IWTrust: Improving User Trust in Answers from the Web

Ilya Zaihrayeu[1] Paulo Pinheiro da Silva[2] Deborah L. McGuinness[2]

[1]University of Trento, Povo, Trento, Italy
[2]Stanford University, Stanford, USA

Abstract. Question answering systems users may find answers without any supporting information insufficient for determining trust levels. Once those question answering systems begin to rely on source information that varies greatly in quality and depth, such as is typical in web settings, users may trust answers even less. We address this problem by augmenting answers with optional information about the sources that were used in the answer generation process. In addition, we introduce a trust infrastructure, IWTrust, which enables computations of trust values for answers from the Web. Users of IWTrust have access to sources used in answer computation along with trust values for those source, thus they are better able to judge answer trustworthiness. Our work builds upon existing Inference Web components for representing and maintaining proofs and proof related information justifying answers. It includes a new TrustNet component for managing trust relations and for computing trust values. This paper also introduces the Inference Web answer trust computation algorithm and presents an example of its use for ranking answers and justifications by trust.

1 Introduction

There is an increasing amount of information sources available to web applications. As information source breadth increases, so does the diversity in quality. Users may find increasing challenges in evaluating the quality of web answers, particularly in settings where answers are provided without any kind of justification. One way our work improves a user's ability to judge answers is by including *knowledge provenance information* along with answers. Knowledge provenance includes information about the origin of knowledge and a description of the reasoning processes used to produce the answers [20]. This paper describes our new work expanding on our work on knowledge provenance, which allows justifications to include trust values with answers. The computation process takes into account the user's (stated or inferred) degree of belief in the sources, answering engines, and in other users who provide sources and/or answering engines. Our framework allows users to define and (locally) maintain individual trust values, and use those values in their evaluation of answers from their own trust "viewpoint". They may also access the trust network to discover other user's trust values of answers and thus also use those values in their evaluation of answers.

In recent work, we have addressed the problem of improving user's trust in answers by providing information about how an answer was calculated [17]. That work provides an infrastructure, called Inference Web (IW), which allows proofs and proof fragments to be stored in a portable, distributed way on the web by using the Proof Markup Language (PML) [19]. Proofs describe information manipulation processes (i.e., information extraction, reasoning, etc.) used to answer questions providing full support for tracking knowledge provenance related to answers. In addition to PML, IW provides IWBase [16], which is a distributed repository of *knowledge provenance elements*. Knowledge provenance elements contain proof-related meta-information such as information about *answering engine*s, *inference rule*s, *representation language*s, and *source*s such as ontologies or documents. PML documents can represent answers varying from a single database lookup operation to a derivation of complex answers in a distributed fashion, involving multiple sources, and using distinct answering engines.

Knowledge provenance alone may not be enough for a user to evaluate how much they should trust an answer. For example, a user may be unfamiliar with a question answering component that was used to find (a part of) the answer. A user may not know much about the question answering system (e.g. reasoning method, correctness and completeness of the reasoner, reasoning assumptions, etc.). Alternatively, a user may know something about the reasoner and would normally expect to trust answers from the reasoner, but in the case where the answer appears to be incomplete or conflict with a user's expectations, a user may need more information before he or she would trust the answer. Also, users may trust a question answering system completely but if the system relies on information from sources that the user does not trust, then the user may not trust the answer. Additional considerations include situations where a source is used that is unknown to the user but the source is known to be trusted by another user (human or agent) that is trusted by the user.

In this paper we introduce an extension of IW, called *IWTrust*, which can quantify users' degree of trust in answers obtained from web applications and services. IWTrust uses trust values between users as well as trust values between users and provenance elements. Thus, IWTrust can store that Louise trusts her friend Deborah's knowledge of wine and food parings, and also trusts the US Department of Agriculture's food pyramid, and her favorite cook book's descriptions of recipes. Trust values of the answer can then be computed relative to a particular user's perspective. The final trust value for the answer uses both a user's trust value of the sources used in the answer as well as a user's trust in other user's trust of sources used to obtain the answer. For example, Louise may ask an agent to answer questions about food and wine pairings and she may trust the answer more if it uses the sources with which she is familiar. She may also trust the system if it uses sources known to be trusted by agents or people she trusts. IWTrust also allows one to compute collective trust measures of (a group of) users in some provenance element. These are expected to be used by users for defining their (starting) trust values for provenance elements. Many light users of IWTrust will never actually input many (or any) of their own per-

sonalized trust values, and instead rely on the aggregated trust from groups. In the example setting, Louise may decide to trust the wine agent if it used sources that a very large number of users trusted to a high degree.

The paper is organized as follows. Section 2 gives an overview of existing trust management systems discussing trust issues important to the IWTrust framework. Section 3 provides an abstract view of how trust components interact with question answering components. Section 4 provides the details of our trust model for IW. Section 5 describes three algorithms for computing trust values using the IWTrust framework. Section 6 provides and example use of IWTrust and the final section summarizes our work.

2 Related Work on Trust Management

Credibility and *resource authentication* (i.e., digital signatures, public keys) are two major issues in trust management literature. Our work focuses on the *credibility* issue, which has been studied in the context of a few areas including social networks (e.g., see [11]), general Semantic Web [15, 11], and Peer-to-Peer computing [13, 1].

