Propagating Multitrust within Trust Networks[*] [†]

Stefano Bistarelli
Department of Sciences, University "G. d'Annunzio" of Chieti-Pescara
Viale Pindaro 87, Pescara, Italy
bista@sci.unich.it

Francesco Santini
IMT Lucca - School for Advanced Studies
Piazza San Ponziano 6, Lucca, Italy
f.santini@imtlucca.it

ABSTRACT

We suggest the concept of *multitrust*, which is aimed at computing trust by collectively involving a group of trustees at the same time: the trustor needs the concurrent support of multiple individuals to accomplish its task. We propose *Soft Constraint Logic Programming* based on semirings as a mean to quickly represent and evaluate trust propagation for this scenario. To attain this, we model the trust network adapting it to a weighted *and-or* graph, where the weight on a connector corresponds to the trust feedback value among the connected nodes. Semirings are the parametric and flexible structures used to appropriately represent trust metrics.

Keywords

Soft Constraint Logic Programming, *and-or* Graphs, Trust Propagation, Trust Network

1. INTRODUCTION

Decentralized trust management [5, 11] provides a different paradigm of security in open and widely distributed systems where it is not possible to rely solely on traditional security measures as cryptography. The reasons usually are that the nodes appear and disappear from the community, span multiple administrative domains, their direct interactions are limited to a small subset of the total number of nodes and, moreover, there is no globally trusted third party that can supervise the relationships. For this reason a computational model is needed to derive a trust value among the individuals of a community, represented as a trust network, in the following abbreviated as *TN*.

Two contributions are given in this paper: first we propose the concept of *multitrust*, i.e. when the relationship of trust concerns one trustor and multiple trustees in a correlated way (the name recalls the *multicast* delivery scheme in networks). An example in peer-to-peer networks is when we download a file from multiple sources at the same time, and we need a reliability feedback for the whole download process. Notice that the trust feedback towards a group of individuals is usually different from merely summing up the component of each one: the characteristics and capabilities of individuals can benefit if suitably collaborating together, or loss something if in contrast. Consider sport teams, for example: having many good players in the same team does not necessarily imply to have the winner team.

Secondly, we outline a model to solve trust propagation in this new scenario: we represent TNs (the same model applies also to related terms in literature as trust graph, web of trust or social network [7, 11, 17]) as *and-or* graphs [14] (i.e. hypergraphs), mapping individuals to nodes and their relationships to directed connectors. The *and* connectors (i.e. hyperarcs) represent the event of simultaneously trusting a group of individuals at the same time. The costs of the connectors symbolize how trustworthy the source estimates the destination nodes, that is a *trust value*.

Afterwards, we propose the *Soft Constraint Logic Programming* (SCLP) framework [1, 3] as a convenient declarative programming environment in which to solve the trust propagation problem for multitrust. In SCLP programs, logic programming is used in conjunction with soft constraints, that is, constraints which have a preference level associated to them. In particular, we show how to translate the *and-or* graph obtained in the first step into a SCLP program, and how the semantics of such a program computes the best trust propagation tree in the corresponding weighted *and-or* graph. SCLP is based on the general structure of a *c-semiring* [1] (or simply, semiring) with two operations $\times$ and $+$. The $\times$ is used to combine the preferences, while the partial order defined by $+$ (see Section 2) is instead used to compare them. Therefore, we can take advantage of the semiring structure to model and compose different trust metrics. SCLP is also parametric with respect to the chosen semiring: the same program deals with different trust metrics by only choosing the proper semiring structure.

This paper is organized as follows: in Sec. 2 we present some background information about trust metrics and the SCLP framework. Section 3.1 depicts how to represent a TN with an *and-or* graph, while in Sec. 4 we describe the way to pass from *and-or* graphs to SCLP programs, showing that the semantic of SCLP program is able to compute the best trust propagation tree in the corresponding *and-or* graph. Section 5 draws the final conclusions and outlines intentions for future works.

[*]Partially supported by Institute for Informatics and Telematics (IIT-CNR) Pisa, Italy.

