Computational Intelligence, Volume 25, Number 2, 2009

FUZZY TRUST AGGREGATION AND PERSONALIZED TRUST INFERENCE IN VIRTUAL SOCIAL NETWORKS

MOHSEN LESANI AND NILOUFAR MONTAZERI
School of Computer and Communication Sciences, EPFL

Virtual marketplaces on the Web provide people with great facilities to buy and sell goods similar to conventional markets. In traditional business, reputation is subjectively built for known persons and companies as the deals are made in the course of time. As it is important to do business with trustful individuals and companies, there is a need to survive the reputation concept in virtual markets. Auction sites generally employ reputation systems based on feedbacks that provide a global view to a cyber dealer. In contrast to global trust, people usually infer their personal trust about someone whose reputation is completely or partially unknown by asking their trusted friends. Personal reputation is what makes a person trusted for some people and untrusted for others. There should be a facility for users in a virtual market to specify how much they trust a friend and also a mechanism that infers the trust of a user to another user who is not directly a friend of her. There are two main issues that should be addressed in trust inference. First, the trust modeling and aggregation problem needs to be challenged. Second, algorithms should be introduced to find and select the best paths among the existing trust paths from a source to a sink. First, as trust to a person can be stated more naturally using linguistic expressions, this work suggests employing linguistic terms for trust specification. To this end, corresponding fuzzy sets are defined for trust linguistic terms and a fuzzy trust aggregation method is also proposed. Comparing the fuzzy aggregation method to the existing aggregation methods shows superiority of fuzzy approach especially at aggregating contradictory information. Second, this paper proposes an incremental trust inference algorithm. The results show improvement in preciseness of inference for the proposed inference algorithm over the existing and recently proposed algorithm named TidalTrust.

Key words: reputation, trust, aggregation, social networks, e-market, rating.

1. INTRODUCTION

An individual usually satisfies his material and service requirements by his own or asks others. As one cannot autonomously satisfy all his needs, the human being has selected to live in society to be able to fulfill his needs through relationships. The advent of Internet has added new dimensions to traditional interpersonal relationships and indeed the way that people satisfy their needs. Web-based social network sites offer many facilities to their users to find their friends, make new ones, and specify their relationships. Millions of people are connected to each other in such social networks, making a large relationship network. People are able to satisfy much of their requirements through virtual acquaintances. Web-based services provide people with facilities to search for the items they need and to acquire them from the providers. These services have opened a new way of business that makes people able to virtually offer their goods and the goods can be browsed or be found by customers that search. The item offered can also be a service such as news and review.

When a material or a service is being purchased from other persons, it is important to have enough trust to the providers to be sure that they will deliver proper material or service. The belief that a person is trusted or untrusted usually emerges as a consequence of the person's reputation. The reputation is the past behavior, i.e., the quality of previously provided material or service. The idea is that in the future, a person will most probably continue his past behavior. Some researchers have tried to prove that the concept of reputation in a society is the incentive for selfish agents to behave honestly. Jurca and Faltings (2007) have proposed a reputation mechanism and have shown that it has an equilibrium where the providers behave as socially desired. Kerr and Cohen (2007) have employed first-order logic to prove security for buyers in a reputation system based on a set of assumptions.

Address correspondence to Mohsen Lesani, EPFL IC IIF LAMP1, INR 329, Station 14, CH-1015 Lausanne; e-mail: mohsen.lesani@gmail.com

While in traditional markets, the reputation was built over time for a known person or company, the contemporary virtual marketplaces offer dealing with complete strangers. A person may know hundreds of people in a social network but do not know many others. The user may know how much she can rely on persons that she directly knows but what about the others? How much should she trust a review that is written by a person that she does not directly know? The question of how to preserve reputation and trust concepts in virtual marketplaces is a matter to be solved or else the virtual marketplace becomes a place for low-quality stuff (Akerlof 1970). The fact that the reputation of a seller is important to customers is also supported by experiments on eBay[1] (Resnick et al. 2002).

There are several methods designed to compute global trust to a user, ignoring the fact that trust is often a personal matter. The well-known auction site, eBay, has a reputation system based on feedbacks that sellers and buyers write for each other (Resnick et al. 2000). The feedback score assigned to each profile is a signed summation of feedbacks of all people who have had a deal with that profile. Hence the feedback score presents a global view to the profile. Aside from the simple summation of feedbacks employed in eBay, an important family of global trust computation algorithms is the flow model category. Algorithms that compute trust by iterating paths and loops of trust graphs are called flow model algorithms. This family includes Google's PageRank (Page et al. 1998), EigenTrust (Kamvar, Schlosser, and Garcia-Molina 2003), Advogato[2] community reputation computation (Levien 2004), and Appleseed (Ziegler and Lausen 2004) algorithms. PageRank used by Google for ranking pages on the Web can be viewed as a reputation computation algorithm where any hyperlink to a Web page is a positive rating for that page. EigenTrust algorithm is proposed to compute trust scores of users in P2P networks for the purpose of determining the most reliable file servants. Advogato's reputation and Appleseed algorithms also assign a global trust score to each member of the community.

In contrast to global trust, a unique person may be viewed trusted by some and untrusted by some others. What the majority of people believe is not what anyone believes. The deviation from the majority becomes more important when the matter that information is requested for is related to personal feelings and taste. As an instance, the trust in a tailor is highly dependent to the styles he is professional in and the styles the customer desires. Besides, people naturally build trust to an unknown person by asking their trusted friends about him. For example, if one has found an unknown person offering a laptop with a reasonably low price, he is interested to know if any of his trusted friends knows him. If they do not know him, he is interested to know if any of his trusted friends can ask from their trusted friends about him; and the chain can get even longer. Recommender Web sites return a list of providers in response to a search for an item. The list is desired to be ordered by the user's direct or indirect trust to the providers. There is a need to infer how much a person individually trusts another one from the limited trust ratings that the users have provided. Obviously, the persons may be not directly connected in the social network. Personalized reputation is computed from information coming from trusted friends. Hence, it is also a way to fight against false high or low reputations occurring as a result of cheating positive or unfair negative feedbacks. When a reputation is built from the feedbacks coming from all the social network users, it can be more easily misled by unknown users than when it is composed from information coming only from trusted friends. By personalized reputation, an incorrect trust rating provided by a cheating user can only mislead the users who directly or indirectly have selected to trust him. Thus computing personalized versus global reputation and trust seems to be more effective.

[1] http://www.ebay.com

[2] http://www.advogato.org/

If the trust relationship in a social network is abstracted to a trust graph, to infer personalized trust from a node to another one, algorithms are needed to find and select from the paths that exist from the first node to the second. Also aggregation methods are needed to combine trust values when the values are obtained from several nodes. Aggregation methods are dependent and are proposed on specific trust modelings. This work has contributions in both trust modeling and aggregation and also trust inference algorithms. In the remainder of this part, an overview of related works and contributions in trust modeling and aggregation and then trust inference algorithms will follow.

Crisp approach models trust as a number in a continuous range, for example [0, 10]. Golbeck (2005) and Golbeck and Hendler (2006) models trust relationships in a social network as a crisp trust graph. Mean method has been used for trust aggregation. Huynh et al. (2004) have offered a trust model called FIRE for open systems, the systems that agents can join and leave at any time. FIRE has a crisp modeling of trust and uses the mean method to aggregate trust ratings that are obtained from interactions, rules about roles, witnesses, and certifications. Aringhieri et al. (2006) have proposed a trust model for P2P networks. They used the term "fuzzy" in the sense that trust is a value between [0, 1]. They employ weighted mean method to aggregate reputations obtained from other peers. Jøsang (2001) has proposed Subjective Logic as a belief modeling and reasoning theory. His modeling considers disbelief and uncertainty besides belief. Belief, disbelief, and uncertainty all take crisp values. He has applied his belief model to trust and the belief combination operators of his theory to trust aggregation. Also in his survey, Jøsang, Ismail, and Boyd (2007) have reviewed different trust modeling methods and different live reputation systems.

In the crisp modeling of trust, users should specify their trusts to other users as numbers, whereas people naturally use linguistic expressions when they are asked to state their trust to others. In fact, people tend to specify their trust in expressions such as very low, medium, or high; and specifying trust in crisp numbers may seem odd to them. A user that trusts another one by a medium high value may find no or little difference between 6 and 7 values in a trust value scale of 10. Similarly people like to hear linguistic expressions when they ask others about their trust to an unknown person. Hence, it seems that linguistic terms are superior to crisp numbers in describing trust, although this can be stated more strongly when a subjective study of users supports it. More importantly, as it is shown in this paper, the aggregation methods proposed for the crisp modeling of trust, such as mean method and Subjective Logic aggregation operators, do not perform well especially in combing contradictory information. It seems that a number does not have enough capacity for conveying an aggregated value especially if the value is the result of aggregating contradictory values.

According to the aforementioned shortcomings in the crisp view of trust, fuzzy sets seem an ideal choice for trust modeling and inference. This research proposes using linguistic terms for specifying trust. A fuzzy set is defined for each trust linguistic term and a fuzzy aggregation method is offered for combining fuzzy trust values. Moreover, the resulting trust fuzzy set is finally converted to its corresponding linguistic expression of trust. Hence, not only trust specification is in linguistic terms, but also the inference result is offered as a linguistic expression. The results of this research show that the fuzzy aggregation method outperforms the mean and subjective logic aggregation methods.

As the other focus of this research is trust inference algorithms, the following paragraphs mention two previously proposed algorithms and the contribution of this research for inferring trust individually from a person to another.

Jøsang, Hayward, and Pope (2006) have proposed a trust inference algorithm called DSPG (Directed Series Parallel Graph)-finding algorithm. The algorithm presents a subgraph comprised of some selected paths between a source and a sink as a closed form of serial and parallel connections. The trust from the source to the sink is then inferred from the obtained

closed form using the Subjective Logic operators for serial and parallel belief combination. The problem is that the time complexity of the DSPG-finding algorithm is exponential. A suboptimal algorithm with time complexity of $O(V + E + M \times L)$ has also been proposed, where M is the average number of paths between any two nodes and L is the average length of these paths in the trust graph.

Golbeck (2005) and Golbeck and Hendler (2006) have researched on trust in social networks and deduced some trust graph properties from real social networks. They have proposed an incremental algorithm named TidalTrust, which performs the inference through the strongest of shortest paths. The time complexity of TidalTrust is $O(V + E)$ where V is the number of vertices and E is the number of edges of the trust graph. Incremental trust inference algorithms infer trust incrementally from the sink level by level back to the source. For each node, trust to the sink is computed locally from the information (i.e., the trust to sink) coming from its neighbors. Computed trust values for nodes in the current level are used in computing trust values for nodes in the next level that is one level farther from the sink.

TidalTrust confines the search to shortest paths while the information inferred from a long chain of people with high trust along it may be much more precise than the information inferred from a short chain of people with low trust along it. In addition, while the most trustworthy information comes from the strongest paths, weaker paths can also convey valuable information. TidalTrust also does not report the preciseness of inference; while the inference along a lowly trusted path—even if it results the same inferred trust value—is less precise than the inference that is done along a highly trusted path. The new incremental algorithm that is proposed by this work not only benefits from shortest and strongest paths but also from longer and weaker ones. The preciseness of inference is also reported in the proposed algorithms. The results of AllLengthStrongestTrust algorithm, which considers not only the shortest paths but also all the longer ones to find the strongest of all the possible paths, show improvement in preciseness of inference in cases that longer paths are the most trusted ones.

