A shorter version published in
Canadian Semantic Web Working Symposium 2006

Applying and Inferring Fuzzy Trust in Semantic Web Social Networks

Mohsen Lesani and Saeed Bagheri

Computer Engineering Department, Sharif University of Technology, Tehran, Iran. mohsen_lesani@mehr.sharif.edu and bagheri@sharif.edu

Abstract. Social networks let the people find and know other people and benefit form their information. Semantic Web standard ontologies support social network sites for making use of other social networks information and hence help their expansion and unification, making them a huge social network. As social networks are public virtual social places much information may exist in them that may not be trustworthy to all. A mechanism in needed to rate coming news, reviews and opinions about a definite subject from users, according to each user preference. There should be a feature for users to specify how much they trust a friend and a mechanism to infer the trust from one user to another that is not directly a friend of the user so that a recommender site can benefit from these trust ratings for showing trustworthy information to each user from her or his point of view from not only her or his directly trusted friends but also the other indirectly trusted users. This work suggests using fuzzy linguistic terms to specify trust to other users and proposes an algorithm for inferring trust from a person to another person that may be not directly connected in the trust graph of a social network. The algorithm is implemented and compared to an algorithm that let the users to specify their trust with a number in a definite range. While according to the imprecise nature of the trust concept writing and reading a linguistic expression for trust is much more natural than a number for users, the results show that the algorithm offers more precise information than the previously used algorithm especially when contradictory beliefs should be composed and also when a more precise inference is potentially possible in searching deeper paths. As the trust graphs and inference are viewed abstractly, they can be well employed in other multi agent systems.

1 Introduction

All the information that we have is what we ourselves have devised or we have gotten from the others. The information we get from others about different subjects is obviously an important part of our knowledge bases. The social networks such as LiveJournal (http://www.livejournal.com), Orkut (http://www.orkut.com) and so on are growing and becoming more popular day by day. Web based social network sites offer many facilities to their users to find their friends, make new ones and specify their relationships. Some of them let their users to rate and write reviews about films, shows, events and so on. Millions of people are connected to each other in each of such social networks making a large relationship graph. Semantic Web community has proposed a general ontology for people relationships in FOAF (friend of a friend) project (http://www.foaf-project.org/) and many social network sites also offer their user relationship information in accordance to this vocabulary in RDF (http://www.w3.org/RDF/) or OWL (http://www.w3.org/2004/OWL/) files. This makes the foundation for a large distributed social network comprising those individual social networks information. A person may know and be connected to hundreds of people but may not know many others. The user may know how much she or he should rely on the persons she or he directly knows but how about the others? How much does she or he trust a review that is written by a person she or he does not directly know? Recommender sites are favored that offer individualized recommendation for each user about a subject e.g. a film and also an ordered list of reviews from the other users that are directly or indirectly reliable for the user. There is a need to infer how much a person trusts another person in the social network that is not directly connected to the person provided that a person can only specify her or his trust to his direct friends.

Golbeck [1] has researched on trust in social networks and deduced some trust graphs properties from real networks and proposed an algorithm that has called TidalTrust algorithm for inferring trust in trust graphs the trust ratings in which can be numbers in a continuous range.

While the users should specify their trusts to other persons as numbers in TidalTrust [1] algorithm, people naturally use linguistic expressions when they are asked about their trust to other individuals. People tend to specify their trust in expressions like very low, medium or high and so on and specifying trust in crisp numbers may seem odd to them. A user that trusts another user by a medium high value may find no or little difference between 6 and 7 trust 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, although this can be told more strongly when a subjective study of users supports it. In connection with the crisp nature of the TidalTrust algorithm, its averaging scheme for composition sometimes infers incorrect trust values. It seems that a crisp number is not enough for conveying accurate information especially contradictory information. Aside form the crisp view of trust, TidalTrust does not report the inference preciseness while the inference along a lowly trusted path even if yielding the same inferred trust value is

less precise than the inference done along a highly trusted path. Also TidalTrust confines the search to shortest paths while the information inferred from a long chain of people with high trust between them may be much more precise than the information inferred from a short chain of people with low trust between them.
According to the aforementioned considerations about crisp view of trust and as trust is generally an ambiguous concept fuzzy logic seems an ideal for trust modeling and inference. This research proposes using fuzzy linguistic terms for specifying trust and offers an algorithm called FuzzyTrust algorithm for inferring trust in social networks with such linguistic terms trust ratings. The inference preciseness is also computed for each inference. The FuzzyTrust algorithm results agree well with the results of TidalTrust algorithm for corresponding trust graphs while it outperforms the TidalTrust in reporting richer information especially in contradictory composition situations. A modified version of FuzzyTrust considers not only the shortest paths but also all the longer ones to find the strongest of all the possible paths and results show inference preciseness improvement when longer paths are more trusted sources.

This paper presents some properties of trust first and then the Golbeck's TidalTrust algorithm is explained and then the proposed FuzzyTrust algorithm is offered. Simulation and Results present the experiments and comparisons of the two algorithms and at the end conclusion and future works sections will conclude the paper.

2 Properties of Trust and Trust Graphs

Although trust can be used in different contexts but its meaning is intuitive in each case. Modeling the persons as nodes and friendships or acquaintances as directed edges and trust values as edge labels, social networks are viewed as large directed graphs. The trust rating on an edge is the trust that the source node has to the sink or equally to the information coming from the sink.

