GUO LEI

(*B. ENG. XI'AN JIAO TONG UNIVERSITY*)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF INDUSTRIAL & SYSTEMS ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2007

ACKNOWLEDGEMENT

First of all, I would like to express my sincere appreciation to my supervisor, Associate Professor Poh Kim Leng for his gracious guidance, a global view of research, strong encouragement and detailed recommendations throughout the course of this research. His patience, encouragement and support always gave me great motivation and confidence in conquering the difficulties encountered in the study. His kindness will always be gratefully remembered.

I would like to express my sincere thanks to the National University of Singapore and the Department of Industrial & Systems Engineering for providing me with this great opportunity and resource to conduct this research work.

Finally, I wish to express my deep gratitude to my parents, sister，brother and my husband for their endless love and support. This thesis is dedicated to my parents.

TABLE OF CONTENTS

SUMMARY

In most real situations, agents are often required to work in the presence of other agents, either artificial or human. These are examples of multi-agent systems (MAS). In MAS, agents adopt cooperation strategy to increase their utilities and they have incentives to tell the truth to other agents. However, when competition occurs, they have incentives to lie. Thus, the decision on which agents to cooperate with is a problem which has attracted a lot of attention. In order to overcome the uncertainties in open MAS, researchers have introduced the concept of "trust" into these systems. The trust evaluation becomes a popular research topic in the multi-agent systems.

Based on the existing trust evaluation mechanisms, we proposed a novel mechanism to help agents evaluate the trust value of the target agent in the multi-agent systems. We present an approach to help agents construct a trust network automatically in a multi-agent system. Although this network is a virtual one, it can be used to estimate the trust value of a target agent. After the construction of the trust network, we use the Bayesian Inference Propagation approach with Leaky Noisy-Or model to solve the trust graph. This is a novel way to solve the trust problem in the multi-agent systems. This approach solves the trust estimation problem based on objective logic which means that there is no subjective setting of weights. The whole trust estimation process is automatic without the intervention of human beings. The experiments carried out by our simulation work demonstrate that our model works better than the models proposed by other authors. By using our model, the whole agents' utility

gained is higher than by using other models (MTM and without trust measure). In addition, our model performs well in a wide range of provider population and it also reconfirmed the fact that our model works well than the models we compared. Moreover, we also demonstrate that more information resource can help the decision maker make a more accurate decision. Last but not least, in the dynamic environment, and the experiment results also demonstrate that our model performs better than the models we compared with.

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Background

Internet makes the geographical and social unrelated communication come true in a twinkle. It enables a transition to peer-to-peer commerce without intermediaries and central institutions. However, online communities are usually either goal or interest-oriented and there is rarely any other kind of bond or real life relationship among the members of communities before the members meet each other online [Zacharia, 1999]. Without prior experience and knowledge about each other, peers are under the risk of facing dishonest and malicious behaviors in the environment. Take the peers as agents, this environment can be seen as a multi-agent system. Large numbers of research have been done to manage the risk of deceit in the Multi-agent Systems. One way to address this uncertainty problem is to develop strategies for establishing trust and developing systems that can assist peers in assessing the level of trust they should place on an eCommerce transaction [Xiong and Liu, 2004].

Traditional trust construction relies on the use of a Central Trusted Authority or trusted third party to manage trust, such as access control list, role-based access control, PKI, etc. [Kagal et al., 2002]. However, in an open Multi-agent system, there are some specific requirements [Despotovic and Aberer, 2006]: (1) The environment is open. The users in this environment are autonomous and independent to each other.

(2) The environment is decentralized. There is no central point in this system and the users are free to trust others. (3) The environment is global. There is no jurisdictional border in the environment. Thus, in the open Multi-agent System, the central trust mechanism cannot satisfy the requirement of mobility and dynamics. These issues have motivated substantial research on trust management in open Multi-agent Systems. Trust management helps to maintain overall credibility level of the system as well as to encourage honest and cooperative behavior.

1.2 Motivations

As traditional trust mechanisms have their disadvantages, this issue has motivated substantial research on Trust Management in MAS. There has been an extensive amount of research on online trust and reputation management [Marsh, 1994, Abdul-Rahman et al., 2000; Sabater, et al., 2002; Yu and Singh, 2002]. Among these research works, there are two ways to estimate the trustworthiness of a given agent, which are probabilistic estimation and social network. However, in the real online community, each agent not only relies on its own experience, but also on the reputation among the whole systems. Thus, how to estimate a given agent's trustworthiness under the direct experience and reputation becomes a new problem that needs to be solved.

1.3 Methodology

A Bayesian Network [Jensen, 1996, Charniak, 1991] is a graphical method of representing relationships among different variables that together define a model of a real-world situation. Formally, it is a Directed Acyclic Graph (DAG) with nodes being the variables and each directed edge representing dependence between two of them. Bayesian Networks are useful in inference from belief-structures and observations [Charniak, 1991 and AI 1999]. Bayesian Networks not only can readily handle incomplete data sets, but also offer a method of updating the belief or the probability of occurrence of the particular event for the given causes. In Bayesian Networks, the belief can be updated by network propagation method and each node has the task of combining incoming evidence and outputting some aggregation of the inputs.

The noisy-OR model is the most accepted and widely applied model to solve the multi-causal interactions network and it leads to a very convenient and widely applicable rule of combination. However, the noisy-OR model is based on two assumptions: accountability and exception [Pearl, 1988]. Accountability states that an event can be presumed false if all its parents are false. Exception requires that the influence of each parent on the child be independent of other parents.

1.4 Contributions

The objective of this research is to develop a trustworthiness estimation system and this dissertation proposes a novel approach among the trust management area.

In our trustworthiness estimation system, we solve the trust network by using Bayesian propagation method and Noisy-or model is used as well. First, based on historical interaction data, each agent constructs graphs to store two trust data which are functional trust and referral trust. When the estimation starts, the agent will check its functional trust data first, and after that, the agent will send requirement to its acquaintances to ask for recommendations. Then, a Trust Network would be constructed between the source agent and the target agent.

To solve the Trust Network, we firstly made some adjustment which is known as parallelization. Secondly, we use Bayesian Propagation to evaluate each chain in the parallelized Trust Network. Thirdly, the Noisy-or model is introduced to obtain the trustworthiness value of the target agent.

One important contribution of this dissertation is in applying Bayesian propagation method to solve the trustworthiness estimation problem. This application is the first time for the Bayesian Network methods to solve the Trust Network problem. It not only extends the application field of Bayesian Networks, but also solves the Trust Network in a novel way.

Another contribution is the derivation of a computational model based on sociological and biological understanding of trust management. Based on the strength of the software development, the introduction of Bayesian Propagation method makes the calculation of trustworthiness become easy and quick.

1.5 Organization of the Thesis

The next chapter presents a state-of-the-art survey of reputation-based trust management. Chapter 3 describes the storage of the data set and the Trust Network construction. Chapter 4 presents the process of trustworthiness evaluation. Chapter 5 proposes an experiment and the results. Chapter 6 briefly concludes this work and points to directions for future research opportunities.

2 LITERATURE REVIEW

2.1 Trust

In 1737, David Hume provides a clear description on the problem involving trust in his Treatise on Human Nature. We rely on trust everyday: we trust that our parents would support us, our friends would be kind to us, we trust that motorists on the road would follow traffic rules; we trust that the goods we buy have the quality commensurate with how much we pay for them, etc [Mui, 2002]. Trust is one of the most important factors in our human society. With the development of the computer technology in the past decades, trust construction in the virtual communities become more and more important.

2.1.1 What Is Trust?

In most real situations, agents are often required to work in the presence of other agents, which are either artificial or human. These are examples of multi-agent systems (MAS). In MAS, when agents adopt cooperation strategy to increase their utilities, they have incentives to tell the truth to other agents. Meanwhile, when competition occurs, they have incentives to lie. Thus, which agents to cooperate with is a problem which has attracted a lot of attention. In order to overcome the uncertainties in open MAS, researchers have introduced the concept “trust” into these systems.

As a research group leaded by Castelfranchi stated, trust is at the same time: a mental attitude towards another agent, a decision to rely on another and a behavior [Falcone et al., 2004].

- Trust as a mental attitude is most common in daily life, and is based on evaluation of past behavior, and on the expectation of future behavior.
- Trust as a decision (the act of entrusting a task) puts a part of the trusting agent's welfare on the line and thus involves risk: however satisfactory the transaction history with another agent might be, it is never guaranteed that this will continue in the future.
- Trust as a behavior emphasizes the actions of trusting agents and the relation between them. The relation generally intensifies as time progresses.

Trust as a mental attitude gives us an important clue of how to determine the trustworthiness of others: we need to analyze past interactions with the agent. Not surprisingly, this is exactly what the majority of trust algorithms do.

2.1.2 Definition of Trust

Although a lot of work has been done on the topic of trust, the definition of trust is still not very clear and different authors have given various definitions for the term trust. The properties of trust must be verified as well. In this thesis, when we need to calculate the value of trust, we use the definition proposed by [Marsh, 1994] which is

commonly accepted in the literature. "*Trust, is a particular level of the subjective probability with which an agent will perform a particular action, both before he can monitor such action (or independently of his capacity to monitor it) and in a context in which it affects his own action*".

Meanwhile, when the trust is used to make a decision, the definition proposed by [McKnight and Chervany, 1996] would be more easier to understand although the meaning is the same as the definition we introduced before: "*Trust is the extent to which one party is willing to depend on something or somebody in a given situation with a feeling of relative security, even though negative consequences are possible."*

2.1.3 Characteristics of Trust

Despite different contexts, trust can be broadly categorized by the relationships between two involved agents [Falcone and Shehory, 2002].

- Trust between a user and her agents: although an agent behaves on its user's behalf, an agent might not act as its user expects. How much a user trusts her agent determines how she delegates her tasks to the agent.
- Trust in service provider: It measures whether a service provider can provide trustworthy services.
- Trust in references: References refer to the agents that make recommendations or share their trust values. It measures whether an agent can provide reliable

recommendations.

- Trust in groups: It is the trust that one agent has in a group of other agents. By modeling trust in different groups, an agent can decide to join a group that can bring it most benefit.

Among various trust relationships, there are three characteristics for trust. [Abdul-Rahman and Hailes, 2000, Montaner. et al., 2002, Sabater and Sierra, 2001].

- Context-specific: Trust depends on some context. That is to say, trust a person to be a good doctor but do not trust her as a good driver.
- Multi-faceted: Even in the same context, there is a need to develop differentiated trust in different aspects of the capability of a given agent. For instance, a customer might evaluate a restaurant from several aspects, such as the quality of food, the price, and the service. For each aspect, a customer can derive a trust different from other aspects.
- Dynamic: Trust increases or decreases with further experience (direct interaction). It also decays with time.

2.2 Reputation

A reputation is an expectation about an agent's behavior based on information about or observations of its past behaviors [Abdul-Rahman, 2000]. It refers to a perception that an agent has of another's intentions and norms.

Similar to trust, reputation is a context-dependent quantity. An individual may enjoy a very high reputation for his/her experience in one domain, while having a low reputation in another.

In the meanwhile, reputation can be viewed as a global or personalized quantity. For social network researchers [Katz, 1953; Freeman, 1979; Marsden, et al., 1982; Krackhardt, et al., 1993], reputation is a quantity derived from the underlying social network. An agent's reputation is globally visible to all agents in a social network. Personalized reputation has been studied by [Zacharia, 1999; Sabater, et al., 2001; Yu et al, 2001], among others. As argued by [Mui, et al., 2002], an agent is likely to have different reputations (Figure 2.1) in the eyes of others, relative to the embedded social network.

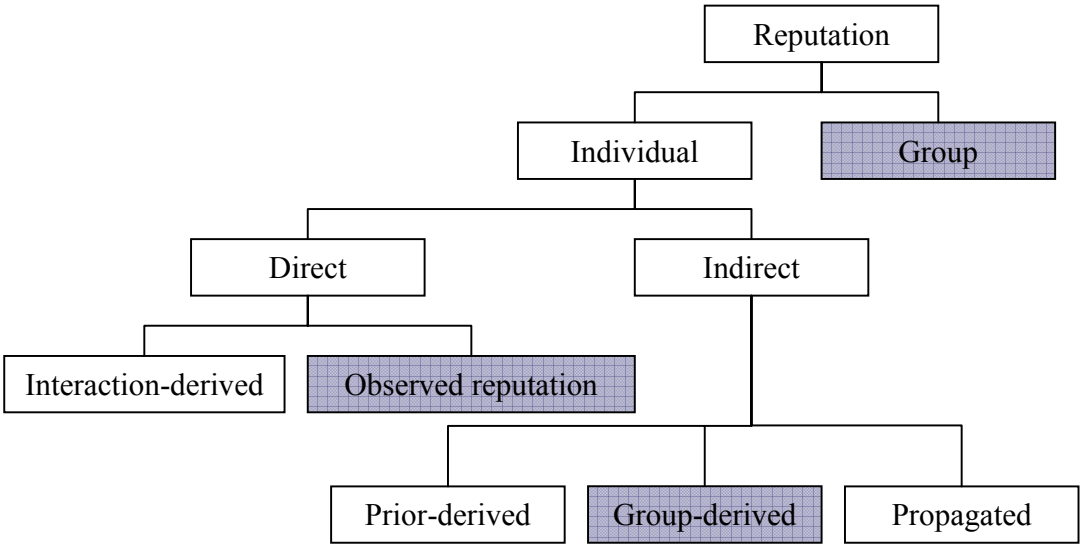

Figure 2.1 Reputation Typology

It is assumed that reputation is context dependent, shaded boxes indicate notions that are likely to be modeled as social (or global) reputation as opposed to being personalized to the inquiring agent.

Here we pick out the reputation we used in this dissertation to give some interpretation.

- Observed reputation: Agent A's observed reputation can be obtained from the other agent's feedback of the direct interaction with agent A.
- Prior-derived reputation: In the simplest inference, agents bring with them prior beliefs about strangers. As in human societies, each of us has different prior beliefs about the trustworthiness of strangers we meet.
- Propagated Reputation: In a Multi-agent System, an agent might be a stranger to the evaluating agent, and the evaluating agent can attempt to estimate the stranger's reputation based on information gathered from others in the environment. As [Abdul-Rahman and Hailes, 2000] have suggested, this mechanism is similar to the "word-of-mouth" propagation of information for humans. Reputation information can be passed from agent to agent.

2.3 Trust Management Approach in Multi-agent Systems

Trust management in Multi-agent Systems is used to detect malicious behaviors and to promote honest and cooperative interactions. Based on the approach adopted to

establish and evaluate trust relationship between agents, trust management in Multi-agent systems can be classified into 3 categories [Suryanarayana, et al., 2004], which are credential and policy-based trust management, reputation-based trust management and social network-based trust management as shown in Figure 2.2.

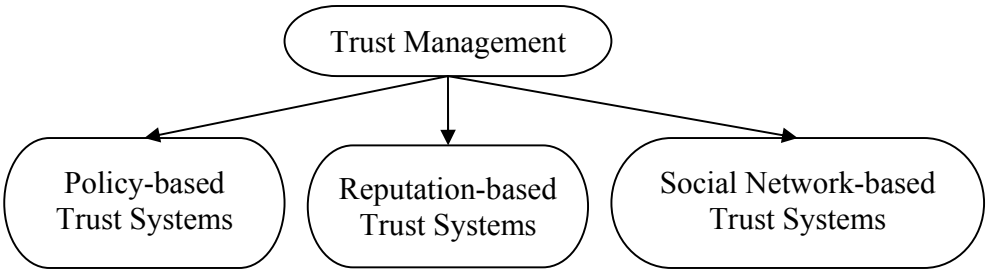

Figure 2.2 Trust Management Taxonomy

2.3.1 Policy-based Trust Management Systems

The research on policy-based trust focuses on problems in exchanging credentials, and generally assumes that trust is established simply by knowing a sufficient amount of credentials pertaining to a specific party. [Donovan and Yolanda, 2006] have pointed out that a credential may be as simple as a signature uniquely identifying an entity, or as complex and non-specific as a set of entities in the Semantic Web, where relationships between entities are explicitly described. The recursive problem of trusting the credentials is frequently solved by using a trusted third party to serve as an authority for issuing and verifying credentials.

Establishing trust under the policy-based trust systems suffers from a problem that a credential may incur a loss of privacy or control of information. [Yu et al., 2001; Yu and Winslett, 2003] have focused on the trade-off between privacy and earning trust. Based on their work, [Winslett et al., 2002] have proposed an architecture named *TrustBuilder* which provides mechanisms for addressing this trade-off. Another system is *PeerTrust* [Nejdl et al., 2004], a more recent policy and trust negotiation language that facilitates the automatic negotiation of a credential exchange. Others working in this area have contributed ideas on client-server credential exchange [Winsborough et al., 2000] and protecting privacy through generalizing or categorizing credentials [Seigneur and Jensen, 2004].