Generally, users do not "know" all other users in a trust network, and at most, they will only define trust values w.r.t. a (small) subset of them. The trust relationship is directional, i.e. the fact that user u_1 trusts user u_2 at some level t does not necessarily mean that u_2 trusts u_1 at the same level. In reality, u_2 may trust u_1 at a different level (including explicit distrust) or u_2 may be unaware of u_1 and thus have unknown trust. Trust relationships between users can be represented as a directed graph with weighted edges, where an edge from node u_1 to node u_2 with weight t means that user u_1 trusts user u_2 at the level of t. This is a restricted form of a *trust graph* where *graph nodes* (also called *resources*) are users. In this paper we are interested in building and using trust graphs where we can observe the following:

- the use of different kinds of resources including *users* (humans or agents querying the web), *sources*, *engines* (answering queries), *statements* told by one or more sources (also called *told information*), statements derived by one or more engines.
- the paths from a user who asks a question to the answer. This path may include sources and trust values (possibly by others) of those sources.

Our work uses trust graphs, and as a result infrastructure for creating and maintaining trust graphs is discussed in this paper. Trust graphs use relations between resources and work on these relations exists in the literature. For example, Despotovic and Aberer [5] distinguish trust among users as *providers* of trustworthy information and/or services from trust among users as *recommenders* of (other) users providing trustworthy information and/or services; and from trust among users as recommenders of (other) recommenders. Further, Matthew et al. [15] introduce the notion of user's belief in the accuracy, credibility and relevance of a *statement*. Gil and Ratnakar [9] operate with the notions of *sources*

and *statements* (acquired from the sources), and associates them with one of the six values from their appropriate scales.

Going back to traditional trust management systems, trust values between users are used for the following:

1. trust-based routing of (search) requests in the network [15]
2. computing trust values between nodes that are not connected by a single edge but are connected through a path of nodes in a trust graph [15, 10]
3. computing a "collective" trust value for a user as an aggregated measure of trust values of a group of users [13, 1, 14]
4. detecting malicious users, who intentionally spread unauthentic information, or inactive peers, who consume but do not contribute, and encouraging them to contribute (e.g. [13, 14])

We know that some users may not provide authentic information about provenance elements, they may intentionally mis-report their credibility level to other users and/or provenance elements, and they may not provide provenance element information that is useful to IW in answering questions. In this paper we address issues (1) – (3) above and we do not address issue (4).

Trust values may be different and may be interpreted differently by alternative approaches. For instance, [10] and [1] define a binary scale for trust values, namely, 0 (or, -1 in [10]) denotes an untrustworthy relation, and 1 denotes a trustworthy relation. Golbeck et al. [11] defines nine levels of trust, defining a specific meaning for each level (from most trusted to least trusted). Trust values may be defined in a continuous range [0,1] with probabilistic interpretation [15, 13, 5]. Intuitively, a trust value t, for a user, means that the user has the probability t of providing trustworthy information and/or services.

Computing estimated trust values for users, or, generally, for resources , is one of the most essential issues addressed in the reputation systems literature. There are two basic approaches: computing a *collective* trust value by aggregating trust values for some particular user from all other users (e.g. [13]), or from a subset (quorum) of other users [14, 1]; and computing an *inferred* (or *personalized*) trust value for a user with respect to another user based on the trust values on the path(s) in a trust graph from one user to another [15, 10]. Initial trust values for potentially unknown users need to be defined. In this case, it is assumed that there is no path in the trust graph between the users, and therefore an estimated trust value can not be computed. The adoption of a collective trust value is an approach for initial trust values (e.g., [14]).

3 Trust for Question Answering

In this section, we will describe a prototypical question answering environment and discuss possible approaches for adding trust-based functionalities.

3.1 Question Answering Environments

In the Inference Web context, question *answering engine* is any kind of software system providing services able to produce answers in response to queries. From this general definition, answering engines can vary from retrieval-based services, such as database management systems and search engines, to reasoning-based services such as automated theorem provers.

We assume an environment where a user (a software system or a human being) interacts with a *query front-end* component[1] formulating and asking a query q to which the query front-end responds with a set of answers A[2]. In addition to interacting with users, the query front-end is responsible for forwarding the query q to the answering engine grouping answers from the answering engine into a set of answers A, and forwarding A to the user. Optionally, the user may provide the specification of a set S of information sources along with the query to be used by the answering engine to retrieve information from sources. Also optionally, the user may specify the answering engine(s), which is not further discussed in this paper. On demand, answering engines may also provide $N(A)$, a set of justifications for answers in A. If justifications are provided with answers, $N(A)$ should have one or more justification for each answer a_k in A.