This paper begins with presenting definition, modeling, and some properties of trust; it continues by explaining different aggregation methods including crisp and fuzzy approaches and then the incremental inference algorithms are offered. The Simulation and Results section presents the experiments and comparison with previous works. Finally, the Conclusion and Future Works section concludes the paper.

2. TRUST

2.1. Trust Definition

Trust is defined differently in different contexts. For the purpose of this research, trust is the relying party's subjective belief in the trusted entity to serve a certain function on which the welfare of the relying party depends. The function can be information provision.

2.2. Trust Graphs

A social network can be modeled as a large directed graph where nodes represent persons, directed edges represent friendships or acquaintances, and edge labels represent trust values. The trust rating on an edge is the trust that the first node has to the second node (i.e., the belief to the correctness of information coming from the second node). Trust graphs not only are theoretically defined but practical instances of them also exist such as FilmTrust.[3] The following properties are considered for trust and trust graphs.

[3] http://trust.mindswap.org/FilmTrust

2.3. Trust Properties

2.3.1. Asymmetry. As two friends have different beliefs and may have seen different behaviors form each other, the trust they have to each other may be different. This is what makes the trust graph asymmetric and directed.

2.3.2. Transitivity. Trust may seem not to be transitive, i.e., if A highly trusts B and B highly trusts C it does not mean that A also highly trusts C. However it is usual that you ask one of your highly trusted friends about an unknown matter and take his opinion as your own. Consider a case when A asks B about a film because A highly trusts B and B does not know about the film; so B asks his highly trusted friend C that knows about the film while A is unaware of this relationship. B takes C's opinion as his own and gives it to A that will also take it as her own opinion. Person A finally takes C's opinion as if C is her trusted friend. While the transitivity of trust is a source of argument, it can be considered that trust is transitive in the sense of information back propagation. It can be defined that person B is transitively trusted by person A in a social network if information can be propagated from B back to A in reverse directions of trust connections in the network. The transitivity property of trust lets information propagate backwards through a trust chain of people. It is obvious that this information can also be the trust value to another person; so trust to an unknown person can also be found from transitive trust paths.

When something is known personally, it is trusted with the maximum trust value. When information is received from another person, the trust to that person affects the trust to the information gotten from him. The less a person is trusted, the more the trust in the information he provides is discounted. If the information propagates back through a person and then another one and then reaches the person at the front of the chain, discounting of the trust to the information happens twice. The longer the chain, the more the trust to the information obtained from the chain is discounted. When you hear news from a witness of an event you believe it more than when you get the news from a deeper chain of people. Similarly, when trust to a person should be inferred, it is more desirable to be inferred from shorter chains of people. Golbeck (2005) showed that shorter paths lead to more accurate trust inference results.

Information is desired to be taken from a more trusted person or when the information is an aggregation of information taken from several persons, it is desired to be affected more by the information provided by more trusted persons. Similarly, when you are to infer the trust to an unknown person and you have the choice of getting this information from two people chains both ending to the unknown person and having the same depth, you certainly choose the chain with more trust between the people along it. Golbeck (2005) showed that paths with higher trust ratings cause better trust inference.

3. TRUST MODELING AND AGGREGATION

3.1. Trust Modeling

Trust is considered a measurable continuous subjective quantity. Trust quantity has a minimum value (that means having no trust) and a maximum value (that means completely having trust) where partial trust comes in-between. Taking 0 as the lower and 1 as the higher bounds of trust results the range of [0, 1] for trust.[4] As in some previous works, the trust range can also be taken [0, 10] without losing any generality.

[4]This range is for trust and not distrust. To support distrust, the range should be extended to negative values. See Conclusions and Future Works section.

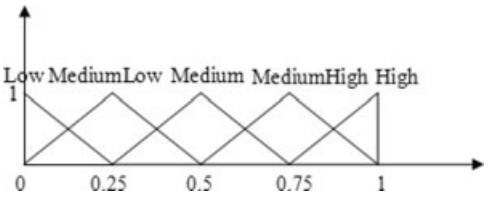

FIGURE 1. Fuzzy membership functions of trust linguistic terms.

Crisp modeling represents trust values as numbers in the trust range. Presenting trust as a number such as 0.6 in the trust range is the approach adopted by most previous works.

Subjective Logic as a belief modeling and reasoning theory is also applied to trust modeling and aggregation (Jøsang et al. 2006). Subjective Logic not only considers belief but also disbelief and uncertainty. The ordering of $w = (b, d, u, a)$ called opinion is the modeling for subjective quantities such as trust where b, d, and $u \in [0, 1]$ represent belief, disbelief, and uncertainty, respectively. Parameter a is called base rate and is the amount of belief considered by default in lack of any knowledge. Belief, disbelief, and uncertainty are all crisp numbers and $b + d + u = 1$. Probability values are equivalent to expectation values $E = b + au$ in subjective logic, and expectation values are mapped to uncertainty-maximized opinions with base rate a. This means that if $E \geq a$ then $w = ((E - a)/(1 - a), 0, (1 - E)/(1 - a), a)$ and if $E < a$ then $w = (0, (a - E)/a, E/a, a)$ (Jøsang 2001).

Similar to any other continuous quantity, fuzzy modeling is possible for trust. This research proposes fuzzy modeling of trust where trust values are represented as fuzzy sets with definite membership functions in the trust range. The fuzzy membership functions for the linguistic terms such as low, medium, medium low, medium high, and high can be defined as depicted in Figure 1. Hence, the linguistic terms can also be used as trust values.

3.2. Trust Aggregation

Several bits of information may be obtained from parallel paths as answers to a single question. For example if A does not know C, she asks her friends (Bs) about C. Different friends (Bs) may have different ideas about that person (C). Person A should combine the different ideas she has received from Bs about C to infer a unique idea about C. Similarly, the final combination phase should take place when the information comes from deeper parallel chains of people. People naturally combine information when they receive them from different sources, assigning more credit to more trusted sources. The belief combination methods, also known as aggregation methods, try to simulate this human ability.

Aggregation methods of trust are dependent to the modeling of trust. First, mean aggregation method and Subjective Logic aggregation operators are presented. Next the fuzzy rule-based aggregation method proposed in this work for fuzzy modeling of trust is offered.

3.2.1. Mean Aggregation Method. Assume that person P has n different information sources for the value of an unknown continuous quantity X. P should aggregate the values that are obtained from these sources. If each source i reports value B_i as his believed value for X and P trusts each source i with crisp trust value T_i, then the result of aggregation B_p, i.e., the value that P believes for X is

$$B_p = \frac{\sum_{i=1}^{n} T_i \times B_i}{\sum_{i=1}^{n} T_i}. \tag{1}$$

The preceding aggregation method can be applied to trust aggregation. Assume that a node N should aggregate different trust values to a sink that are obtained from its neighbors. If trust values from different neighbors to the sink are known to be $TrustToSink_1, TrustToSink_2, \ldots, TrustToSink_n$ and node N's direct trust to its neighbors are $Trust_1, Trust_2, \ldots, Trust_n$, respectively, the trust of node N to the sink as the result of aggregation is

$$TrustToSink_N = \frac{\sum_{i \in Neighbors(N)} Trust_i \times TrustToSink_i}{\sum_{i \in Neighbors(N)} Trust_i}. \tag{2}$$

3.2.2. Subjective Logic Combining Operators. The following operators are the two operators Subjective Logic has defined for combining two serial and parallel opinions. Opinion is the Subjective Logic's modeling of trust that is proposed by Jøsang, which had been described in Section 3.1.

Reduction Operator

If A's trust to B in recognition of correctness of information x is $\omega_B^A = (b_B^A, d_B^A, u_B^A, a_B^A)$ opinion and B's opinion about the information x is $\omega_x^B = (b_x^B, d_x^B, u_x^B, a_x^B)$ then to infer A's opinion to the information x, B's opinion about the information x is reduced with the A's opinion in B as follows:

$$\omega_x^{A:B} = \omega_B^A \otimes \omega_x^B = \begin{cases} b_x^{A:B} = b_B^A b_x^B \\ d_x^{A:B} = b_B^A d_x^B \\ u_x^{A:B} = d_B^A + u_B^A + b_B^A u_x^B \\ a_x^{A:B} = a_x^B. \end{cases} \tag{3}$$

The superscript denotes the trusting and the subscript denotes the trusted.

The effect of reduction operator is to increase uncertainty and decrease belief and disbelief in transitive chains.

Consensus Operator

Consensus between two opinions is an opinion that reflects both of the initial opinions fairly. If A and B believe in the information x with corresponding beliefs of $\omega_x^A = (b_x^A, d_x^A, u_x^A, a_x^A)$ and $\omega_x^B = (b_x^B, d_x^B, u_x^B, a_x^B)$, their aggregation is: (Assuming that $a_x^A = a_x^B$.)

$$\omega_x^{A \diamond B} = \omega_x^A \oplus \omega_x^B = \begin{cases} b_x^{A \diamond B} = \left(b_x^A u_x^B + b_x^B u_x^A\right) / \left(u_x^A + u_x^B - u_x^A u_x^B\right) \\ d_x^{A \diamond B} = \left(d_x^A u_x^B + d_x^B u_x^A\right) / \left(u_x^A + u_x^B - u_x^A u_x^B\right) \\ u_x^{A \diamond B} = \left(u_x^A u_x^B\right) / \left(u_x^A + u_x^B - u_x^A u_x^B\right) \\ a_x^{A \diamond B} = a_x^A. \end{cases} \tag{4}$$

The effect of the consensus operator is to increase belief and disbelief and decrease uncertainty.

It is obvious that the information x can be trust to a sink and hence the operator can be used to aggregate different trust values to a sink.

3.2.3. Fuzzy Aggregation Method. Fuzzy modeling and aggregation is the first contribution of this work. Assume that person P should aggregate different values for a quantity

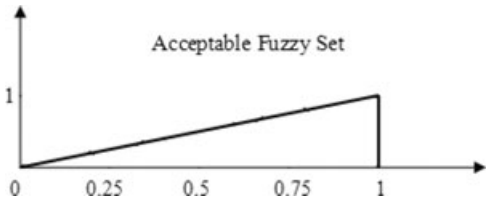

FIGURE 2. Acceptable fuzzy set.

X that are obtained from his known sources. Assume that $<Trust_i, BelievedValue_i> 1 \leq i \leq n$ are n pairs of fuzzy sets where $BelievedValue_i$ is the value that source i believes for X and $Trust_i$ represents trust from P to source i. For each pair, a result fuzzy set named $Result_i$ is obtained from fuzzy implication with the following fuzzy rule.

$$if\,(Trust_i\, is\ Acceptable)$$
$$then\,(Result_i\ is\ BelievedValue_i).$$

In the [0, 1] trust range, the membership function of Acceptable fuzzy set is defined as a linear function with membership value of 0 at trust value of 0 and membership value of 1 at trust value of 1 as in Figure 2.