Several standards for representation of credentials and policies have been proposed to facilitate the exchange of credentials. *WS-Trust* [WS-Trust, 2005], an extension of WS-Security, specifies how trust is gained through proofs of identity, authorization, and performance. *Cassandra* [Becker and Sewell, 2004] is a system using a policy specification language that enforces how trust may be earned through the exchange of credentials. [Leithead et al., 2004] have presented another idea by using ontologies to flexibly represent trust negotiation policies.

Using credentials-based trust systems, one problem that should be solved is the credentials are also subject to trust decisions (i.e., can you believe a given credential to be true?). A typical solution in this case is to employ a common trusted third party to issue and verify credentials. However, it can be undesirable to have a single

authority responsible for deciding who and when someone is trusted. This problem is broadly described as *trust management*. [Blaze et al., 1996] have presented a system called *PolicyMaker. PolicyMaker* is a trust management system that facilitates the development of security features including privacy and authenticity for different kinds of network applications. Following PolicyMaker, a system called *KeyNote* is presented by [Blaze et al., 1999], which provides a standard policy language which is independent of the programming language used. KeyNote provides more application features than PolicyMaker, and the authors compare their idea of trust management with other existing systems at the time.

The policy-based access control trust mechanisms do not incorporate the need of the requesting agent to establish trust in the resource-owner; therefore, they by themselves do not provide a complete generic trust management solution for all decentralized applications.

2.3.2 Reputation-based Trust Management Systems

Reputation is a measure that is derived from direct or indirect knowledge on earlier interactions of agents, and it is used to access the level of trust an agent puts into another agent. Reputation-based trust management is a mechanism to use personal experience or the experiences of others, possibly combined, to make a trust decision about an entity. Reputation management avoids a hard security approach by distributing reputation information, and allowing an individual to make trust

decisions instead of a single, centralized trust management system. The trust value assigned to a trust relationship is a function of the combination of the peer's global reputation and the evaluating peer's perception of that peer.

[Abdul-Rahman and Hailes, 1997] have advocated an approach based on combing in a distributed trust model with a recommendation protocol. They focus on providing a system in which individuals are empowered to make trust decisions rather than automating the process. The main contribution of this work is to describe a system where it can be acknowledged that malicious entities coexist with the innocent, achieved through a decentralized trust decision process. In this model, a trust relationship is always between exactly two entities, is non-symmetrical, and is conditionally transitive. Decentralization allows each peer to manage its own trust. In the meanwhile, trust is context dependent. Trust in a peer varies depending on the categories. In a large decentralized system, it may be impossible for a peer to have knowledge about all other peers. Therefore, in order to cope with uncertainty arising due to interaction with unknown peers, a peer has to rely on recommendations from known peers about these unknown peers.

[Abdul-Rahman and Hailes, 2000] have proposed that when one peer trusts another, it constitutes a direct trust relationship. But if a peer trusts another peer to give recommendations about another peer's trustworthiness, then there is a recommender trust relationship between the two. Trust relationship exists only within each peer's own database and hence there is no global centralized map of trust relationships.

Corresponding to the two types of trust relationships, two types of data structures are maintained by each peer: one for direct trust experiences and another for recommender trust experiences. Recommender trust experiences are utilized for computing trust only when there is no direct trust experience with a particular peer.

[Aberer and Despotovic, 2001] have presented the P-Grid trust management approach which focuses on an efficient data management technique to construct a scalable trust model for decentralized applications. The global trust model described is based on binary trust. Peers perform transactions and if a peer cheats in a transaction, it becomes untrustworthy from a global perspective. This information in the form of a complaint about dishonest behavior can be sent to other peers. Complaints are the only behavior data used in this trust model. Reputation of a peer is based on the global knowledge on complaints. While it is easy for a peer to have access to all information about its own interactions with other peers, in a decentralized scenario, it is very difficult for it to access all the complaints about other agents. P-Grid [Aberer, 2001] is an efficient data storage model to store trust data. Trust is computed by using P-Grid as storage for complaints. A peer can file a complaint about another peer and send it to other peers using *insert* messages. When a peer wants to evaluate the trustworthiness of another peer, it searches for complaints on it and identifies peers that store those complaints. Since these peers can be malicious, their trustworthiness needs to be determined. In order to limit this process and to prevent the entire network from being explored, if similar trust information about a specific peer is achieved from a sufficient number of peers, no further checks are carried out.

[Damiani, di Vimercati et al., 2002] have introduced the XREP approach which primarily focuses on P2P file-sharing applications. In this system, each peer not only evaluates resources accessed from peers, but also models the reputations of peers in the system. A distributed polling algorithm is used to allow these reputation values to be shared among peers, so that a peer requesting a resource can assess the reliability of the resource offered by a peer before using it. Each peer named as a "servant" in the application plays the role of both server and client by providing and accessing resources respectively. XREP is a distributed protocol that allows the reputation values to be maintained and shared among the servants. It consists of the following phases: resource searching, resource selection and vote polling, vote evaluation, best servant check, and resource downloading.

[Lee, Sherwood et al., 2003] have proposed NICE, a platform for implementing distributed cooperative applications. NICE provides three main services: resource advertisement and location, secure bartering and trading of resources, and distributed trust evaluation. The objective of the trust inference model is to: a) identify cooperative users so that they can form robust cooperative groups, and b) prevent malicious peers and clusters to critically affect the working of the cooperative groups. NICE uses two trust mechanisms to protect the integrity of the cooperative groups: trust-based pricing and trust-based trading limits. In trust-based pricing, resources are priced according to mutually perceived trust. In trust-based trading limits, instead of varying the price of the resource, the amount of the resources bartered is varied. This ensures that when transacting with a less trusted peer, a peer can set a bound on the

amount of resources it loses. The trust inference algorithm can also be expressed using a directed graph called the trust graph. In such a trust graph, each vertex corresponds to a peer in the system. A directed edge from peer **A** to peer **B** exists if and only if **B** holds a cookie signed by **A** which implies that at least one transaction occurred between them. The value of this edge signifies the extent of trust that **A** has in **B** and depends on the set of **A**'s cookies held by **B**. If, however, **A** and **B** were never involved in a transaction and **A** wants to compute **B**'s trust, it can infer a trust value for **B** by using directed paths that end at **B**. Two trust inference mechanisms based on such a trust graph are described in NICE approach. One is the strongest path mechanism and the other is the weighted sum of strongest disjoint paths mechanism.

[Dragovic, Kotsovinos et al., 2003] have proposed Xeno Trust which is a distributed trust and reputation management architecture used in the XenoServer Open Platform. There are two levels of trust in XenoTrust: authoritative trust and reputation-based trust. Here we only focus on the reputation-based trust. The reputation-based trust in this system is built through interaction between peers based on individual experiences. In order to accommodate newcomers to the system who have no initial experience with other partners, exchanging of reputation information between partners is advocated. All the information gathered about each participant's reputation is aggregated in XenoTrust. This information is updated as new reputation information is received from peers.

2.3.3 Social Network-based Trust Management Systems

Social network-based trust management systems utilize social relationships between agents when computing trust and reputation values. In particular, these systems form conclusions about agents through analyzing a social network that represents the relationships within a community. The key feature of the social network-based trust management approach is that in any case, no matter how the system is solved, it is clear that one needs to explore the entire trust multi-graph in order to assess the trustworthiness of a single agent.

[Yu and Singh, 2000] were one of the first to explore the effect of social relationships of agents belonging to an online community on reputation in decentralized scenarios. It models an electronic community as a social network. Agents can have reputations for providing good services and referrals. In such a system, agents assist users working with them in two ways. First, they help to decide whether or how to respond to requests received from other agents in the system. And second, they help to evaluate the services and referrals provided by other agents in order to enable the user to contact the referrals provided by the most reliable agent. In this approach, agent evaluates the target agent not only by its direct observation, but also the referrals given by its neighbors. When a user poses a query to its corresponding agent, the agent uses the social network to identify a set of potential neighboring agents whom it believes has the expertise to answer the query. The query is then forwarded to this set of neighbors. A query sent to a peer contains three things: the question, the requestor agent's ID and address, and a number specifying the upper bound on the number of

referrals requested. When a query is received by a agent, it decides whether the query suits the user and if it should be shown to the user. The agent answers only if it is confident that its expertise matches the query. The agent may also respond with referrals to other trusted users whom it believes has the necessary expertise to answer the query. Thus, a response may include an answer to the query, or a referral, or both, or neither.

[Sabater and Sierra, 2001] have proposed a similar concept to TrustNet [Schillo, Funk et al., 2000] and the social dimension of agents and their opinions in the reputation model. Regret adopts the stance that the overall reputation of an agent is an aggregation of different pieces of information instead of relying only on the corresponding social network as a TrustNet. Regret is based on three dimensions of reputation: individual, social and ontological. It combines these three dimensions to yield a single value of reputation. When a member agent depends only on its direct interaction with other members in the society to evaluate reputation, the agent uses the individual dimension. If the agent also uses information about another peer provided by other members of the society, it uses the social dimension. The social dimension relies on group relations. In particular, since a peer inherits the reputation of the group it belongs to, the group and relational information can be used to attain an initial understanding about the behavior of the agent when direct information is unavailable. Thus, there are three sources of information that help agent "**A**" decide the reputation of agent "**B**", which are individual dimension between **A** and **B**, witness reputation from the information **A**'s group has about **B**, neighborhood

reputation from the information **A**'s group has about **B**'s group. Regret believes reputation to be multi-faceted. To combine the different types of reputation and obtain new types of reputation is defined by the *ontological dimension*.

[Pujol, Sanguesa et al., 2002] have introduced NodeRanking, like TrustNet and Regret, which utilizes social community aspects of agents to determine their reputation. The goal behind reputation systems in NodeRanking is to remove dependence upon the feedback received from other users, and instead explore other ways to determine reputation. NodeRanking views the system as a social network where each member has a position in the community. The location of a given member of a community in the network can be used to infer properties about the agent's degree of expertise or reputation. Members who are experts are well-known and can be easily identified as highly connected nodes in the social network graph. This information can be used by agents directly instead of having to resort to explicit ratings issued by each agent.

[Mui, 2002] has presented a computational model of trust and reputation. In this model, the author considered *Reciprocity* which is an important strategy in the real world society. The relationship of trust, reputation and reciprocity can be seen in Figure 2.3.

The direction of the arrow indicates the direction of influence among the variables. The dashed line indicates a mechanism not discussed.

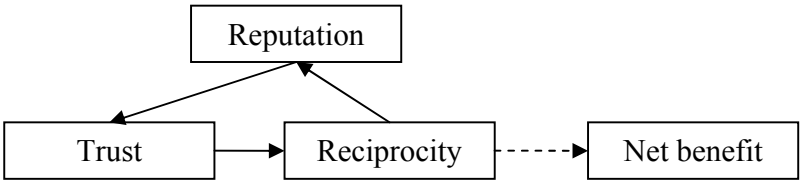

Figure 2.3 The Reinforcing Relationships among Trust, Reputation and Reciprocity

For an agent a_i in the embedded social network **A**, the relationships of trust, reputation and reciprocity are as follows:

- Increase in agent a_i's reputation in its embedded social network **A** should also increase the trust from the other agent for a_i.
- Increase in agent a_j's trust of a_i should also increase the likelihood that a_j will reciprocate positively to a_i's action.
- Increase in a_i's reciprocating actions to other agents in its embedded social network **A** should also increase a_i's reputation in **A**.

The reputation in this work is defined as the perception that an agent creates through past actions about its intentions and norms and it is the perception that suggests an agent's intentions and norms in the embedded social network that connects two agents. Trust is termed as a subjective expectation an agent has about another's future behavior based on the history of their encounters. When there are only two agents considered, the reputation can be estimated by using Beta distribution and the level of reciprocity is used to measure the confidence on the parameter estimation. When there are numbers of chains between two agents, the reputation can be obtained by using combination methods, which are additive and multiplicative.

2.4 Trust Propagation Mechanisms in Trust Graph

We have reviewed the works that have been done on trust management. One of the problems is how to inference the reputation in Trust Graph. The problem can be seen in the reputation-based trust management and social network-based trust management systems. The relationship between these problems is shown in Figure 2.4.

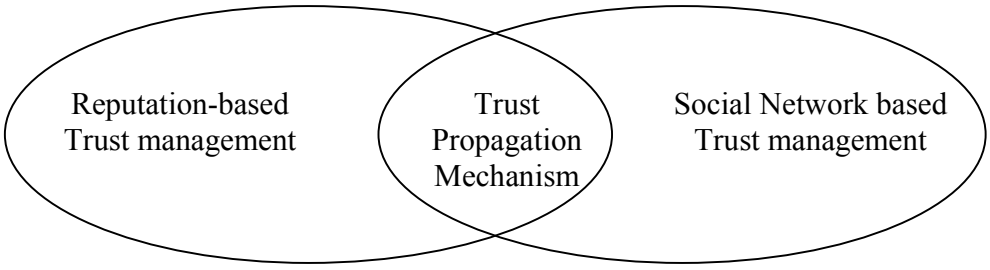

Figure 2.4 The Relationship between the Trust Management Systems and the Trust Propagation Mechanism

[Zacharia, 1999] has introduced a method to propagate the trust value in the highly connected communities. When a user submits a query for the Histos reputation value of another user, the systems will perform the following computation:

- Use a Breadth First Search algorithm to find all the directed paths connecting the two agents.
- Keep the chains whose length are less than or equal to N. And the chronologically *q* most recent ratings are only cared about.

After constructing the Trust Graph, the reputation propagation can be calculated as follows: Let $W_{jk}(n)$ denote the rating of user A_j for user $A_k(n)$ at a distance *n* from

user A_0, and $R_k(n)$ denote the personalized reputation of user $A_k(n)$ from the perspective of user A_0. At each level n away from user A_0, the users $A_k(n)$ have a reputation value given by:

$$R_k(n) = D \bullet \sum_j (R_j(n-1) \bullet W_{jk}(n))) / \sum_j R_j(n-1)$$
$$\forall jk, such \quad that \quad W_{jk}(n) \geq 0.5$$
$$m_k(n) = \deg(A_k(n)) = |W_{jk}(n)|$$

Where $\deg(A_k(n))$ is the number of connected paths from A_0 to $A_k(n)$ and D is the range of reputation values.

[Esfandiari and Chandrasekharan, 2001] have proposed that when considering the weakly transitive of trust, the propagation can be calculated as:

$$T_{prop}(a,c) = T(a,b_1) \times \cdots \times T(b_{1n},c)$$
$$with \quad (b_i) \quad being \quad the \quad \text{int}ermediate \quad agents \quad in \quad a \quad path \quad from \quad a \quad to \quad c.$$

[Yu and Singh, 2002] have analyzed the reputation management by using Dempster-Shafer Theory. TrustNet is used to systematically incorporate the testimonies of the various witnesses regarding a particular party. Suppose A_r wishes to evaluate the trustworthiness of V_g. After a series of l referrals, a testimony about agent V_g is returned from agent A_j. Given a series of referrals $\{r_1, r_2, ..., r_n\}$, the requester A_r constructs a TrustNet by incorporating each referral $r_i =< A_i, A_j >$ into TrustNet. A_r adds r_i to R if and only if $A_j \notin A$ and $depth(A_i) \leq depthLimit$. The testimonies propagation through a TrustNet is shown in Figure 2.5. Suppose agent A_r wants to evaluate the trustworthiness of agent V_g, and $\{w_1, w_2, ..., w_L\}$ are a group of

witnesses towards agent V_g. The testimonies from witnesses can be incorporated into the rating of a given agent as follows: Let τ_{A_i} and π_{A_i} be the belief functions corresponding to agent A_i's local and total beliefs, respectively.