[†]Supported by the MIUR PRIN 2005-015491

2. BACKGROUND

Trust and Metrics. No universal agreement on the definition of trust and reputation concepts has been yet reached in the community [13]. However, we adopt the following definitions: trust describes a nodes belief in another nodes capabilities, honesty and reliability based on its own direct experiences, while reputation is based on recommendations received also from other nodes. The main difference between trust and reputation is that trust systems produce a score that reflects the relying party's subjective view of an entity's trustworthiness, whereas reputation systems produce an entity's (public) reputation score as seen by the whole community.

Trust and reputation ranking metrics have primarily been used for public key certification in distributed or hierarchical systems (as respectively PGP or PKI systems), rating and reputation systems part of online communities, peer-to-peer networks, semantic web and also mobile computing fields [11, 13, 16, 17]. Each of these scenarios favors different trust metrics. Trust metrics are used to predict trust scores of users by exploiting the transitiveness property of relationships (thus, we are considering transitive trust chains): if two nodes, say node A and node C in Fig. 1a, do not have a direct edge connecting them, the TN can be used to generate an inferred reputation/trust rating. A TN represents all the direct trust relationship in a community. An example of a classical TN is provided in Fig. 1a, where we can see that trust is usually represented as a 1-to-1 relationship between only two individuals: the edges are directed from the trustor to the trustee. If node A knows node B, and node B knows node C, then A can use the path to compose the inferred rating for C. This process is called *trust propagation* by concatenation, and it is a necessary requirement since in most settings a user has a direct opinion only about a very small portion of nodes in the TN. Therefore, trust needs to be granted also by basing on third-party recommendations.

According to [16], trust metrics can be classified in:

- *Basic*: based only on the reachability of the nodes in the TN (as in Fig. 1a): if a target node is reachable, then it can be trusted by the source node. The length of the path can be bounded, i.e. *bounded reachability*. One example is X.509 [16], in which an extension of the certificate can limit the length of the remaining segment of the certification path.

- *Weighted*: if the edges representing trust relationships are weighted, and the trust score is obtained by finding the cost of the paths between the source and the target nodes. Examples are Maurer or Rahman-Hailes metrics [16].

- *Flow Related*: where a "flow" or an "energy" amount is injected in the graph and the trustworthiness of a node is defined by the quantity of flow which reaches it. These metrics show a greater resistance to forgery-based attacks [16, 17].

Soft Constraint Logic Programming. The SCLP framework [1, 3], is based on the notion of *c-semiring* introduced in [4]. A c-semiring S is a tuple $\langle A, +, \times, 0, 1\rangle$ where A is a set with two special elements $(0, 1 \in A)$ and with two operations $+$ and $\times$ that satisfy certain properties: $+$ is defined over (possibly infinite) sets of elements of A and thus is commutative, associative, idempotent, it is closed and 0 is its unit element and 1 is its absorbing element; $\times$ is closed, associative, commutative, distributes over $+$, 1 is its unit element, and 0 is its absorbing element (for the exhaustive definition, please refer to [4]).

Table 1: A simple example of an SCLP program.

```
s(X)   :- p(X,Y).
p(a,b) :- q(a).
p(a,c) :- r(a).
q(a)   :- t(a).
t(a)   :- 2.
r(a)   :- 3.
```

The $+$ operation defines a partial order $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$; we say that $a \leq_S b$ if b represents a value *better* than a. Other properties related to the two operations are that $+$ and $\times$ are monotone on $\leq_S$, 0 is its minimum and 1 its maximum, $\langle A, \leq_S\rangle$ is a complete lattice and $+$ is its lub. Finally, if $\times$ is idempotent, then $+$ distributes over $\times$, $\langle A, \leq_S\rangle$ is a complete distributive lattice and $\times$ its glb.

Semiring-based constraint satisfaction problems (SCSPs) are constraint problems where each variable instantiation is associated to an element of a c-semiring A (to be interpreted as a cost, level of preference or, in this case, as a trust/reputation level), and constraints are combined via the $\times$ operation and compared via the $\leq_S$ ordering. Varying the set A and the meaning of the $+$ and $\times$ operations, we can represent many different kinds of problems, having features like fuzziness, probability, and optimization. In Sec. 3, the set A is used to collect the trust values, the $\times$ operator to combine them into a result for a complete tree, and $+$ to find the best (i.e. the most trustworthy) tree.

