Lightweight Distributed Trust Propagation

Daniele Quercia, Stephen Hailes, Licia Capra
Department of Computer Science, University College London, London, WC1E 6BT, UK
{D.Quercia, S.Hailes, L.Capra}@cs.ucl.ac.uk

Abstract

Using mobile devices, such as smart phones, people may create and distribute different types of digital content (e.g., photos, videos). One of the problems is that digital content, being easy to create and replicate, may likely swamp users rather than informing them. To avoid that, users may organize content producers that they know and trust in a web of trust. Users may then reason about this web of trust to form opinions about content producers with whom they have never interacted before. These opinions will then determine whether content is accepted. The process of forming opinions is called trust propagation. *We design a mechanism for mobile devices that effectively propagates trust and that is lightweight and distributed (as opposed to previous work that focuses on centralized propagation). This mechanism uses a graph-based learning technique. We evaluate the effectiveness (predictive accuracy) of this mechanism against a large real-world data set. We also evaluate the computational cost of a J2ME implementation on a mobile phone.*

1 Introduction

Researchers are realizing that mobile devices may engage people in many different ways. For example, mobile devices allow people to take photos or shoot videos and distribute them to their local communities at very low cost. Content distribution may help to engage people in, for example, urban planning or creative expression [4, 19]. But what happens if everybody is distributing content? In that case, to paraphrase Italo Calvino, we would live in an unending rainfall of content [5].

To avoid content overload, we need new ways of filtering content (of deciding which content to accept). Conventional wisdom holds that one such way is to maintain a web of trust [11, 24] of content producers. A web of trust is a network of trust relationships: we trust (link to) only a handful of other people; these people, in turn, trust (link to) a limited number of other individuals; overall, these trust relationships form a network (a web of trust) of individuals linked by trust relationships. Based upon this web of trust, individuals may form opinions of other individuals (in technical parlance, they *propagate trust* in other individuals) from whom they have never received content before. Individuals then decide whether to accept content according to these opinions.

Section 2.2 will show that existing ways of propagating trust cannot be readily applied in mobile computing because they are usually designed to work on a centrally stored web of trust and to run on high-end machines. Thus we set out to design a novel way of propagating trust that works in distributed settings (e.g., P2P networks) and runs on (resource-constrained) mobile phones. Our core contributions include:

- A new trust propagation model that exploits a graph-based semi-supervised learning scheme [12, 23], carefully adapted to our domain. The key idea is that each mobile device stores a very limited subset of the web of trust; on that subset, it then applies a machine learning technique for propagating trust. Section 2.3 introduces the technique and Section 3 describes it in details.

- Evaluation of the accuracy of our proposition on a real and large web of trust (Section 4.1).

- Evaluation of its robustness against simulated uncooperative users (Section 4.2).

- Evaluation of the computational overhead of a J2ME implementation on a Nokia mobile phone (Section 4.3).

2 Overview

We now describe our research problem more formally. We will then demonstrate that existing solutions are not suitable for mobile devices. Finally, we will briefly introduce our proposed solution.

1550-4786/07 $25.00 © 2007 IEEE
DOI 10.1109/ICDM.2007.64

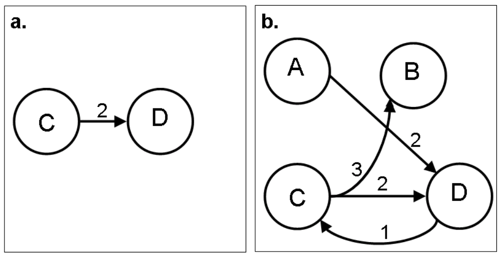


Figure 1. (a) A simple web of trust representing the statement "C rates 2 its trust in D". (b) A web of trust of four people connected by their trust relationships.

2.1 Problem Statement

One may represent the statement "C trusts D" as a web-of-trust of two persons and one trust relationship going from C to D (Fig. 1(a)). The relationship may also be labeled with a rating representing the extent to which C trusts D in a given range of trust values (for example, in Fig. 1(a), C rates its trust for D as 2 in a discrete range $\{1, 2, 3\}$[1]). Now, consider the web of trust in Fig. 1(b), where we have four people A, B, C and D. Not everyone has interacted with everyone else. For example, A and B have never interacted (no link between them). For the sake of argument, suppose that A now wishes to interact with B and, as a consequence, it has to form an opinion about B. A may do so by predicting its trust for B. The goal of this paper is to study how a mobile user A may form an opinion about another user B without prior interaction.