Agent A_r could update its local belief value of agent V_g as follows:

$$\pi_{A_r} = \tau_{w_1} \oplus \cdots \oplus \tau_{w_L}$$

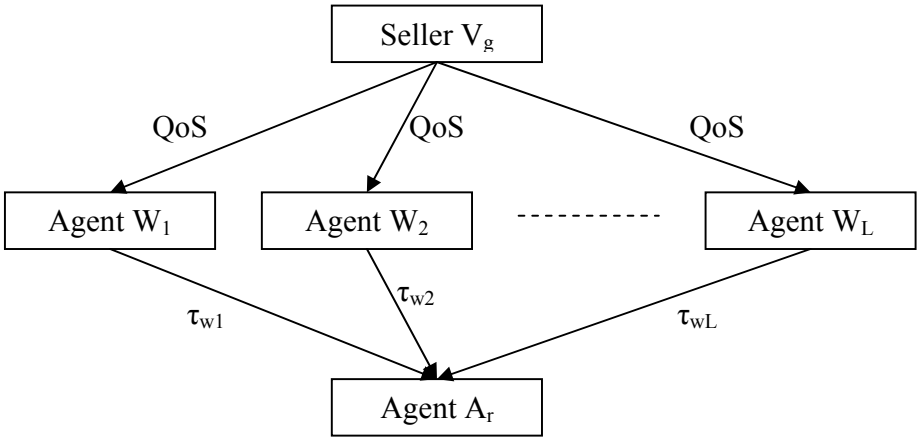

Figure 2.5 Testimony Propagation through a TrustNet

[Mui, 2002] has proposed mechanisms for inferring reputation. When the acquaintances are in the parallel networks as in Figure 2.6, the reputation can be inferred as follows:

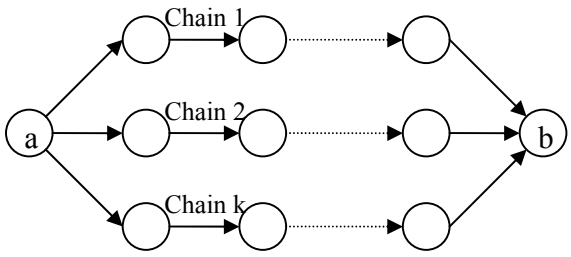

Figure 2.6 Illustration of a Parallel Network between Two Agents *a* and *b*

There are *k* chains between two agents of interest, where each chain consists of at least one link. For each chain in the parallel network, the total weight can be tallied by using additive method or multiplicative method. The form of a multiplicative estimate for chain *i*'s weight (w_i) can be: $w_i = \prod_{j=1}^{l_i} w_{ij}$ *where* $0 \le i \le k$, where l_i refers to the total number of edges in chain *i* and w_{ij} refers to the j^{th} segment of the i^{th} chain. w_{ij} can be calculated as follows: $w_{ij} = \begin{cases} \frac{m_{ij}}{m} & if \quad m_{ij} < m \\ 1 & Otherwise \end{cases}$, where m_{ij} is the number of encounters between agents *i* and *j*, *m* represents the minimum number of encounters necessary to achieve the desired level of confidence and error. Once the weights of all chains of the parallel network between the two end nodes are calculated, the estimate across the whole parallel network can be sensibly expressed as a weighted sum across all the chains: $R_{ab} = \sum_{i=1}^{k} r_{ab}(i)\overline{w}_i$, where $r_{ab}(i)$ is *a*'s estimate of *b*'s reputation using path *i* and $\overline{w}_i$ is the normalized weight of path *i* (summing $\overline{w}_i$ over all *i* yields 1). R_{ab} can be interpreted as the overall perception that *a* garnered about *b* using all paths connecting the two. Along each chain, the Bayesian estimate rating

method can be used to infer the reputation of second degree indirect neighbors scheme: $\rho_{ik}(c) = \rho_{ij}(c)\rho_{jk}(c) + (1-\rho_{ij}(c))(1-\rho_{jk}(c))$. $\rho_{ij}(c)$ is the probability that *i* approves of another *j*'s opinion for an object in the context *c*. This logic is based on the fact that *i* would approve of *k*'s opinion given the intermiediate agent *j* is the sum of the following 2 probabilities: *i* approves of *j* and *j* approves of *k*; *i* disapproves of *j* and *j* disapproves of *k*. However, when one chain is long enough, the trust value would be too limited because the reputation of second degree indirect neighbors is obtained by the summation of the both approval and disapproval. There exists another situation which is the generalized network of acquaintances. In this network, there are complex relations between the nodes in the network. To infer reputation in the generalized network, the author proposed one important step, which is Graph Parallelization. After the parallelization, the network can be solved as before.

[Lee, Sherwood et al., 2003] have introduced NICE trust inference model. The trust inference algorithm is expressed using a directed graph called the trust graph (see Figure 2.7). Two trust inference mechanisms based on such a trust graph are described in the NICE approach. These are the strongest path mechanism and the weighted sum of strongest disjoint paths mechanism. In the strongest path mechanism, the strength of a path can be computed either as the minimum valued edge along the path or the product of all edges along the path, and thus, agent **A** can infer agent B's trust by using the minimum trust value on the strongest path between **A** and **B**. In the weighted sum of strongest disjoint paths, agent **A** can compute a trust value for **B** by computing the weighted sum of the strength of all the strongest disjoint paths.

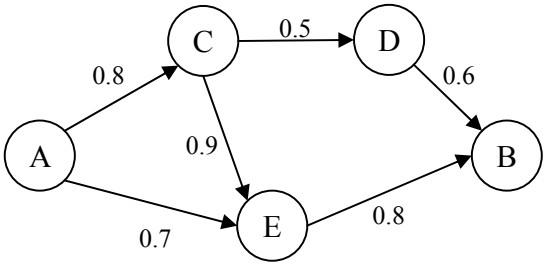

Figure 2.7 NICE Trust Graph (Weights Represent the Extent of Trust the Source Has in the Sink)

[Wang and Singh, 2006] have presented a trust propagation method which is based on the concatenation operator and aggregation operator. Given a trust network, these two operators can be used in the path algebra to merge the trust. The combination can be shown in details below.

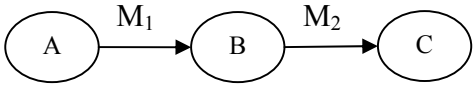

Figure 2.8 Transformation Trust Path

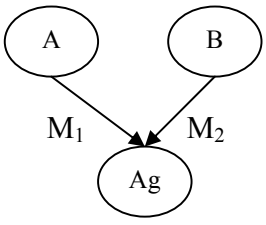

Figure 2.9 Combination Trust Path

This approach is based on the following two cases. Case 1: As shown in Figure 2.8, agent *A* has a trust M_1 in agent *B*'s references and *B* has a trust M_2 in agent *C*. Then

A's trust in *C* due to the reference from *B* is $M = M_1 \otimes M_2$. Here $\otimes$ is the concatenation operator. Case 2: In Figure 2.9, agents *A* and *B* have trust M_1 and M_2, respectively, in A_g. Then the combined trust of *A* and *B* in A_g is captured via the aggregation operator $\oplus$, as in $M_1 \oplus M_2$. For a given trust network, the beliefs can be combined as follows: For any agent $A_i \in A$, suppose $\{B_1, B_2, ..., B_m\}$ are the neighbors of A_i. Suppose the trust ratings that A_i assigns to $B_1, B_2, \ldots, B_m$ are $M_1, M_2, \ldots M_m$. Suppose that all the neighbors have already obtained their trust ratings in A_g, and let these be $M_1', M_2', ..., M_m'$. Then we obtain the trust of A_i in A_g, M, by:

$$M = (M_1 \otimes M_1') \oplus (M_2 \otimes M_2') \oplus \cdots \oplus (M_m \otimes M_m')$$

If the neighbor has not obtained the trust in A_g, the algorithm can be run recursively to obtain the trust from merging and combining the trust from the neighbor's neighbors, since all the leaves in the trust network are the witnesses who have their trust values in A_g computed from their direct interactions with A_g. So the trust ratings can be merged in a bottom up fashion, from the leaves of the trust network up to its root A_r.

[Jøsang, et al., 2006a] analyzed the trust network by using subjective logic. In order to solve the trust network, they introduce the network simplification, rather than normalization which was used by a lot of research work on the trust network analysis before. Simplification of a trust network consists of only including certain arcs in order to allow the trust network between the source trustor and the target trustee to be formally expressed as a canonical expression. DSPG (directed series-parallel graphs) is the type of network which needs no normalization because a DSPG does not have loops and internal dependencies. To evaluate the trust between source and sink, the

first step is to determine all possible paths from a given source to a given target. In this step, the authors proposed an algorithm written in Seudo-code and the transitive trust graphs can be stored and represented on a computer in the form of a list of directed trust arcs with additional attributes. The second step is to select a subset of those paths for creating a DSPG. The definition of the *canonical expression* says that an expression of a trust graph in structured notation where every arc only appears once is called canonical. Thus, to create the DSPG, all the expressions except the non-canonical ones are used. However, among all the DSPGs, only one will be selected for deriving the trust measure. The optimal DSPG is the one that results in the highest confidence level of the derived trust value. This principle focuses on maximizing certainty in the trust value, and not on others such as deriving the strongest positive or negative trust value. Here there is a trade-off between the time it takes to find the optimal DSPG, and how close to the optimal DSPG a simplified graph can be. In order to solve this, the author introduced an *exhaustive* method that is guaranteed to find the optimal DSPG and a *heuristic* method that will find a DSPG close to, or equal to the optimal DSPG. After DSPG's construction and optimization, the subjective logic can be used to derive the trust value.

2.5 Research Gaps

Trust work in multi-agent systems has been introduced in this chapter. The overviews of trust, trust management and the trust propagation mechanisms in trust network have been figured out. As the trust and reputation have been used in virtual

communities, how to acquire the trust value in this artificial environment is a challenge for the researchers. However, none of the work has solving the trust network by using artificial intelligence techniques. The works have been done either based on normalization or on simplification. To infer messages in a network, one of the most efficient methods is Bayesian Inference method. Thus, in this dissertation, we will solve the trust inference problem in trust network by using Bayesian Inference method. In the next Chapters 3 and 4, we will propose the modeling of trust and evaluation trustworthiness in trust network. In Chapter 5, we will propose a simulation experiment and provide the results.

3 TRUST MODELING AND TRUST NETWORK CONSTRUCTION

Trust is often built over time by accumulating personal experience with others. This experience is used to predict how they will perform in a yet- to- be observed situation. However, when assessing our trust in someone with whom we have no direct personal experience, we often ask others about their experiences with this individual. This collective opinion of others regarding an individual is known as the individual's *reputation* and it is the reputation of a trustee that we use to assess its trustworthiness, if we have no personal experience.

Given the importance of trust and reputation in open multi-agent systems, the computational trust and reputation model should be developed meeting requirements for the domain to which they apply. In our case, the requirements can be summarized as follows [Patel, el al., 2005]:

- The model must provide a trust metric that represents a level of trust in an agent. Such a metric allows comparisons between agents so that one agent can be inferred as more trustworthy than another. The model must be able to provide a trust metric given the presence or absence of personal experience.
- The model must reflect an individual's *confidence* in its level of trust for another agent. This is necessary so that an agent can determine the degree of influence the trust metric has on its decision about whether or not to interact with another individual. Generally speaking, higher confidence means a greater impact on the decision-making process, and lower confidence means

lower impact.

To meet the above requirements, we have modeled trust and reputation by using TRAVOS model.

3.1 Trust Modeling

As we have described before, trust and reputation are context based. Thus, in the following discussion, we model the trust and reputation only in one particular context. The model equips an agent with three ways of assessing the trustworthiness of another agent on one context. These are from direct interaction, witnesses' reputation and both.

Owing to the characteristic of open MAS, we have made an assumption about the agents and their environment [Huynh et al, 2006].

Assumption 3.1: Agents are willing to share their experiences with others (as witness).

3.1.1 Basic Notation

In this section, we will give some notations which are used to represent the trust problem.

Let $A = \{a_1, a_2, ..., a_n\}$ be the set of all agents. Over time, distinct pair of agents $\{a_i, a_j\} \subseteq A, i \neq j$, may interact with each other and in one time slot, there might be more than one pair of agents interacting. *S* represents the truster or trust source; *T* represents the trustee or trust target. In an environment, each agent can be the truster or the trustee.

In order to distinguish the trust on one agent's recommendation and ability to fulfill some function, we give the following two definitions which were mentioned by [Josang *et al*., 2006b].

Definition 3.1: *Functional trust* is a type of trust one agent puts on the target agent based on the latter's competence to supply some particular service.

Definition 3.2: *Referral trust* is a type of trust one agent puts on the target agent based on the ability to give recommendation.

Let $f\tau_{a_i,a_j}$ and $r\tau_{a_i,a_j}$ represent the functional trust and referral trust of a_i to a_j respectively.

3.1.2 Modeling

Functional Trust Modeling

This work is very similar to the trust modeling which has been done by [Patel et al., 2005]. Let $O^{t}_{a_i,a_j}, i \neq j$ denotes the outcome of an interaction between agents i and j at time t. We represent a contract outcome with a binary variable for simplicity: $O_{a_i,a_j} = \begin{cases} 1 & contract\ \ fulfilled\ \ by\ \ a_j \\ 0 & otherwise \end{cases}$

During the time period $[t_0, t_1]$, the history of interaction between agents a_i and a_j is recorded as a tuple, $\Re^{t_0:t_1}_{a_i,a_j} = (m^{t_0:t_1}_{a_i,a_j}, n^{t_0:t_1}_{a_i,a_j})$ where the value of $m^{t_0:t_1}_{a_i,a_j}$ is the number of successful interaction of a_i and a_j, and $n^{t_0:t_1}_{a_i,a_j}$ is the number of unsuccessful interaction between a_i and a_j.

B_{a_i,a_j} is the expected value of O_{a_i,a_j} given complete information about a_j's decision processes and all environment factors that affect its capabilities. $B_{a_i,a_j} = E[O^{t_0:t_1}_{a_i,a_j}], where \quad B_{a_i,a_j} \in [0,1]$.

The functional trust can be evaluated by using the method proposed by [Patel et al., 2005].

$$f\tau_{a_i,a_j} = E[B_{a_i,a_j} \left| O^{t_0:t_1}_{a_i,a_j} \right.]$$

The expected value of a continuous random variable depends on the probability density function used to model the probability that the variable will have a certain value. In Bayesian analysis, the beta family of pdfs is commonly used as a priori

distribution for random variables that take on continuous values in the interval [0,1]. According to the work by [Patel et al., 2005], the functional trust can be calculated as follows:

$$f\tau_{a_i,a_j} = E[B|\alpha,\beta] = \frac{\alpha}{\alpha+\beta}$$

where $\alpha = m_{a_i,a_j}^{t_0:t_1} + 1$ *and* $\beta = n_{a_i,a_j}^{t_0:t_1} + 1$, $t_0:t_1$ is the time period of an assessment.

Referral Trust Modeling

The functional trust is not equal to referral trust because a good customer might not be a good recommender. The referral trust is the trust that the truster places on the trustee who can recommend a third agent who can supply function service or have ability to recommend others. The referral trust can be estimated by using the information supplied by witnesses to the truster. At each time slot, there is not only an exchange of functional trust, but also the referral trust. For instance, an agent **A** can send requirements to some agents (**B, C, D**) and ask them to evaluate agent **E**. Owing to the assumption that all the agents report their information accurately and truthfully, agent **A** can compare the announcement to the outcome which **A** interacted with **E**. There are three results: (1) the witness agent says that it does not know agent **E** (2) the witness agent's evaluation is the same as the interaction outcome (3) the witness agent's evaluation is not the same as the interaction outcome. We classify these three situations into two, which are: type 1, the evaluation is the same as the interaction outcome and type 2, otherwise. To some extent, the interaction between the truster and the witness could be seen to be the same as the interaction between the

truster and the trustee, so the referral trust can be estimated by using the same method which is used to evaluate functional trust.

$T^t_{a_i,a_j}, i \neq j$ denotes the outcome of an detection between agents a_i and a_j at time t. We represent detection with a binary variable for simplicity.

$$T_{a_i,a_j} = \begin{cases} 1 & a_j\text{'s report is the same as the fact} \\ 0 & otherwise \end{cases}$$

During time period $[t_0, t_1]$, the history of interaction between agents a_i and a_j is recorded as a tuple, $R^{t_0:t_1}_{a_i,a_j} = (g^{t_0:t_1}_{a_i,a_j}, h^{t_0:t_1}_{a_i,a_j})$ where the value of $g^{t_0:t_1}_{a_i,a_j}$ is the number of successful interaction of a_i and a_j, and $h^{t_0:t_1}_{a_i,a_j}$ is the number of unsuccessful interactions between a_i and a_j.

D_{a_i,a_j} is the expected value of T_{a_i,a_j} given complete information about a_j's decision processes and all environment factors that affect its capabilities.

$D_{a_1,a_j} = E[T^{t_0:t_1}_{a_i,a_j}], where \quad D_{a_i,a_j} \in [0,1]$.

$$r\tau_{a_i,a_j} = E[D_{a_i,a_j} \left| T^{t_0:t_1}_{a_i,a_j} \right.]$$

$$r\tau_{a_i,a_j} = E[D \left| \alpha', \beta' \right.] = \frac{\alpha'}{\alpha' + \beta'}$$

where $\alpha' = g^{t_0:t_1}_{a_1,a_j} + 1 \quad and \quad \beta' = h^{t_0:t_1}_{a_i,a_j} + 1$, $t_0 : t_1$ is the time period of an assessment.