2.1 Asymmetry

As two friends have different beliefs and have seen different behaviour form each other, although not usually, the trust they have to each other may be different. This is what makes the trust graph asymmetric.

2.2 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. But it is usual that you ask one of your highly trusted friends about an unknown person 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 her or his highly trusted friend C that knows about the film while A is unaware of this relationship. B takes C's opinion as her or his own and gives it to A that will also take it as her or his own. A finally takes C's opinion. It is seen that trust is transitive in this sense. Transitivity lets the trust to a person to pass back through a chain of people.

2.3 Composability

If A does not know C, she or he asks her or his friends (Bs) about C. Different friends (Bs) may have different ideas about that person (C). Person A should compose the different ideas she or he receives about C from Bs to infer a unique idea about C. People naturally compose trust value when they receive them from different sources maybe giving higher importance to more trusted sources.

2.4 The Stronger Paths, the More Accurate Inference

When you are to infer the trust to a person and you have the choice of two people chains that have the same depth, you certainly choose the chain that more trust is between the people along it. Golbeck [1] showed that paths with higher trust ratings cause better trust inference.

2.5 The Shorter Paths, the More Accurate Inference

When you hear news from a witness of an event you believe it more than when you hear from some one who has heard from a witness or when you get the news from a deeper chain of people. Similarly when the trust should be inferred it is more desirable to be inferred from a shorter chain of people. Golbeck showed that shorter paths lead to more accurate trust inference results.

She also has presented the following properties. People tend to use higher values of trust when they rate their connections. This is because people are naturally friend with people that they trust more. The people that are connected with high trust ratings agree more and rate other people more similarly than people that are connected with low trust ratings. Also people with high trust rating connections have more common friends than people with lower ones.

3 Tidal Trust Algorithm

The TidalTrust algorithm supports the trust values to be numbers in a continuous range. A simple algorithm for inferring trust in binary trust networks in which only 0 or 1 trust values are allowed where 1 indicates having trust to and 0 indicates having no idea of another person is presented first as a basis for the Tidal-

Trust algorithm. While the pseudo code is a brief explanation, the detailed code is also presented for more clarity in the appendix.

```
If source is adjacent to the sink
   return trust rating in the trust graph from the source to
the sink.
else
  compute the trust from the trusted neighbors of the source
  to the sink recursively.
  compute the inferred trust as a rounded average between
  the trust of the trusted neighbors to the sink.
```

This algorithm simply returns the trust rating from the source to the sink if they are directly connected and if not, all the trusted neighbors trust to the sink are computed recursively and then an average of them and rounding makes the result. It should be noted that the neighbors with 1 trust rating participate in average computation and all 0 trust rated neighbors are ignored as they are believed to convey no trustworthy information. An extension to this algorithm is to round the trust only at the original source and retain the intermediate trust results as numbers in [0, 1] range.

In order to extend the mentioned average to continuous trust ratings, weighted average is used. If $trust_{sr}$ represents trust rating and $iTrust_{sr}$ represent the inferred trust from s to r, $iTrust_{sr}$ is given by:

$$iTrust_{sr} = \frac{\sum_{i \in adjacent(s)} trust_{si} \times iTrust_{ir}}{\sum_{i \in adjacent(s)} trust_{si}}$$

TidalTrust makes use of the aforementioned properties of the shorter and the stronger paths. The most trustworthy information comes from highest trusted paths and lower trusted ones give lower trustworthy information. 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 path strength from one node to another (considering all paths) is defined as the maximum path strength in all the paths between them and equally the path from source to sink strength value is the maximum trust value that at least one path from source to the sink with all the trust ratings along it excluding the last greater than or equal to the value can be found; this strength will be called the required strength. The reason behind excluding the last trust rating in path strength computation is that although its value is of trust nature but it is the information that we get from the path and all the previous trust ratings along the path contribute to the trust we have to this information. The strength of a path with zero length is the max value by definition.

While the computation excluding the last trust rating is called strength computation, the counterpart including the last trust rating is called the complete strength computation. Note that strength computation in the manner defined is only for the strength from the source to the sink computation and all the strength from the

source to midway nodes computations in the incremental strength computation as presented in the algorithm are complete strength computations. The algorithm presented in Golbeck dissertation does not attention to the difference of the last rating but the algorithm presented here for TidalTrust has fixed this.

TidalTrust algorithm proposes to limit the information used in averaging to higher trusted paths i.e. giving zero weight to neighbors along lower trusted ones. For inferring trust from a node to the sink, from the node's neighbors that have a path to the sink with a strength greater than or equal to the required strength and hence the trust from them to the sink can be inferred sufficiently strongly those with trust ratings to them greater than or equal to the required strength participate in averaging in TidalTrust algorithm so the weighted averaging equation will become.

$$iTrust_{sr} = \frac{\sum_{i \in Participating(s)} trust_{si} \times iTrust_{ir}}{\sum_{i \in Participating(s)} trust_{si}}$$

Where
Participating(s) =
{n ∈ Nodes |
adjacent(s, n) and trust(s, n) ≥ RequiredStrength
and
pathFromTo(p, n, r) and strength(p) ≥ RequiredStrength}

As it is mentioned shorter paths lead to more accurate inference so TidalTrust seeks the shortest paths from the source to the sink by an algorithm similar to breadth first search and any other paths longer than the shortest ones are ignored and the nodes along them do not participate in the inference.