2.2 Existing Solutions: Unfit for Mobile Computing

To form an opinion about B, A may use the ratings of its past experiences [6, 17, 18]. However, that is not possible if A has never interacted with B. In that case, literature suggests that A may create a web of trust from third-party ratings and, based on that, may then *set* its trust (*propagate its trust*) for B.

There is substantial literature on *how to* propagate trust. That literature breaks roughly into two camps. In the first, techniques assign a *global* trust value to each user. That is, A's trust in B corresponds to a global trust value in B. By global, we mean a trust value that is accepted and shared by all users. In peer-to-peer networks, EigenTrust [13] assigns a global trust rating to each peer similar to how Google's PageRank [16] ranks web pages. Global ratings are then used by peers to exclude untrustworthy peers (which send inauthentic files) and to select peers from whom to download files. As a consequence, the number of inauthentic files in the network decreases. In a free software developer community, Advogato [15] assigns a global trust to each community member. It does so by arranging ratings in a web of trust and by composing ratings between members using max flow from designated trusted members. The idea of max flow is that between any two nodes, the quantity of trust flowing from one node to another cannot be greater than the weakest rating somewhere on the path between the two nodes. This way of composing ratings has proved to be attack resistant - it successfully isolates unreliable members. More recently, Ziegler and Lausen [24] proposed to rank all users by spreading "activation models" (by arranging ratings in a matrix and finding the principal eigenvector), while Dell'Amico [7] focused on peer-to-peer networks and proposed to rank peers by using link-analysis techniques in a fully distributed setting.

By contrast, in the second literature camp, techniques assign a pairwise (*local*) trust rating to each pair of users. That is, A assigns a personalized trust rating in B. Personalizing trust is beneficial because it makes it possible for two individuals to have different opinions about the trustworthiness of the same person (which may well happen in reality). In 2003, Golbeck *et al.* [9] proposed different algorithms for propagating trust in this way. For example, they proposed a variation of max flow that accounts for path length. To Domingos *et al.* [20] and Guha *et al.* [11] goes the merit of presenting the first comparative studies of different trust propagation algorithms in which pairwise ratings are computed. These algorithms have been evaluated against Epinions [1], a large collection of binary ratings[2]. By binary we mean that each rating simply expresses whether an individual trusts another individual or not. Despite being evaluated on binary ratings, these algorithms are general in the sense that they can take discrete ratings (not necessarily binary).

Most of the work on assigning pairwise trust ratings is based on a simple, yet effective mechanism: A finds all paths leading to B; for each path, A then concatenates the ratings along the path; A finally aggregates all path concatenations into a single trust rating for B. Algorithmically, this is equivalent to A arranging trust ratings into a matrix and, over a series of iterations, propagating trust by, for example, *direct* propagation: if A trusts C and C trusts B, then trust propagates from A to B. The resulting matrix values are then rounded into a single trust rating. Unfortunately, this way of propagating trust suffers from two main limitations:

- Literature has proved direct trust propagation to be extremely effective, but it has done so only on data sets of *binary* ratings. There is no published work on how

[1]For simplicity's sake, our examples use a trust scale $\{1, 2, 3\}$. However, our model by no means requires this as it works on any scale.

[2]In Epinions, quality assessments of products are on a scale $\{1, \ldots, 5\}$, but user ratings are binary.

direct propagation would perform on a *large* data set of *discrete* ratings, not necessarily binary[3]. An individual may express whether she trusts another individual or not, and, if she does, she may then express the extent to which she trusts by a discrete value. For example, in Advogato [15], users express their trust with 3-level ratings (ratings in $\{1,2,3\}$). Given that, one may wonder how direct trust propagation would perform against the Advogato data set. We have evaluated direct trust propagation on Advogato as Section 4 will describe. Unfortunately, as Fig. 2 shows, the predictive accuracy (fraction of correct predictions) is low. More precisely, considering a large sample of Advogato's trust ratings (55.455 ratings), direct trust propagation correctly predicts 50.76% of the ratings (as a reference, consider that a naive prediction (random guess) correctly predicts 31.1% of the ratings[4]). Therefore, direct trust propagation does not predict the actual rating in roughly half of the cases.