In an implication, the firing rate of the rule is the truth degree of the antecedent. For the preceding rule, the firing rate is:

$$FiringRate = Super(Trust_i \cap Acceptable), \tag{5}$$

where for fuzzy sets S_1 and S_2, the membership function of $S_1 \cap S_2$ is the minimum of membership functions of S_1 and S_2. Also for a fuzzy set S, Super(S) is the maximum value in the membership function of S.

If Larsen method is used as the implication method, then

$$Result_i = FiringRate \times BelivedValue_i, \tag{6}$$

where for a number c and a fuzzy set S, the membership function of $c \times S$ is the membership function of S that is scaled down by c. Larsen implication method scales down the rule consequence by the firing rate.

The fuzzy set that is named Result is the fuzzy union of all the $Result_i$ fuzzy sets.

$$Result = \bigcup_{i=1}^{n} Result_i, \tag{7}$$

where for fuzzy sets S_1 and S_2, the membership function of $S_1 \cup S_2$ is the maximum of membership functions of S_1 and S_2.

The final Result of fuzzy aggregation, FinalResult fuzzy set, is obtained by normalizing the Result fuzzy set. Normalization of a fuzzy set S is $(1/c) \times S$ where $c = \text{super}(S)$. In other words, normalization scales a fuzzy set to make its super() value equal to 1.

The preceding general fuzzy rule can be simply applied to trust aggregation. Assume that node N should aggregate different trust values to a sink that are obtained from its neighbors. If $TrustToSink_i$ represents trust from neighbor i to the sink, then $BelievedValue_i$ would be $TrustToSink_i$ and the inference rule would become

$$if\,(Trust_i\ is\ Acceptable)$$
$$then\,(Result_i\ is\ TrustToSink_i).$$

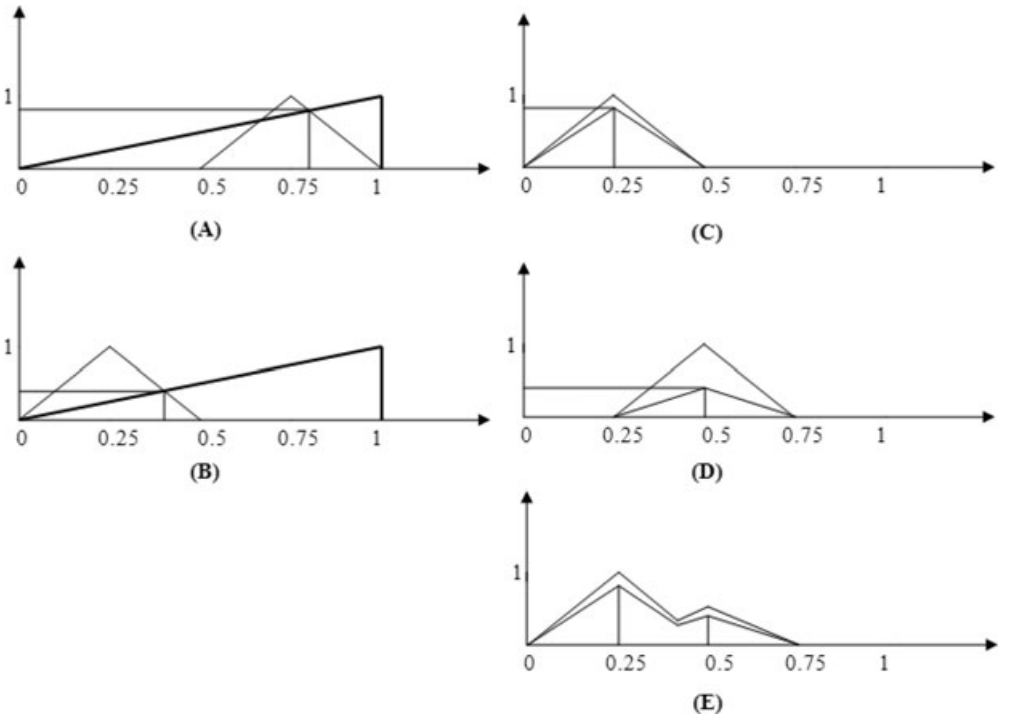

FIGURE 3. Fuzzy aggregation example.

As an instance, Figure 3 depicts aggregation process for <MediumHigh, MediumLow> and <MediumLow, Medium> pairs. Figures 3(a) and (b) depict computation of firing rates of the rule. The firing rates are shown with vertical lines. $TrustToSink_1$ and $TrustToSink_2$ fuzzy sets (i.e., MediumLow and Medium) are scaled by the firing rates to produce $Result_1$ and $Result_2$ fuzzy sets as shown in Figures 3(c) and (d). Figure 3(e) shows Result fuzzy set as the fuzzy union of $Result_1$ and $Result_2$ fuzzy sets (the lower membership function) and *FinalResult* as the normalization of Result (the upper membership function).

It is notable that the fuzzy rule is similar to how people get a friend's opinion if they acceptably trust the friend. As the believed fuzzy sets are scaled by firing rates, the contribution of each believed fuzzy set to the final result is proportional to the trust value to that believed fuzzy set.

Although the lowly trusted believed fuzzy sets are scaled down, they are not eliminated from the final result. Hence, even if a lowly trusted information source reports the sink as not highly trusted, his belief is present in the final result and the sink is not considered as a fully trusted node. This characteristic that is absent in other aggregation methods results in success of the fuzzy aggregation method in the following simulations.

What the fuzzy aggregation procedure returns is a fuzzy set. If the result is to be fed to an artificial deciding agent, the at hand fuzzy set form of the result is the proper form; but if the client is a user, the result should be converted to a more readable format. To make the aggregated trust value comprehendible to the user, the result fuzzy set should be approximated to the most similar fuzzy set of the known terms (such as low, medium, and high) or the known terms with hedges (such as very and somehow) or a disjunction of them and report the corresponding linguistic expression such as "medium or somehow high" to the user.

There is a simple procedure for yielding an approximating linguistic expression for the inferred fuzzy set although more intelligent algorithms may be possible. The first step is to construct all the possible conjuncted pairs of the terms with hedges and their corresponding fuzzy sets. We call these fuzzy sets linguistic fuzzy sets. In the second step, the inferred fuzzy set is compared against each of the linguistic fuzzy sets and the most similar fuzzy set is selected. Finally, the linguistic expression corresponding to the selected linguistic fuzzy set is reported as the approximating linguistic expression for the inferred fuzzy set. The similarity of two fuzzy sets A and B is defined as

$$S = \frac{|A \cup B|}{|A \cap B|}, \tag{8}$$

where | | is the cardinality of a fuzzy set

$$|A| = \int_{x \in SupportSet} \mu_{A(x)}. \tag{9}$$

As a result fuzzy set can be represented by its corresponding linguistic expression, not only the input to but also the result from the fuzzy aggregation method can be in linguistic terms.

4. TRUST INFERENCE ALGORITHMS

To infer an unknown trust value from a node to another in a trust graph, algorithms are needed to identify trust paths from the first node to the second. Trust inference algorithms use trust aggregation methods through the identified paths to finally infer the trust from the source to the sink. Incremental trust inference algorithms infer trust incrementally from the sink level by level back to the source.

Regarding the shortest paths and strongest paths as more proper paths for inference, four different algorithms named ShortestStrongestTrust, ShortestAllStrengthTrust, AllLength-AllStrengthTrust, and AllLengthStrongestTrust are possible. The algorithms that seek for shortest paths (i.e., ShortestStrongetTrust and ShortestAllStrengthTrust) never requeue a node that is previously visited because a shorter path to the node is previously found; hence these algorithms are fundamentally Graph Search algorithms (Russel and Norvig 2003). In contrast, algorithms that search for paths of any length (i.e., AllLengthAllStrengthTrust and AllLengthStrongestTrust) do revisit previously visited nodes and are therefore basically Tree Search algorithms (Russel and Norvig 2003).

ShortestStrongestTrust algorithm was first introduced by Golbeck (2005), which she named TidalTrust algorithm. As this algorithm is improved in this work it is explained in the following subsection.

4.1. ShortestStrongestTrust (TidalTrust) Algorithm

As indicated before, shorter paths are more desirable paths for information acquisition. Also the most reliable information comes from the strongest trust paths. The algorithm makes use of the strongest of the shortest paths hence it is called ShortestStrongestTrust here. ShortestStrongestTrust algorithm searches for shortest paths from the source to the sink similar to the breadth first search algorithm. Any path longer than the shortest paths is ignored and nodes along it do not participate in trust inference. Also from all the shortest paths only the strongest ones participate in trust inference.

The strength property for a path is defined as the minimum trust rating along the path excluding the last trust rating in the path. The reason behind excluding the last trust rating is that although it's a trust value, it is the information that is obtained from the path. All the previous trust ratings along the path contribute to the trust to this information. The strength of a path with zero length is the maximum value by definition. The strength from a source to a sink is defined as the maximum path strength in all the paths between them.

ShortestStrongestTrust considers only the strongest paths among the shortest ones. Considering all the shortest paths form the source to the sink, only the nodes that are along the strongest ones participate in trust inference. For the purpose of explaining the ShortestStrongestTrust, the strength of the strongest of shortest paths from the source to the sink will be called the required strength. Hence, the trust from the source to the sink should be inferred from the shortest paths that have the required strength.

The trust from intermediate nodes to the sink is to be computed during incremental inference of trust from the sink back to the source. For inferring trust from an intermediate node to the sink, information from only those neighbor nodes are used that are along the paths from the source to the sink with the strength value equal to the required strength. We call the set of such neighbors the Participating set. The Participating set for a node S is defined as follows:

$$
\begin{aligned}
&\textit{Participating}\ (s) = \\
&\{ \\
&\quad n \in \textit{Nodes} \mid \\
&\qquad \textit{adjacent}(s, n)\ \textit{and}\ \textit{trust}\ (s, n) \geq \textit{RequiredStrength} \\
&\qquad \textit{and} \\
&\qquad \textit{strength}(\textit{pathFormTo}(n, \textit{sink})) \geq \textit{RequiredStrength} \\
&\}
\end{aligned}
$$

The ShortestStrongestTrust algorithm is as follows:

```
ShortestStrongestTrust (TidalTrust) algorithm pseudo code
inferShortestStrongestTrust( Node source, Node sink )
{
if ( source = sink )
       return MAX_TRUST_VALUE

queue.queue( source )
nextLevelQueue = new Queue()
maxDepth = DUMMY_MAX_VALUE
depth = 1

strengthFromSource[ source ] = MAX_TRUST_VALUE

while ( (not queue.isEmpty()) and (depth ≤ maxDepth) )
{
       node = queue.dequeue()
       tideStack.push( node )
       if ( not trustGraph.isAdjacent(node, sink) )
       {
              for each adjacentNode in adjacents of node
              {
                     if ( not visiteds[ adjacentNode ] )
                     {
                            visiteds[ adjacentNode ] = true
                            nextLevelQueue.queue( adjacentNode )
                     }
```

```
                    if ( nextLevelQueue.contains( adjacentNode ) )
                    {
                          pathStrength = min(
                                strengthFromSource[ node ],
                                trustGraph.getTrust( node, adjacentNode ) )

                          strengthFromSource[ adjacentNode ]= max(
                                strengthFromSource[ adjacentNode ],
                                pathStrength )
                          // When one of the parameters has a dummy value,
                          // the min and max functions return the other parameter.
                    }
              }
        }
        else // if ( trustGraph.isAdjacent(node, sink) )
        {
              maxDepth = depth
              pathStrength = strengthFromSource[ node ]

              strengthFromSource[ sink ] = max(
                    strengthFromSource[ sink ],
                    pathStrength )
        }

        if ( queue.isEmpty() )
        {
              queue = nextLevelQueue
              depth = depth + 1
              nextLevelQueue = new Queue()
        }
}


if ( maxDepth == DUMMY_MAX_VALUE )
      return DUMMY
// If the maxDepth value has not changed since the beginning then
//    no path is found to the sink.

requiredStrength = strengthFromSource[ sink ]
// The required strength is computed in the forward wave.
// The Backward wave:
while ( not tideStack.isEmpty() )
{
      node = tideStack.pop()
      computeTrustToSinkFromParticipatingSetOfNeighbors( node )
}
return <trustsToSink[ source ], requiredStrength>
}
```

The algorithm is named TidalTrust because computation flows from the source to the sink and then back from the sink to the source. The algorithm starts with a search queue that only contains the source node. It iterates the adjacent nodes level by level similar to the breadth first search algorithm. Whenever a node is iterated, it is pushed onto a stack to take the nodes in a reverse order later.