3.2 Trust Network Construction

3.2.1 Trust Transitivity

Trust Transitivity in our work has the same meaning as proposed by [Jøsang et al., 2006b]. It means, for example, that if Alice trusts Bob who trusts Eric, then Alice will also trust Eric. However, trust is not always transitive in real life. For example the fact that Alice trusts Bob to look after her child, and Bob trusts Eric to fix his car, does not imply that Alice trusts Eric for looking after her child, or for fixing her car. However, under certain semantic constraints [Jøsang and Pope, 2005], trust can be transitive, and a trust system can be used to derive trust.

Separating trust into referral trust and functional trust makes trust transitivity become true. An actual example is that Alice needs to have her car serviced, so she asks Bob for his advice about where to find a good car mechanic in town. Bob does not actually know any car mechanics himself, but he knows Claire and he believes that Claire knows a good car mechanic. As it happens, Claire is happy to recommend the car mechanic named Eric. As a result of transitivity, Alice is able to derive trust in Eric. As already mentioned, trust in the ability to recommend represents referral trust, and is precisely what allows trust to become transitive. At the same time, referral trust always assumes the existence of a functional trust scope at the end of the transitive path, which in this example is about being a good car mechanic. The "referral" variant of a trust scope can be considered to be recursive, so that any transitive trust

chain, with arbitrary length, can be expressed using only one trust scope with two variants. This principle is captured by the following criterion.

Definition 3.3: *Functional Trust Derivation Criterion:* Derivation of functional trust through referral trust requires that the last trust arc represents functional trust, and all previous trust arcs represent referral trust.

3.2.2 Trust Network Construction

Related definition and question description

Definition 3.4: *Agents Trust Relation Graph (ATRG): ATRG* is a directed graph, which denotes the trust relations among agents. *ATRG= (V, E),* where: *V* is the set of agents in the graph and $E=V\times V$ denotes the trust relation among agents and T_{v_1,v_2} denotes the trust value that agent v_1 has on agent v_2.

Definition 3.5: *Agents Functional Trust Sub-Graph (AFTSG): AFTSG* is a directed graph, which denotes the functional trust information contained in agent *i*. $AFTSG=(V_f^{'},E_f^{'})$, where: $V_f^{'}\subseteq V$, which denotes agents that have functional trust relation with agent *i*; $E_f^{'}=V_f^{'}\times V_f^{'}$ denotes the functional trust relation among the agents of $V_f^{'}$ and $T_{v_1^{'},v_2^{'}}^{f}$ denotes the functional trust value that agent $v_1^{'}$ has on agent $v_2^{'}$.

Definition 3.6: *Agents Referral Trust Sub-Graph (ARTSG): ARTSG* is a directed graph, which denotes the referral trust information contained in agent *i*. $AFTSG = (V_r^{'}, E_r^{'})$, where: $V_r^{'} \subseteq V$, which denotes agents that have referral trust relation with agent *i*; $E_r^{'} = V_r^{'} \times V_r^{'}$ denotes the referral trust relation among the agents of $V_r^{'}$ and $T_{v_1^{'}, v_2^{'}}^{r}$ denotes the referral trust value that agent $v_1^{'}$ has on agent $v_2^{'}$.

Definition 3.7: *Trust Path (TP):* Trust path from agent *i* to agent *j* can be defined as an agent sequence $\{i, i+1, ..., j-1, j\}$ where *i* has referral trust with *i+1*, *i+1* has referral trust with *i+2,…, j-1* has functional trust with *j*. The *TP* indicates that agent *i* can get the functional trust of agent *j* after a series of trust delegation.

Each agent has two sub-graphs which are *AFTSG* and *ARTSG*. *AFTSG* is used to store the functional trust and *ARTSG* is used to store the referral trust which the graph owner has with other agents. The *ATRG* describes the global trust information when one agent needs to evaluate another. In real multi-agent systems, no *ATRG* exists, and each agent only stores the trust information related to itself. Each time, different target agents and source agents may accomplish their evaluation process with different *ATRGs*.

When agent *i* needs to decide whether to cooperate with agent *j*, the following procedure can be used to find out the *ATRG*.

- First agent *i* will check the *AFTSG* in its own database. If there is a record in its *AFTSG*, *i* can make a decision right away. The agent also has another choice which is to use recommendation to evaluate the target agent and this can be accomplished by the following steps. This situation occurs when there is no record with *j* in its *AFTSG*, or the source agent has to evaluate the target agent.
- Agent *i* should send the inquiry to the agents in the *ARTSG*. Each agent in *i*'s *ARTSG* will check its *AFTSG* to see whether there is an interaction record. If they have, they will report the functional trust value to *i*, otherwise, they will send inquiry to the agents in their *ARTSG*. Each agent does this till all the agents which have functional trust of agent *j* are found out.
- An *ATRG* is constructed including all the agents who supply information to the inquiry process.

Construction of ATRG

In open multi-agent systems, each agent can only communicate with a few other agents called acquaintances to exchange the trust information they have. To evaluate the target agent's trustworthiness, the source agent has to construct the *ATRG* to obtain the outcome. The following example can explain this process well.

When agent *i* needs to evaluate agent *j*'s functional trust. It first searches among its own functional trust dataset and finds out whether there is a record of *j*'s functional

trust. At the same time, *i* sends inquiries to its acquaintances and asks them to supply the information about agent *j*. If the acquaintances have functional trust records of *j*, they will send it to agent *i*, otherwise, they will ask their acquaintances to do the same thing. Untill all the agents who know *j*'s functional trust are found out, the ATRG is accomplished.

Agent *i*'s functional trust dataset and referral trust dataset are shown in Figure 3.1 and Figure 3.2.

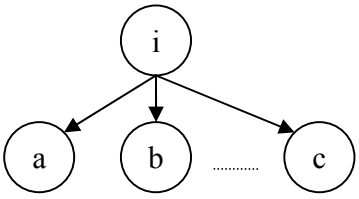

Figure 3.1 Agent *i*'s functional trust dataset

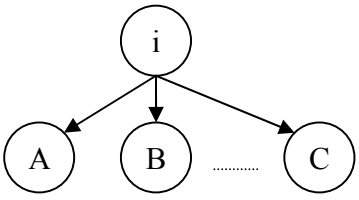

Figure 3.2 Agent *i*'s referral trust dataset

In agent *i*'s functional trust dataset, there is no interaction record with agent *j*. Thus, agent *i* should send request to its acquaintances. The acquaintances will check their functional trust dataset first and if there are records, they will report to agent *i*,

otherwise, they will send requirements to their acquaintances. We assume that the functional trust records of agent *j* are shown in Figure 3.3, and agents *A* and *C* have referral trust interaction with agents *k*, *l* respectively. Thus, we can get the partial *ATRG* of agents *i* and *j*, which is shown in Figure 3.4.

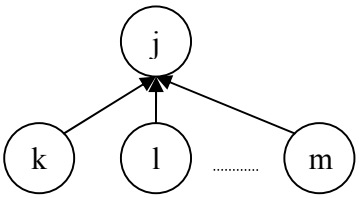

Figure 3.3 Agent *j*'s functional trust dataset

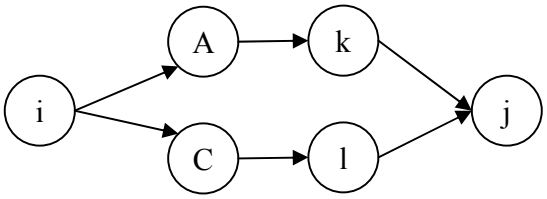

Figure 3.4 Agent *i*'s partial *ATRG* with agent *j*

After finishing the construction of the *ATRG*, an intact *ATRG* can be seen and through each path, the trust information is transferred to another agent. However, every arc in the Trust Graph has the same trust scope and the transitive trust propagation is possible with two variants of a single trust scope.

In the following chapter, we will propose an approach to evaluate the trust value in the trust network we have constructed.

4 TRUSTWORTHINESS EVALUATION

As stated in Chapter 3, we have constructed an *ATRG* step by step. After obtaining the Trust Graph, we still cannot know the trustworthiness of the target agent yet. In order to analyze the trustworthiness of target agent, one of the most important processes is to solve the Trust Graph and get a comparable value of trust which can help the agent to make a decision.

In this Chapter, we will propose a novel approach to solve the Trust Graph and in the following Chapter, an experiment will be presented.

4.1 Evaluation

After constructing the Trust Graph, the most important procedure is to solve the Trust Graph and show a readable value to the decision maker. The novel approach we proposed is based on the Bayesian inference method.

4.1.1 Introduction

In our model, the trust value is only in the range of [0, 1] and there is no negative trust value. So the higher the trust value is, the more trust the source has on the target. From the Trust Graph we have constructed in Chapter 3, we can see that there is at least one chain to link the source agent and the target agent. As we know, it is

common to collect advice from several sources in order to be better informed while making decisions. However, having a lot of information from different sources and how to combine them, say, how to get the conclusion which reflects the fact is a problem that needs to be solved. By using the Parallel Trust Combination [Jøsang et al., 2006b], we can conclude that parallel combination of positive trust has the effect of strengthening the derived trust. The combination is shown in Figure 4.1.

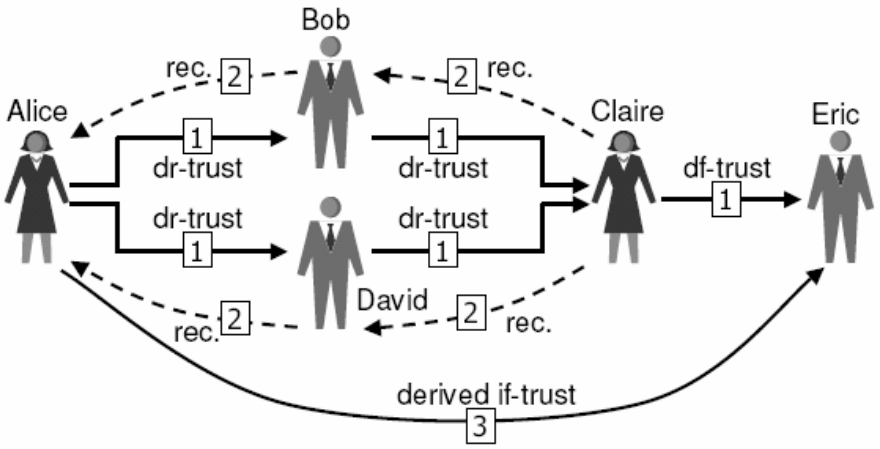

Figure 4.1 Trust Derived by Parallel Combination of Trust Paths

When receiving conflicting recommended trust, the subjective logic is used to combine these recommendations to derive the trust in the target agent.

Another way to solve the Trust Network is normalization. In a Trust Network, for each chain from source agent to target agent, the link has its weight. The final derived trust value can be obtained by normalization [Mui, 2002].

In our work, we use the Bayesian inference method to combine the information and derive the trust value.

Bayesian Probability Theory and Bayesian Networks

Bayesian probability theory is the statistical theory of making statements about uncertain events θ. Initially events of interest are assigned a prior belief $p(\theta)$ which reflects existing knowledge about the event and the problem area. Later, as new information D becomes available, the subjective beliefs are updated using the Bayes' rule [Nurmi, 2005]:

$$posterior = \frac{likelihood \times prior}{normalizer} = p(\theta|D) = \frac{p(D|\theta)\,p(\theta)}{p(D)} \tag{4.1}$$

The likelihood term $p(D|\theta)$ measures the probability of seeing particular realizations of the event θ, whereas the normalizer $p(D)$ is used to ensure that the values of $p(\theta|D)$ sum up to one and thus define a proper probability distribution. After updating, the values of the posterior $p(\theta|D)$ are used as the new priors $p(\theta)$.

Bayesian Networks are directed acyclic graphs that model relationships between variables using probability theory. In the causal interpretation, two (or more) variables are connected through an edge only if there is a direct causal relationship between the variables. Although we cannot see the whole Trust Network as a Bayesian Network, we can see each parallel chain in the Trust Network as a Bayesian Network and introduce the message passing algorithm to solve each chain.

This logic can work in the Trust Network, and we will explain this as follows.

1. The trust values that each agent have on others are modeled by using Beta distribution. That is to say, the prior probabilities are assigned using Beta Distribution; the resulting posterior beliefs are of Beta distribution.
2. The trust value agent *A* tells to agent *C* on agent *B* may not be the actual trust value of agent *B*, it depends on agent *A*'s attitude and agent *B*'s action. Thus, if we assume that agent *A* is reliable to agent *C*, the trust value will be the conditional trust value agent *B* has given agent *A* speaks. To explain this in another way, agent *B* can be seen as an information source, agent *A* tells to agent *C* on reliability of agent *B*, is the conditional reliability of agent *B* given agent *A* says. If you see the reliability as the trust value, we can model the trust value agent *A* has on agent *B* as the conditional probability $P(B|A)$.
3. Along the whole network, the target agent's trust value can be obtained by the Bayesian inference method. Belief updating by network propagation in networks is calculating the posterior probability given some known evidence. Although there is no casual relation between each agent, the trust value agent *A* speaks out on agent *B* is the interaction history of the agent *A* with agent *B*. It can be said as the trust value of agent *B* given the interaction history with agent *A* when agent *A* is reliable, when i.e, it tells the truth.

4.1.2 The Proposed Approach

This approach is based on autonomous agent trust network construction process. After constructing the Trust Graph, the agent can use the following procedure to derive the trust value of the target agent.

However, in order to make our approach more flexible and to make it usable in other Trust Networks rather than the Trust Graph constructed by us, we introduce a procedure called parallelization. If the Trust Network's structure is not the same as our Trust Graph, the parallelization should be applied before using our trust estimation mechanism.

Parallelization

In the *Trust Network*, the relation between the target agent and the source agent might not be parallelized. In order to overcome the dependence, we choose to parallelize the Trust Graph at first and the following algorithm can be used to fulfill this requirement.

```
DEFINE PROCEDURE Parallelization TO BE
  Define chain () as the dataset
  Set node 1 as the current node
  Set node n as the node which has the direct functional
  connection with the target node
  k=0
  Flag=true
  Do while Flag=true
   Do while (current node is not node n)
    If (have another node connect to current node) then
        If (this node does not exist in the Stack) then
            Push current node into Stack
            Move to next node which is connected to current
  node
         End if
    Else
      If (Stack is empty) then
           Set Flag as false & exit the Parallelization process
      Else
           Pop one node from Stack and set current node as this
  node
      End if
    End if
  Loop
  If the top node in the Stack is node n then
       k=k+1
      Store all the nodes in the Stack into chain(k)
      Pop one node from Stack
      Pop one node from Stack and set current node as this
  node
  End if
Loop
```

In this program, node **1** represents the source node and node **n** represents target node. **k** is the total number of chains of one Trust Network.

In the *ATRG* (parallelized or not), all the chains cannot be used for further calculation. From [Zacharia, 1999], we only keep the length less than or equal to N and the chronologically q most recent recommendations given from each witness. The two thresholds can be set by the domain experts.

The reason behind this is that trust is weakened or diluted through transitivity. The longer the trust chain is, the more unreliable the trust value derived from this chain. In the meanwhile, the older the interaction occurs, the more questionable is the trust value obtained from the interaction to estimate the recent trust value.

Regularizing the Numerical Structure

All the agents only have two states which are reliable and unreliable. The value on each arc is the trust value of the child in the eyes of the parent, that is to say, the child's conditional probability of being reliable given its parent's reliable announcement. We assume the child's conditional probability of being reliable given its parent's unreliable announcement is as follows:

$\min(\frac{1}{n_q}, p(witness \quad is \quad reliable))$, where n_q is the number of possible outcomes for each witness, here it is 2 [Barber and Kim, 2001].

Mathematically, let $V_{i,j}$ represent the trust value on the arc from a_i to a_j and $V_{s,t}$ represents the truster's direct trust value on the trustee. To simplify the formula, we use *i* and *j* to represent the agents instead of a_i and a_j. For all the agents, $i^1 \equiv (i = reliable), i^0 \equiv (i = unreliable)$. In this part, the source agent and target agent is represented as *s* and *t*, while other agents can be represented as *i* and *j*. From the Trust Network, we can get conditional probability as follows:

$$P(j^1 \mid i^1) = V_{i,j}; P(j^0 \mid i^1) = 1 - V_{i,j} \tag{4.2}$$

$$P(j^1 \mid i^0) = \min(\frac{1}{2}, P(i^1)); P(j^0 \mid i^0) = 1 - P(j^1 \mid i^0) \tag{4.3}$$

$$P_0 = \begin{cases} V_{s,t} & truster\ \ has\ \ direct\ \ trust\ \ value\ \ of\ \ trustee \\ 0.1 & otherwise \end{cases} \tag{4.4}$$

The reasons for regularization like this have been given in the introduction part. We still need to point out that when the agent does not tell the truth, say, does not tell the true interaction history of another agent, the trust value will be the conditional trust value given the speaking agent tells a lie.

Probability Propagation

Step 1: evaluating the nodes whose only child is the target node.