The TidalTrust algorithm which supports inference in trust graphs with continuous trust rating is as follows. Firstly the pseudo code is presented that conveys the main ideas and then a more delved explanation will come afterwards. Besides the pseudo codes, some simplified code snippets are presented in the appendix that are reviewed and simplified for reading while their formal object oriented structure is preserved. The reader is recommended to follow the simplified codes along the algorithm explanation.

```
The TidalTrust algorithm pseudo code
Forward wave: The required strength computation
      Iterate the nodes from the source to the sink similar
      to the breadth first search level by level to find
      shortest paths.
      Set the path strength from source to any node in the
      next level using the previously set path strengths
      from the source to current level nodes.
Backward wave: The trust from nodes to the sink inference
```

```
Iterate the nodes form the sink to the source level
by level.
If the node is adjacent to the sink then its inferred
trust is simply its trust rating to the sink in the
trust graph.
Else (If the node is not adjacent to the sink) do
weighted averaging on the neighbors with trust rat-
ings to them greater than or equal to the required
strength and that also have a path to the sink with a
strength greater than or equal to the required
strength and hence the trust from them to the sink is
previously inferred strongly enough.
```

The algorithm is named TidalTrust because computation flows from source to sink and then back from sink to source. The algorithm starts with the source node and iterates the adjacent nodes level by level similar to the breadth first search algorithm. Whenever a node is seen it is pushed to a stack to take the nodes in reverse order later. When a node that is not adjacent to the sink is visited, all its neighbors are marked visited and added to the next level search queue if they are not visited before. No previously visited nodes are placed in the next level queue nodes since they have been placed in the previous level queues and shorter paths to them are considered. The path strength to each of the next level search queue nodes are updated whenever a new path containing the current node from the source to them is found and as all the current level nodes are iterated before next level nodes, path strength values for next level nodes is finalized before they are iterated and hence can be used in updating their next level nodes path strength values. Note that the trust ratings from the last level nodes directly to the sink do not contribute to the strength computation form the source to the sink.

If a node is seen that is adjacent to the sink the minimum path length from the source to the sink is set and path strength from source to sink is updated. The functions min and max are supposed to return the minimum and maximum of their parameters respectively and the other parameter when one of the parameters has an invalid value. When all the current level nodes are iterated, depth variable is incremented, the next level search queue replaces the current one and iteration proceeds until all the nodes with a depth below or equal to the minimum depth are iterated. If the depth is equal to max integer value after the loop it means that it is never updated i.e. all the nodes in the connected component containing the source are iterated and no path from the source to the sink could be found, the trust can not be inferred and a dummy value is returned instead of an inferred trust value.

The nodes are popped from the stack and iterated in the reverse order. 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 inferred trusts of its neighbors to the sink according to the aforementioned weighted averaging formula. As the nodes are reversely iterated level by level from the sink to the source, the trusts values from all of a node neighbors to sink that are required for the trust inference of that node are set before the trust to sink value for the node is inferred. Only the neighbor nodes that the trust rating to them is greater than or equal to the required strength and are along a path to the sink stronger or

as strong as the required strength should participate in the trust computation. If a neighbor node has a path to the sink with its strength greater than or equal to the required strength, its trust to the sink value is previously set to a trust value other than the dummy initial value. Aside with checking the inequality of the neighbor's trust to the sink to the dummy value, the trust rating from the node to the neighbor should also be higher than or equal to the required strength. The last trust value that is computed is the source inferred trust to the sink that is returned as the result.

4 Fuzzy Trust Algorithm

Fuzzy Trust algorithm supports the linguistic terms as trust rating of a node for another in the trust graph. The fuzzy membership functions for the linguistic terms such as low, medium, medium low, medium high and high can be defined as depicted in Fig. 1.

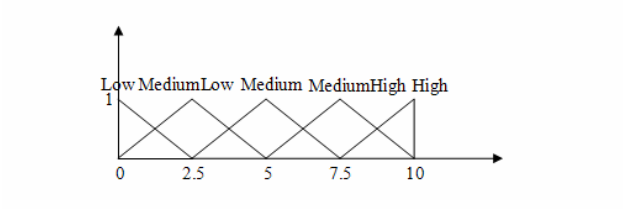

Fig. 1. Fuzzy membership functions of trust linguistic terms

The algorithm tries to compute trust from stronger and shorter paths similar to the TidalTrust algorithm so it performs a breadth-first-like search through the nodes to find shortest paths and also to find the path from source to sink strength fuzzy set. The path from source to sink strength fuzzy set is the maximum trust term fuzzy set that at least a path from the source to the sink with all the trust rating fuzzy sets along it excluding the last larger than or equal to that fuzzy set can be found; this will be called the required strength fuzzy set. It is considered that low, medium, medium low, medium, medium high and high fuzzy sets are in ascending order.

```
The FuzzyTrust algorithm pseudo code
Forward wave: The required strength fuzzy set computation
       Iterate the nodes from the source to the sink similar
       to the breadth first search level by level to find
       shortest paths.
       Set the path strength fuzzy set from source to any
       node in the next level using the previously set path
       strengths fuzzy sets from the source to current level
       nodes.
Backward wave: The trust from nodes to the sink inference
```

```
Iterate the nodes form the sink to the source level
by level.
If the node is adjacent to the sink then its inferred
trust fuzzy set is simply its trust rating fuzzy set
to the sink in the fuzzy trust graph.
Else (If the node is not adjacent to the sink) use
fuzzy inference procedure on the neighbors with trust
ratings to them greater than or equal to the required
strength fuzzy set and that also have a path to the
sink with a strength greater than or equal to the re-
quired strength fuzzy set and hence the trust from
them to the sink is previously inferred strongly
enough.
```