- Direct trust propagation does not scale on mobile devices. Direct trust propagation is meant for Web applications in which centralized servers store full webs of trust upon which trust is then propagated by multiplying vectors and matrices whose dimensions are extremely high. As a consequence, it is computationally expensive and would not scale well on any existing portable device. Moreover, mobile devices would only know a very small subset of the web of trust at any given time (it is unrealistic to assume complete knowledge) because of, for example, network partition, device (un)availability, and limited resources.

It thus seems that a different way of propagating trust in mobile computing scenarios is needed. But what sort of method should we use?

2.3 Our Proposal

Our problem is to find a way of propagating trust that is both *effective* and *scalable*. To do so, we propose to use a class of semi-supervised learning techniques that have been proved to be effective when applied to various problem domains (e.g., to predict movie reviews [10], to recognize digits [12], to classify text [23]).

These techniques work on a graph in which: not links but *nodes* are either rated or unrated, and those nodes are then connected to each other if they are *related* (the techniques consider that *two nodes are related if their ratings are similar*). Informally, these techniques exploit knowledge already present in the graph (rated nodes) to construct a function that is capable of predicting unrated nodes. To choose the most *effective* function (the function with the highest predictive accuracy), the techniques impose that: on input of each rated node in the graph, the chosen function returns the node's actual rating (this serves to choose a function that is *consistent* with existing ratings); given two connected nodes x_i and x_j, the chosen function returns $f(x_i)$ similar to $f(x_j)$ (this serves to choose a function that assigns similar ratings to connected nodes).

Let us now imagine a graph whose nodes are trust relationships (we call this graph *'relationship graph'*). A node is rated if the corresponding relationship in the web of trust is known and rated (e.g., in Fig. 3(a), $C \rightarrow B$ is known and rated, so that in the relationship graph the node representing this relationship will be rated). By contrast, a node is unrated if the corresponding relationship is unknown and has to be predicted (e.g., $A \rightarrow B$ in Fig. 3(a)). Predicting a trust relationship thus means finding a function that effectively rates the corresponding node. Finding such a function is exactly the problem solved by the semi-supervised learning techniques described above. In order to exploit these techniques, we first need to represent (part of) a web of trust as a relationship graph. We explain how to do so in the following section.

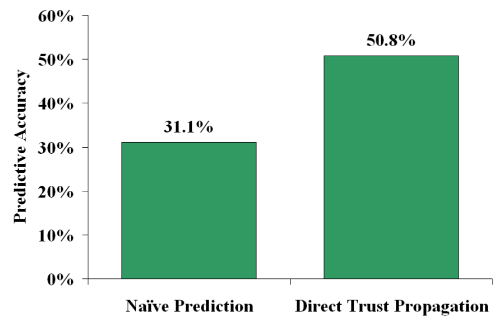


Figure 2. Predictive Accuracy. The fraction of correct predictions for naive prediction (random guess) and direct trust propagation.

3 The Proposed Model

To describe our model, we refer to our running example of how A may propagate its trust in B given the web of trust shown in Fig. 3(a). The following four steps are required, each of which is described in the following subsections:

1. A determines the trust relationships that our propaga-

[3]One may point out the existence of PGP's web of trust [25]. Unfortunately, that web of trust is not suitable for what we call trust propagation (i.e., computation of *single* pairwise ratings). On that web of trust, it makes only sense to *combine* pairwise ratings (as PGP does) - each of these ratings cannot be propagated but must be set by a user. Any research on PGP (including that in mobile settings) solves a problem different from ours, i.e., how to make it possible to combine (not to compute) pairwise ratings.

[4]Its predictive accuracy is 31.1% and not 33% because ratings are not uniformly distributed across the three possible discrete values.

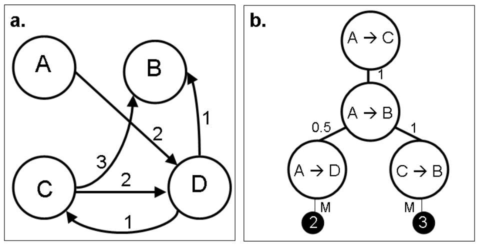


Figure 3. (a) **A web of trust and** (b) **the corresponding relationship graph for predicting** $A \rightarrow B$**'s rating.**

tion scheme may find relevant for predicting A's trust in B (Section 3.1).

2. A restricts its attention to the subset of the web of trust that it knows and that includes those relationships (Section 3.2).

3. From this subset, A builds a relationship graph (Section 3.3).

4. A finally applies the machine learning technique to determine a function that predicts $A \rightarrow B$ (Section 3.4).