When a node that is not adjacent to the sink is iterated, all its neighbors are marked visited and added to the next level search queue if they are not visited before. The previously visited nodes are ignored; because if they belong to the next level, they have already been put in next level queue. Otherwise, they belong to the current or a previous level and hence they lead to longer paths while the algorithm searches for shortest ones.

The strength from the source to each of the neighbor nodes that are in the next level search queue is updated to the maximum of its previous value and the strength value of the new path (that starts at the source, passes through the current node, and ends at the neighbor node). Note that the strength from source to a node in the next level queue may be updated several times as new edges from the current level nodes to it are found.

If the current node is adjacent to the sink, search has reached the goal; so to stop the search, the upper limit of iteration depth (i.e., the maximum depth variable) is set to the current depth. This prevents advancing to nodes in the next levels. Also, as other paths to the sink can still be found, the strength from the source to the sink is updated. Note that the trust edges from the nodes in the last level to the sink do not contribute to the computation of strength from the source to the sink. This is according to the definition of strength property of a path stated before. The strength property for a path is defined as the minimum trust rating along the path except the last trust rating. The ShortestStrongestTrust algorithm presented in Golbeck dissertation has not excluded the last trust rating but this problem is fixed here. While the computation excluding the last trust rating is called strength computation, the counterpart that includes the last trust rating is called the complete strength computation. All the strength values from the source to midway nodes in the incremental strength computation are computed as complete strengths to finally lead to normal strength computation from the source to the sink.

When all the current level nodes are traversed and the current queue becomes empty, depth variable is incremented and the next level search queue replaces the current one. Iteration proceeds until all the nodes with depth lower or equal to the maximum depth variable are iterated or the search queue is empty. If the iteration terminates because the queue is empty, this means that all the nodes accessible from the source node are traversed and the sink node could not be found. Hence, there is no path from the source to the sink. As the maximum depth variable is updated when the sink is reached, to find out whether any path from the source to the sink is found, the current value of the maximum depth variable is compared to its initial value; if the variable is never updated from its initial value, the algorithm returns a dummy value instead of an inferred trust value to show that the trust cannot be inferred.

At the end of the forward wave, the required strength from the source to the sink is found. The nodes are popped up from the stack for iteration in a reverse order in the backward wave. If a node is adjacent to the sink, then its inferred trust to the sink is simply its trust rating to the sink in the trust graph. If not, the trust to the sink is computed from the previously defined participating set of neighbors. As the trust ratings of the current node to its neighbors are available in the trust graph and the nodes are iterated backwards level by level from the sink to the source, the trust values of neighbors of a node to the sink are computed before the trust value from the node to the sink is inferred.

As inference is done along the paths with at least the required strength, the minimum trust rating to any neighbor node participating in trust inference is the required strength. This means that the inference is trusted as much as the required strength value. Therefore, although preciseness of inference is not reported by Golbeck's algorithm, the algorithm that is presented here returns the preciseness of inference in addition to the inference result.

4.2. AllLengthStrongestTrust Algorithm

The main concern of this algorithm is to infer trust from the strongest paths. The algorithm finds the strongest paths even if they are longer than the shortest ones; so it does not stop the search when the sink node is visited for the first time; but the search continues

until no other unvisited nodes can be found. This algorithm leads to a more precise inference when deeper paths conduct more trusted information.

```
AllLengthStrongestTrust algorithm pseudo code
inferAllLengthStrongestTrust ( Node source, Node sink )
{
if ( source = sink )
       return MAX_TRUST_VALUE

queue = new Queue()
priorityQueue = new Queue()

queue.queue( source )

strengthFromSource[ source ] = MAX_TRUST_VALUE
strengthFromSource[ sink ] = -1

while ( (not queue.isEmpty()) or (not priorityQueue.isEmpty()) )
{
       if ( priorityQueue.isEmpty() )
              node = queue.dequeue()
       else
              node = priorityQueue.dequeue()

       isVisited[ node ] = true

       if ( not trustGraph.isAdjacent( node, sink ) )
       {
              for each adjacentNode in adjacents of node
              {
                     pathStrength = min(
                            strengthFromSource[ node ],
                            trustGraph.getTrust( node, adjacentNode ) )

                     previousStrength =
                            strengthFromSource[ adjacentNode ]

                     newStrength = max(
                            previousStrength,
                            pathStrength )

                     strengthFromSource[ adjacentNode ] = newStrength

                     if ( newStrength >= strengthFromSource[ sink ] )
                     {
                            // Add the adjacentNode to the queues only if it
                            // may lead to a path to the sink stronger than
                            // or as strong as the previously found paths.
                            if ( not isVisited[ adjacentNode ] )
                                   queue.queue( adjacentNode )
                            else
                                   if ( newStrength ≠ previousStrength )
                                          priorityQueue.queue( adjacentNode )
                     }
              }
       }
```

```
        else // if ( trustGraph.isAdjacent(node, sink) )
        {
                pathStrength = strengthFromSource[ node ]

                strengthFromSource[ sink ] = max(
                        strengthFromSource[ sink ],
                        pathStrength )
        }
}


if ( maxDepth == DUMMY_MAX_VALUE )
        return DUMMY
// If the maxDepth value has not changed since the beginning then
// no path is found to the sink.

requiredStrength = strengthFromSource[ sink ]

initialize all the isVisited[ i ] to false

adjacentsToSink = trustGraph.getAdjacentsTo( sink )
for all adjacentsToSink[ i ]
{
        trustsToSink[ adjacentsToSink[ i ] ] =
                trustGraph.getTrust( adjacentsToSink[ i ], sink )
        queue.queue( adjacentsToSink[ i ] )
}

while ( (not priorityQueue.isEmpty()) or (not queue.isEmpty()) )
{
        if ( not priorityQueue.isEmpty() )
                currentNode = priorityQueue.dequeue()
        else
                currentNode = queue.dequeue()
        isVisited[ currentNode ] = true

        adjacentsTo = trustGraph.getAdjacentsTo( currentNode )
        for all adjacentTo in adjacentsTo[ i ]
        {
                if ( adjacentTo = sink )
                        continue

                if ( not isVisited[ adjacentTo ] )
                {
                        updateTrustToSink( adjacentTo )
                        queue.queue( adjacentTo )
                }
                else
                {
                        previousTrust = trustsToSink[ adjacentTo ]
                        newTrust = updateTrustToSink( adjacentTo )
```

```
                    if (previousTrust ≠ newTrust)
                        priorityQueue.queue( adjacentTo )
                }
        }
}
return <trustsToSink[ source ], requiredStrength>
}
```

In the first flow, the nodes are iterated level by level from the source to find the required strength. When a node is iterated, a new path from source is found to each of its neighbors. The strength from the source to the neighbors of the node is updated to the strength of the new path if it is greater than previously found strength. If a neighbor node is not visited before, it is added to the queue. If it is visited before, the current edge is a backward edge, i.e., an edge to a node in a previous level. The reiteration of a node that belongs to one of the previous levels requires reiteration of some nodes in its subsequent levels; hence it is important to iterate them before advancing to the next level to prevent propagation of reiteration to next levels. Therefore, revisited nodes should be iterated before iterating any node in the queue. Thus, the priority queue is used to hold revisited nodes. The nodes in the priority queue are prior to the nodes in the queue for iteration. Any neighbor node having strength from the source lower than the currently found strength of the source to the sink is ignored. Such neighbor nodes are ignored because they can only be on paths to the sink that are weaker than the currently found paths. The first flow is finished when all the nodes accessible from source are iterated. The strength from the source to the sink that is called the required strength is finalized at the end of the first flow.

In the second flow, trust values from nodes to the sink are computed from the sink level by level back to the source. Trust values should be computed thorough only the nodes that are on the paths with strength equal to the required strength. Hence, trust from each node that is adjacent to the currently iterated node is updated if the trust rating from it to the current node is greater than or equal to the required strength. If such adjacent node is not visited before, it is queued to be iterated later. Similar to the first flow, backward edges are possible; hence the priority queue is employed to hold revisited nodes. The trust from a node to the sink may be updated for several times as new adjacent nodes of it are iterated. When the backward flow finishes, the trust value from source to the sink is finalized.

It is notable that if a node is adjacent to the sink in the trust graph (i.e., there is an edge from the node to the sink), the algorithm does not ignore valuable information from other longer paths from this node to sink. This corresponds to the situation when a person knows another person but also gets information about him from trusted friends. As previously mentioned about aggregation, the participation of information obtained from a source is proportional to the trust value to that source. A node's own direct trust rating to the sink participates with the maximum trust value while the trust values obtained from longer paths participate with a trust value equal to the strengths of the paths. The intuition is that people usually trust their own knowledge more than knowledge of anyone other.