The algorithm is structurally similar to the TidalTrust algorithm excluding the inference. When the nodes are popped and iterated in the reverse order if a node is adjacent to the sink its inferred trust to the sink is simply its trust rating to the sink in the fuzzy trust graph. If the node is not adjacent to the sink, its inferred trust fuzzy set to the sink should be computed from all the neighbors that have a sufficiently strong path to the sink and the trust rating from the node to them is larger than the required strength fuzzy set. Only the neighbors that satisfy these two conditions participate in the trust inference. If a node's neighbor has an enough strong path i.e. a path with the strength fuzzy set of larger than or equal to the required strength fuzzy set, its trust to the sink is set to a value other than the dummy initial value before the trust inference for the node is performed.

For the fuzzy inference a fuzzy set named Acceptable trust fuzzy set is defined according to the required strength fuzzy set with a linear membership function having value 0 at 0 and value 1 at 10. The Acceptable fuzzy set for medium low required strength is shown in Fig. 2.

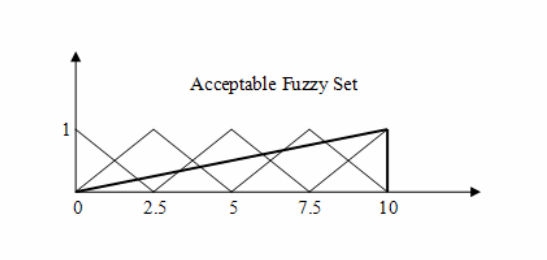

Fig. 2. Acceptable Fuzzy Set

The fuzzy inference is done according to the fuzzy inference rules. If the trust rating fuzzy set from the node to the neighbor is denoted by TrustToNeighbor and the neighbor's trust fuzzy set to the sink is denoted by NeighborTrustToSink, the only fuzzy rule is as follows:

$$if\ \left(TrustToNeighbor\ is\ Acceptable\right)$$
$$then\ \left(TrustToSink\ is\ NeighborTrustToSink\right)$$

Note the similarity of the rule to how people get a friend opinion as their own about an unknown matter if they acceptably trust the friend. The Acceptable fuzzy set is defined so that the firing rate plays the same role as the weight in weighed averaging that is giving an importance to the neighbor trust to the sink proportional to the node trust rating to the neighbor. The inference with different neighbors' data yields different experiences and the fuzzy union of them makes the final result fuzzy set.

What the FuzzyTrust algorithm returns is the inferred trust fuzzy set. To make the inferred trust value comprehendible to the user the result fuzzy set should be approximated to the most similar fuzzy set of the known terms fuzzy sets or the known terms with hedges such as very, more or less, more than and so on fuzzy sets or a disjunction of them and report the corresponding linguistic expression such as "medium or more or less high" to the user. The similarity of two fuzzy sets A and B is defined as:

$$S = \frac{\left|A \bigcap B\right|}{\left|A \bigcup B\right|} \tag{4}$$

Where || is the cardinality of a fuzzy set that is defined as:

$$\left|A\right| = \int_{x \in SupportSet} \mu_A(x) \tag{5}$$

A simple procedure for yielding approximating linguistic expressions is to prepare all the possible combination of the terms and hedges and find the most similar one by an exhaustive similarity computation for them and the inferred fuzzy set although more intelligent algorithms may be possible. The whole fuzzy inference procedure can be summarized as in Fig. 3.

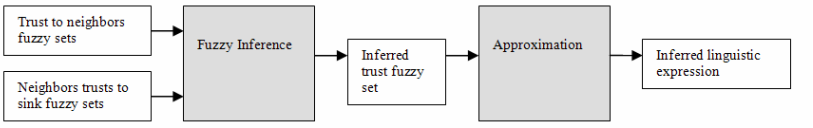

Fig. 3. The fuzzy trust inference

As inference is done along the paths with at least the required strength, the minimum trust rating to any neighbor node participating in the trust inference is the required strength so the inference is trusted at least by the required strength value. Therefore the required strength computed along the inference procedure is reported as the inference preciseness.

The basic algorithm is modified to benefit from longer paths. The AllLengthFuzzyTrust algorithm 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. Allowing longer paths to participate in the trust inference makes information from other nodes available aside from its own trust rating to the sink for trust inference from a source that the sink is its direct neighbor. This corresponds to the situation when someone knows another but also gets help about his trusted friends about him. So the pseudo code for AllLengthFuzzyTrust algorithm is:

```
The AllLengthFuzzyTrust algorithm pseudo code
Forward wave: The required strength fuzzy set computation
      Iterate the nodes from the source to the sink similar
      to the breadth first search level by level to find
      all the paths.
      Set the path strength fuzzy set from source to any
      node in the next level using the previously set path
      strengths fuzzy sets from the source to current level
      nodes.
Backward wave: The trust from nodes to the sink inference
      Iterate the nodes form the sink to the source level
      by level.
      If the node is not adjacent to the sink use fuzzy in-
      ference procedure on the neighbors with trust ratings
      to them greater than or equal to the required
      strength fuzzy set and that also have a path to the
      sink with a strength greater than or equal to the re-
      quired strength fuzzy set and hence the trust from
      them to the sink is previously inferred strongly
      enough.
      Else (If the node is adjacent to the sink) then its
      inferred trust fuzzy set is the result of fuzzy in-
      ference on its direct trust rating fuzzy set to the
      sink in the fuzzy trust graph with a high trust to it
      among information coming from the qualified neighbors
      with conditions of the same as in the if part.
```

This algorithm leads to more precise inference when deeper paths conduct more trusted information.