Before describing these steps in details, we spell out our *assumptions*:

- We opt to target a class of applications in which users do not maliciously modify ratings. That is, for example, the case for the Advogato community - most users rate themselves (they judge their own expertise in developing software) with the lowest level of trust. More generally, this assumption holds for all applications whose users share a value system (e.g., connecting with other people, creating an online identity, expressing oneself, etc.) similar to the one that has recently contributed to the success of Wikipedia, open source software, tagging systems, and many other technologies reported in a recent book by Tapscott and Williams [21]. However, there are applications in which users may maliciously modify their ratings. To tackle this problem, we are currently studying whether a distributed hash table on top of a mobile ad-hoc network may store user ratings and prevent their malicious modification.

- We consider the web of trust to be a network in which the small world phenomenon holds (i.e., the distance between any two people is short). We argue this is a perfectly plausible assumption given that the web of trust is a social network, for which the small world phenomenon emerges.

- We consider that anonymous tokens identify users. This may protect user anonymity but makes the system vulnerable to Sybil attacks (whose impact is discussed in Section 5).

3.1 Determining Relevant Trust Relationships

To begin with, A determines the trust relationships that our propagation scheme may find relevant for predicting A's trust in B, that is, those relationships *related* to $A \rightarrow B$. As defined in Section 2.3, *two relationships are related if their ratings are similar*. Hence we consider related any:

- *Two relationships with the same rater.* For example, $A \rightarrow B$ and $A \rightarrow D$ are related as they have the same rater A, and the more alike B and D *perform*, the more related $A \rightarrow B$ and $A \rightarrow D$ are. We call this kind of relation *performing relation*, and denote it as $rel_p(B,D)$. More formally, the weight between the i^{th} trust relationship "A trusts B" and the j^{th} trust relationship "A trusts D" is

$$w_{ij} = c \cdot rel_p(B,D), \qquad (1)$$

where c is the weight given to performing relations. How do we quantify the performing relation between B and D? We may do so by considering that the more alike B and D perform, the closer the ratings about them. Along these lines, we: *(1)* take all people who have rated both B and D. In our example (Fig. 3(a)), this is only C. *(2)* For each such person, compute the absolute difference between her rating in B and that in D. In our example, the absolute difference between $C \rightarrow D$ and $C \rightarrow B$ is $3 - 2 = 1$. *(3)* Normalize those differences in $[0,1]$, and aggregate them. We will shortly describe two ways of aggregating differences. For the time being, consider that, in our example, there is only one person who has rated both B and D, thus only one difference (so no need for any aggregation). Given that ratings are given in a discrete scale $[1,3]$, the normalized difference value is $(3-2)/(3-1) = 1/2$. Having the aggregated value, we may then say that the bigger this is, the less alike B and D perform (the lower $rel_p(B,D)$). For this reason, we define $rel_p(B,D) = 1 - v$, where v is the aggregated value. In our example, we had only one difference so we did not need any aggregation at the third step. In general, we may have more than one person who has rated both B and D, thus more than one difference (for example, let us assume we have two normalized differences $1/2$ and $2/2$). In that case, we need to aggregate those differences. The simplest way to do so is to compute the average difference (in our

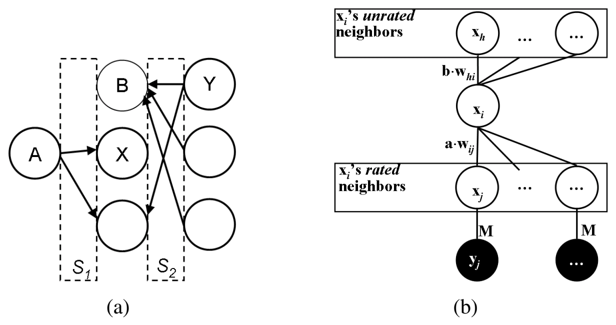


Figure 4. (a) A's view of the web of trust. (b) A schematic representation of a relationship graph. On that graph, our algorithm predicts x_i's rating.

example, that is $\frac{1/2+2/2}{2} = 3/4$). However, whether the average is a good choice depends on the number of the differences across which we average and their variance. The average is a good choice when there are many differences whose variance is very low. However that is not necessarily true. Therefore, we consider a second way of aggregating that accounts for both number of differences and their variance: we compute the confidence interval of the average (with 95% of confidence) and take the higher extreme. We will evaluate which aggregation leads to the highest predictive accuracy in Section 4.1.