5. SIMULATION AND RESULTS

5.1. Mean versus Fuzzy Aggregation

The crisp trust graph depicted in Figure 4 is used for simulation of crisp mean aggregation method. The [0, 1] and [0, 10] trust ranges can obviously be interchanged by a multiplication

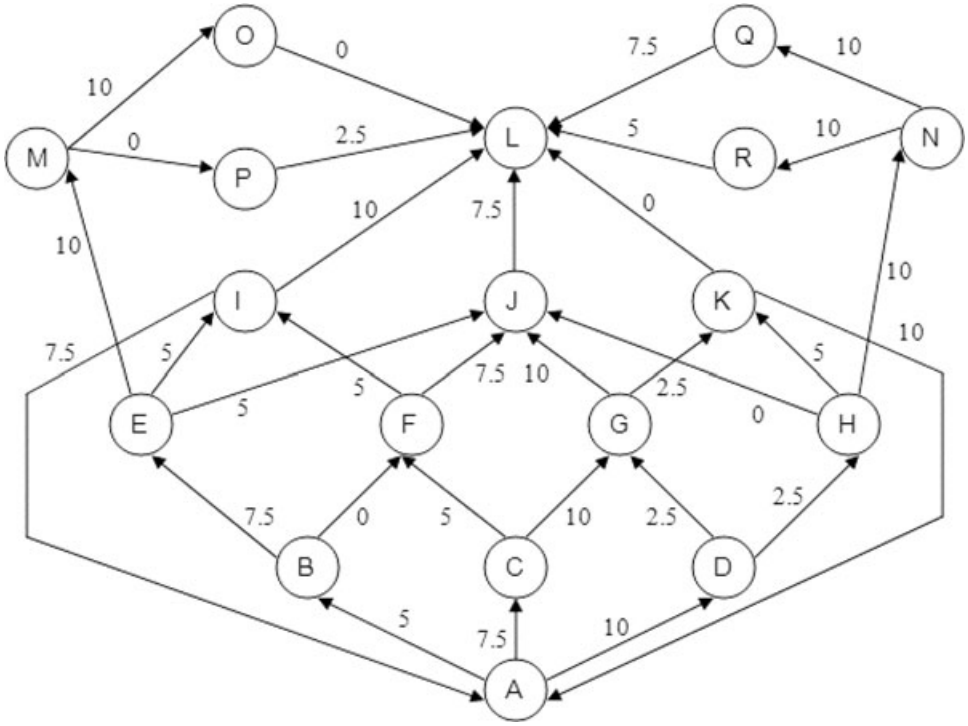

FIGURE 4. Crisp trust graph.

or division by 10. All the trust values in the graph are 0, 2.5, 5, 7.5, or 10. These values are selected deliberately because all of them only and completely belong to one of the fuzzy sets of Low, MediumLow, Medium, MediumHigh, and High linguistic terms; and therefore the corresponding fuzzy trust graph for Figure 4 is the fuzzy trust graph in Figure 5. This correspondence helps for comparing the two crisp mean and fuzzy aggregation methods. To compare the two aggregation methods, the inference algorithm should be the same for both aggregation methods. We used ShortestStrongestTrust as the inference algorithm for both mean and fuzzy aggregation methods.

The results of trust inference is reported for every node in the set $\{A, B, \ldots, L\}$ to any other one in the same set in following tables. The result values of ShortestStrongestTrust algorithm with crisp mean aggregation method are shown in Table 1. The results of ShortestStrongestTrust algorithm with fuzzy aggregation method after conversion to approximating linguistic expressions are presented in Table 2. The inferred fuzzy sets are approximated to linguistic expressions such as "medium or very high." Term abbreviations are used for the sake of brevity.

Tables 3 and 4 compare the results of the crisp and fuzzy aggregation methods. Table 3 shows the membership value of each result of the crisp method in the corresponding result of the fuzzy method. The absolute difference between the results of the crisp method and defuzzified results of the fuzzy method are shown in Table 4. The defuzzification is done using the center of gravity defuzzification method.

The average near 1 in Table 3 and the average near 0 in Table 4 show that the two aggregation methods agree generally in the results. The results of crisp mean and fuzzy aggregation methods differ more when contradictory information should be combined. Among the zero

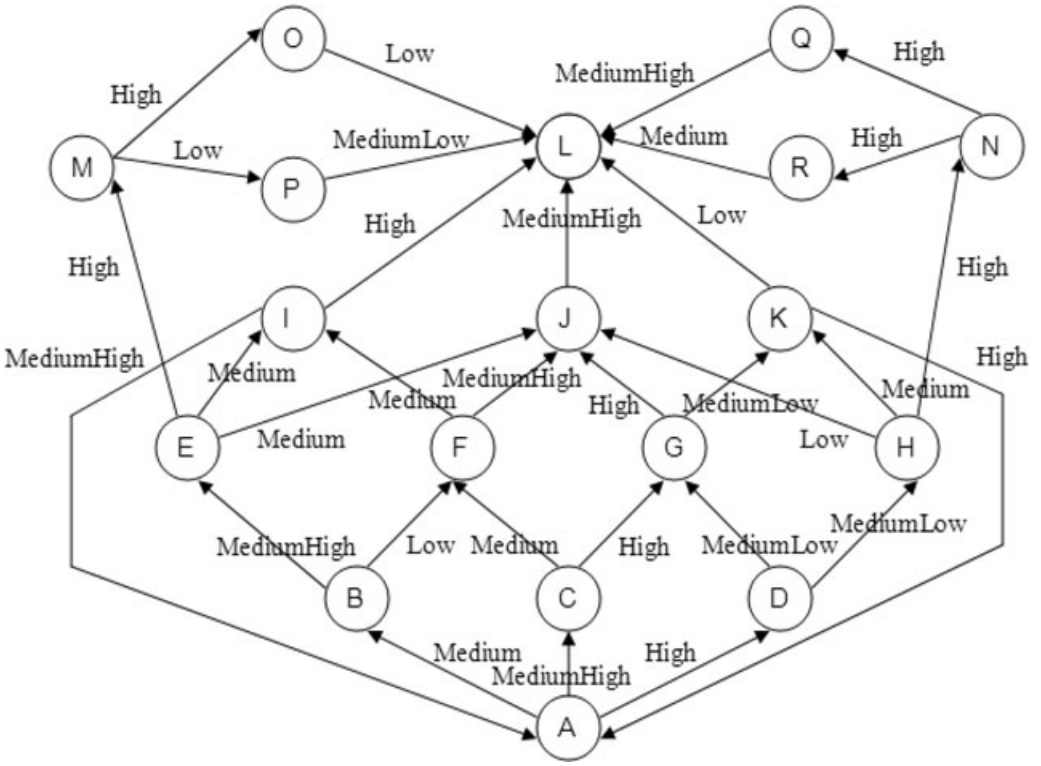

FIGURE 5. Fuzzy trust graph.

TABLE 1. Mean Method Results

	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*
A	—	1	0.8	1	0.8	1	0.3	0.3	1	1	0.3	0.8
B	0.8	—	0.8	1	0.8	0	0.6	0.3	1	0.5	0.3	0.9
C	0.8	1	—	1	0.8	1	1	0.3	1	1	0.3	0.8
D	1	1	0.8	—	0.8	0	0.3	0.3	1	0.5	0.4	0.3
E	0.8	1	0.8	1	—	0	0.6	0.3	1	0.5	0.3	0.9
F	0.8	1	0.8	1	0.8	—	0.6	0.3	1	0.8	0.3	0.8
G	1	1	0.8	1	0.8	0	—	0.3	1	1	0.3	0.8
H	1	1	0.8	1	0.8	0	0.6	—	1	0	0.5	0
I	0.8	1	0.8	1	0.8	1	0.6	0.3	—	1	0.3	1
J	—	—	—	—	—	—	—	—	—	—	—	0.8
K	1	1	0.8	1	0.8	1	0.3	0.3	1	1	—	0
L	—	—	—	—	—	—	—	—	—	—	—	—

values in Table 3 is the value corresponding to trust from D to J. The zero membership value means that the results of the two methods are completely different. There are two shortest paths from D to J that both participate in trust computations. The two neighbors G and H are trusted equally, so they take part in trust inference from D to J equally. The neighbor nodes G and H have contrary trust recommendations for node J; trust of G to J is high (10) while

TABLE 2. Fuzzy Method Results (as Linguistic Expressions)

	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*
A	—	M	MH	H	MH	M	ML	ML	M	H	ML	MH
B	MH	—	MH	H	MH	L	ML or H	ML	M	M	ML	MH or H
C	MH	M	—	H	MH	M	H	ML	M	H	ML	MH
D	H	M	MH	—	MH	L or M	ML	ML	M	L or H	ML or M	L or MH
E	MH	M	MH	H	—	L or M	ML or H	ML	M	M	ML	MH or H
F	MH	M	MH	H	MH	—	ML or H	ML	M	MH	ML	MH
G	H	M	MH	H	MH	L or M	—	ML	M	H	ML	MH
H	H	M	MH	H	MH	L or M	ML or H	—	M	L	M	L
I	MH	M	MH	H	MH	M	ML or H	ML	—	H	ML	H
J	—	—	—	—	—	—	—	—	—	—	—	MH
K	H	M	MH	H	MH	M	ML	ML	M	H	—	L
L	—	—	—	—	—	—	—	—	—	—	—	—

TABLE 3. The Membership Values of the Crisp Method's Results in Fuzzy Method's Results

	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*
A	—	1	1	1	1	1	1	1	1	1	1	1
B	1	—	1	1	1	1	0	1	1	1	1	0.5
C	1	1	—	1	1	1	1	1	1	1	1	1
D	1	1	1	—	1	0.2	1	1	1	0	0.5	0
E	1	1	1	1	—	0.2	0	1	1	1	1	0.5
F	1	1	1	1	1	—	0	1	1	1	1	1
G	1	1	1	1	1	0.2	—	1	1	1	1	1
H	1	1	1	1	1	0.2	0	—	1	1	1	1
I	1	1	1	1	1	1	0	1	—	1	1	1
J	—	—	—	—	—	—	—	—	—	—	—	1
K	1	1	1	1	1	1	1	1	1	1	—	1
L	—	—	—	—	—	—	—	—	—	—	—	—

Average = 0.8945945945945947.

trust of H to J is low (0). The crisp mean aggregation method simply takes the average of 0 and 10, resulting trust value 5 that is just in between (medium), a value that none of the neighbors believe in fact. The linguistic expression resulted from fuzzy aggregation method is "Low or high." This argument also accounts for other disagreements between the results of the mean and fuzzy methods.

The averaging as the combining strategy is incapable to combine contradictory information. In fact, a crisp number has no capacity to convey enough information. Fuzzy aggregation method, on the other hand, benefits from the capacity of membership functions to preserve and convey exactly the information that exists in the trust graph.

5.2. Comparison of Aggregation Methods

Different aggregation methods yield different inferred trust values. A criterion is needed for comparison of aggregation methods. We defined a criterion named Probabilistic Flow for comparison of different trust aggregation methods.

TABLE 4. Absolute Difference between Crisp Method's Results and Defuzzified Values of Fuzzy Method's Results

	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*
A	—	0	0	0.08	0	0	0	0	0	0.08	0	0
B	0	—	0	0.08	0	0.08	0.13	0	0	0	0	0.08
C	0	0	—	0.08	0	0	0.08	0	0	0.08	0	0
D	0.08	0	0	—	0	0.09	0	0	0	0	0	0.23
E	0	0	0	0.08	—	0.09	0.13	0	0	0	0	0.08
F	0	0	0	0.08	0	—	0.13	0	0	0	0	0
G	0.08	0	0	0.08	0	0.09	—	0	0	0.08	0	0
H	0.08	0	0	0.08	0	0.09	0.13	—	0	0.08	0	0.08
I	0	0	0	0.08	0	0	0.13	0	—	0.08	0	0.08
J	—	—	—	—	—	—	—	—	—	—	—	0
K	0.08	0	0	0.08	0	0	0	0	0	0.08	—	0.08
L	—	—	—	—	—	—	—	—	—	—	—	—

Average = 0.030582530582530604.

5.2.1. Reliability. By definition, a node is reliable if the value of trust to it is not below the half of the maximum trust value in the trust value range; otherwise, the node is unreliable. Hence in the trust range of [0, 1], a node is considered to be reliable if the crisp trust value to it is greater than 0.5. In the fuzzy modeling, a node is reliable if the trust fuzzy set to it has nonzero membership values only in the [0.5, 1] range.