Query languages for q vary accordingly to the kinds of answering engines used in the environment. For example, q may be a SQL query if the answering engine is a relational database management system or it can be an OWL-QL [7] query if the answering engine is a hybrid automated reasoner such as JTP [8]. The set of justifications $N(A)$ is represented in PML. The features of PML that we use, and in fact that we claim are essential for trust applications include:

- Full support for tracking provenance information. Thus, trust relations can be established between (told) information and their sources.
- Options for representing information manipulation traces at different digress of formality and specificity. PML has been used to represent formal proofs such as those defined in traditional proof-theory textbooks while simultaneously being used to represent extraction and text analytics processes [6] exemplifying quite different levels of formality and specificity.
- Options for representing multiple justifications for an answer within a single data-structure. Thus, alternative justifications, and consequently alternative trust values, can be offered to users as they evaluate answers and their support.

3.2 A Trust Component for Question Answering

In general, *trust components* are responsible for computing trust values for resources such as users, i.e., the degree of trust a given user gives to other users.

[1] User interfaces of answering engines interacting directly with users can be considered to be query front-ends for the engines.

[2] In this paper, capital letters denote sets while corresponding lowercase letters denote set members, i.e., a_k is one answer in the set of answers A.

In a question answering environment, in addition to computing trust values for users, we believe *trust components* should be able to compute trust values for trust graph resources such as sources, told information, and derived information. We will use a trust component for computing trust values for answers.

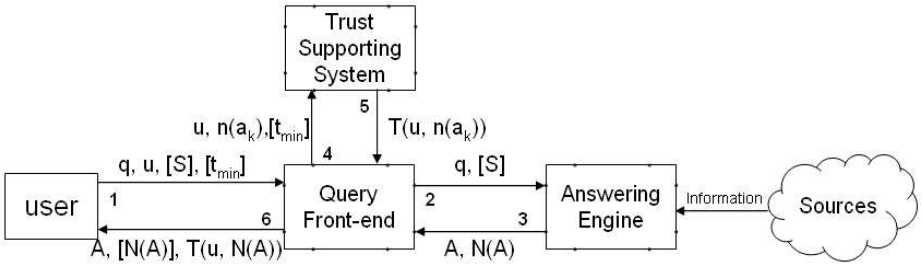

Fig. 1. Trust component in question answering systems

The computation of trust values for answers presented in Figure 1 provides a scenario where the trust component tries to assign trust values to every (intermediate) conclusion c_m during the question answering process. Trust is user-specific information returned by the query front-end to users. Different users may trust other users and sources differently. Thus, an user identification u_i is required along with the query q if the environment is expected to manipulate trust information. The algorithm for computing the trust value for a given u_i and a justification of a conclusion, $T(u_i, n(c_m))$, is described in Section 5. Trust values for answers are computed on demand in the same way that justifications are computed on demand. Thus, if positive answer trust values are required, the answering engine should return answer justifications (step 3 in Figure 1), even if justifications are not asked by the user. Indeed, justifications are required for computing trust values for answers. From an answer and its justifications, the trust component computes (using the underlying trust network) user's trust values $T(u_i, n(a_k))$ for answer a_k , which are returned to the query front-end (step 5). The query front-end consolidates available trust values for justifications of answers in A into a single set $T(u_i, N(A))$[3]. Finally, the query front-end returns $T(u_i, N(A))$ to the user (step 6). The handling of multiple trust values for a single answer may be confusing for users. So, if a tool using IWTrust trust values for answer decides to have one trust value for an answer then the tool can select the highest trust value in the range to be the trust value for the answer.

4 IWTrust Framework

In this section we introduce the IWTrust framework, referred to in the rest of the paper as IWTrust. Figure 2 shows IWTrust in action where a user u_1, submits a query q and obtains a set of answers $\{a_1, \ldots, a_n\}$ with their associated

[3] $T(u_i, N(A))$ is defined as $\{X \mid \forall a_k \in A, T(u_i, n(a_k)) \in X\}$.

trust values $\{t_{11}, t_{12} \ldots, t_{n1}, t_{n2}, \ldots\}$. The user is connected by trust relations to provenance elements (sources and answering engines in the diagram) in the IWBase that are used in answering the query. The user, u_1, is directly connected to some of them such as e_2. The user is connected to others such as s_2, s_3 through other users (u_4 in the TrustNet in this diagram). Provenance elements are connected to told assertions in proofs by provenance relations. Finally, Figure 2 identifies proof fragments, queries, IWBase and TrustNet as IW components supporting the trust graph as discussed in this section.

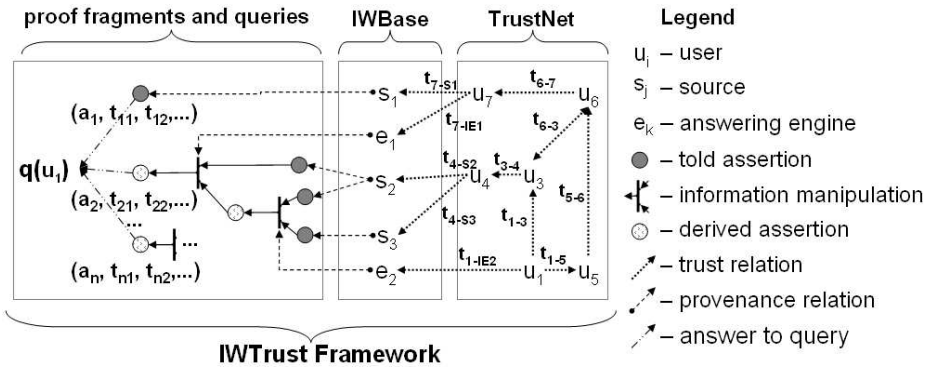

Fig. 2. IWTrust Framework

4.1 Proof Fragments and Queries

PML is used to build OWL documents representing proof fragments and queries. PML *NodeSet* is the primary building block of proofs and is used to represent information manipulation from queries to answers and from answers to sources. PML *Query* is used to represent user queries. A PML query identifies the node sets containing conclusions used in answering the query. Nevertheless, PML is an ontology written in W3C's OWL Semantic Web representation language [18] allowing justifications to be exchanged between Semantic Web services and clients using XML/RDF/OWL.