- *Two relationships in which the same person is rated.* $A \rightarrow B$ and $C \rightarrow B$ are an example, as they both rate B. People may differ in the way they rate. We thus have to compute a *judging relation* for which, the more alike two raters (A and C) have judged the same person (B), the higher their relation. We denote this as $rel_g(A, C)$. The corresponding weight between the two relationships is

$$w_{ij} = d \cdot rel_g(A, C), \qquad (2)$$

where d is the weight given to judging relations. To compute $rel_g(A, C)$, we take all persons who have been rated by both A and C; for each person, we compute the difference between A's and C's ratings; finally, we aggregate all differences. Again, we aggregate by computing the average difference and its confidence interval.

3.2 Taking Part of the Web of Trust

To use our propagation scheme for predicting its trust for B, A needs to know part of the web of trust. To see which part is needed, we must consider the two steps through which A predicts its trust for B. In so doing, we will refer to Fig. 4(a) and consider the following sets: S_1, that is, the set of ratings of A's outgoing relationships; and S_2, that is, the set of all ratings from nodes that have rated B.

Step 1. A determines the trust relationships related to $A \rightarrow B$. Those relationships take the form $A \rightarrow X$ and $Y \rightarrow B$ (where X and Y are generic persons different from A and B). To determine those relationships, A needs the ratings of its outgoing relationships (set S_1) plus the ratings of B's incoming relationships (relationships in S_2 going into B).

Step 2. A determines the extent to which each pair of those edges are related. More specifically:

- A determines the extent to which any pair of edges $A \rightarrow X$ and $A \rightarrow B$ are related. This requires to quantify the (performing) relation between B and X — to quantify how alike B and X perform. To do so, A needs the ratings of the relationships in S_2 plus those in S_1.
- A determines the extent to which any pair of edges $Y \rightarrow B$ to $A \rightarrow B$ are related. This requires to quantify the (judging) relation between Y and A — to quantify how alike Y and A rate. Again, to do so, A needs the ratings of the relationships in S_2 plus those in S_1.

Overall, to use our propagation scheme for predicting its trust in B, A needs the ratings of the relationships in S_1 and in S_2. The former are readily available, as we can assume that A stores locally the ratings it has produced. And the latter are received from B. In fact, those ratings have been generated by the people who have rated B; therefore, B may have all the ratings of the relationships in S_2, as it has received them from its raters. As a result, each user stores a very small subset of the web of trust — she stores the ratings she generates plus those generated by her raters.

3.3 Building the Relationship Graph

At this point, A knows which trust relationships are related to the one that has to be propagated, and the extent to which they are so; A can then build a relationship graph. The key idea is to create a graph whose nodes include the trust relationship to be predicted plus related relationships. Fig. 3(b) shows one such graph whose nodes are: $A \rightarrow B$ (trust relationship to be predicted), $A \rightarrow C$, $A \rightarrow D$, and $C \rightarrow B$ (related relationships). Nodes are linked and the label on a link expresses the extent to which the two linked nodes are related.

More generally, there are n trust relationships $x_1, \ldots, x_n$, of which r are rated $(x_1, y_1), \ldots, (x_r, y_r)$

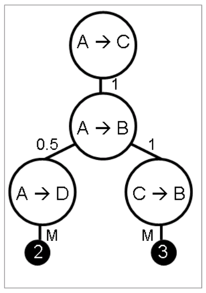
and u are unrated $x_{r+1}, \ldots, x_{r+u}$ (an 'unrated' relationship is a relationship that has no rating on the web of trust). The numerical ratings are defined as being $y_1, \ldots, y_r \in L$, where $L = \{l_1, \ldots, l_p\}$ with $l_1 < \ldots < l_p$. For example, a system with $p = 3$ possible rating levels may have $L = \{1, 2, 3\}$. Our problem is now to build a connected graph $G = (V, E)$ with nodes V corresponding to the n trust relationships. We do so step by step with reference to an example (Fig. 3(b)) and to a general representation (Fig. 4(b)). To understand the rationale behind this construction, we must remember our end goal of finding a predictive function $f : V \rightarrow \mathbb{R}$ on G capable of assigning ratings to unrated nodes (note that f assigns a real value to a trust relationship; this value will then be mapped to the nearest discrete rating in L). In particular, let x_i be the trust relationship we wish to predict (e.g., $A \rightarrow B$ in Fig. 3(b)) among the unrated ones.