A simple example of a SCLP program over the semiring $\langle N, min, +, +\infty, 0\rangle$, where N is the set of non-negative integers and $D = \{a, b, c\}$, is represented in Tab. 1. The intuitive meaning of a semiring value like 3 associated to the atom $r(a)$ (in Tab. 1) is that $r(a)$ costs 3 units. Thus the set N contains all possible costs, and the choice of the two operations min and $+$ implies that we intend to minimize the sum of the costs. This gives us the possibility to select the atom instantiation which gives the minimum cost overall. Given a goal like $s(x)$ to this program, the operational semantics collects both a substitution for x (in this case, $x = a$) and also a semiring value (in this case, 2) which represents the minimum cost among the costs for all derivations for $s(x)$. To find one of these solutions, it starts from the goal and uses the clauses as usual in logic programming, except that at each step two items are accumulated and combined with the current state: a substitution and a semiring value (both provided by the used clause). The combination of these two items with what is contained in the current goal is done via the usual combination of substitutions (for the substitution part) and via the multiplicative operation of the semiring (for the semiring value part), which in this example is the arithmetic $+$. Thus, in the example of goal $s(X)$, we get two possible solutions, both with substitution $X = a$ but with two different semiring values: 2 and 3. Then, the combination of such two solutions via the min operation give us the semiring value 2.

3. PROPAGATING MULTITRUST

We introduce the concept of multitrust, which extends the usual trust relationship from couples of individuals to one trustor and multiple trustees in a correlated way:

DEFINITION 3.1. *Given a set of entities S in the considered trust domain, multitrust is defined as a relationship R_{mt} between a trustor $t \in S$ and a set of trustees $T \subset S$, where $t \notin T$ and $|T| \geq 1$. R_{mt} can be described in terms of time (e.g. at the same time), modalities (e.g. with the same behavior) or collaboration among the trustees in T w.r.t. t.*

Consequently, this trust relation R_{mt} is 1-to-n (no more 1-to-1 as in all the classical trust systems [17]) and can be created by concurrently involving all the interested parties in a shared purpose. Some everyday examples can be found when downloading a file from multiple sources at the same time in peer-to-peer networks, or, in general, when a task must/can be accomplished with the help of many individuals acting together and a trust feedback must be found for the whole process. Another general application can be for *team effectiveness* [8]: suppose we have a decentralized community of open-source programmers and we want to know if a subset of them can be reliably assigned to a new project. Delegating the search to a group of 3 individuals, for example, could significantly enhance the result since we suppose they will accurately collaborate together by joining their skills and obtaining a better result w.r.t. 3 different and independent searches. Thus, the group will be more trustworthy than the single individuals, and even the final trustees will benefit from this group collaboration: they will be reached with an higher score during the propagation of trust in the TN.

According to the classification in [17], also *group* metrics are very well suited to computing membership in a group, since they parallelly evaluate groups of trust assertions [17] for a target node. However, notice that our definition of multitrust for groups is different, since only one 1-to-n assertion is considered among a trustor and its trustees, and not a group of them. Moreover, the idea of the correlated event is not present in *group* metrics.

3.1 From Trust Networks to *And-or* Graph

An *and-or* graph [14] is defined essentially as a hypergraph. Namely, instead of arcs connecting pairs of nodes there are hyperarcs connecting an n-tuple of nodes ($n = 1, 2, 3, \ldots$). The arcs are called *connectors* and they must be considered as directed from their first node to all others. Formally an *and-or* graph is a pair $G = (N, C)$, where N is a set of *nodes* and C is a set of connectors

$$C \subseteq N \times \bigcup_{i=0}^{k} N^i.$$

Note that the definition allows 0-connectors, i.e. connectors with one input and no output node. In the following of the explanation we will also use the concept of *and* tree [14]: given an *and-or* graph G, an *and* tree H is a *solution tree of G with start node n_r*, if there is a function g mapping nodes of H into nodes of G such that: *i)* the root of H is mapped in n_r, and *ii)* if $(n_{i_0}, n_{i_1}, \ldots, n_{i_k})$ is a connector of H, then $(g(n_{i_0}), g(n_{i_1}), \ldots, \ldots, g(n_{i_k}))$ is a connector of G.