A node set $n(c)$ represents a step in a proof whose conclusion c is justified by a set of inference steps associated with the node set. PML adopts the term "node set" since each instance of NodeSet can be viewed as a set of nodes gathered from one or more proof trees having the same conclusion. The conclusion c represents the expression concluded by the proof step. Every node set has one conclusion which is the element in the trust network that requires a trust value. Of particular interest for queries is when the conclusion of a node set is a query answer (rather than an intermediate conclusion)

Each inference step of a node set represents an application of an inference rule that justifies the node set's conclusion. A node set can have any number of inference steps, including none. The inference step's antecedents, answering engines, and sources are all important to the framework. The antecedents of an inference step form a sequence of node sets each of whose conclusion is a

premise of the application of the inference step's rule. The sequence can contain any number of node sets including none. Each source of an inference step refers to an entity representing original statements from which the conclusion was obtained. An inference step can have any number of sources including none. An inference step's source supports the justification of the node set conclusion when the step's rule is a *DirectAssertion*.

4.2 Provenance Elements and the IWBase

Provenance is captured for objects that may be manipulated and shown in proofs. In this paper, we limit our discussion of provenance elements that are important for our trust work. A more detailed description of PML provenance elements is available in [19] and they are formally specified in OWL[4]. A *Source* represents an entity which is the source of the original data. A source can be either an *Organization*, an *AnsweringEngine*, a *Team*, a *Person* or a *CreatedSource*. *CreatedSources* can be *Ontologies*, *Websites* or, generally, *Documents*. Provenance relations among sources are important during trust evaluation. For instance, a person may belong to zero or more teams and/or organizations. This source is important for trust since organization structure may provide information that is useful in trust evaluation (e.g., "I trust that person because I trust the organization she belongs to"). Moreover, a user may trust a document more if she trusts the author(s), publisher(s) and/or submitter(s) of the document. Created sources serve as providers of statements, which are then used in the the computation of answers to user queries.

4.3 TrustNet

TrustNet is a trust network. In this network, users may define trust values w.r.t. other users, answering engines, and sources. In addition to these trust relations, the TrustNet trust graph represents provenance relations between sources. An edge in the graph may connect an authority source node to a created source node, provided that the authority source is the author (publisher or submitter) of the created source. Using the information in these edges, we can compute trust for a created source based on the trust of the authority source that created it. All edges in the graph are associated with two values: *length* and *trust value*. Intuitively, the length of an edge represents the trust "distance" between the origin (i.e., users or authority sources) and destination nodes.

Trust values are defined in the range [0,1], and are given a probabilistic interpretation. Namely, a trust value means the probability of either:

1. A source contains relevant and correct information
2. An answering engine correctly applies rules to derive statements (as conclusions of node sets)
3. A user provides a a reference to a source that meets the requirements from 1, and/or to an answering engine that meets the requirements from 2

[4] http://iw.stanford.edu/2004/07/iw.owl

4. A user recommends other user(s) who can provide trustworthy references to source(s) and/or to answering engine(s)

Edges connecting sources have length values equal to 0 and trust values equal to 1. These values represent the connection between created sources and their associated sources. All other edges have length 1 and may have an arbitrary trust value, which is computed as follows: each statement, used in query answering, and originated from some source, may be evaluated by the user either as correct or as incorrect. Users aggregate this information, and define their trust value of a source as the ratio of correct statements w.r.t. all evaluated statements from the source. The general formula for computing trust values follows:

$t = \frac{t_p * n_p + n_t}{n_p + n_t + n_u}$, where n_t is the number of interactions evaluated as trustworthy; n_u is the number of interactions evaluated as untrustworthy; t_p is the level of anticipated trust; and n_p is a hypothetical number of interactions. t_p predetermines the starting trust level of a user of a source, another user, or an answering engine; and n_p defines the level of confidence of the user that t_p is correct – the higher n_p, the slower t changes its value, while "recording" actual interactions of the user, from the value of t_p. This approach is not absolutely new, and used in a similar form, for instance in [22]

5 Answer Trust Computation Algorithm

The *answer trust computation algorithm* (ATCA) computes $T(u_i, n(a))$, which is a set of user u_i's trust values, one value for each justification extracted from a node set n concluding a. An optional t_{min} minimum trust value may be used for eliminating the use of sources trusted at a level below a minimum threshold.