5 Simulation and Results

The graph depicted in Fig. 4 is used as the graph for TidalTrust Algorithm simulation. All the trust values are 0, 2.5, 5, 7.5 or 10 although they could be any number in the [0, 10] range. These values are selected deliberately because all of them are only and completely belonging to one of the terms fuzzy sets and so the corresponding fuzzy trust graph of the trust graph in Fig. 4 is the fuzzy trust graph in Fig. 5. This correspondence helps in comparing the two algorithms. The trust in-

ference is reported for every node from A to L to any other one form A to L in the simulations for brevity.

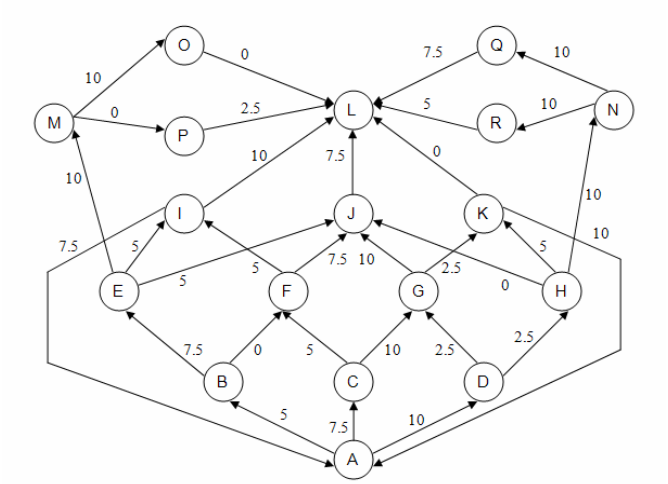

Fig. 4. Trust Graph for TidalTrust Algorithm

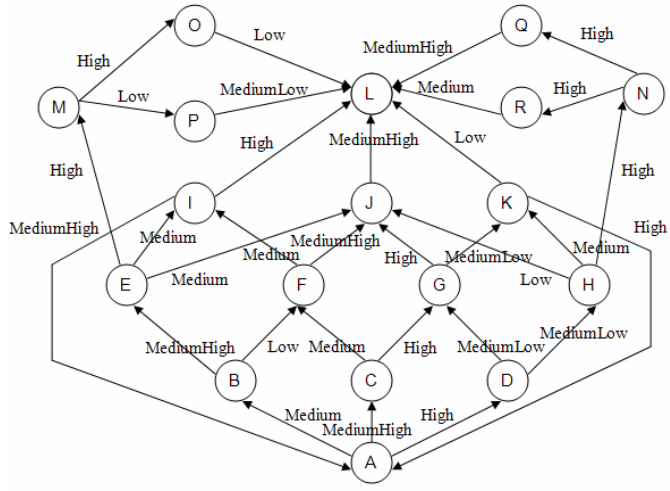

Fig. 5. Trust Graph for FuzzyTrust Algorithm

The result values of TidalTrust algorithm is shown in Table 1. The linguistic expressions of approximating fuzzy sets of FuzzyTrust algorithm result fuzzy sets by Larsen implication method is shown in Table 2. The inferred fuzzy sets are approximated to linguistic expressions that could contain one “or” connective on trust terms that could have very or somewhat hedges. The term abbreviations are

written for the sake of briefness. The shown fuzzy expressions are the expressions the fuzzy set of which are most similar to the inferred fuzzy sets. Table 3 shows the inference preciseness for each node pair trust inference. Membership values of the TidalTrust algorithm result values in the approximating fuzzy sets with the Larsen implication method is shown in Table 4.

Table 1. TidalTrust results

	A	B	C	D	E	F	G	H	I	J	K	L
A	----	5	7.5	10	7.5	5	2.5	2.5	5	10	2.5	7.5
B	7.5	----	7.5	10	7.5	0	5.7	2.5	5	5	2.5	8.8
C	7.5	5	----	10	7.5	5	10	2.5	5	10	2.5	7.5
D	10	5	7.5	----	7.5	3	2.5	2.5	5	5	3.8	3
E	7.5	5	7.5	10	----	3	5.7	2.5	5	5	2.5	8.8
F	7.5	5	7.5	10	7.5	----	5.7	2.5	5	7.5	2.5	7.5
G	10	5	7.5	10	7.5	3	----	2.5	5	10	2.5	7.5
H	10	5	7.5	10	7.5	3	5.7	----	5	0	5	0
I	7.5	5	7.5	10	7.5	5	5.7	2.5	----	10	2.5	10
J	----	----	----	----	----	----	----	----	----	----	----	7.5
K	10	5	7.5	10	7.5	5	2.5	2.5	5	10	----	0
L	----	----	----	----	----	----	----	----	----	----	----	----