Along each chain, we use the following method to find out the marginal probability of the tail agent. After obtaining the prior probability of one node, its parent node can be removed from the chain. Thus, at the end of this step, all the nodes left are the

target node and its parents. Figure 4.2 is an example of Bayesian Network and the node D's prior probability can be calculated using the formulas below.

Figure 4.2 The Bayesian Inference of Prior Probability

We can calculate the marginal probability of node D in Fig 4.2 by using the following formulas.

$$P(B) = \sum_{A} P(B|A)P(A); P(C) = \sum_{B} P(C|B)P(B); P(D) = \sum_{c} P(D|C)P(C) \qquad (4.5)$$

Step 2: Evaluating the probability of the target node.

After finishing step 1, the Trust Network becomes a converging connection Bayesian Network in which there is only one child node which is trustee and *n* parents nodes which are the agents who have functional trust interaction with the trustee. As we know, after the parallelization, each chain's tail node becomes the parent of the trustee. Each tail node carries the whole chain's information and it is not its own. In Figure 4.3, examples are listed.

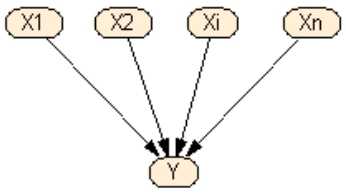

Figure 4.3 Converging Connection Bayesian Network. i=1,2…n.

Let $P_i = P(Y \mid X_i)$. From Neapolitan [Neapolitan, 1990], we can get the formula to compute the trust value of the trustee within converging connection Bayesian Network:

$$P(Y) = \sum_{a_1=0}^{1} \sum_{a_2=0}^{1} \ldots \sum_{a_n=0}^{1} P(Y \mid x_1^{a_1}, x_2^{a_2}, \ldots, x_n^{a_n}) P(x_1^{a_1}) P(x_2^{a_2}), \ldots, P(x_n^{a_n}) \tag{4.6}$$

In order to find out $P(Y \mid X_1, \ldots X_k)$, we introduce the leaky noisy-OR model to fix it. The noisy-OR model is the most accepted and widely applied model to solve the multi-causal interactions network and it leads to a very convenient and widely applicable rule of combination. However, the noisy-OR model is based on two assumptions: accountability and exception [Pearl, 1988]. Accountability states that an event can be presumed false if all its parents are false. Exception requires that the influence of each parent on the child is independent of other parents.

In our case, both of these two assumptions can work. On one hand, the leaky noisy-OR model releases the accountability assumption and it introduces a leak probability P_0 which is the probability that the effect will be produced by the unmodeled causes in absence of all the modeled causes. In other words, we can say P_0 is the prior probability of the effect before modeling the problem. Thus, in the Trust Network, we assume the truster's trust value on the trustee is P_0. On the other hand, each chain's information is independent to each other and it is only affected by the agents on the chain.

According to Figure 4.3, we can use the following formula [Henrion, 1989] to solve the problem.

$$P(y^0 \mid X_1, X_2, ..., X_n) = (1 - P_0) \prod_{i:X_i = x_i^1} \frac{1 - P_i}{1 - P_0} \tag{4.7}$$

$$P(y^1 \mid X_1, X_2, ..., X_n) = 1 - (1 - P_0) \prod_{i:X_i = x_i^1} \frac{1 - P_i}{1 - P_0} \tag{4.8}$$

Integrate formula 4.7 and 4.8 into formula 4.6, we can get:

$$P(y^1) = \sum_{a_1=0}^{1} \sum_{a_2=0}^{1} ... \sum_{a_n=0}^{1} P(x_1^{a_1}) P(x_2^{a_2}), ... P(x_n^{a_n}) [1 - (1 - P_0) \prod_{i:X_i = x_i^1} \frac{1 - P_i}{1 - P_0}] \tag{4.9}$$

4.2 Numerical Example

In order to describe the whole evaluation process more specifically, we give the following example in this section. The original Trust Network is shown in Figure 4.4. There are six witnesses and four of them have functional trust of the trustee. The truster has direct experience of the trustee as well.

(1) Parallelization

The parallelization Trust Network can be seen in Figure 4.5. In this example, suppose we only keep the chains which include three or less than three witnesses. The agent 5 and agent 3's interaction is too old and this link should be deleted. The revised parallelization network is shown in Figure 4.6.

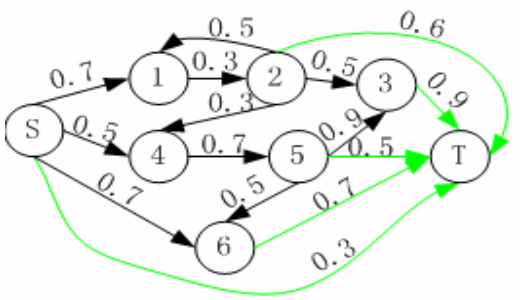

Figure 4.4 Trust Network with trust values

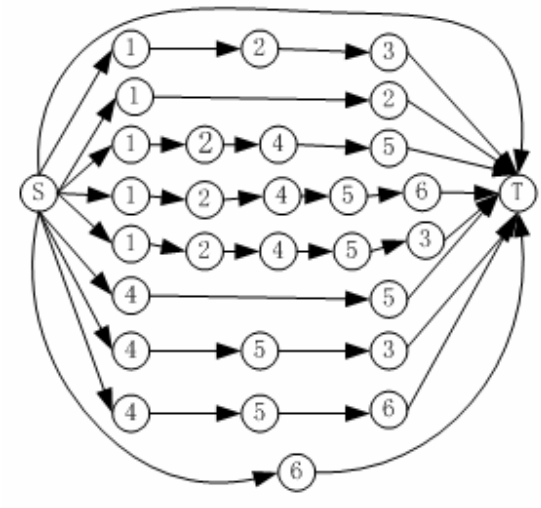

Figure 4.5 Parallel network of example Trust Network

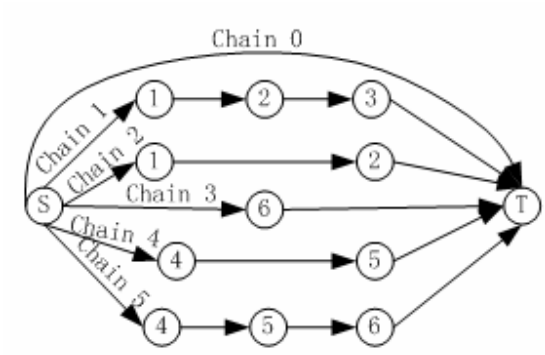

Figure 4.6 Revised parallel network of example Trust Network

(2) Evaluation

Step 1: Evaluation of the target agent's parents.

We solve the probability chain by chain and get the results presented in Table 4.1:

Table 4.1 The prior probability of the trustee's parents on each chain

Chain	1	2	3	4	5
Probability	0.41	0.36	0.7	0.6	0.5

Step2: Evaluation of the target agent.

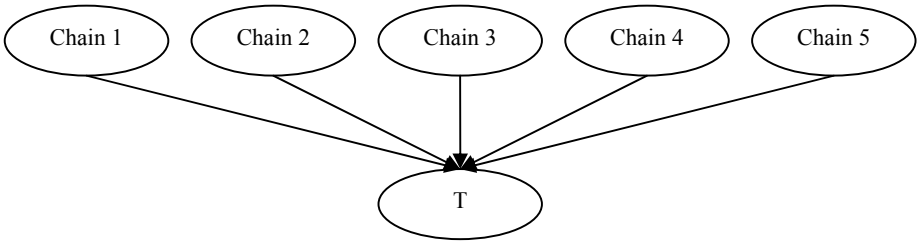

Figure 4.7 Target agent and its parents in the parallelized Trust Network

After obtaining the prior probability of the target agent's parents, we get the Bayesian Network as shown in Figure 4.6. In this step, we use formula (4.9) to get the final result which is 0.8637.

From the example, we can see that although the trust value is only 0.3 from the direct interaction between truster and trustee, the final trust value after evaluation is 0.8637. The truster might trust the trustee this time. However, if the trustee cheats the truster this time, the truster will adjust its records about the trustee and the witnesses, say, records as unsuccessful. After a while, the deceitful agents will be isolated out and isolated from the environment gradually.

In the following chapter, experiments and results will be given to illustrate our mechanism's performance.

5 EXPERIMENTS AND RESULTS

In order to empirically evaluate the approach we have proposed, we designed a test-bed that simulates the relationships and interactions between agents in which trust models are used for selecting interaction partners (see Section 5.1). The test-bed's environment characterizes an open multi-agent system. The methodology used for the evaluation is described in Section 5.2.

5.1 Experimental System

The test-bed environment we designed for evaluating our approach is a multi-agent system consisting of agents providing service (called providers) and agents using those services (called consumers). We assume that the performance of a provider (and effectively its trustworthiness) in a particular services it provide (e.g. news service) is generally independent from that in another services (e.g. weather service or banking service). Therefore, without loss of generality, and in order to reduce the complexity of the test-bed's environment, it is assumed that there is only one type of service in the test-bed. Hence, all the provider agents offer the same service. However, their performance (i.e. the quality of the service) differs and determines the utility that a consumer gains from each interaction (called UG).

The agents are situated randomly on a spherical world whose radius is 2.0 (see Figure 5.1). Each agent has a radius of operation (depicted by a dotted circle around an agent

in Figure 5.1) that models the agent's capability in interacting with others (e.g. the available bandwidth or the agent's infrastructure) and any agents situated in that range are the agent's acquaintances.

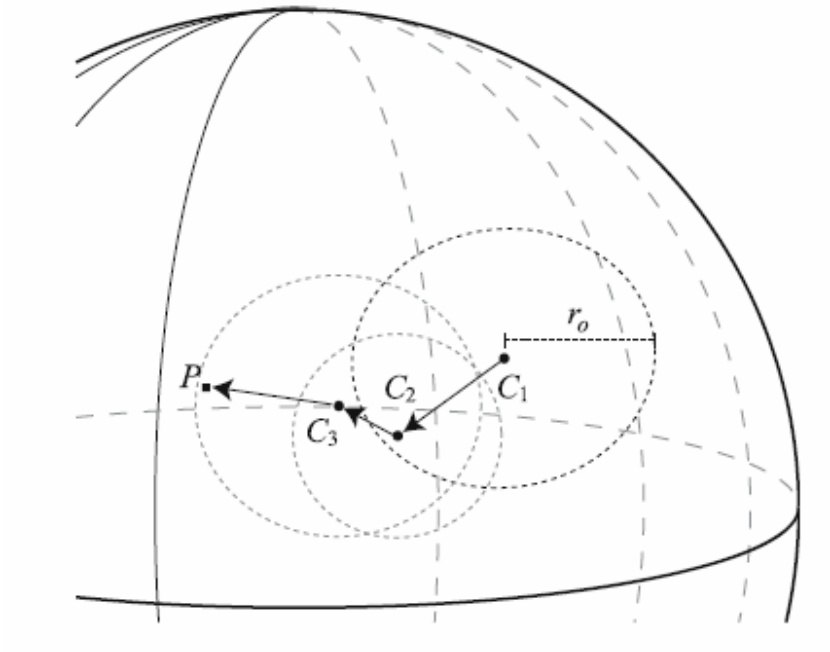

Figure 5.1 The spherical world and an example referral chain from consumer C_1 (through C_2 and C_3) to provider P via acquaintances

For a provider, its radius of operation serves as the normal operational range in which it can provide its service at its full capacity without loss of quality. For consumers outside that provider's normal operational range, the quality of service they receive from it is gradually reduced. This simulates the phenomenon that each agent usually has particular circumstances which affect service delivery. For example, two distant agents may experience significant network latency during their interaction, or a seller

agent in Singapore may charge another agent extra for shipping goods abroad and the goods may arrive much later than usual.

Simulations are run in the test-bed in rounds (of agent interactions). Event that takes place in the same round are considered simultaneous. The round number is used as the time value for events. In each round, if a consumer agent needs to use the service it can contact the environment to locate nearby provider agents (in terms of the distance between the agents on the spherical world). The consumer agent will then select one provider from the list to use its service. The selection process relies on the agent's trust model to decide which provider is likely to be the most reliable. However, consumer agents without a trust model randomly select a provider from the list.

On the other hand, each agent with a trust model would face the following problem: Not all the providers' trustworthiness located by the environment can be determined. Thus, under some situations, a consumer faces two options:

1. Selecting the provider with the highest trust value in the set *HasTrustValue*, which according to the trust model is likely to yield the highest UG.
2. Selecting a random provider from the set *NoTrustValue*, allowing it to learn about the performance of an unknown provider.

When the set *NoTrustValue* is empty, the agent only chooses according to option 1 and vice versa.

When the two sets are not empty, there is a dilemma strategy named *exploit-vs-explore* that can be used to help the agent make a decision. Using this strategy, an agent tends to explore its environment first and then gradually move its stance towards exploitation when it learns more about the environment.

Having selected a provider, the consumer agent then uses the service of the selected provider and gains some utility from the interaction (UG). The value of UG is in [-10, 10] (see Table 5.1) and depends on the level of performance of the provider in that interaction. A provider can serve many users at a time. As in real situations, a consumer agent, however, does not always use the service in every round. The probability it needs and requests the service, called its *activity level* and denoted by α, is selected randomly when the consumer is created. In other words, the activity level of a consumer determines how frequently it uses the service.

In our test-bed, the only difference in each situation is the performance of the provider agents. We consider five types of provider agents: best, good, ordinary, bad and worst. Each of them has a mean level of performance, denoted by μ_p. Its actual performance follows a normal distribution around this mean. The values of μ_p and the associated standard deviation of these types of providers, denoted by σ_p, are given in Table 5.2.

Since agents can freely join and leave an open Multi-agent system, the agent population can be very dynamic. Moreover, since agents are owned and controlled by

various stakeholders, the performance of an agent may not be consistent over time. Therefore, in order to simulate such dynamism, we introduce the following factors in our test-bed:

- The population of the agents: In an open multi-agent system, agents can come and leave the system at anytime. This is simulated by removing a number of randomly selected agents from the test-bed and adding the new ones into it. The numbers of agents added and removed after each round vary, but have an upper limit of some predefined percentage of the whole population. Since providers are usually more established than consumers, the characteristics of the newly added agents are set randomly but they are uniformly distributed over the initial agent populations.
- The location of the agents: During their life cycle, agents break the old relationships and make the new ones. In our test-bed, this type of change is described by the change in an agent's location on the spherical world. When a consumer changes its location, it will have a new set of acquaintances according to its r_0. In addition, the location of an agent in the test-bed also reflects its individual situation covering things such as its knowledge about other local agents and the service delivery between providers and consumers. Therefore, changing an agent's location will change its relationships with others, as well as its individual situation.
- The behavior of the providers: In many environments, provider performance may alter over time. A provider may even change its behavior completely. In our test-bed, the average performance of a provider can be changed by an

amount of $\Delta\mu$ randomly selected in [-M, +M], and this happens in each round with the probability of $P_{\mu c}$.

Table 5.1 Performance level constants

Performance Level	Utility gained
PL_PERFECT	+10
PL_GOOD	+5
PL_ ORDINARY	0
PL_BAD	-5
PL_WORST	-10

Table 5.2 Profiles of provider agents (performance constants defined in Table 5.1)

Profile	Range of μ p	σ p
Best	[PL_GOOD, PL_PERFECT]	1.0
Good	[PL_ ORDINARY, PL_GOOD]	2.0
Ordinary	[PL_BAD, PL_ ORDINARY]	2.0
Bad	[PL_WORST, PL_BAD]	2.0
Worst	[PL_WORST, PL_BAD]	1.0

The changes on the test-bed's environment are applied only after each round of interaction has finished. The nature and degree of dynamism are specified in each experiment.

5.2 Experimental Methodology

In each experiment, the test-bed is populated with provider and consumer agents. Each consumer agent is equipped with a specific trust model, which helps it select a provider when it needs to use a service. Since the only difference among consumer agents is the trust model that they use, the utility gained by each agent through simulation will reflect the performance of its trust model in selecting reliable providers for interactions. Therefore, the test-bed records the total utility gained (TUG) of the whole consumer environment along with the trust model used. In order to obtain an accurate result for performance comparisons between trust models, each model will be employed by a large number of consumer agents (N_c). In addition, the total utility gained of the whole environment will change over time with different trust model. The result of an experiment is presented in a graph with the y-axis, ploting the TUG of the whole environment and the x-axis ploting the interaction by time.

The experimental variables are presented in Table 5.3 and their values will be used in all cases unless otherwise specified. Although a 'typical' provider population may differ in various applications, the space of possibilities is vast and exploring it

completely would be impossible. Therefore, we choose provider populations which we believe are more common than others for our experiments. Here, we consider a typical provider population to consist of even providers. That is to say, the number of providers with different performance is the same.