- The node x_i is connected to its k most related nodes that are *rated*. We call these nodes x_i's rated neighbors. Connecting x_i to those nodes serves to impose that the function f rates x_i and its rated neighbors with similar ratings. Let x_j denote one of x_i's rated neighbors. The weight of the edge between x_i and x_j is $a \cdot w_{ij}$. The coefficient a is the weighting factor for rated neighbors, and w_{ij} is determined as per formula (1) or (2) (depending on whether the relation between x_i and x_j is performing or judging relation). For example, in Fig. 3(b), $A \rightarrow B$ is connected to $A \rightarrow D$ and to $C \rightarrow B$ and the corresponding weights are 0.5 and 1, respectively.

- The node x_i is also connected to the k' most related nodes that are *unrated* (e.g., in Fig. 3(b), $A \rightarrow B$ is connected to $A \rightarrow C$). We call these nodes x_i's *unrated* neighbors. Connecting x_i to those nodes serves to impose that the function f rates x_i and its unrated neighbors with similar ratings. Let x_h denote one of those unrated nodes. The weight of the edge between x_h and x_i is $b \cdot w_{hi}$. The coefficient b is the weighting factor for unrated neighbors, and w_{hi} is determined as per formula (1) or (2).

- Each rated node x_j is connected to an "observed node" (dark circles in both reference figures) whose value is the rating y_j. The observed node is a 'dongle' because it connects only x_j. The edge weight is a large number M. Setting M to a large number serves to pull $f(x_j)$ towards the true rating y_j (in particular, if $M \rightarrow \infty$ then $f(x_j) = y_j$). That corresponds to the first condition for choosing an effective function f (Section 2.3): making the function consistent with existing ratings.

The coefficients in a relationship graph are thus M, a, b, k, and k'. M is set to an arbitrary large number (10^6).



$$y = \begin{bmatrix} 2 \\ 3 \\ 0 \\ 0 \end{bmatrix} \quad C = \begin{pmatrix} M & 0 & 0 & 0 \\ 0 & M & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\bar{W} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.5 & 1 & 1 & 0 \end{pmatrix} \quad f = \begin{bmatrix} 2 \\ 3 \\ 2.7 \\ 2.7 \end{bmatrix}$$

Figure 5. A relationship graph and corresponding matrices. In the relationship graph, there are two rated nodes $x_1 : A \rightarrow D$ and $x_2 : C \rightarrow B$; and two unrated nodes $x_3 : A \rightarrow C$ and $x_4 : A \rightarrow B$. The prediction algorithm populates the rating matrix $\mathbf{y}$, the diagonal dongle matrix C, and the weight matrix $\bar{W}$. Then, it computes the predictive function $\mathbf{f}$.

The remaining coefficients will be set by cross validation (Section 4.1).

3.4 Finding a Predictive Function on the Graph

Having the relationship graph, A now has to find a function that predicts all unrated nodes in that graph, including that of interest (e.g., $A \rightarrow B$).

Let us formalize the problem and the construction of its solution. We have a graph G of n nodes $x_i, i \in [1, n]$, r of which have ratings y_i, and the remaining u $(x_{r+1}, \ldots, x_{r+u})$ are unrated. The set R contains the indices of the *rated* nodes and U those of unrated nodes. Our problem is to seek a function f that rates each of the *unrated* nodes (the nodes whose indices are in U). Section 2.3 mentioned that to choose a function that rates *effectively*, one has to impose that: on input of each rated node in the graph, the chosen function returns the node's actual rating (that has been done in the previous Section 3.3 by setting the coefficient M to a large number); on input of either of two connected nodes, the chosen function's outputs are similar.