ATCA is described in Algorithm 1. There, it first initializes a relation R_T of tuples following the format $tr(n_o, n_d, t, l)$. In a tr tuple, n_o is the origin node, n_d is the destination node, t is the trust value of n_o to n_d, and l is the edge length between n_o and n_d. The trust value of n_o to n_d, reachable by some path, is computed by multiplying the trust values of all edges in the path. We call the result of the computation, the *path trust* value; and we write t_p to denote the path trust value for path p (or alternatively t_{od} to denote the trust value of a path between nodes o and d). Also, we write l_p to denote the length of p. Path trust values have a straightforward probabilistic interpretation: if origin node x trusts node y at the level of t_{xy}, i.e. the probability of that y provides trustworthy information (reference to other node) according to x is t_{xy}; and node y trusts destination node z at level t_{yz}, then the probability of that z provides trustworthy information according to x is $t_{xy} * t_{yz}$, which we consider to be the trust value of x to z.

In algorithm 1, ATCA calls the *provenance retrieval* algorithm (line 2), which returns $S(a)$, a set of provenance elements associated with $n(a)$. For a node set $n(a)$, a set of provenance elements $S(a)$ can be retrieved by using a tree search algorithm, i.e., depth-first search. The algorithm should traverse node set inference steps, and from inference steps, traverse their antecedent nodes that

Algorithm 1 Answer Trust Computation

input: u_i, $n(a)$, (optional t_{min}); **output**: R_T

1: $R_T \leftarrow \emptyset$;
2: $S(a) \leftarrow provenanceRetrieval(n(a))$
3: $R_a \leftarrow trustPathComputation(u_i, S(a), t_{min})$;
4: $J(a) \leftarrow proof(n(a))$
5: **for all** $(j_m \in J(a))$ **do**
6: $\quad R_T \leftarrow R_T + answerTrustComposition(u_i, j_m, R_a, t_{min})$;
7: **end for**

are also node sets. Thus, in the process of traversing all possible justifications for a, the algorithm records sources including the answering engines associated with each inference step reached in the search.

ATCA then calls the *trust path computation* algorithm (line 3) in order to enumerate paths from u_i to each provenance element in $S(a)$ and consequently to compute element's t and l values. The trust path computation algorithm is described in Section 5.1. Currently, for those provenance elements in $S(a)$, for which trust can not be computed, the user, likely after performing some analysis of these provenance elements, sets her trust values manually with path length of 1. Alternatively, the user may take a default value of trust for elements based simply on a preset value or, in a future system, based on aggregated trust settings for this or similar elements.

Trust values need to be computed for each proof j_m of a rather than for a set of justifications $n(a)$. Therefore, a set of proofs J needs to be "extracted" from $n(a)$ by using the *proof extraction* algorithm described in Section 3.2 in [19][5](line 4). Finally, ATCA passes each node $j_m \in J$ to the *answer trust composition* algorithm (line 6) that computes and adds an answer trust value to R_T. The answer trust composition algorithm is described in Section 5.2.

5.1 Trust Path Computation

We are not interested in paths from users to provenance elements of any possible length, and restrict the length to some (small) number of edges (e.g. 5-6 edges). There are two reasons for this. First, people tend not to trust another person much if the only path they have between themselves and the other person goes through a long chain of acquaintances, even if each link along the way has a high level of trust. Second, it has been also shown that WWW exhibits that two people in the world are separated on average by six acquaintances [2] as well as random graphs in general [21].

There may be several paths between a user and a provenance element with distinct path trust and length values. We address this issue by considering two possible strategies for selecting paths: *shortest path* and *highest trust*. In the shortest path approach the user chooses a path with the smallest number of edges, giving a preference to a path with the highest trust value if there are

[5] Each single justification from a conclusion c is called a "proof from c".

several paths of equal length. In the highest trust approach the user chooses a path with the highest trust value, giving a preference to the shortest path if there are several paths with the same trust value.

Using the requirements above, we can now formalize the trust path computation problem as follows: given a user u_i, a provenance element $p_{element}$, a maximum path length value p_{max}, a minimum trust value t_{min}, compute a path p from u_i to $p_{element}$, such that $l_p \leq p_{max}$, $t_p \geq t_{min}$, and:

- (a): $l_p \leq l_{p'}$ for any path $p' \neq p$, and $t_p \geq t_{p'}$ if $l_p = l_{p'}$ (shortest path strategy); or
- (b): $t_p \geq t_{p'}$ for any path $p' \neq p$, and $l_p \leq l_{p'}$ if $t_p = t_{p'}$ (highest trust strategy).

With this definition the trust path computation problem represents a path computation problem [12]. This class of problems deals with the enumeration of paths between two given nodes in a graph, and the computation of some particular properties of these paths (or finding a path with certain properties). A property of a path is a function of the labels assigned to the edges in the path, and may be, for example, the summation or multiplication of the edge weights, their maximum or minimum values, etc [3]. In particular, [3] proposes a generalized version of the *semi-naive* algorithm [4] to compute transitive closure of a relation in an iterative manner, producing at each iteration a set of new transitively held relations.