As discussed in Section 4.1, the calculation of the trust value of the target agent is accomplished by using Noisy-OR model. In this model, one of the important parameters is the leaky probability $P_{0.}$ In our experiment, P_0 can be seen as the source of agent's prior trust value on the target agent without any information. As we know, we have different first impressions to different people. The P_0 is the first expression without any rational thinking. Thus, P_0 is created randomly from (0, 0.2].

Table 5.3 Experimental variables

Simulation Variable	Symbol	Value
Number of simulation rounds	N	300
Total number of provider agents	N_p	200
Best providers	N_{pb}	40
Good providers	N_{pg}	40
Ordinary providers	N_{po}	40
Bad providers	N_{pd}	40
Worst providers	N_{pw}	40
Number of consumer agents in each group	N_c	200
Range of consumer activity level	α	[0.25, 1.00]

5.3 Results

Having presented the test-bed and the proposed methodology in this section, we will evaluate the experiments. In particular, we concentrate on the benefit of using our approach for selecting interaction partners with different provider populations and the comparison of our approach with the computational model proposed by [Mui, 2002] as well as without trust model (Section 5.3.1). In addition, we also compare the estimation with combining recommendations to without combining recommendations (Section 5.3.2). Moreover, we test our model under the dynamic environment as well (Section 5.3.3).

5.3.1 Overall Performance of Bayesian-based Inference Approach

In order to evaluate the overall performance of our approach, we compare it with the computational model proposed by [Mui, 2002] (whose operation is described in Section 2.4) and a group of agents with no trust model. Hence, there are three groups of consumer agents: Bayesian-based Inference Approach (BTM), Mui-Proposed Approach (MTM) and NoTrust. Through out the whole chapter, we call the approach proposed by us as BTM, and the approach proposed by Mui as MTM.

The first thing to test is whether BTM helps consumer agents select profitable providers from the population and, by so doing, helps them gain better utility than

without BTM. In this section, the test-bed's environment is static. The NoTrust group selects providers randomly without any trust evaluation. To compare each model, we use the total utility which can be calculated as follows: each agent in the environment has its utility gained from the interaction. In one time slot, the summation of all the agents' utility gained in the whole environment can be seen as the total utility gained (TUG) in that time slot.

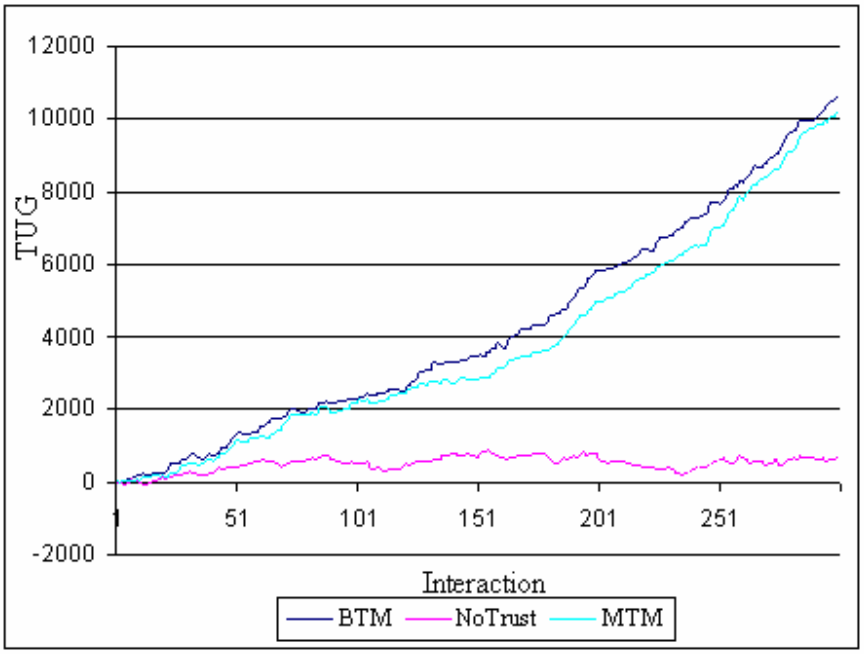

Figure 5.2 Performance of BTM, MTM and NoTrust Model

Figure 5.2 shows that the NoTrust group that selects providers randomly without any trust evaluation, performs consistently the lowest (as we would expect). On the other hand, both the BTM and MTM prove to be beneficial to consumer agents, helping them to obtain significantly higher utility. This shows that the tested trust models can

learn about the provider population, and allow their agents to select profitable providers for interaction.

In addition, from Figure 5.2, we can also see that the total utility gained by using BTM is higher than using MTM, i.e., the BTM outperforms MTM. We can get the same result from Table 5.4. In Table 5.4, we can also see that in the first few interactions, BTM can learn about the providers quicker than MTM as the BTM group achieves its superiority from the first interaction quicker than MTM. The total utility gained by BTM in each interaction after interaction 3 is higher than what MTM gained. Although there is fluctuation in the first 6 interactions in BTM, the fluctuation occurs in MTM during the first 10 interactions. This situation illustrated that BTM can reach stable situation quicker than MTM.

Table 5.4 The Performance of BTM and MTM in the first 10 interactions

Interaction	1	2	3	4	5	6	7	8	9	10
BTM	-10	-40	55	50	75	60	80	155	175	195
MTM	-90	-30	30	25	40	55	45	55	60	35

The performance difference of BTM and MTM is that BTM is accounted by the way to calculate the trust value along each chain, while MTM uses the Bayesian estimate rating propagation, which is given by

$\rho_{ik}(c) = \rho_{ij}\rho_{jk} + (1-\rho_{ij})(1-\rho_{jk})$. As we have explained before, the requirement of this method is too strict to get the final trust value of the source agent to the target agent along one chain. Thus, the agents using MTM obtain the lower total utility than using BTM. In the meantime, the MTM uses using the weighted sum to figure out the last trust value, but the approach may introduce rating noise by giving different weights to different trust chains in the Trust Network. In contrast, BTM introduces Bayesian inference propagating approach to solve the Trust Network, and the trust value along each chain is calculated by the approach used to solve Bayesian Networks. It avoids the risk of repeatedly using the trust value of each agent in one chain. In addition, the BTM leaves the weighted sum/ multiple method out drastically. Thus, there is no rating noise caused by giving weights to each chain.

We repeated the same experiment but with the provider population consisting of providers of only one profile (e.g. best, good, ordinary, bad, and worst) to see how different types of providers may affect the BTM performance. These experiments aim to test the stability and consistency of BTM. From Figure 5.3, we can see that the total utility gained with 100% best providers and 100% worst providers are symmetrical and the only difference is that one is positive and the other is negative. Similar observations are also obtained for the good and bad providers. This demonstrates that our model is very stable and consistent when solving the trust evaluation problem and also our model can work well in a wide range of provider population.

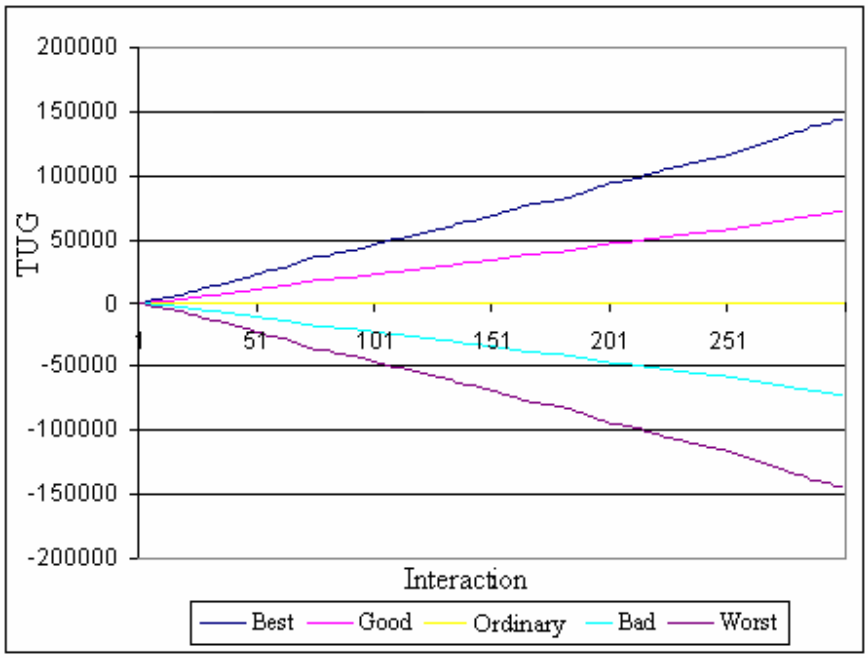

Figure 5.3 Performance of BTM with different providers

5.3.2 Comparison of with and without Combining Recommendations

We argued that the witnesses' recommendation is crucial in the trust evaluation process. However, many of the authors only take into consideration of the direct interaction when it is available. Unless there is no direct interaction, they will use recommendations. In BTM, we not only consider the direct interaction (direct experience), but also the recommendations from witnesses. As we know, only one information resource may not reflect the real situation of the target. Thus, whether the direct interaction is available or not, we take into consideration of all the information about the target agent. Figure 5.4 shows that the total utility gained by only using direct experience and by using all the information. From Figure 5.4, we can see that BTM outperforms the model that only uses the direct experience. The experiment

demonstrates that the witnesses' recommendations make the estimation more accurate than without the recommendations. In some real situations, the agent has to pay for the information. Thus, there is a tradeoff between the charge and the income.

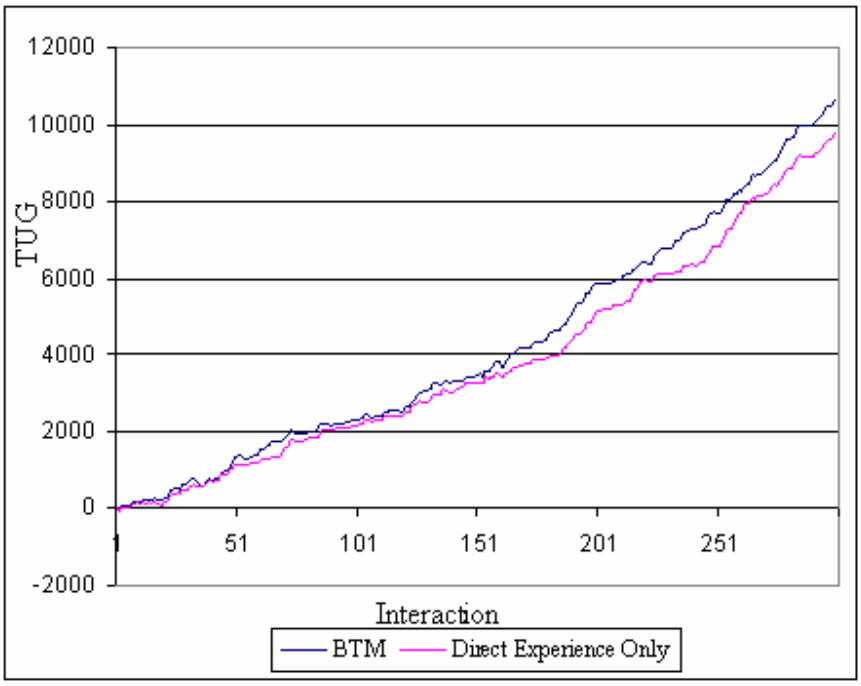

Figure 5.4 The Total Utility Gained by using direct experience only and by BTM.

5.3.3 The effects of dynamism

The environment of a realistic open multi-agent system is always changing because of its openness. Hence, a trust model designed for open multi-agent systems should be able to function properly in such a dynamic environment. This section concentrates

on testing the hypothesis that our model still maintains its properties in a changing environment.

Similarly to the experiments in Section 5.3.1, we compare the performance of BTM, MTM, Direct Experience Only and NoTrust in the same changing environment. The higher the total utility gained, the better the trust model works. We run the experiments with the following conditions:

1. The provider population changes at a maximum of 5% for every 50 rounds.
2. A provider switches into a different (performance) profile randomly for every 50 rounds.
3. A provider moves to a new location in the spherical world at a randomly selected direction and distance for every 50 rounds.

In the following parts, we will give some details of the experiments and the results.

Provider population changes

The experiment carried out under this condition aims to simulate the situation that the agent may come in and go out of the environment freely. In this experiment, we make the total number of agents in the environment unchanged. We simulate this situation through the following method: in every 50 rounds, we pick out 10 agents and add in 10 agents as the newcomers. The trust value of the newcomer is set randomly.

The results are shown in Figure 5.5. From the figure, we can see that BTM gained more utilities than other models. It demonstrates that our model can work well in the open multi-agent environment. Although we keep the total number of agents constant, this assumption will not affect our model's performance when the total number of agents changes.

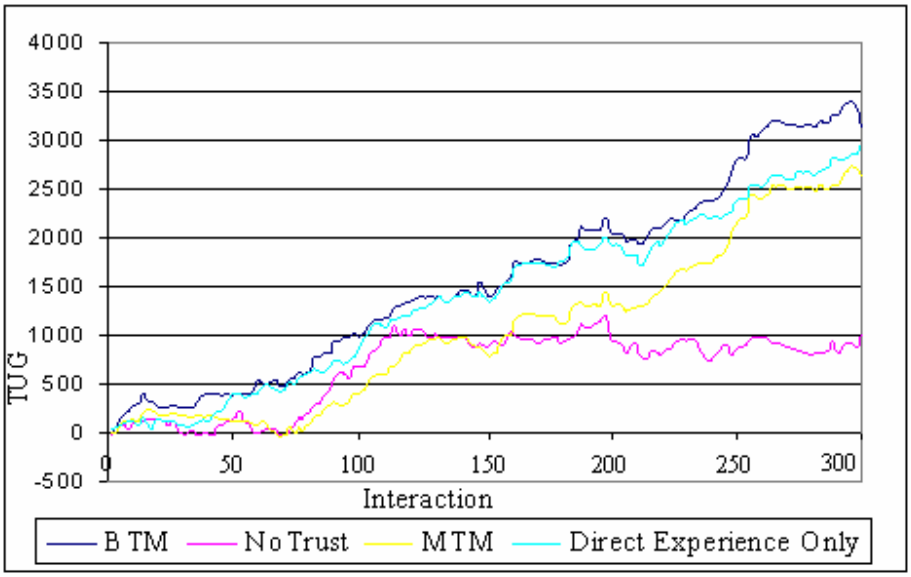

Figure 5.5 The performance of the four models under condition 1

Providers switch into a different performance profile

The experiment here is to simulate the situation that a provider's performance would change or be different in some rounds. We change some providers' performance in every 50 rounds. If we take $t_1, t_2, ...t_n$ as the time points that the provider's performance is changed, in each time point the providers picked out to change their performance are not the same. This will make the experiment more similar to the actual situation.

The results are shown in Figure 5.6. From the results, we can see that BTM dominates other models. At the same time, the TUG gained by the model with direct experience only is higher than what MTM gained; this demonstrates that our approach is more efficient than MTM. Although using the direct experience only, our model still works well in the dynamic environment where the providers change their performance.

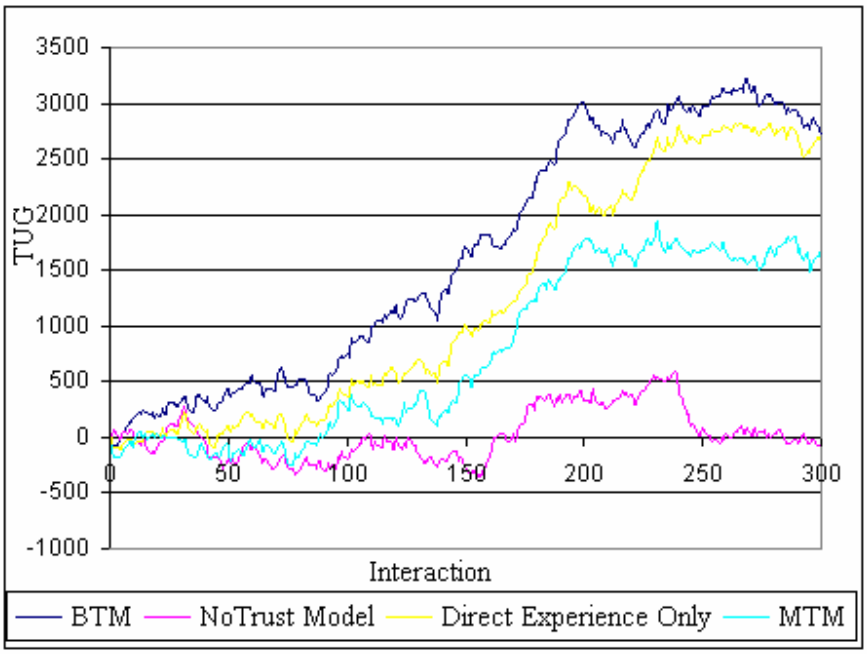

Figure 5.6 The performance of the four models under condition 2.