5.2.2. Reliability Probability. A path from a node to another one in a trust graph introduces the second node as an information source for the first one. If trust value from a source node to a sink is to be inferred, the nodes that have direct edges to the sink and are at the end of paths from the source provide different trust recommendations. Nodes at the end of stronger paths provide more reliable trust recommendations. According to these facts, the following flow model criterion is proposed.

A flow starts from the source as the current state. It iteratively changes its state to one of the neighbors based on the following probability distribution. If the current node has t_1, $t_2, \ldots, t_n$ crisp trust values to its neighbors N_1, $N_2, \ldots, N_n$, the probability of selecting neighbor N_i as the next node is

$$P(N_i) = \frac{t_i}{\sum_{i=1}^{n} t_i} \tag{10}$$

The probability assignment to each neighbor is proportional to the crisp trust rating to that neighbor. This makes selection of more trusted neighbors more probable and hence stronger paths are more probable to be traversed. The flow continues until it reaches a node that has a direct edge to the sink. The trust rating on this edge is the trust value that is resulted from the flow. According to the trust value that is resulted and the definition of reliability, the flow denotes the sink as reliable or unreliable.

The flow is repeated for several times. Repetitions of the flow may traverse different paths and hence result in different trust values to the sink. Some flows denote the sink node as reliable and other flows denote it as unreliable. We name the percentage of flows that denote the sink node as reliable, the reliability probability. As the probabilistic flows

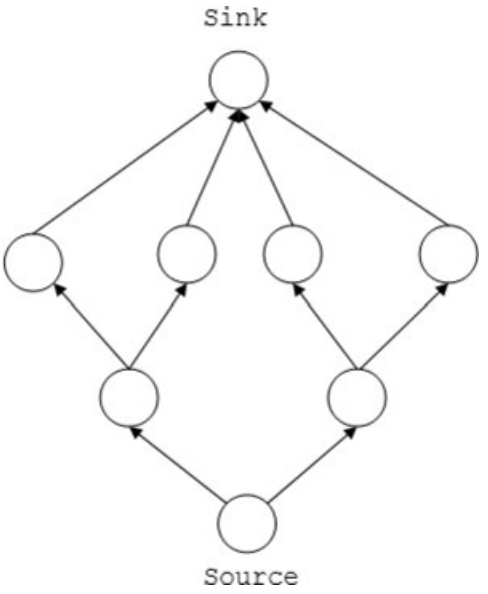

FIGURE 6. The graph that is used for comparison of the three aggregation methods.

traverse stronger paths more probably, results from stronger paths are repeated more and hence stronger paths contribute more to the reliability probability. Increasing the number of repetitions of the flow increases the accuracy of the reliability probability. In our simulations, we repeat the flow until it is done greater than or equal to a definite number of times and all the paths from the source to the sink are traversed at least once. Computing the reliability probability is very time consuming and this accounts for why it is not directly used as a trust inference method and is proposed as a criterion to evaluate performance of aggregation methods.

5.2.3. Simulation. The graph depicted in Figure 6 is used for comparison of the three aggregation methods: mean aggregation, subjective logic aggregation, and fuzzy aggregation methods.

Fuzzy trust ratings are High, MediumHigh, Medium, MediumLow, and Low. The membership functions of these fuzzy terms are shown in Figure 1. To compare the methods, crisp and opinion counterparts of the fuzzy values should be computed. Defuzzification of the fuzzy sets results in the crisp trust ratings. Center of gravity method is used for defuzzification. Crisp probability values are equivalent to expectation values in subjective logic. The mapping from the expectation value to subjective logic opinion that is presented in Section 3.1 is used to compute the trust rating opinions. The default base rate $a = 0.5$ is used in the mappings.

As the graph has 10 edges and there are five different trust terms, there are 10^5 different assignments of terms to the graph edges. The simulation is done for each of the 10^5 graphs. A definite trust aggregation method reports an inferred trust value for each of the graphs. The inferred value indicates whether the aggregation method categorizes the sink node as reliable or unreliable. Hence the graphs are categorized into two sets that are denoted as Reliable and Unreliable in the following tables. Repeating the flow on each graph results a percentage as the reliability probability. The flow is repeated 100 times on each graph for the following results. The average, minimum, and maximum of reliability probability of the

TABLE 5. Simulation Results for Mean Aggregation Method

	Average of reliability probability	Minimum of reliability probability	Maximum of reliability probability
Reliable set	63.11%	00.12%	100.00%
Unreliable set	17.32%	00.00%	78.00%

TABLE 6. Simulation Results of Subjective Logic Aggregation Method

	Average of reliability probability	Minimum of reliability probability	Maximum of reliability probability
Reliable set	67.70%	07.00%	100.00%
Unreliable set	33.52%	00.00%	100.00%

TABLE 7. Simulation Results of Fuzzy Aggregation Method

	Average of reliability probability	Minimum of reliability probability	Maximum of reliability probability
Reliable set	100.00%	100.00%	100.00%
Unreliable set	38.42%	00.00%	99.92%

TABLE 8. Comparison of Different Aggregation Methods

	Denoting only the nodes with a reliability probability of 100% as reliable	Denoting all the nodes with a reliability probability of 100% as reliable
Fuzzy	Yes	Yes
Mean	No	Yes
Subjective logic	No	No

graphs in each set are reported in the following tables. Tables 5–7 show the results for each of the three aggregation methods.

Table 7 shows that all the graphs that the fuzzy aggregation method categorizes in the Reliable set have a reliability probability of 100%. This means that when the fuzzy aggregation method denotes the sink as reliable, all the flows denote it as reliable too. Also no graph with a reliability probability of 100% is present in the Unreliable set. This means that all the graphs for which the flows denote a reliability probability of 100% are also denoted as reliable by the fuzzy aggregation method. Hence, not only every node that the fuzzy aggregation method denotes as reliable has a reliability probability of 100%, but also any node with reliability probability of 100% is denoted as reliable by the fuzzy aggregation method. Table 8 summarizes these two properties for the three aggregation methods that are acquired from Tables 5–7. Only the fuzzy aggregation method denotes all and only the nodes with reliability probability of 100% as reliable. This shows the superiority of the fuzzy modeling and aggregation to the other two methods. Although SL is shown not to

TABLE 9. Fuzzy AllLengthStrongestTrust Algorithm Results.

	ShortestStrongestTrust	ShortestStrongestTrust preciseness	AllLengthStrongestTrust	ShortestStrongestTrust preciseness
A to L	MH	MH	MH	MH
B to L	MH or H	M	*L	*MH
C to L	MH	H	MH	H
D to L	L or MH	ML	*M or MH	ML
E to L	MH or H	M	*L	*H
F to L	MH	MH	MH	MH
G to L	MH	H	MH	H
H to L	L	M	*M or MH	*H
I to L	H	H	H	H
J to L	MH	H	MH	H
K to L	L	H	L	H
L to L	—	—	—	—
M to L	L	H	L	H
N to L	M or MH	H	M or MH	H
O to L	L	H	L	H
P to L	ML	H	ML	H
Q to L	MH	H	MH	H
R to L	M	H	M	H

be competitive here, the strength of it is modeling of distrust in addition to trust. Fuzzy modeling can also be extended to accommodate distrust.

5.3. AllLengthStrongestTrust Versus ShortestStrongestTrust

AllLengthStrongestTrust and ShortestStrongestTrust algorithms are compared on the graph in Figure 5. The simulation results are reported in Table 9. The results of the Fuzzy ShortestStrongestTrust algorithm and preciseness of these results are presented in columns one and two. The results of the Fuzzy AllLengthStrongestTrust algorithm and preciseness of these results are in third and fourth columns, respectively where star depicts a difference between the results of the two algorithms. The three stars in the fourth column that all show improvement in preciseness correspond to the node pairs B to L, E to L, and H to L where paths stronger than the shortest paths exist from the first node to the second; and these stronger paths are well utilized for a more precise inference by the Fuzzy AllLengthStrongestTrust algorithm. The results show improvement in preciseness of inference for AllLengthStrongestTrust in comparison with ShortestStrongestTrust.

6. CONCLUSIONS AND FUTURE WORKS

Trust to people that are dealt with is an important aspect of any business. Auction sites have tried to employ reputation mechanisms to provide trust ratings to other dealers. While the contemporary mechanisms for trust inference generally compute a global trust to dealers, trust can be more precisely modeled as a (direct or indirect) relationship between every pair of dealers. This leads to the personalized view of trust. People naturally try to ask their trusted friends if they themselves do not know someone. To infer personalized trust from a dealer to another, not only algorithms called trust inference algorithms should be proposed to identify

the trust paths, but also the theoretical problem of trust modeling and aggregation needs to be addressed. This research proposes fuzzy modeling of trust and a fuzzy aggregation method. New personalized trust inference algorithms extending previous works are also the contribution of this research.

Aside from the fact that linguistic expressions are naturally more desirable for users to write and read about trust, the results show the superiority of fuzzy aggregation method over mean and Subjective Logic aggregation methods especially when contradictory information should be aggregated. The preciseness of inference that this work reports for each inference is improved by AllLengthStrongestTrust algorithm that performs the inference from the most trusted paths and does not confine the search to shortest paths.

Besides the advantages, it should be mentioned that fuzzy aggregation has a heavier computational load than the two other aggregation methods. The model should also be extended to accommodate distrust in addition to trust.

In addition, although AllLengthStrongestTrust performs the inference more precisely, the time complexity of the algorithm is exponential. The AllLength algorithms may not be utilized in a real-time online application due to their time complexities but can be well employed when a trust graph such as a research citation graph is to be precisely analyzed.

Although this work targeted the virtual social networks, ideas in trust modeling, trust graphs and trust inference are proposed abstractly. Hence, the proposed material can also be used in multiagent systems, distributed systems, and ubiquitous computing. The following paragraphs present some future works that are suggested for researchers interested in the subject.

Length has priority over strength in ShortestStrongestTrust algorithm. Another algorithm can also be proposed that gives priority to strength over length. As stronger paths are favored to weaker but shorter ones, strength of a path seems to be more important than its length.

Users can be asked to determine and specify their trust to others in a limited number of concepts; but it happens to require trust inference in some other concepts. It is desired to infer trust in a concept from the available trust ratings in some other related concepts. This seems possible regarding the fact that related concepts have semantic overlap. A semantic analysis of concepts is needed to accompany trust inference to infer trust in a concept from the trust in other concepts with semantic overlap.

When the deviation in the result of trust inference is great (for example, if the result is "low or high"), usually it is not much helpful for the user. One method to avoid generating such results is to utilize information redundancy. In other words, when combining different information that the neighbors of a node provide, more attention should be paid to the information that the majority of neighbors agree with and infrequent information should be faded. This can also be viewed as a need for a noise cancellation technique. The redundancy that is in fact ignored by the fuzzy aggregation method can be useful in preference of some bits of information to some others to offer more concentrated results.

Another subject is time. A trust rating given yesterday is much more important than a trust rating given one year ago. A mechanism to decrease trust rates with time should be proposed.

REFERENCES

AKERLOF, G. 1970. The Market for Lemons: Quality Uncertainty and the Market Mechanism. Quarterly Journal of Economics, **84**(August):488–500.

ARINGHIERI, R., E. DAMIANI, S. D. DI VIMERCATI, S. PARABOSCHI, and P. SAMARATI. 2006. Fuzzy Techniques for Trust and Reputation Management in Anonymous Peer-to-Peer Systems. Special Topic Section on Soft

Approaches to Information Retrieval and Information Access on the Web. Journal of the American Society for Information Science and Technology, **57**(4):528–537.