Informally, a solution tree of an *and-or* graph is analogous to a path of an ordinary graph: it can be obtained by selecting exactly one outgoing connector for each node.

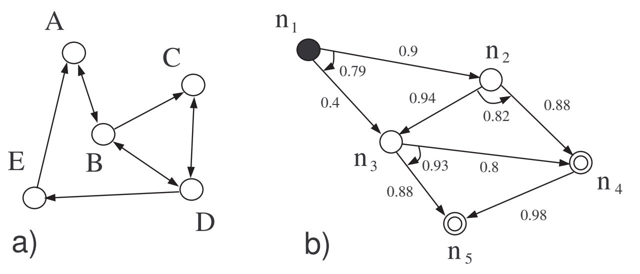

Figure 1: *a)* a classical trust network, and *b)* an *and-or* graph representing multitrust.

In Fig. 1b we directly represent a TN for multitrust as a weighted *and-or* graph, since for its characteristics, this translation is immediate. Each of the individuals can be easily cast in a corresponding node of the *and-or* graph. In Fig. 1b we represent our trustor as a black node (i.e. n_1) and the target trustees as two concentric circles (i.e. n_4 and n_5). Nodes n_2 and n_3 can be used to propagate trust.

To model the classical trust relationship between two nodes we use 1-connectors, which correspond to usual TN arcs: the 1-connectors in Fig. 1b are (n_1, n_2), (n_1, n_3), (n_2, n_3), (n_2, n_4), (n_3, n_4), (n_3, n_5), (n_4, n_5). We remind that the connectors are directed, and thus, for example the connector (n_4, n_5) means that the input node n_4 trusts the individual represented by n_5. Moreover, since we are now dealing with multitrust, we need to represent the event of trusting more individuals at the same time. To attain this, in Fig. 1b we can see the three 2-connectors (n_1, n_2, n_3), (n_2, n_3, n_4) and (n_3, n_4, n_5): for example, the first of these hyperconnectors defines the possibility for n_1 to trust both n_2 and n_3 in a correlated way. In Fig. 1b we draw these n-connectors (with $n > 1$) as curved oriented arcs where the set of their output nodes corresponds to the nodes traversed by the curved arc. Considering the ordering of the nodes in the tuple describing the connector, the input node is at the first position and the output nodes (when more than one) follow the orientation of the related arc in the graph (in Figure 1b this orientation is lexicographic). Notice that in the example we decided to use connectors with dimension at most equal to 2 (i.e. 2-connectors) for sake of simplicity. However it is possible to represent whatever cardinality of trust relationship, that is among a trustor and n trustees (i.e. with a n-connector).

Each of the connectors in Fig. 1b is labeled with a trust value in the range $[0, 1]$. A trust value close to 1 indicates that the output nodes of the connector have gained good reputation in terms of their past performance and thus are more trustworthy, whereas a low trust value means the nodes showed relatively poor QoS in the past and are rated with low reputations. In general, we could have trust expressed with a k-dimensional vector representing k different metrics.

Collecting the trust values to assign to the labels of the connectors is out of the scope of this work, since they can be described in terms of specificity/generality dimensions (one or more aspects) and subjective/objective dimensions (personal, as e-Bay, or formal criteria, as credit rating) [13]. However, for n-connectors with $n \geq 2$, we can suppose also the use of a composition operation $\circ$ which takes n 1-dimensional vectors (e.g. $tvalue_1, \ldots, tvalue_n$) as operands and returns the estimated trust value for that n-connectors

($tvalue_{nc}$): $\circ\,(tvalue_1, tvalue_2, \ldots, tvalue_n) \longrightarrow tvalue_{nc}$.