Table 2. FuzzyTrust results (Linguistic expressions of the most similar fuzzy sets approximating the inferred fuzzy sets)

	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	very L or M	ML	ML	M	L or H	ML or M	L or MH
E	MH	M	MH	H	----	very 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	very L or M	----	ML	M	H	ML	MH
H	H	M	MH	H	MH	very 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. Inference preciseness

	A	B	C	D	E	F	G	H	I	J	K	L
A	----	H	H	H	M	MH	H	H	M	MH	MH	MH
B	M	----	M	M	H	H	M	M	MH	MH	M	M
C	M	M	----	M	M	H	H	M	M	H	H	H
D	ML	ML	ML	----	ML	ML	H	H	ML	ML	ML	ML
E	M	M	M	M	----	M	M	M	H	H	M	M
F	M	M	M	M	M	----	M	M	H	H	M	MH
G	ML	ML	ML	ML	ML	ML	----	ML	ML	H	H	H
H	M	M	M	M	M	M	M	----	M	H	H	M
I	H	MH	MH	MH	M	MH	MH	MH	----	MH	MH	H
J	----	----	----	----	----	----	----	----	----	----	----	H
K	H	H	H	H	M	MH	H	H	M	MH	----	H
L	----	----	----	----	----	----	----	----	----	----	----	----

Table 4. TidalTrust result values membership in FuzzyTrust result fuzzy sets

	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	1	1	1	0	0.5	0
E	1	1	1	1	----	0	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	----	1	1	1	1	1
H	1	1	1	1	1	0	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												

Table 5. Absolute difference between TidalTrust values and defuzzified values of FuzzyTrust inferred fuzzy sets

	A	B	C	D	E	F	G	H	I	J	K	L
A	----	0	0	1	0	0	0	0	0	0.8	0	0
B	0	----	0	1	0	1	1.3	0	0	0	0	0.8
C	0	0	----	1	0	0	0.8	0	0	0.8	0	0
D	0.8	0	0	----	0	1	0	0	0	0	0	2.3
E	0	0	0	1	----	1	1.3	0	0	0	0	0.8
F	0	0	0	1	0	----	1.3	0	0	0	0	0
G	0.8	0	0	1	0	1	----	0	0	0.8	0	0
H	0.8	0	0	1	0	1	1.3	----	0	0.8	0	0.8
I	0	0	0	1	0	0	1.3	0	----	0.8	0	0.8
J	----	----	----	----	----	----	----	----	----	----	----	0
K	0.8	0	0	1	0	0	0	0	0	0.8	----	0.8
L	----	----	----	----	----	----	----	----	----	----	----	----
Average = 0.3058253058253062												

Table 6. AllLengthFuzzyTrust algorithm

	FuzzyTrust	FuzzyTrust preciseness	AllLengthFuzzyTrust	AllLengthFuzzyTrust 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

The average near one shows that the two algorithms agree generally in the results. The two algorithm results differ more when contradictory information

should be composed for trust inference. One of the membership values in Table 4 that is 0 is the membership value from D to J meaning that the two algorithms results 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 to the same amount so they take part the same in D to J trust value inference. The neighbor nodes G and H have contrary trust recommendations for node J that is trust of one of them to J is high (10) while the other one's is low (0). The TidalTrust algorithm simply takes the average of 0 and 10, yielding the 5 value that is a value just in between (medium), none of the neighbors believe in it in fact. The FuzzyTrust algorithm with Larsen method result linguistic expression is "Low or high". While FuzzyTrust result is exactly the information that exists in the trust graph, TidalTrust algorithm fails to exactly convey the information existing in the trust graph due to its improper composition scheme that is averaging. A crisp number is incapable of conveying enough information in fact. Similar arguments can be brought for other node pairs that two algorithm results do not completely agree. That averaging as the composition strategy infers trust wrongly when the neighbors give contradictory information is mainly why these differences occur.

The Absolute difference between TidalTrust results and defuzzified FuzzyTrust inferred fuzzy sets are shown in Table 5. Note that the difference for D to J is zero meaning that the result of TidalTrust algorithm can be gotten from the FuzzyTrust algorithm result. The average of 0.3 in the [0, 10] range of trust indicates very little difference. This little average difference suggests that the TidalTrust results are contained in or can be computed from FuzzyTrust results by defuzzification.

As TidalTrust crisp trust values can also be approximated to the trust term fuzzy set with the maximum membership function value, the two heterogeneous trust inference systems that one of them computes trust with TidalTrust algorithm and the other one with FuzzyTrust algorithm can also cooperate successfully and benefit from each other's trust graph information and inference.

The AllLengthFuzzyTrust algorithm is compared to the FuzzyTrust algorithm in Table 6. The FuzzyTrust algorithm results and their preciseness are written in column two and three and the AllLengthFuzzyTrust algorithm results and preciseness are in fourth and fifth respectively where star depicts a result difference between the algorithms. The three stars in the fifth column correspond to the node pairs (B to L, E to L and H to L) that more strong paths than the shortest ones exist for them that are well utilized for a more precise inference by the AllLengthFuzzyTrust algorithm.