We extend the semi-naive algorithm to include our requirements. The pseudo-code of our algorithm is shown as Algorithm 2. Lines 2, 3, 4, 10 and 11 are the core steps of the semi-naive algorithm, whereas line 9 is a step of the algorithm extended to our needs. Particularly, at step 9 we compute a set of transitively held trust relations where: the origin of a trust relation must be u_i; resulting trust value must be greater or equal than the minimal trust value ($t_1 * t_2 \geq t_{min}$); and the path length should be less or equal to the maximum path length ($(l_1 + l_2) \leq p_{max}$). The $d_1 = o_2$ condition in line 9 represents the join condition. By introducing the $o_1 \neq d_2$ condition, we avoid the computation of paths containing loops.

Algorithm 2 works as follows. R_a represents the answer relation which is initialized to the empty set at line 1. R_t is the initial set of trust tuples, R_f is the transitive closure for R_t, and R_Δ represents the set of new trust tuples computed at each iteration. Both R_f and R_Δ are initially set to be equal to R_t (lines 2 and 3). At each iteration we remove from R_Δ (line 6) and add to R_a (line 7) tuples containing answer information, i.e. trust values of u_i to the provenance elements in $S(a)$ (line 5). $S(a)$ contains only those provenance elements, which are not directly connected with u_i by an edge. Note, that the trust value check at line 5 ($t \geq t_{min}$) is only introduced for the first iteration tuples, as tuples in all following iterations will already satisfy the requirement due to the identical condition in line 9. Line 10 of the pseudo code removes from R_Δ all trust tuples which originally existed (in R_t) or were computed in previous iterations. In line 11 the distinct tuples are added to the transitive closure relation R_f.

Algorithm 2 Trust Path Computation

input: u_i, $S(a)$, R_t, t_{min}; **output**: R_a

1: $R_a \leftarrow \emptyset$
2: $R_f \leftarrow R_t$
3: $R_\Delta \leftarrow R_t$
4: **while** $R_\Delta \neq \emptyset$ **do**
5: **for all** $(tr(o,d,t,l) \in R_\Delta | o = u_i, d \in S(a), t \geq t_{min})$ **do**
6: $R_\Delta \leftarrow R_\Delta \setminus \{tr\}$
7: $R_a \leftarrow R_a \cup \{< u_i, d, t, l >\}$
8: **end for**
9: $R_\Delta(o_1, d_2, t_1 * t_2, l_1 + l_2) \leftarrow R_\Delta(o_1, d_1, t_1, l_1), R_t(o_2, d_2, t_2, l_2),$
$o_1 = u_i, d_1 = o_2, o_1 \neq d_2, t_1 * t_2 \geq t_{min}, (l_1 + l_2) \leq p_{max}$
10: $R_\Delta \leftarrow R_\Delta \setminus R_f$
11: $R_f \leftarrow R_f \cup R_\Delta$
12: **end while**

As [3] points out, it is important to define a proper semantics for how tuples with identical o and d values are treated in the union and difference operations (as in lines 7 and 10 of our algorithm). We define the union and difference semantics based on the path trust computation strategy. In the shortest path approach, when the o and d values are identical for a set of tuples, we leave a tuple with the smallest l value and drop the others. If two or more tuples have equivalent l values, then the one with the highest t value is left. Analogously, in the highest trust approach we leave a tuple with the highest trust value giving a preference to the shortest path when trust values are equivalent. The algorithm eventually terminates when no new tuples are produced (checked at line 4). At this point, R_a contains all provenance elements from $S(a)$ which are "reachable" from u_i given path length and trust constraints.

5.2 Answer Trust Composition

Starting from the last step in proof j concluding c and proceeding through the set of antecedents C of the proof step deriving c (line 7 in Algorithm 3), the answer trust composition computes t_c and l_c recursively. The answer trust composition algorithm terminates when conclusions inside a proof have no antecedents ($C = \emptyset$). In this case, it is assumed an implicitly trust relation with value of 1.0 and of length 1 between the conclusion and its source s and t_c and l_c for the resulting tuple are inferred as the trust and the length for s (line 3).

For a conclusion c from a proof step with one or more antecedents, the algorithm works as follows: it first computes a weighted average over t_j for all $c_j \in C$, whereas the weights are inversely proportional to the path lengths of nodes in C. $Watv$ and wl are used to compute the answer weighted average trust value.

By computing a trust value for the answering engine input as the ratio between $watv$ and wl we are fully trusting the answering engine. However, we may

Algorithm 3 Trust Composition

input: u_i, R_a, j; **output**: $< u_i, c, t_c, l_c >$

note: c is the conclusion of proof j; s is the source of c, if any; e and C are the engine and set of antecedents for the last step in j deriving c

1: **if** $C = \emptyset$ **then**
2: $t_c, l_c \leftarrow R_a(u_i, s)$;
3: **else**
4: $watv, wl, length \leftarrow 0$;
5: **for all** $c_j \in C$ **do**
6: $t_j, l_j \leftarrow trustCompositionAlgorithm(u_i, c_j, R_a, t_{min})$;
7: $watv \leftarrow watv + (t_j/l_j)$;
8: $wl \leftarrow wl + (1/l_j)$;
9: $length \leftarrow total + l_j$;
10: **end for**
11: $t_e, l_e \leftarrow R_a(u_i, e)$
12: $t_c \leftarrow (watv/wl) * t_e$;
13: $l_c \leftarrow INT(length/|C|) + 1$
14: **end if**

have reasons for not trusting an engine that much. For instance, the engine may not be sound for some kind of questions. Thus, we "weigh" the input trust values against path lengths as shorter paths are likely to be more "credible" than longer ones. Then, we compute the trust value for t_c, l_c, by multiplying the weighted average trust value ($watv/wt$) by t_e (line 13). We compute the path length to c as the integer part of the average of all c_j, incremented by 1 (line 14).