CORMEN, T. H., C. E. LEISERSON, R. L. RIVEST, and C. STEIN. 2001. Introduction to Algorithms. *In* The Massachusetts Institute of Technology, 2nd Edition, 1180 pages.

GOLBECK, J. 2005. Computing and Applying Trust in Web-based Social Networks. Ph.D. dissertation, University of Maryland, College Park, 200 pages.

GOLBECK, J., and J. HENDLER. 2006. Inferring Trust Relationships in Web-Based Social Networks. ACM Transactions on Internet Technology, **6**(4):497–529.

HUYNH, T. D., N. R. JENNINGS, and N. R. SHADBOLT. 2004. Developing an Integrated Trust and Reputation Model for Open Multi-Agent Systems. *In* Proceedings of 7th International Workshop on Trust in Agent Societies.

JØSANG, A., R. ISMAIL, and C. BOYD. 2007. A Survey of Trust and Reputation Systems for Online Service Provision. Decision Support Systems, **43**(2):618–644.

JØSANG, A., L. GRAY, and M. KINATEDER. 2006. Simplification and Analysis of Transitive Trust Networks. Web Intelligence and Agent Systems Journal, **4**(2):139–161.

JØSANG, A., R. HAYWARD, and S. POPE. 2006. Trust Network Analysis with Subjective Logic. *In* Australasian Computer Science Conference (ACSC'06), Hobart.

JØSANG, A. 2001. A Logic for Uncertain Probabilities. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, **9**(3):279–311.

JURCA, R., and B. FALTINGS. 2007. Obtaining Reliable Feedback for Sanctioning Reputation Mechanisms. Journal of Artificial Intelligence Research (JAIR), **29**:391–419.

KAMVAR, S. D., M. T. SCHLOSSER, and H. GARCIA-MOLINA. 2003. The EigenTrust Algorithm for Reputation Management in P2P Networks. *In* Proceedings of the Twelfth International World Wide Web Conference, Budapest.

KERR, R., and R. COHEN. 2007. Towards Provably Secure Trust and Reputation Systems in E-Marketplaces. *In* Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007).

LEVIEN, R. 2004. Attack Resistant Trust Metrics. Ph.D. dissertation, University of California at Berkeley.

PAGE, L., S. BRIN, R. MOTWANI, and T. WINOGRAD. 1998. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project.

RESNICK, P., R. ZECKHAUSER, E. FRIEDMAN, and K. KUWABARA. 2000. Reputation Systems. Communications of the ACM, **43**(12):45–48.

RESNICK, P., R. ZECKHAUSER, J. SWANSON, and K. LOCKWOOD. 2002. The Value of Reputation on e-Bay: A Controlled Experiment. Experimental Economics, **2**(9):2006.

RUSSEL, S., and P. NORVIG. 2003. Artificial Intelligence, a Modern Approach. *In* Pearson Education, 2nd Edition, 1080 pages.

ZIEGLER, C. N., and G. LAUSEN. 2004. Spreading Activation Models for Trust Propagation. *In* Proceedings of the IEEE International Conference on e-Technology, e-Commerce, and e-Service (EEE '04), Taipei.

APPENDIX

1.1. Distributivity

A Directed Series Parallel Graph (DSPG) defined in Jøsang et al. (2006) is a trust graph (subgraph) containing some paths from a source to a sink that can be represented as a closed form. Two operations for serial and parallel combination denoted by : and $\diamond$ symbols are used to represent a DSPG. As an instance the trust graph in Figure 7(a) is represented as $((A_1 : C_1) \diamond (A_2 : C_2)) : B$.

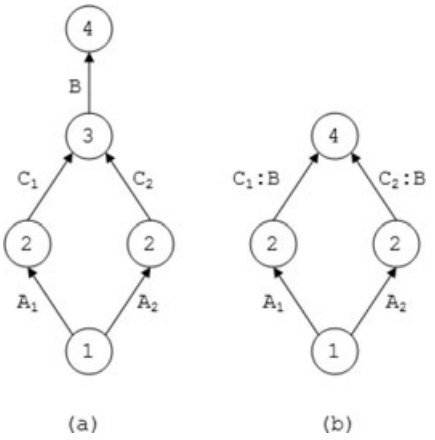

FIGURE 7. (a) The trust graph that is viewed as (b) by incremental sink to source traverse.

To infer the trust from the source to the sink, the corresponding closed form should be evaluated; therefore the closed form of a DSPG is needed to be obtained first. Less computation would be needed if the trust could be evaluated from the graph itself. The incremental trust inference algorithm infers trust not from the closed form but incrementally level by level back from the sink to the source. For example after two levels of trust inference, the graph in Figure 7(a) is reduced to the graph in Figure 7(b). The trust value from the source to the sink is defined to be the value obtained from the evaluation of the closed form. Hence, only a trust aggregation method that results the same value from node 1 to node 4 in Figures 7(a) and (b) can be used with the incremental trust inference algorithm. This means that the incremental computation approach can only be benefited with trust aggregation methods that satisfy the following distribution for : and $\diamond$ operators.

$$((A_1 : C_1)\diamond(A_2 : C_2)) : B = (A_1 : (C_1 : B))\diamond(A_2 : (C_2 : B)). \tag{13}$$

Subjective Logic reduction and consensus operators are not distributive to each other (Jøsang 2001) and therefore Subjective Logic aggregation method cannot be used with the incremental trust inference algorithm. However it will be shown that crisp mean and fuzzy rule-based aggregation methods satisfy the distribution and hence both can be used with incremental trust inference algorithms.

For crisp mean aggregation method it can be defined that

$$(A_1 : B_1)\diamond(A_2 : B_2) = \frac{(A_1 \times B_1) + (A_2 \times B_2)}{A_1 + A_2} \tag{14}$$

and when there is no parallel path

$$A_1 : B_1 = (A_1 : B_1)\diamond(0 : 0) = \frac{A_1 \times B_1 + 0 \times 0}{A_1 + 0} = B_1 \tag{15}$$

Hence the equality is proved as follows.

$$T_a = ((A_1 : C_1)\diamond(A_2 : C_2)) : \mathrm{B} = \left(\frac{(A_1 \times C_1) + (A_2 \times C_2)}{A_1 + A_2}\right) : \mathrm{B}$$
$$= \left(\frac{\frac{(A_1\times C_1)+(A_2\times C_2)}{A_1+A_2} \times \mathrm{B}}{\frac{(A_1\times C_1)+(A_2\times C_2)}{A_1+A_2}}\right) \tag{16}$$

$$T_b = (A_1 : (C_1 : B))\diamond(A_2 : (C_2 : B)) = (A_1 : (B))\diamond(A_2 : (B))$$
$$= \frac{A_1 \times (B) + A_2 \times (B)}{A_1 + A_2} = B \tag{17}$$

For fuzzy rule based aggregation method : and $\diamond$ operators are defined as

$$\begin{aligned} A_1\diamond A_2 &= A_1 \cup A_2 \\ A_1 : A_2 &= \mathit{FuzzyInference}(A_1, A_2). \end{aligned} \tag{18}$$

Where $\mathit{FuzzyInference}(A_1, A_2)$ is the result of inference with the fuzzy rule introduced in subsection 3.2.3 with A_2 as the *BelievedValue* and A_1 as the Trust fuzzy set. According to the fuzzy aggregation method the : operator can be written as

$$A_1 : A_2 = \mathit{FuzzyInference}(A_1, A_2) = \mathit{Super}(A_1 \cap \mathit{Acceptable}) \times A_2 \tag{19}$$

Where Super() is a function that gets a fuzzy set as the input and returns the value with the maximum membership in the support set of the fuzzy set. Multiplication of a scalar to a fuzzy set scales down the fuzzy set.

$$\begin{aligned} &T_b \\ &= (A_1 : (C_1 : B))\diamond(A_2 : (C_2 : B)) \\ &= \mathit{FuzzyInference}(A_1 : (C_1 : B)) \cup \mathit{FuzzyInference}(A_2 : (C_2 : B)) \\ &= [\mathit{Super}(A_1 \cap \mathit{Acceptable}) \times (C_1 : B)] \cup [\mathit{Super}(A_2 \cap \mathit{Acceptable}) \times (C_2 : B) \\ &= [\mathit{Super}(A_1 \cap \mathit{Acceptable}) \times \mathit{FuzzyInference}(C_1, B)] \\ &\quad \cup[\mathit{Super}(A_2 \cap \mathit{Acceptable}) \times \mathit{FuzzyInference}(C_2, B)] \\ &= [\mathit{Super}(A_1 \cap \mathit{Acceptable}) \times \mathit{Super}(C_1 \cap \mathit{Acceptable}) \times B] \\ &\quad \cup[\mathit{Super}(A_2 \cap \mathit{Acceptable}) \times \mathit{Super}(C_2 \cap \mathit{Acceptable}) \times B] \\ &= max\left\{\begin{array}{l}[\mathit{Super}(A_1 \cap \mathit{Acceptable}) \times \mathit{Super}(C_1 \cap \mathit{Acceptable})], \\ [\mathit{Super}(A_2 \cap \mathit{Acceptable}) \times \mathit{Super}(C_2 \cap \mathit{Acceptable})]\end{array}\right\} \times B \\ &= max\left\{\begin{array}{l}\mathit{Super}[\mathit{Super}(A_1 \cap \mathit{Acceptable}) \times (C_1 \cap \mathit{Acceptable})], \\ \mathit{Super}[\mathit{Super}(A_2 \cap \mathit{Acceptable}) \times (C_2 \cap \mathit{Acceptable})]\end{array}\right\} \times B \\ &= \mathit{Super}\left\{\begin{array}{l}[\mathit{Super}(A_1 \cap \mathit{Acceptable}) \times (C_1 \cap \mathit{Acceptable})]\cup \\ [\mathit{Super}(A_2 \cap \mathit{Acceptable}) \times (C_2 \cap \mathit{Acceptable})]\end{array}\right\} \times B \\ &= \mathit{Super}\left\{\begin{array}{c}\{[\mathit{Super}(A_1 \cap \mathit{Acceptable}) \times C_1] \cup [\mathit{Super}\,(A_2 \cap \mathit{Acceptable}) \\ \cap \mathit{Acceptable}\end{array}\right\} \times B \\ &= \mathit{Super}\{(A_1 : C_1)\diamond(A_2 : C_2) \cap \mathit{Acceptable}\} \times B \\ &= (A_1 : C_1)\diamond(A_2 : C_2) : B \end{aligned} \tag{20}$$