6 Conclusion and Future Work

The inference simulations show that the FuzzyTrust algorithm results generally agree with TidalTrust results but FuzzyTrust algorithm reports richer expressions that match more with the information existing in the trust graph than TidalTrust algorithm especially when contradictory information should be composed for the trust inference. Also the fuzzy linguistic expressions are much more desirable for

the user to write and read about trust values. The inference preciseness values that are also reported for each inference are improved by AllLengthFuzzyTrust algorithm that searches not only the shortest paths but also any other to perform the inference from the most trusted existing paths.

The algorithms make use of strongest paths while other path may also have useful information. Less strong paths are pruned in the algorithms while they have even little information. The future work of this research involves benefiting from weaker paths in the inference (AllLengthAllStrengthFuzzyTrust algorithm).

Besides the advantages it should be mentioned that fuzzy inference has a more computational load that averaging. The simulations show that FuzzyTrust infers trust well theoretically but a study of real world network of users will show clearly if the fuzzy calculated trust values superior in with real users.

Although the work targeted the social networks, ideas are proposed abstractly enough in trust graph modeling. The abstract node can be a social network user, a software mobile agent, a robot of a cooperating team or a semantic web URI. TidalTrust and FuzzyTrust that are structurally similar need global trust information in the trust graph format. This need can be satisfied in a society of social networks size but not easily for network of the web size. The most important issue to address before deploying the algorithm in semantic web trust layer may be distributiveness. The two algorithms can be distributively employed if the nodes are computational entities and each compute path strength form the source to it, convey it to its neighbors and infer its own trust to the sink but a lot of synchronization that is needed for the algorithms level by level computations should be done that seems highly impractical in the web. Although investigating different paths is valuable for the sake of more trusted information and better inference but scalability may be more important. A model and algorithm should be well distributed to become the trust modeling and inference algorithm for the semantic web trust layer. The path searching can be eliminated towards localization and each computational node can infer its trust from all the information coming from all the neighbors according to its trust to them using the same fuzzy inference rule or other fuzzy information composition procedures that are under study.

References

1. Golbeck J (2005) Computing and Applying Trust in Web-based Social Networks. Doctor of Philosophy Dissertation, University of Maryland, College Park, 200 pages
2. Golbeck J (November 2005) Semantic Web Interaction through Trust Network Recommender Systems. In: End User Semantic Web Interaction Workshop at the 4th International Semantic Web Conference
3. Golbeck J, Hendler J (January 2005) Inferring Trust Relationships in Web-based Social Networks. ACM Transactions on Internet Technology (in press)
4. Golbeck J and Hendler J (July 2004) Reputation network analysis for email filtering. In: Proceedings of the First Conference on Email and Anti-Spam, Mountain View, California.

5. Golbeck J, Hendler J (2004) Accuracy of Metrics for Inferring Trust and Reputation in Semantic Web-based Social Networks. In: Proceedings of EKAW 04.
6. Golbeck J, Parsia B, Hendler J (2003) Trust Networks on the Semantic Web. In: Proceedings of Cooperative Intelligent Agents 2003, Helsinki, Finland
7. Ziegler C, Golbeck J (2005) Investigating Correlations of Trust and Interest Similarity - Do Birds of a Feather Really Flock Together? Submitted to the Journal of Artificial Intelligence Research

Appendix

1. BinaryTrust Algorithm

```
int inferBinaryTrust(TrustGraph graph, int source, int sink)
{
  if (graph.isAdjacent(source, sink)
    return graph.getTrust(source, sink);
  else
  {
    int[] adjacents = graph.getAdjacents(source);
    int trustedCount = 0;
    double sum = 0;
    for (int i = 0; i < adjacents.length; i++)
    if (graph.getTrust(source, i) == 1)
    {
      Sum += inferTrust(i, sink);
      trustedCount++;
    }
    return (int) (sum / trustedCount);
  }
}
```

2. TidalTrust Algorithm

```
double inferTidalTrust(int source, int sink)
{
if (source == sink)
  return null;

boolean[] visiteds = new
  boolean[graph.getNodeCount()];
double[] trustToSink = new
  double[graph.getNodeCount()];
double[] pathFromSourceStrength = new
  double[graph.getNodeCount()];

for (int i = 0; i < visiteds.length; i++)
{
  visiteds[i] = false;
  trustToSink[i] = -1;
  pathFromSourceStrength[i] = -1;
}

Queue searchQueue = new Queue();
searchQueue.queue(source);
Queue nextLevelSearchQueue = new Queue();
Stack tideStack = new Stack();
```

```
int maxDepth = Integer.MAX_VALUE;
int depth = 1;
pathFromSourceStength[source] = 10;

while (!searchQueue.isEmpty() && depth <= maxDepth)
{
  int node = searchQueue.dequeue();
  tideStack.push(node);
  if (!graph.isAdjacent(node, sink))
  {
    int[] adjacents = graph.getAdjacents(node);
    for (int i = 0; i < adjacents.length; i++)
    {
      int adjacentNode = adjacents[i];
      if (visiteds[adjacentNode] != true)
      {
        visiteds[adjacentNode] = true;
        nextLevelSearchQueue.queue(adjacentNode);
      }
      if (nextLevelSearchQueue.contains(adjacentNode))
      {
        double pathStrength =
          min(pathFromSourceStrength[node],
            graph.getTrust(node, adjacentNode));

        pathFromSourceStrength[adjacentNode]=
          max(pathFromSourceStrength[adjacentNode], pathStrength);
      }
    }
  }
  else
  {
    maxDepth = depth;
    double pathStrength =
      pathFromSourceStrength[node];

    pathFromSourceStrength[sink] =
      max(pathFromSourceStrength[sink], pathStrength);
  }

  if (searchQueue.isEmpty())
  {
    searchQueue = nextLevelSearchQueue;
    depth = depth + 1;
    nextLevelSearchQueue = new Queue();
  }
}


if (maxDepth == Integer.MAX_VALUE)
  return -1;
//If its value has not changed since the beginning then no path is found to
//    the sink.

double pathFromSourceToSinkStrength =
  pathFromSourceStrength[sink];

while (!tideStack.isEmpty())
{
  int node = tideStack.pop();
  if (graph.isAdjacent(node, sink))
    trustToSink[node] = graph.getTrust(node, sink);
  else
  {
    int[] adjacents = graph.getAdjacents(node);
    double numerator = 0;
    double denumerator = 0;
    for (int i = 0; i < adjacents.length; i++)
    {
```

```
        int adjacentNode = adjacents[i];
        if ((graph.getTrust(node, adjacentNode) >= pathFromSourceToSinkStrength)
         && (trustToSink[adjacentNode] != -1))
        {
          numerator +=
          graph.getTrust(node, adjacentNode) * trustToSink[adjacentNode];

          denumerator += graph.getTrust(node, adjacentNode);
        }
      }

      if (denumerator > 0)
        trustToSink[node] = numerator / denumerator;
    }
  }

  return trustToSink[source];
  }
```