This $\circ$ operation can be easily found for objective ratings, since they are the result of applying formal aspects that have been clearly defined, while automating the computation of subjective ratings is undoubtedly more difficult. Notice also, as said before, that such a $\circ$ operation is not only a plain "addition" of the single trust values, but it must take into account also the "added value" (or "subtracted value") derived from the combination effect. For example, considering the connector (n_3, n_4, n_5) in Fig. 1b, its cost, i.e. 0.93, significantly benefits from simultaneously trusting n_4 and n_5, since both the trust values of (n_3, n_4) and (n_3, n_5) are sensibly lower (i.e. respectively 0.8 and 0.88). On the other hand, n_2 does not consider n_3 and n_4 to be so "collaborative" since the trust label of (n_2, n_3, n_4), i.e. 0.6, is decidedly worse than the costs of (n_2, n_3) and (n_2, n_4) (i.e. 0.94 and 0.88). In the example in Fig. 1b we supposed to use subjective ratings, and therefore the trust values for 2-connectors do not follow any specific $\circ$ function.

So far, we are able to represent an entire TN with a weighted *and-or* graph, but still we need some algebraic framework to model our preferences for the connectors, to use during trust propagation. For this reason, we decided to use the semiring structure (see Sec. 2) usually adopted for *Probabilistic SCSPs* [1], since the confidence of a node is often seen as a probability score in many works [7, 16]: $S_{Trust} = \langle [0,1], \max, \times, 0, 1\rangle$. With this semiring we compose the independent probabilities with the product operation (i.e. the arithmetic $\times$) and we maximize the global trust result with max.

Notice that other semirings can be used to model other trust metrics: for example, the *Fuzzy Semiring* $\langle [0,1], \max, \min, 0, 1\rangle$ can be used if we decide that the score of a trust chain corresponds to the weakest of its links. Or we can select the *Weighted Semiring*, i.e. $\langle \mathcal{R}^+, \min, +, \infty, 0\rangle$, to count negative referrals in reputation systems as e-Bay [13]. Moreover, since the cartesian product of semirings is still a semiring [1], trust can be propagated by considering several independent criteria (i.e. metrics) at the same time. If the k-dimensional costs of the connectors are not elements of a totally ordered set, it may be possible to obtain several Pareto-optimal trees.

In Fig. 1b we avoided cycles to keep the graph readable; however, the *Probabilistic Semiring* automatically avoids cycles in the SCLP program, since it maximizes the result [3].

4. *AND-OR* GRAPHS USING SCLP

In this Section, we represent the *and-or* graph in Fig. 1b with a program in SCLP. Since SCLP is a declarative programming environment, it is relatively easy to specify trust relationships; moreover, the c-semiring structure is a very parametric tool where to represent different trust metrics.

Using this framework, we can easily find the trust propagation trees over the hypergraph built in Sec. 3.1. In fact, in this paper our aim is to find the best tree simultaneously reaching all the desired trustees, which is only one of the possible choices when computing trust [7]: according to *multipath propagation*, when multiple propagation paths (in this case, trees) exist between A and C (in this case, several trustees at the same time), all their relative trust scores can be composed together in order to have a single result balanced with every opportunity.

Table 2: The CIAO program representing all the AND trees over the *and-or* graph in Fig. 1b

```
:- module(trust,_,_).
:- use_module(library(lists)).

times(T1, T2, T) :-
      T is T1 * T2,!.

Leaves
leaf([n1], 1).
leaf([n2], 1).
leaf([n3], 1).
leaf([n4], 1).
leaf([n5], 1).

Connectors
connector(n1,[n2], 0.9).
connector(n1,[n3], 0.4).
connector(n1,[n2,n3], 0.79).
connector(n2,[n3], 0.94).
connector(n2,[n4], 0.88).
connector(n2,[n3,n4], 0.82).
connector(n3,[n4], 0.8).
connector(n3,[n5], 0.88).
connector(n3,[n4,n5], 0.93).
connector(n4,[n5], 0.98).

1)  tree(X,[X], T):-
          leaf([X], T).

2)  tree(X, Z, T):-
          connector(X,W, T1),
          treeList(W,Z, T2),
          times(T1, T2, T).

3)  treeList([],[], 1).

4)  treeList([X|Xs],Z, T):-
          tree(X, Z1, T1),
          append(Z1, Z2, Z),
          treeList(Xs, Z2, T2),
          times(T1, T2, T).
```

To represent the connectors in SCLP we can write clauses like $c(n_i, [n_j, n_k]) :-\ tvalue$, meaning that the graph has connector from n_i to nodes n_j and n_k with $tvalue$ cost. Then, other SCLP clauses describe the structure of the tree we desire to search over the graph.