1.2. ShortestAllStrengthTrust Algorithm

This algorithm infers trust from a source to a sink benefiting from all the shortest paths. The concern of this algorithm is to infer trust from the shortest paths possible; hence it is especially useful in trust graphs in which longer inference paths inherently decrease the validity of inference. If the trust ratings in the trust graph are taken from the users for an atomic concept, the number of transitions or the path length is not an important issue; but when the trust rating is asked in a general or broad concept that comprises some other concepts, the path length matters. For example, consider a social network where users are asked for trust to their friends in a general concept of C, where C includes two other concepts C_1 and C_2. Assume that person P_1 trusts person P_2 in concept C_1 and person P_2 trusts person P_3 in concept C_2; So P_1 rates P_2 and P_2 rates P_3 as trusted in concept C. There are trust ratings from P_1 to P_2 and from P_2 to P_3 but inferring that P_3 is trusted by P_1 in C and automatically directing information in C_1 from P_3 to P_1 is not correct. If the concept that the trust is asked for, is so general that is at high risk of change from a person to another, this algorithm is the proper one. Because the algorithm is presented in detailed pseudo-code, the verbal explanation is omitted not to extend the text.

```
ShortestAllStrengthTrust algorithm pseudo code
Trust inferShortestAllStrengthTrust( Node source, Node sink )
{
       // To infer the trust from the source node to the sink node from
       // shortest but with any strength paths.
       // In-context variables:
       //     TrustGraph trustGraph:
       //           Representing the trust graph
       //     Trust[] trustsToSink:
       //           Representing the array of trust from all the nodes in
       //           the trust graph to the sink node that is computed in
       //           the procedure.

if ( source = sink )
       return MAX_TRUST_VALUE

nodeCount = trustGraph.getNodeCount()
trustsToSink = new Trust[ nodeCount ]
initialize each trustsToSink[ i ] to DUMMY
isVisited = new boolean[ nodeCount ]
initialize each isVisited[ i ] to false

queue = new Queue()
nextLevelQueue = new Queue();
// The Queue class queues an object if
//    it is not previously present in the queue.
sourceFound = false

adjacentsToSink = trustGraph.getAdjacentsTo( sink )
for each adjacentToSink in adjacentsToSink
       queue.queue( adjacentToSink )

while ( not queue.isEmpty() )
{
       currentNode = queue.dequeue()

       if ( not sourceFound )
       {
              computeTrustToSink( currentNode )
              if (not trustGraph.isAdjacent( source, node ) )
```

```
                {
                        adjacentsTo = trustGraph.getAdjacentsTo( currentNode )
                        for each adjacentTo in adjacentsTo
                                if ( not isVisited[ adjacentTo ] )
                                {
                                        isVisited[ adjacentTo ] = true
                                        nextLevelQueue.queue( adjacentTo )
                                }
                }
                else
                        sourceFound = true
        }
        else
                if ( trustGraph.isAdjacent( source, currentNode ) )
                        computeTrustToSink( currentNode )

        if ( queue.isEmpty() and (not sourceFound) )
        {
                queue = nextLevelQueue
                nextLevelQueue = new Queue()
        }
}

computeTrustToSink( source )

return trustsToSink[ source ]
}

Trust computeTrustToSink( Node node )
{
        // Used in the inferShortestAllStrengthTrust algorithm to compute the
        //    trust from a node to the sink node.
        // In-context variables:
        //     TrustGraph trustGraph:
        //             Representing the trust graph
        //     Trust[] trustsToSink:
        //             Representing the array of trust from all the nodes in
        //             the trust graph to the sink node

if ( trustGraph.isAdjacent( node, sink ) )
        trustsToSink[ node ] = trustGraph.getTrust( node, sink )
else
{
        participatingBeliefs = new Set()
        adjacents = trustGraph.getAdjacents( node )
        if (adjacents.length = 0)
                return DUMMY;
        for each adjacent in adjacents
                if ( trustsToSink[ adjacent ] ≠ DUMMY )
                {
                        belief = new Belief(
                                trustGraph.getTrust( node, adjacent ),
                                trustsToSink[ adjacent ] )
                        // Support to the believed value as the first
                        //    argument and the believed value as the second
```

```
                    //    argument.
                    participatingBeliefs.add( belief )
                }
        trustsToSink[ node ] = combineBeliefs( participatingBeliefs )
}

return trustsToSink[ node ]
}
```

1.3. AllLengthAllStrengthTrust Algorithm

Although the shortest and strongest paths conduct the most reliable information, longer and weaker paths also convey information. This information can also contribute to the inference. AllLengthAllStrengthTrust algorithm infers the trust from a source to a sink in the trust graph benefiting from all the existing paths.

The algorithm is basically a Tree Search with the breadth first strategy (Russel and Norvig 2003) with the sink as the initial node and source as the goal node. The difference is that a tree search terminates as the goal is found but AllLengthAllStrength algorithm continues iteration even if the source node is found to traverse all the paths back from the sink to the source. Also employing the priority queue makes the search tree to grow in some branches more rapidly than others.

The algorithm pseudo-code is as follows. As the pseudo-code is written clearly in detail, it is not explained.

```
AllLengthAllStrengthTrust algorithm pseudo code
Trust inferAllLengthAllStrengthTrust( Node source, Node sink )
{
        // To infer the trust from the source node to the sink node from
        //     any length and any strength paths.
        // In-context variables:
        //     TrustGraph trustGraph:
        //           Representing the trust graph
        // Trust[] trustsToSink:
        //           Representing the array of trust from all the nodes in
        //              the trust graph to the sink node
if ( source = sink )
        return MAX_TRUST_VALUE

nodeCount = trustGraph.getNodeCount()
trustToSink = new Trust[ nodeCount ]
initialize all the trustsToSink[ i ] to DUMMY
isVisited = new boolean[ nodeCount ]
initialize all the isVisited[ i ] to false
queue = new Queue()
priorityQueue = new Queue()
// The Queue class queues an object if
//    it is not previously present in the queue.

adjacentsToSink = trustGraph.getAdjacentsTo( sink )
for all adjacentsToSink[ i ]
{
        trustsToSink[ adjacentsToSink[ i ] ] =
        trustGraph.getTrust( adjacentsToSink[ i ], sink )
        queue.queue( adjacentToSink )
}
```

```
while ( (not priorityQueue.isEmpty()) or (not queue.isEmpty()) )
{
        if ( not priorityQueue.isEmpty() )
        currentNode = priorityQueue.dequeue()
        else
                currentNode = queue.dequeue()

        isVisited[ currentNode ] = true
        adjacentsTo = trustGraph.getAdjacentsTo( currentNode )
        for all adjacentsTo[ i ]
        {
                if ( adjacentsTo[ i ] = sink )
                        continue
                // Computing how much the sink trusts himself is useless.

                if ( not isVisited[ adjacentsTo[ i ] ] )
                {
                        updateTrustToSink( adjacentsTo[ i ] )
                        queue.queue( adjacentsTo[ i ] )
                }
                else // if ( isVisited[ adjacentsTo[ i ] ] )
                {
                        previousTrust = trustsToSink[ adjacentTo ]
                        newTrust = updateTrustToSink( adjacentTo )
                        if ( previousTrust ≠ newTrust )
                                priorityQueue.queue( adjacentTo )
                }
        }
}
return trustsToSink[ source ]
// If no path is found to the source, trustsToSink[ source ] is DUMMY that
//    is returned.
}


Trust updateTrustToSink( Node node )
{
        // Used in the inferAllLengthAllStrengthTrust algorithm to update the
        //     trust from a node to the sink node.
        // In-context variables:
        //     TrustGraph trustGraph:
        //             Representing the trust graph
        //     Trust[] trustsToSink:
        //             Representing the array of trust from all the nodes in
        //                the trust graph to the sink node

participatingBeliefs = new Set()
adjacents = trustGraph.getAdjacents( node )
for all adjacents[ i ]
        if ( trustsToSink[ adjacents[ i ] ] ≠ DUMMY )
        {
                if ( adjacents[ i ] == sink )
                        belief = new Belief(
                                MAX_TRUST_VALUE,
                                trustGraph.getTrust( node, adjacents[ i ] )
                else
                        belief = new Belief(
                        trustGraph.getTrust( node, adjacents[ i ] ),
                        trustsToSink[ adjacents[ i ] ] )
                // Support to the believed value as the first
```

```
            //    argument and the believed value as the second
            //    argument.
            participatingBeliefs.add( belief )
        }
trustsToSink[ node ] = combineBeliefs( participatingBeliefs )
return trustsToSink[ node ]
}
```

1.4. Time complexity

In the following time complexity analysis for the four incremental inference algorithms, V is the number of vertices and E is the number of edges in the trust graph.

As ShortestStrongestTrust and ShortestAllStrengthTrust algorithms are Graph Search algorithm with the breadth first strategy and the time complexity of a Graph Search with the breadth first strategy is $O(V + E)$ (Cormen et al. 2001), therefore the time complexity of both algorithms is $O(V + E)$. The following two paragraphs approve this time complexity by analyzing the pseudo-code in detail.

First the time complexity of ShortestStrongestTrust algorithm is explained. Within the first while loop, each node that is iterated and added to the queue is marked as visited and no previously visited node is added to the queue. This means that any node can only be dequeued from the queue and iterated once. Hence the time complexity for the while statement is $O(V)$. The total time needed for the statements in the while loop that iterate the adjacency list of each node is $O(E)$. The second while loop iterates the same nodes as the first loop; so it also runs in $O(V)$. Within this loop, the trust computation for each node needs iteration of that node's adjacency list; hence the trust computation totally runs in $O(E)$ time. Thus the total time of the algorithm is $O(V + E)$.

There is a traverse from the sink back to the source in the ShortestAllStrengthTrust algorithm. As the shortest paths are to be found, any link back to previous levels is ignored. At most all the nodes of the graph are queued; so the total run time of the while loop statement is $O(V)$. Computing the trust from each node to the sink requires iterating its adjacent nodes; so the total run time of trust computation is $O(E)$. The for loop that iterates the nodes that are adjacent to the dequeued node also runs in a total time of $O(E)$. Thus, the time complexity of the algorithm is $O(V + E)$.

In AllLengthAllStrengthTrust, the while loop implements a Tree Search with a breadth first strategy with two differences. First, as the priority queue makes the search tree to grow in some branches deeper than the current depth of breadth first search, the grown branches can be deeper than the final depth of breadth first search in the worst case. Second AllLengthAllStrengthTrust continues until the queues become empty while Tree Search terminates sooner when the goal node is found for the first time. Hence, in the worst case the number of nodes iterated by AllLengthAllStrengthTrust is greater than the number of nodes iterated by Tree Search. It is known that the time complexity of Tree Search with the breadth first strategy is $\Omega(b^d)$ where b is the branching factor, i.e., the average out-degree of nodes and d is the length of the shortest path from the initial node to the goal node (Russel and Norvig 2003). Therefore, the time complexity of AllLengthAllStrengthTrust algorithm is also $\Omega(b^d)$.

AllLengthStrongestTrust algorithm performs a search from the source to the sink and then another search from the sink back to the source. The second search is exactly similar to the search of AllLengthAllStrengthTrust algorithm that was explained to have time order of $\Omega(b^d)$. The first search from the source to the sink is a Tree Search with the same two differences between AllLengthAllStrengthTrust algorithm and Tree Search. Same arguments

can be offered to explain that the time order of the first search is also $\Omega(b^d)$. Thus the time complexity of AllLengthStrongestTrust algorithm is totally $\Omega(b^d)$.

The exact time complexity of AllLength algorithms is dependent on the number of reiterations that is related to structure of the trust graph and its trust ratings. As the time complexity of the problem of finding all the paths from a source to a sink in a graph is exponential, the algorithms that are to infer trust from all the existing paths naturally have exponential time complexity. Any algorithm that claims a lower time complexity does not search sufficiently deep in the graph and runs the risk of missing some paths.