Providers move to a new location

The experiment aims to simulate the situation that the agent breaks the old relationship and constructs the new ones in some rounds. We change some providers' location in every 50 rounds. If we take $t_1, t_2, ...t_n$ as the time points the provider's location is changed, in each time point the providers picked out to change their location are not the same. This will make the experiment more similar to the actual situation.

The results are shown in Figure 5.7. From the results, we can see that BTM gained more utility than other models. We can obtain the same results as before. The TUG

gained by the model with direct experience only is higher than what MTM gained; this demonstrates that our approach is more efficient than MTM. Although only using the direct experience, our model still works well in the dynamic environment where the providers change their performance.

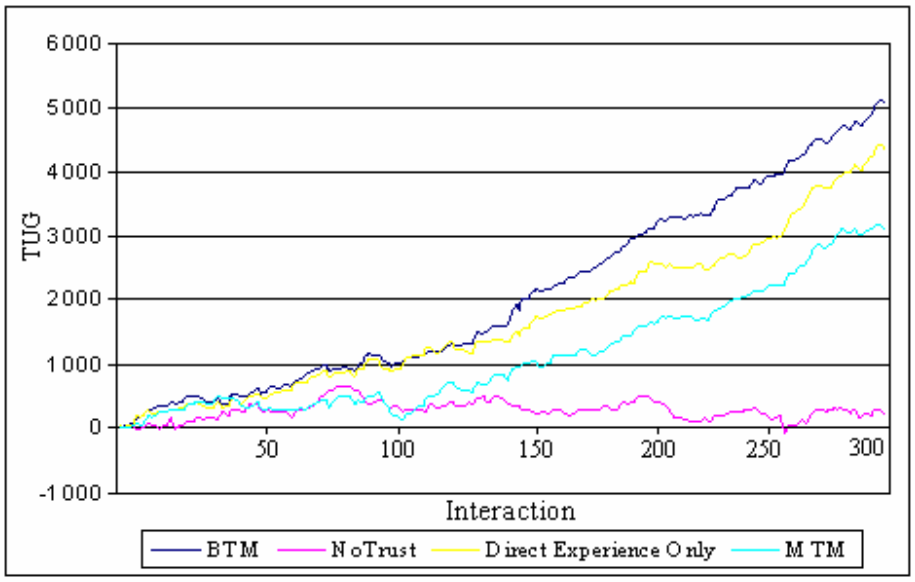

Figure 5.7 The performance of the four models under condition 3.

In summary, dynamism, as it introduces noise to the environments, adversely affects the performance of BTM and MTM in all the experiments reported here. Specifically, and as what we would expect, their performance is lower than that in the static environment. Nevertheless, although having lower levels of performance than in a static environment, the BTM still outperforms other models.

5.4 Summary

From the simulation results, we can see that the trust estimation approach we proposed works well in a wide range of providers. In the meantime, our approach outperforms to the approach proposed by Mui, as well as the mechanisms without trust evaluation mechanism. In addition, we demonstrate that the use of the direct experience only is not good enough to evaluate the trust value. However, combing in the direct experience and the witnesses' recommendation is a better way to evaluate the target agent's trust value and it gained more in the long run. Moreover, our model performs better in the dynamic environment than other models and this conclusion is confirmed by the fact that in our approach that uses direct experience only, the gained utilities are higher than that of MTM and NoTrust.

6 Conclusions and Future Work

This chapter concludes this thesis with a summary of the accomplishments and the future work.

6.1 Summary of Contributions

To fill the research gap of the trust estimation we found in the open multi-agent system, we present a new approach to estimate the trust value for the multi-agent systems. This work aims to introduce the trust evaluation problem into the artificial intelligence area and solve the problem by using the methodology in the Bayesian Network area.

We present an approach to help agents construct a trust network automatically in a multi-agent system. Although this network is a virtual one, it can be used to estimate the trust value of the target agent.

The second part is to solve the trust network constructed by our methodology. In this part, we use the Bayesian Inference Propagation approach with Leaky Noisy-OR model to solve the trust network. This is a novel way to solve the trust problem in the multi-agent systems. This approach solves the trust estimation problem based on objective logic, which means that there is no subjective weight setting. The whole trust estimation process becomes automatic without the intervention of human beings.

Lastly, we demonstrate the advantage of our approach by carrying two groups of experiments in the simulation. Experiments in group one are in the static environment and experiments in group two are under dynamic environment. From the experiments, we find out that our model works better than the model proposed by other authors as by using our model, the whole agents' utility gained is higher than by using other models (MTM and without trust measure). In addition, we run the experiment in different provider situations. The results tell us that our model performs well in a wide range of provider population and it also reconfirmed the fact that our model works better than the models we compared. Moreover, in order to demonstrate that more information resources can help the decision maker make a more accurate decision, we develop another experiment which compares the performance of our model and the model that only uses direct experience unless the direct experience is not available. The results of the experiment confirm our proposed viewpoint, which is that combining the recommendation and the direct experience works better than using direct experience. In addition, we test our approach in the dynamic environment and the results illustrate that our approach gains more TUG than MTM and without trust measure under three different dynamic environments.

By using the framework we proposed, the trust estimation problem can be solved in a new way which is based on Bayesian Propagation. Comparing to the existing methods, this approach can solve the trust problem in the virtual communities

automatically and the results are more reliable which could help agents in the environment obtain more earnings.

6.2 Recommendations for Future Work

From the simulation results, we find that our model works better. However, we need more studies on the trust modeling and trust evaluation can be done.

There are several possible future works that can be done to improve our work:

1) The assumption of the trust value given the witness tells a lie needs to be relaxed or to find a better way to solve this problem.
2) In this work, all the trust values proposed are all falling into the confidence level. However, in some problems, witnesses' information might be out of the confidence level and the chain will be broken, and it needs to be improved in the Trust Network construction part.
3) The approach we proposed has a disadvantage, which is the increase of computation time as the complexity of the trust graph increases. The more of the quantity the chains, the more computation that needs to be done.

In addition, we may also improve our work by modeling the trust value in the Bayesian Network. It is possible to construct a Bayesian Network to expect the trust value or the target agent's intention in the future interaction.

In summary, this thesis presents an overview of trust estimation approaches and develops a trust network construction algorithm, as well as a mechanism to estimate the trust value in a trust network. More research and application of such approaches need to be followed in the future.

REFERENCES

Abdul-Rahman, A. and Hailes, S. (1997). A Distributed Trust Model. New Security Paradigms Workshop, Langdale, Cumbria UK.

Abdul-Rahman, A., and Hailes, S., Supporting Trust in Virtual Communities, in: Proceedings of the 33rd Hawaii International Conference on System Sciences (IEEE Computer Society, 2000).

Aberer, K. (2001). P-Grid: A self-organizing access structure for P2P information systems. 9th International Conference on Cooperative Information Systems, Trento, Italy.

Aberer,K. and Despotovic, Z. (2001) Managing Trust in a Peer-to-Peer Information System. Conference on Information and Knowledge Management, Atlanta, Georgia.

Barber, K. B. and Kim, J., Belief Revision Process based on Trust: Agents Evaluating Reputation of Information Sources, in: Lecture notes in computer science 2246 (Springer, 2001) 73-82.

Becker, M.Y. and Sewell, P., Cassandra: Distributed access control policies with tunable expressiveness. In Proceedings of the 5th IEEE International

Workshop on Policies for Distributed Systems and Networks, Pages 159, 2004.

Blaze, M., et al., Decentralized trust management. In Proceedings of IEEE Symposium on Security and Privacy, pages 164-173, 1996.

Blaze, M., et al., The role of trust management in distributed system security. Lecture Notes in Computer Science, 1603: 185-210, 1999.

Damiani, E., di Vimercati, S. D. C. et al. (2002) A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. 9th ACM Conference on Computer and Communication Security, Washington DC.2002.

Despotovic, Z., Aberer, K., P2P reputation management: Probabilistic estimation vs. social networks. Computer Networks 50(2006), 485-500.

Dragovic, B., Kotsovinos, E., et al (2003) Xeno Trust: Event-based distributed trust management. Second International Workshop on Trust and Privacy in Digital Business, Prague, Czech Republic.

Esfandiari, B. and Chandrasekharan, S. (2001) On How Agents Make Friends: Mechanisms for Trust Acquisition. 4th Workshop on Deception, Fraud and Trust in Agent Societies, Montreal.

Falcone R. and Shehory O., Trust Delegation and Autonomy: Foundations for Virtual Societies". AAMAS tutorial 12, July 16, 2002.

Falcone, R., et al., Why a cognitive trustier performs better: Simulating trust-based contract nets. In Proceedings of AAMAS 2004.

Freeman, L.C., Centrality in Social Networks: I. Conceptual Clarification, Social Networks, 1:215-239, 1979.

Guo, L., Poh, K.L., and Li, G.L., Trust Estimation within Trust Network for Multi-agent Systems: A Bayesian Propagation Approach, Submitted to The Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM), 2007.

Henrion, M., Some Practical Issues in Constructing Belief Networks, Uncertainty in Artificial Intelligence 3, Amsterdam: North-Holland, (1989).

Huynh, T. D., Jennings, N.R. and Shadbolt, N. R. "An intergrated trust and reputation model for open multi-agent systems", Auton Agent Multi-agent Sys (2006) 13: 119-154.

Jøsang, A. and Pope, S. "Semantic Constraints for Trust Transitivity", In Proceedings of the Asia-Pacific Conference of Conceptual Modeling (APCCM) (Volume

43 of Conference in Research and Practice in Information Technology), S. Hartmann and M. Stumptner, Eds., Newcastle, Australia, February, 2005.

Jøsang, A., Hayward, R., and Pope, S. Trust Network Analysis with Subjective Logic. Australasian Computer Science Conference 2006.

Jøsang, A. et al. (2006) "*Simplification and Analysis of Transitive Trust Networks"* Web Intelligence and Agent Systems Journal 4(2) 2006, pp.139-161.

Kagal, L. et al, Developing secure agent systems using delegation based trust management. In Security of Mobile Multi-agent Systems (SEMAS 02) held at Autonomous Agents and MultiAgent Systems (AAMAS 02), 2002.

Katz, L., New Status Index Derived from Sociometric Analysis. Psychometrika, 18:39-43, 1953.

Krackhardt, D., et al., KrackPlot: A Picture's Worth a Thousand Words. Connections, 16:37-47, 1993.

Lee, S., Sherwood, R., et al. (2003) Cooperative peer groups in NICE. IEEE Infocom, San Francisco, USA.2003.

Leithead, T., et al., How to exploit ontologies for trust negotiation. In ISWC Workshop on Trust, Security, and Reputation on the Semantic Web, Volume 127 of CEUR Workshop Proceedings, Hiroshima, Japan. Technical University of Aachen (RWTH), 2004.

Marsden,P.V. and Lin, L., Social Structure and Network Analysis, Newbury Park, CA: Sage, 1982.

Marsh, S. P., Formalising trust as a computational concept, Department of Mathematics and Computer Science, University of Stirling, 1994.

McKnight, D. and Chervany, N. The meaning of trust. University of Minnesota, Management Information Systems Research Center, Tech, Rep. MISRC Working Paper Series 96-04, 1996.

Montaner M. and L'opez B., Opinion based filtering through trust. In Proceeding of the 6th International Workshop on Cooperative Information Agents (CIA'02), Madrid (Spain), September 18-20, 2002.

Mui, L. (2002) Computational models of Trust and Reputation: Agents, Evolutionary Games, and Social Networks. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology. 2002.

Neapolitan, R.E., Probabilistic reasoning in expert systems: theory and algorithms (Wiley, New York, 1990).

Nejdl, W., et al., Peertrust: Automated trust negotiation for peers on the semantic web. In Proceedings of Workshop on Secure Data Management in a Connected World in conjunction with the 30th International Conference on Very Large Data Bases, pages 118-132, 2004.

Nurmi, P., "Bayesian game theory in practice: A framework for online reputation systems", Department of Computer Science, Series of Publications C, Report C-2005-10, University of Helsinki, Finland, 2005.

Patel, J., W.T.L. Teacy, N.R. Jennings, M. Luck, A Probabilistic Trust Model for Handling Inaccurate Reputation Sources, in: 3rd Int. Conf. on Trust Management (Roquencourt, France, 2005) 193-209.

Pearl .J, Probabilistic Reasoning in Intelligent Systems (Morgan Kaufmann, San Francisco, California, 1988).

Pujol, J., Sanguesa, R., et al. Extracting reputation in multi-agent systems by means of social network topology. 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy. 2002.

Sabater,J. and Sierra, C. REGRET: A Reputation Model for Gregarious Societies. 4th Workshop on Deception, Fraud and Trust in Agent Societies, Montreal, Canada. 2001.

Schillo, M., Funk, P., et al. Using trust for detecting deceitful agents in artificial societies. Applied Artificial Intelligence Journal, Special Issue on Trust, Deception and Fraud in Agent Societies. 2000.

Seigneru, J.M. and Jensen, C.D., Trust enhanced ubiquitous payment without too much privacy loss. In SAC'04: Proceedings of the 2004 ACM symposium on Applied computing, pages 1593-1599, New York, NY, USA. ACM Press. 2004.

Suryanarayana, G., et al, A survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications, ISR Technical Report# UCI-ISR-04-6, 2004.

Wang, Y.H. and Singh, M.P. *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)* July 2006.

Winsborough, W., et al., Automated trust negotiation, Technical Report, North Carolina State University at Raleigh, Raleigh, NC, USA, 2000.

Winslett, M., et al., Negotiating trust on the web. IEEE Internet Computing, 6(6): 30-37, 2002.

WS-Trust. http://www.128.ibm.com/developerworks/library/specifications/ws-trust/.

Xiong, L. and Liu, L., PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities, IEEE Transactions on Knowledge and Data Engineering, Vol.16, NO.7, July 2004.

Yu, T., Winslett, M., and Seamons, K. E., Interoperable strategies in automated trust negotiation. In CCS'01: Proceedings of the 8th ACM conference on Computer and Communications Security, pages 146-155, New York, NY, USA. ACM Press, 2001.

Yu, B. and Singh, M. P. A Social Mechanism of Reputation Management in Electronic Communities. Fourth International Workshop on Cooperative Information Agents.2000.

Yu, B., Singh, M.P., Distributed Reputation Management for Electronic Commerce, Computational Intelligence 18 (2002) 535-549.

Yu, T. and Winslett, M., Policy migration for sensitive credentials in trust negotiation. In WPES'03: Proceedings of the 2003 ACM workshop on Privacy in the electronic society, Pages 9-20, New York, NY, USA. ACM Press, 2003.

Zacharia, G., Collaborative Reputation Mechanisms for Online Communities, Dept. of Architecture. Program in Media Arts and Sciences, Massachusetts Institute of Technology, 1999.