The interpretation of trust and path length values for answers is different from the one for sources. The heuristics we use in the trust composition algorithm do not allow us to give a probabilistic interpretation to the resulting trust value, and we can not treat path length value as a number of acquaintances from the user to the answer. Trust values should be treated on a relative basis. An answer with higher trust value is more likely to be correct than an answer with lower trust value.

6 Trusting Answers: An Example

IWTrust's typical use of trust values is for comparison and ordering of answers. We do not expect that typical users will be interested in looking at raw trust values such as 0.234 but we do expect that they will be interested in knowing that some answers were trusted more than others. In this section, we present an example showing how trust values can be used to rank both answers and justifications.

Figure 3 shows a proof tree supporting the answer to a question concerning the type of Tony's Specialty. This particular proof tree encodes information justifying that Tony's specialty is a shellfish dish. In this example, Source 2 states that TonysSpecialty is a CRAB dish and Source 1 states that type is

transitive (thus if a dish is of type crab and crab is a kind of shellfish, then the dish is of type shellfish). Thus, using generalized modus ponens (GMP), the proof concludes that TonysSpecialty has the type of all of the superclasses of CRAB. Further, Source 2 and Source 3 state that SHELLFISH is a superclass of CRAB. Thus, using GMP again, the proof concludes that TonysSpecialty is a SHELLFISH dish. In the figure, the leaf nodes display their trust and path length values and the non-leaf nodes display trust values of answering engines, as well as computed trust and path length values.

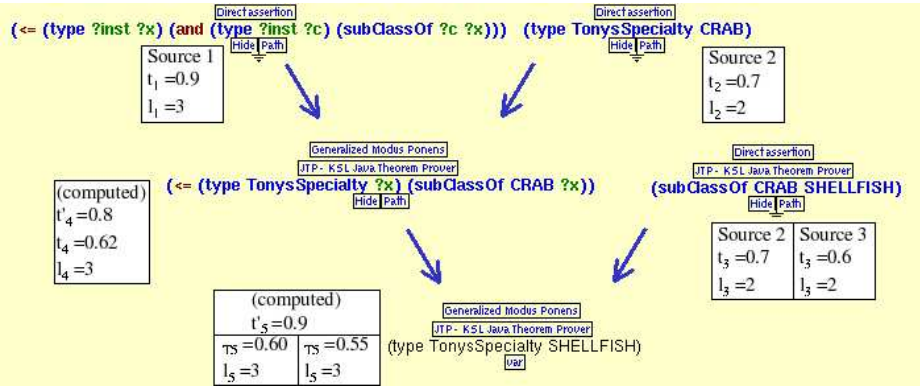

Fig. 3. Trust Composition Algorithm: an example

The proof shows that the question has at least two answers: one that TonysSpecialty is a SHELLFISH dish, and also that it is a CRAB dish. The proof also shows that the SHELLFISH answer has two justifications: one based on statements from sources 1, 2 and 3; and the other based on statements from sources 1 and 2 only. The example is simple but a question may have multiple answers and each answer may have multiple justifications when answering engines use the web as a search space. PageRank "Relevance" has proved to be a useful technique for ranking answers but we also expect trust to be a valuable ranking method. If we rank answers by trust values in this example, we would present crab before shellfish. However, if another source with trust level higher than Source 2 states that TonysSpecialty is a SHELLFISH dish than the SHELLFISH answer may appear first.

Trust rankings may be valuable for ranking of justifications and their components. For instance, if the user asks for a list of sources for the SHELLFISH answer, we may not include Source 3 since the justification based on Source 3 (t=0.55) has a trust value lower than the justification without it (t=0.60). However, a Source 4 also stating that SHELLFISH is a superclass of CRAB could be listed if the Source 4's trust value was high enough to make a justification based on its statement higher than 0.60.

This example shows that our work provides additional methods for filtering and ranking answers. In addition to ordering methods based on specificity of

answer or things like reverse links, we can also use trust of sources and trust in answers for choosing presentation of answers and justification options.

7 Conclusions

In this paper, we have introduced IWTrust as a solution framework supporting trust in question answering environments. IWTrust leverages the Inference Web infrastructure to provide explanations from a variety of question answering environments ranging from retrieval-intensive systems, such as database management systems and search engines, to reasoning-intensive systems such as theorem provers. It then enhances these explanations with user-customized trust values, which can then be used to determine trust of answers, sources, and answer justifications.