We chose to represent an *and-or* graph with a program in *CIAO Prolog* [6], a system that offers a complete Prolog system supporting ISO-Prolog and constraint programming expressions. CIAO Prolog has also a fuzzy extension, but since it does not completely conform to the semantic of SCLP defined in [3] (due to interpolation in the interval of the fuzzy set). For this reason, we inserted the cost of the connector in the head of the clauses, differently from SCLP clauses which have the cost in the body of the clause.

From the *and-or* graph in Fig. 1b we can build the corresponding CIAO program of Tab. 2 as follows. First, we describe the connectors of the graph with facts like

$$connector(source\_node, [list\_of\_destination\_nodes], tvalue)$$

e.g. the fact $connector(n_1, [n_2, n_3], 0.79)$ represents the connector of the graph (n_1, n_2, n_3) with a trust value of 0.79. The set of connector facts is highlighted as *Connectors* in Tab. 2. The *Leaves* facts of Tab. 2 represent the terminations for the Prolog rules. Their cost must not influence the final trust score, and then it is equal to the unit element of *times*, i.e. 1. The *Times* clause in Tab. 2 mimics the $\times$ operation of the c-semiring proposed in Section 3.1: $S_{Trust} = \langle [0,1], \max, \times, 0, 1\rangle$. At last, the rules *1-2-3-4* of Tab. 2 describe the structure of the trees we want to find over the graph. *Rule 1* represents a tree made of only one leaf node, *Rule 2* outlines a tree made of a connector plus a list of sub-trees with root nodes in the list of the destination nodes of the connector, *Rule 3* is the termination for *Rule 4*, and *Rule 4* is needed to manage the junction of the disjoint sub-trees with roots in the list $[X|Xs]$. When we compose connectors and trees (*Rule 2* and *Rule 4*), we use the *Times* clause to compose their costs together.

To make the program in Tab. 2 as readable as possible, we omitted two predicates: the *sort* predicate, needed to order the elements inside the list of destination-nodes of connectors and trees (otherwise, the query $tree(n_1, [n_4, n_5], T)$ and $tree(n_1, [n_5, n_4], T)$ would produce different results), and the *intersection* predicate to check that multiple occurrences of the same node do not appear in the same list of destination

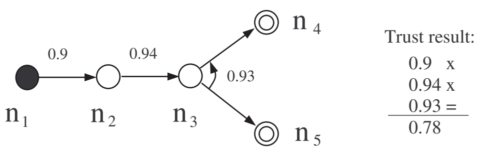

Figure 2: One of the trust propagation tree that can be found with the program in Tab. 2, and its cost.

nodes, if reachable from different connectors (otherwise, for example, $tree(n_1, [n_4, n_4, n5]$ would be a valid result).

To solve the *and-or* graph problem it is enough to perform a query in Prolog language: for example, if we want to compute the cost of all the trees rooted at n_1 and having as leaves the nodes representing the receivers (in this case, $\{n_4, n_5\}$), we have to perform the query $tree(n_1, [n_4, n_5], T)$, where T will be instantiated with the trust cost of the found trees. One of the outputs of the CIAO program for this query corresponds to the cost of the tree in Fig. 2, i.e. 0.78. Using multiple metrics can be clearly ore informative than a plain number in the interval $[0, 1]$.