3. FuzzyTrust Algorithm

```
FuzzySet inferFuzzyTrust (int source, int sink)
{
if (source == sink)
  return null;

boolean[] visiteds = new
  boolean[graph.getNodeCount()];
double[] trustToSink = new
  double[graph.getNodeCount()];
FuzzySet[] pathStengthFuzzySet = new
  double[graph.getNodeCount()];

for (int i = 0; i < visiteds.length; i++)
{
  visiteds[i] = false;
  trustFuzzySetToSink[i] = null;
  pathStrengthFuzzySet[i] = null;
}

Queue searchQueue = new Queue();
searchQueue.queue(source);
Queue nextLevelSearchQueue = new Queue();
Stack tideStack = new Stack();
int maxDepth = Integer.MAX_VALUE;
int depth = 1;
pathStengthFuzzySet[source] = FuzzyTrustGraph.HIGH;

while (!searchQueue.isEmpty() && depth <= maxDepth)
{
  int node = searchQueue.dequque();
  tideStack.push(node);

  if (!graph.isAdjacent(node, sink))
  {
    int[] adjacents = graph.getAdjacents(node);
    for (int i = 0; i < adjacents.length; i++)
    {
      int adjacentNode = adjacents[i];
      if (visiteds[adjacentNode] != true)
      {
        visiteds[adjacentNode] = true;
        nextLevelSearchQueue.queue(adjacentNode);
      }

      if (nextLevelSearchQueue.contains(adjacentNode))
      {
        FuzzySet thisPathStrengthFuzzySet =
```

```
            min(pathStengthFuzzySet[node],
              graph.getTrustFuzzySet(node, adjacentNode));

          pathStengthFuzzySet[adjacentNode] =
            max(pathStengthFuzzySet[adjacentNode], thisPathStrengthFuzzySet);
        }
      }
    }
    else
    {
      maxDepth = depth;
      FuzzySet thisPathStrengthFuzzySet = pathStengthFuzzySet[node];

      pathFromSourceStrengthFuzzySet[sink] =
        max(pathStengthFuzzySet[sink],
          thisPathStrengthFuzzySet);

    }
    if (searchQueue.isEmpty())
    {
      searchQueue = nextLevelSearchQueue;
      depth = depth + 1;
      nextLevelSearchQueue = new Queue();
    }
}

if (maxDepth == Integer.MAX_VALUE)
  return null;
//If its value has not changed since the beginning then no path is found to
//    the sink.

FuzzySet pathToSinkStrengthFuzzySet =
  pathStengthFuzzySet[sink];

while (!tideStack.isEmpty())
{
  int node = tideStack.pop();
  if (graph.isAdjacent(node, sink))
  {
       trustFuzzySetToSink[node] = graph.getTrustFuzzySet(node, sink);
  }
  else
  {
    int[] adjacents = graph.getAdjacents(node);
    Set trustToAdjacentsFuzzySets =
      new Set();
    Set trustFromAdjacentsFuzzySets =
      new Set();

    int validAdjacentCount = 0;
    for (int i = 0; i < adjacents.length; i++)
    {
      int adjacentNode = adjacents[i];
      if ((trustFuzzySetToSink[adjacentNode] != null) &&
      (graph.getTrustFuzzySet(node, adjacentNode) >= pathToSinkStrengthFuz-
    zySet)
      )
      {
        trustToAdjacentsFuzzySets.add(
          graph.getTrustFuzzySet(node,adjacentNode));

        trustFromAdjacentsFuzzySets.add(
          trustFuzzySetToSink[adjacentNode]);

        validAdjacentCount++;
      }
    }
    if (validAdjacentCount == 0)
      ; //trustFuzzySetToSink[node] = null;
```

```
    else
      trustFuzzySetToSink[node] =
        fuzzyInfer(
          trustToAdjacentsFuzzySets,
          trustFromAdjacentsFuzzySets);
  }
}

return trustFuzzySetToSink[source];
}
```