Appendix-A Parallelization

```
Parallalization Procedure in C Sharp code
       private Array parallalization(int source, int sink)
       {
           Stack myStack = new Stack();
           ArrayList chain = new ArrayList();
       int i, j;
       int k = 0;
       bool flag = true;
       String Currentnode = source.ToString();
       i = source;
       j = 0;
       int jpre = i;
       maxlength = System.Convert.ToInt16(textBox3.Text);
           while (flag == true)
           {
              while ((j <= nodeia.GetUpperBound(1)) &
(myStack.Count < (maxlength - 2)))
              {
                 if (j != jpre)
                 {
                    if (Convert.ToInt16(nodeia.GetValue(i,
j)) != 0)
                    {
                       if (myStack.Contains(Currentnode))
                       {
                          //Here is to prevent the cycle
occurance
                          j = j + 1;
                       }
                       else
                       {
                          myStack.Push(Currentnode);

                          Currentnode = j.ToString();
                          if (Currentnode ==
sink.ToString())
                          {
                             goto a;
                          }
                          jpre = 0;
                          i = j;
                          j = 0;
                       }
```

```
                    }
                    else
                    {
                        j = j + 1;
                    }
                }
                else
                {
                    j = j + 1;
                }

            }

            //don't find the terminal chain, then back
            if (myStack.Count > 0)
            {
                jpre = Convert.ToInt16(Currentnode);
                i = Convert.ToInt16(myStack.Pop());
                Currentnode = i.ToString();
                j = jpre + 1;
            }
            else
            {
                flag = false;
            }
            goto e;

        //finaliza the chain
        a: myStack.Push(Currentnode);
            //copy all the nodes in stack into one chain()
            Object[] myStandardArray = myStack.ToArray();
            MyStringBuilder.Remove(0,
MyStringBuilder.Length);
            for (j = myStandardArray.GetUpperBound(0); j >
-1; j--)
            {
                if (MyStringBuilder.Length == 0)
                    MyStringBuilder.Append("Node" +
myStandardArray.GetValue(j).ToString());
                else
                    MyStringBuilder.Append("+" + "Node" +
myStandardArray.GetValue(j).ToString());
            }

            chain.Add(MyStringBuilder.ToString());
            k = k + 1;
```

```
            //back two steps
            jpre = Convert.ToInt16(myStack.Pop());
            i = Convert.ToInt16(myStack.Pop());
            Currentnode = i.ToString();
            j = jpre + 1;

        e: ;
        }
        return chain.ToArray();
    }
```

Appendix-B BTM Core Code

```
private double calculatetrust(int source, Array sinkarray,
int finalsink, Array degradevalueinmodel)
{
          int n, i,chainlength;
          double P0 = rand.Next(1, 2000) / 10000.0;
//p0=0.0001~0.2
          Array chain;
          double trust = P0;
          trustchain.Clear();
          for (n = 0; n < sinkarray.Length; n++)
          {
              if
(System.Convert.ToInt16(sinkarray.GetValue(n)) == source)
//source has direct functional relationship with finalsink
              {
                   trustchain.Add("sink=" +
source.ToString() + ";final sink=" + finalsink.ToString() +
";partial Trust value=" + nodefunctiontrust.GetValue(source,
finalsink).ToString());
              }
              else //source didn't have the direct functional
relationship with finalsink
              {
                  chain = parallalization(source,
System.Convert.ToInt16(sinkarray.GetValue(n)));
                  chainlength = chain.Length;
                  if (chainlength != 0)
                  {
                      for (i = 0; i < chainlength; i++)
                      {

trustchain.Add(calculatetrustvalue(chain.GetValue(i).ToSt
ring(), finalsink));
                      }
                  }
              }
          }
          if (trustchain.ToArray().Length != 0)
          {
             trust = analyzetrustchain(trustchain.ToArray(),
P0);
          }
```

```
          if
(System.Convert.ToDouble(degradevalueinmodel.GetValue(sou
rce, finalsink)) <= 0)
          {
             trust = trust *
Math.Exp(System.Convert.ToDouble(degradevalueinmodel.GetV
alue(source, finalsink)));
          }
          else
          {
             trust = (1 +
System.Convert.ToDouble(degradevalueinmodel.GetValue(sour
ce, finalsink)) / 100.0) * trust;
             if (trust > 1)
             {
                trust = 1;
             }

          }
      return trust;
}

       ArrayList analyzetrustchain1 = new ArrayList(); //to
store the sink node
       ArrayList analyzetrustchain2 = new ArrayList(); //to
store the final sink node
       ArrayList analyzetrustchain3 = new ArrayList(); //to
store the trust value
       //analyze the array of trustchain
       private double analyzetrustchain(Array mytrustchain,
double myp0)
       {
          analyzetrustchain1.Clear();
          analyzetrustchain2.Clear();
          analyzetrustchain3.Clear();
          int i=mytrustchain.Length;
          int j;
          int n;
          string analyzerstring;
          Array aplit_result =
Array.CreateInstance(typeof(double), 3);

          for (j = 0; j < i; j++)
          {
             analyzerstring =
mytrustchain.GetValue(j).ToString();
```

```
            string[] split =
analyzerstring.Split(";".ToCharArray(), 3);
            for (n = 0; n < 3; n++) //need to separate
considering the last node and the other nodes
            {

aplit_result.SetValue(System.Convert.ToDouble(split.GetVa
lue(n).ToString().Substring(1 +
split.GetValue(n).ToString().IndexOf("="))), n);
            }

analyzetrustchain1.Add(System.Convert.ToInt16(aplit_resul
t.GetValue(0)));

analyzetrustchain2.Add(System.Convert.ToInt16(aplit_resul
t.GetValue(1)));

analyzetrustchain3.Add(System.Convert.ToDouble(aplit_resu
lt.GetValue(2)));
          }
          Array myprY_Xi =
Array.CreateInstance(typeof(double),
analyzetrustchain1.ToArray().Length);
          Array mypriorXi =
Array.CreateInstance(typeof(double),
analyzetrustchain1.ToArray().Length);
          Array temp1 = analyzetrustchain1.ToArray();
          Array temp2 = analyzetrustchain2.ToArray();
          Array temp3 = analyzetrustchain3.ToArray();
          for (j = 0; j < analyzetrustchain1.ToArray().Length;
j++)
          {

myprY_Xi.SetValue(System.Convert.ToDouble(conditionalfunc
tiontrust.GetValue(System.Convert.ToInt16(temp1.GetValue(
j)), System.Convert.ToInt16(temp2.GetValue(j)))), j);

mypriorXi.SetValue(System.Convert.ToDouble(temp3.GetValue
(j)), j);

          }
          return calculatefinaltrust(myp0, myprY_Xi,
mypriorXi);
       }
```

```
       //calculate the trust between the last second node and
the final sink node
       private double calculatefinaltrust(double p0, Array
myprY_Xi, Array mypriorXi)
       {
          int i, j;
          Array prY_Xi = myprY_Xi;
          Array priorXi = mypriorXi;
          int n = prY_Xi.Length;
          int mm;
          Array parentValues =
Array.CreateInstance(typeof(double), n);
          for (i = 0; i < n; i++)
             parentValues.SetValue(0, i);

          Double configurations =
System.Convert.ToDouble(Math.Pow(2, n));
          Array CPT = Array.CreateInstance(typeof(double),
System.Convert.ToInt64(configurations), 2);
          Array priors =
Array.CreateInstance(typeof(double), 1, 2);
          Array currentProbs =
Array.CreateInstance(typeof(double), 1, 2);
          for (i = 0; i < 1; i++)
             for (j = 0; j < 2; j++)
                priors.SetValue(0, i, j);

          for (i = 0; i < configurations; i++)
          {
             CPT.SetValue((1 - p0), i, 0);
             for (j = 0; j < n; j++)
             {
                if
(System.Convert.ToInt16(parentValues.GetValue(j)) == 1)
                {

CPT.SetValue((System.Convert.ToDouble(CPT.GetValue(i, 0))
* (1 - System.Convert.ToDouble(prY_Xi.GetValue(j))) / (1 -
p0)), i, 0);
                }
             }
             CPT.SetValue((1 -
System.Convert.ToDouble(CPT.GetValue(i, 0))), i, 1);
             double prob_config = 1; //the probability of the
current parent configuration
             for (j = 0; j < n; j++)
             {
```

```
                if
(System.Convert.ToDouble(parentValues.GetValue(j)) == 1)
                {
                   prob_config = prob_config *
System.Convert.ToDouble(prY_Xi.GetValue(j));
                }
                else
                {
                   prob_config = prob_config * (1 -
System.Convert.ToDouble(prY_Xi.GetValue(j)));
                }
             }
             for (j = 0; j < 2; j++)

currentProbs.SetValue(System.Convert.ToDouble(CPT.GetValu
e(i, j)), 0, j);

             for (j = 0; j < n; j++)
             {
                if
(System.Convert.ToDouble(parentValues.GetValue(j)) == 1)
                {

currentProbs.SetValue(System.Convert.ToDouble(currentProb
s.GetValue(0, 0)) *
System.Convert.ToDouble(priorXi.GetValue(j)), 0, 0);

currentProbs.SetValue(System.Convert.ToDouble(currentProb
s.GetValue(0, 1)) *
System.Convert.ToDouble(priorXi.GetValue(j)), 0, 1);

                }
                else
                {

currentProbs.SetValue(System.Convert.ToDouble(currentProb
s.GetValue(0, 0)) * (1 -
System.Convert.ToDouble(priorXi.GetValue(j))), 0, 0);

currentProbs.SetValue(System.Convert.ToDouble(currentProb
s.GetValue(0, 1)) * (1 -
System.Convert.ToDouble(priorXi.GetValue(j))), 0, 1);

                }
             }
             for (mm = 0; mm < 2; mm++)
```

```
priors.SetValue((System.Convert.ToDouble(priors.GetValue(
0, mm)) + System.Convert.ToDouble(currentProbs.GetValue(0,
mm))), 0, mm);


            j = n - 1;

parentValues.SetValue((System.Convert.ToDouble(parentValu
es.GetValue(j)) + 1), j);
            while ((j > 0) &
(System.Convert.ToDouble(parentValues.GetValue(j)) > 1))
            {
                parentValues.SetValue(0, j);
                j = j - 1;

parentValues.SetValue((System.Convert.ToDouble(parentValu
es.GetValue(j)) + 1), j);
            }
        }
        return System.Convert.ToDouble(priors.GetValue(0,
1));
    }


    private string calculatetrustvalue(string onechain,
int finalsink)
    {
        double partialtrust;
        char[] tempy = onechain.ToCharArray();
        int nn = 0;
        double mintemp;
        foreach (char x in tempy)
        {
            if (x == 43)
                nn = nn + 1;
        }
        nn = nn + 1;
        string[] split = onechain.Split("+".ToCharArray(),
nn);
        Array aplit_result =
Array.CreateInstance(typeof(int), split.Length);
        int n;
        for (n = 0; n < split.Length; n++) //need to separate
considering the last node and the other nodes
        {
```

```
aplit_result.SetValue(System.Convert.ToInt16(split.GetVal
ue(n).ToString().Substring(4)), n);
          }

          if (split.Length==2)
          {

partialtrust=System.Convert.ToDouble(nodetrust.GetValue(S
ystem.Convert.ToInt16(aplit_result.GetValue(0)),
System.Convert.ToInt16(aplit_result.GetValue(1))));
          }
          else
          {

partialtrust=System.Convert.ToDouble(nodetrust.GetValue(S
ystem.Convert.ToInt16(aplit_result.GetValue(0)),
System.Convert.ToInt16(aplit_result.GetValue(1))));
             for (n = 1; n < split.Length-1;n++ )
             {
                mintemp = Math.Min(0.5, partialtrust);

partialtrust=System.Convert.ToDouble(conditionalreference
trust.GetValue(System.Convert.ToInt16(aplit_result.GetVal
ue(n)), System.Convert.ToInt16(aplit_result.GetValue(n +
1))))*partialtrust+(1-partialtrust)*mintemp;

             }
  //next is to calculate the function trust between last
second node to the finalsink node
          }
          if (aplit_result.Length == 2)
          {
             return "sink=" +
aplit_result.GetValue(n-1).ToString() + ";final sink=" +
finalsink.ToString() + ";partial Trust value=" +
partialtrust.ToString();
          }
          else
          {
             return "sink=" +
aplit_result.GetValue(n).ToString() + ";final sink=" +
finalsink.ToString() + ";partial Trust value=" +
partialtrust.ToString();
          }
 }
```

Appendix-C MTM Core Code

```
private double calculatetrustformodel4(int source, Array
sinkarray, int finalsink, Array degradevalueinmodel)
      {
         int n, i, chainlength;
         double P0 = rand.Next(1, 2000) / 10000.0;
//p0=0.0001~0.2
         Array chain;
         double trust = P0;
         trustchain.Clear();

         for (n = 0; n < sinkarray.Length; n++)
         {
            if
(System.Convert.ToInt16(sinkarray.GetValue(n)) == source)
//source has direct functional relationship with finalsink
            {
               trustchain.Add("partial Trust value=" +
nodefunctiontrust.GetValue(source, finalsink).ToString() +
";weight=0.5"); //for model 4
            }
            else
            {
               chain = parallalization(source,
System.Convert.ToInt16(sinkarray.GetValue(n)));
               chainlength = chain.Length;
               if (chainlength != 0)
               {
                  for (i = 0; i < chainlength; i++)
                  {

trustchain.Add(calculatetrustvalue_inmodel4(chain.GetValu
e(i).ToString(), finalsink));
                  }
               }

            }
         }
         if (trustchain.ToArray().Length != 0)
         {
            trust =
analyzetrustchain_inmodel4(trustchain.ToArray());
         }
```

```
          if
(System.Convert.ToDouble(degradevalueinmodel.GetValue(sou
rce, finalsink)) <= 0)
          {
             trust = trust *
Math.Exp(System.Convert.ToDouble(degradevalueinmodel.GetV
alue(source, finalsink)));
          }

          else
          {
             trust = (1 +
System.Convert.ToDouble(degradevalueinmodel.GetValue(sour
ce, finalsink)) / 100.0) * trust;
             if (trust > 1)
             {
                trust = 1;
             }

          }
          return trust;
       }

       private double analyzetrustchain_inmodel4(Array
mytrustchain)
       {

          int chainnumber = mytrustchain.Length;
          double i = 0.0; //return value
          double j = 0.0; //summarize all the weight
          int n_model4 = 0;
          ArrayList weight_model4 = new ArrayList();
          ArrayList trust_model4 = new ArrayList();
          for (n_model4 = 0; n_model4 < chainnumber;
n_model4++)
          {
             //analyze the number
             //"partial Trust value=" +
nodefunctiontrust.GetValue(source, finalsink).ToString() +
";weight=0.5"

weight_model4.Add(System.Convert.ToDouble(mytrustchain.Ge
tValue(n_model4).ToString().Substring(1 +
mytrustchain.GetValue(n_model4).ToString().LastIndexOf("=
"))));
```

```
trust_model4.Add(mytrustchain.GetValue(n_model4).ToString
().Substring(1 +
mytrustchain.GetValue(n_model4).ToString().IndexOf("="),
System.Convert.ToInt16(mytrustchain.GetValue(n_model4).To
String().IndexOf(";")) - 1 -
System.Convert.ToInt16(mytrustchain.GetValue(n_model4).To
String().IndexOf("=")));
          }

          Array weightarray = weight_model4.ToArray();
          Array trustarray_model4 = trust_model4.ToArray();

          for (n_model4 = 0; n_model4 < chainnumber;
n_model4++)
          {
              j = j +
System.Convert.ToDouble(weightarray.GetValue(n_model4));
              i = i +
System.Convert.ToDouble(weightarray.GetValue(n_model4)) *
System.Convert.ToDouble(trustarray_model4.GetValue(n_mode
l4));
          }
          return i / j;

      }

      private string calculatetrustvalue_inmodel4(string
onechain, int finalsink)
      {
          double partialtrust;
          char[] tempy = onechain.ToCharArray();
          int nn = 0;
          foreach (char x in tempy)
          {
              if (x == 43)
                  nn = nn + 1;
          }
          nn = nn + 1;
          string[] split = onechain.Split("+".ToCharArray(),
nn);
          Array aplit_result =
Array.CreateInstance(typeof(int), split.Length);
          int n;
          for (n = 0; n < split.Length; n++) //need to separate
considering the last node and the other nodes
          {
```

```
aplit_result.SetValue(System.Convert.ToInt16(split.GetVal
ue(n).ToString().Substring(4)), n);
          }
          if (split.Length == 2)
          {  partialtrust = 1 + 2 *
System.Convert.ToDouble(nodetrust.GetValue(System.Convert
.ToInt16(aplit_result.GetValue(0)),
System.Convert.ToInt16(aplit_result.GetValue(1)))) *
System.Convert.ToDouble(conditionalfunctiontrust.GetValue
(System.Convert.ToInt16(aplit_result.GetValue(1)),
finalsink)) -
System.Convert.ToDouble(conditionalfunctiontrust.GetValue
(System.Convert.ToInt16(aplit_result.GetValue(1)),
finalsink)) -
System.Convert.ToDouble(nodetrust.GetValue(System.Convert
.ToInt16(aplit_result.GetValue(0)),
System.Convert.ToInt16(aplit_result.GetValue(1))));
             return "partial Trust value=" +
partialtrust.ToString() + ";weight=0.75";
          }
          else
          {
             partialtrust = 1 + 2 *
System.Convert.ToDouble(nodetrust.GetValue(System.Convert
.ToInt16(aplit_result.GetValue(0)),
System.Convert.ToInt16(aplit_result.GetValue(1)))) *
System.Convert.ToDouble(conditionalreferencetrust.GetValu
e(System.Convert.ToInt16(aplit_result.GetValue(1)),
System.Convert.ToInt16(aplit_result.GetValue(2)))) -
System.Convert.ToDouble(conditionalreferencetrust.GetValu
e(System.Convert.ToInt16(aplit_result.GetValue(1)),
System.Convert.ToInt16(aplit_result.GetValue(2)))) -
System.Convert.ToDouble(nodetrust.GetValue(System.Convert
.ToInt16(aplit_result.GetValue(0)),
System.Convert.ToInt16(aplit_result.GetValue(1))));
             partialtrust = 1 + 2 * partialtrust *
System.Convert.ToDouble(conditionalfunctiontrust.GetValue
(System.Convert.ToInt16(aplit_result.GetValue(2)),
finalsink)) -
System.Convert.ToDouble(conditionalfunctiontrust.GetValue
(System.Convert.ToInt16(aplit_result.GetValue(2)),
finalsink)) - partialtrust;
             return "partial Trust value=" +
partialtrust.ToString() + ";weight=1";
          }
       }
```