The representation of TN for multitrust given in Sec. 3.1 can lead to computability problems because of the outdegree of the nodes: in theory, for each of the N individuals in the *and-or* graph we could have a connector towards each of the subsets of individuals in the network, whose number is $O(2^N)$. The model could be composed by a total of $O(N \cdot 2^N)$ connectors, thus in the worst case it is exponential in the number of graph nodes. However, the complexity of the tree search can be reduced by using *tabling* (or *memoing*) techniques [9]. The calls to *tabled* predicates are stored in a searchable structure together with their proven instances, and subsequent identical calls can use the stored answers without repeating the computation. However, notice that most of social networks bear significant traits of *small-world* networks in which most nodes are not neighbors of one another [17]: therefore, the complexity problem is limited.

5. CONCLUSIONS

We have defined the concept of multitrust and we have described a method to represent and solve the trust propagation problem in such scenario, with the combination of *and-or* graph and SCLP programming. Our framework can be fruitfully applied to have a quick and elegant formal-model where to compute the results of different trust metrics. We think that multitrust can be used in many real-world cases: trusting a group of individuals at the same time can lead to different conclusions w.r.t. simply aggregating together the trust values of the single individuals in the group.

Our future goal is to find a structure able to aggregate distinct paths and trees (for multitrust) in a sigle trust value, i.e. to compute *multipath* propagation (e.g. an average cost of the independent paths). Moreover, we would like to model network-flow related metrics, as Levien and Reiter-Stubblebine metrics [16], or energy-propagation metrics as Appleseed [17], because all these group metrics show a greater resistance to forgery-based attacks. .

Moreover, we are considering to adapt *Subjective Logic* [12] to semirings and use the results of this paper for the propagation of *opinion metric* [12]. We also plan to use the *expectation semiring* [10], where an element of the semiring set corresponds to the couple "trust-probability", that is a trust value and the confidence in that value (a *referral trust* [17]). Then, we would like to study in our framework the *correlated evidence* problem described in [15].

At last, we would like to introduce the notion of "distrust" in the model and to propagate it by using operations based on the inverse of the semiring $\times$ operator [2].

6. REFERENCES

[1] S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *LNCS*. Springer, 2004.

[2] S. Bistarelli and F. Gadducci. Enhancing constraints manipulation in semiring-based formalisms. In *ECAI 2006*, pages 63–67. IOS Press, 2006.

[3] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint logic programming. In *Proc. IJCAI97*, pages 352–357. Morgan Kaufman, 1997.

[4] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

[5] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *SP '96: Symp. on Security and Privacy*, page 164. IEEE Computer Society, 1996.

[6] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla. The CIAO prolog system: reference manual. Technical Report CLIP3/97.1, School of Computer Science, Technical University of Madrid (UPM), 1997.

[7] G. Caronni. Walking the web of trust. In *WETICE '00*, pages 153–158. IEEE Computer Society, 2000.

[8] A. C. Costa, R. A. Roe, and T. Taillieu. Trust within teams: the relation with performance effectiveness. *European Journal of Work and Organizational Psychology*, 10(3):225–244, 2001.

[9] B. Cui and D. Warren. A system for tabled constraint logic programming. In *Computational Logic*, pages 478–492. Springer-Verlag, 2000.

[10] J. Eisner. Parameter estimation for probabilistic finite-state transducers. In *ACL '02*, pages 1–8. Association for Computational Linguistics, 2001.

[11] J. Jonczy and R. Haenni. Credential networks: a general model for distributed trust and authenticity management. In *PST*, 2005.

[12] A. Jøsang. A logic for uncertain probabilities. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 9(3):279–311, 2001.

[13] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43(2):618–644, 2007.

[14] A. Martelli and U. Montanari. Optimizing decision trees through heuristically guided search. *Commun. ACM*, 21(12):1025–1039, 1978.

[15] M. Schillo, P. Funk., and M. Rovatsos. Who can you trust: dealing with deception. In *Proc. of of the Autonomous Agents Workshop on Deception, Fraud and Trust in Agent Societies*, pages 95–106, 1999.

[16] A. Twigg and N. Dimmock. Attack-resistance of computational trust models. In *WETICE '03*, pages 275–280. IEEE Computer Society, 2003.

[17] C.-N. Ziegler and G. Lausen. Propagation models for trust and distrust in social networks. *Information Systems Frontiers*, 7(4-5):337–358, 2005.