Our primary contributions are in two areas. First we provide an implemented solution infrastructure that can provide explanations to a wide range of question answering systems that has been integrated with a trust network. Second, we provide a design and prototype of a trust network with trust functionalities supporting trust computation in a distributed question answering environment such as the web. While others have provided trust algebras previously, and implemented explanation solutions exist for different types of question answering paradigms, our work is the first we know of that addresses the wide range of question answering systems with a trust integration and simultaneously provides an extensible architecture. Our primary contributions in the trust area include our design in the TrustNet layer presented in Section 4.3 and the answer trust computation algorithms presented in Section 5.

In summary, we have provided a proof-based solution for explanations enhanced with trust values that take into account user-context. The implemented solution includes our extensible explanation infrastructure, Inference Web, enhanced with a trust network and our trust computation algorithms. The resulting answers are filterable by trust and thus provide more ways for users to obtain more reliable and explainable answers in distributed web settings.

References

1. Karl Aberer and Zoran Despotovic. Managing trust in a Peer-2-Peer information system. In Henrique Paques, Ling Liu, and David Grossman, editors, *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM-01)*, pages 310–317, New York, November 5–10 2001. ACM Press.
2. Lada A. Adamic. The small world web. In S. Abiteboul and A.-M. Vercoustre, editors, *Research and Advanced Technology for Digital Libraries. Third European Conference, ECDL'99, Paris, France, September 22-24, 1999, Proceedings*, volume 1696 of *Lecture Notes in Computer Science*, pages 443–452. Springer, Heidelberg, Germany, 1999.
3. R. Agrawal, S. Dar, and H. Jagadish. Direct transitive closure algorithms: Design and performance evaluation. *ACM Transactions on Database Systems.*, 15(3):427–458, September 1990.

4. F. Bancilhon. Naive evaluation of recursively defined relations. In M. L. Brodie and J. Mylopoulos, editors, *On Knowledge Base Management Systems*, pages 165–178. Springer, New York, 1986.
5. Z. Despotovic and K. Aberer. A probabilistic approach to predict peers' performance in p2p networks. In *Proceedings of the Eighth International Workshop on Cooperative Information Agents, CIA*, Erfurt, Germany, September 2004.
6. D. Ferrucci and A. Lally. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Journal of Natural Language Engineering*, 10(3/4):327–348, June 2004.
7. Richard Fikes, Pat Hayes, and Ian Horrocks. DAML Query Language (DQL) Abstract Specification. Technical report, W3C, 2002.
8. Richard Fikes, Jessica Jenkins, and Gleb Frank. JTP: A System Architecture and Component Library for Hybrid Reasoning. Technical Report KSL-03-01, Knowledge Systems Laboratory, Stanford University, Stanford, CA, USA, 2003.
9. Yolanda Gil and Varun Ratnakar. Trusting information sources one citizen at a time. In *Proceedings of the First International Semantic Web Conference (ISWC), Sardinia, Italy*, pages 163–176, June 2002.
10. Jennifer Golbeck and James Hendler. Accuracy of Metrics for Inferring Trust and Reputation in Semantic Web-based Social Networks. In *Proceedings of 14th International Conference on Knowledge Engineering and Knowledge Management (EKAW'04)*, 2004.
11. Jennifer Golbeck, Bijan Parsia, and James Hendler. Trust networks on the semantic web. In *Proceedings of Cooperative Intelligent Agents*, Helsinki, Finland, August 2003.
12. Yannis E. Ioannidis and Raghu Ramakrishnan. Efficient transitive closure algorithms. In *Fourteenth International Conference on Very Large Data Bases, August 29 - September 1, 1988, Los Angeles, California, USA, Proceedings*, pages 382–394. Morgan Kaufmann, 1988.
13. Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the Twelfth International World Wide Web Conference (WWW)*, 2003.
14. Sergio Marti and Hector Garcia-Molina. Limited reputation sharing in p2p systems. In *ACM Conference on Electronic Commerce (EC'04)*, pages 91–101, May 2004.
15. R. Matthew, R. Agrawal, and P. Domingos. Trust management for the semantic web. In *Proceedings of the Second International Semantic Web Conference*, Sanibel Island, Florida., October 2003.
16. Deborah L. McGuinness and Paulo Pinheiro da Silva. Registry-based support for information integration. In S. Kambhampati and C. Knoblock, editors, *In Proceedings of IJCAI-2003's Workshop on Information Integration on the Web (IIWeb-03)*, pages 117–122, Acapulco, Mexico, August 2003.
17. Deborah L. McGuinness and Paulo Pinheiro da Silva. Explaining Answers from the Semantic Web. *Journal of Web Semantics*, 1(4):397–413, October 2004.
18. Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.
19. Paulo Pinheiro da Silva, Deborah L. McGuinness, and Richard E. Fikes. A Proof Markup Language for Semantic Web Services. *Information Systems*, 2005. (to appear).
20. Paulo Pinheiro da Silva, Deborah L. McGuinness, and Rob McCool. Knowledge Provenance Infrastructure. *In Data Engineering Bulletin Vol.26 No.4*, pages 26–32, December 2003.
21. Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.
22. Wang Y. and Vassileva J. Trust-based community formation in peer-to-peer file sharing networks. *Proc. of IEEE/WIC/ACM International Conference on Web Intelligence (WI 2004), Beijing, China*, September 2004.