The latter condition is equivalent to saying that (refer to Fig. 4(b)): for any pair of related nodes x_i and x_j, the difference of their ratings $f(x_i)$ and $f(x_j)$ should be minimum. In other words, $(f(x_i) - f(x_j))^2$ should be minimum. This expression represents the rating difference *over one edge*. One may compute the difference *over the graph* by summing the rating differences of all edges. We denote this difference as $\mathcal{L}(f)$. We will see that such a difference depends on the chosen function f. Our problem is to find the function f for which the difference over the graph is minimum. We do so in Appendix A and find that such a function

Factor	Description	Tuning Range
$f_r = \frac{k}{\vert R\vert}$	Fraction of rated nodes used as neighbors.	$\{0.05, 0.1, 0.15, 0.2, 0.3\}$
$f_u = \frac{k'}{\vert U\vert}$	Fraction of unrated nodes used as neighbors.	$\{0.05, 0.1, 0.15, 0.2, 0.3\}$
$\beta = \frac{b}{a}$	Relative weight between rated and unrated nodes.	$\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$
$\gamma = \frac{d}{c}$	Relative weight between judging and performing relations.	$\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$
α	How to compute $rel(A,B)$, i.e., the extent to which A and B perform or judge alike.	Two ways: average difference or confidence interval

Table 1. Parameters of our prediction algorithm. To find the optimal values, we tune the parameters in the reported ranges.

is $\mathbf{f} = \Big(C + L\Big)^{-1} C\mathbf{y}$. Let us now introduce this notation and the three steps of the algorithm for computing **f** (refer to Fig. 5):

1. Populate 3 matrices:
   - The rating matrix $\mathbf{y} = (y_1, \ldots, y_r, 0, \ldots, 0)^\top$. This is an $n \times 1$ matrix whose first r rows contain the ratings of the rated nodes, and any remaining row contains zero. For example, Fig. 5 shows a relationship graph and the corresponding rating matrix.
   - The diagonal dongle weight matrix C. The elements that correspond to rated nodes contain M (otherwise 0):
   $$C_{ii} = \begin{cases} M, & i \in R \\ 0, & i \in U \end{cases}$$
   In Fig. 5, the first two nodes are rated.
   - The $n \times n$ weight matrix $\bar{W}$:
   $$\bar{W}_{ij} = \begin{cases} aw_{ij}, & j \in RN(i) \\ bw_{ij}, & j \in UN(i) \\ 0, & \text{otherwise} \end{cases}$$
   $RN(i)$: set of indices of i's *rated* neighbors;
   $UN(i)$: set of indices of i's *unrated* neighbors.
2. Compute: the symmetrized version of $\bar{W}$: $W = max(\bar{W}, \bar{W}^\top)$; the diagonal degree matrix $D_{ii} = \sum_{j=1}^{n} W_{ij}$ (we consider a node's degree to be the sum of its edge weights); and the combinatorial Laplacian matrix $L = D - W$.
3. Finally, compute $\mathbf{f} = \Big(C + L\Big)^{-1} C\mathbf{y}$. This has the form $\mathbf{f} = (f(x_1), \ldots, f(x_n))^\top$, and its last u elements are the predicted ratings for the nodes in U. Appendix A shows that the so computed **f** minimizes the rating difference over the relationship graph. For example, in the relationship graph of Fig. 5, $A \rightarrow B$'s rating is predicted to be $f_4 = 2.7$.

That concludes the description of our model. In the next section, we turn to evaluating it.

4 Evaluation

The goal of our algorithm is to predict trust ratings on portable devices. To ascertain the effectiveness of our algorithm at meeting this goal, our evaluation ought to answer three questions:
(1) *(Predictive Accuracy)* How accurate is our algorithm in predicting trust ratings?
(2) *(Prediction Robustness)* What is the impact of uncooperative users upon the algorithm's accuracy?
(3) *(Overheads)* What time, storage, and communication overheads does our algorithm impose on a mobile phone?

To see whether our algorithm effectively predicts trust and whether it is usable on portable devices, we need a *large-scale deployment*. Only so can we separate statistical significant answers from plausible insights gained by a small-scale deployment. Plus, a deployment needs to be evaluated in the *long-term* to see whether our algorithm is robust against, for example, uncooperative users.

Unfortunately, we do not have a long-term evaluation of a large-scale mobile computing deployment. We do, however, have a large rating data set from the Advogato community that has been around for more than a decade[5]. Using this data set (described next), we evaluate whether our algorithm is effective in predicting real trust ratings (Section 4.1). Then, to evaluate how robust our algorithm is, we emulate how users may rationally turn to be uncooperative (Section 4.2). Finally, we implement our algorithm to assess whether it is usable on a mobile phone (Section 4.3).

To begin with, let us describe the Advogato data set. Advogato is a community discussion board for free software developers. Using the Advogato's trust metric [15], each

[5]Since webs of trust tend to contain private information, only a limited number are publicly available, and those few are quite small (they do not allow for thorough evaluation). The only exception is the Advogato data set.

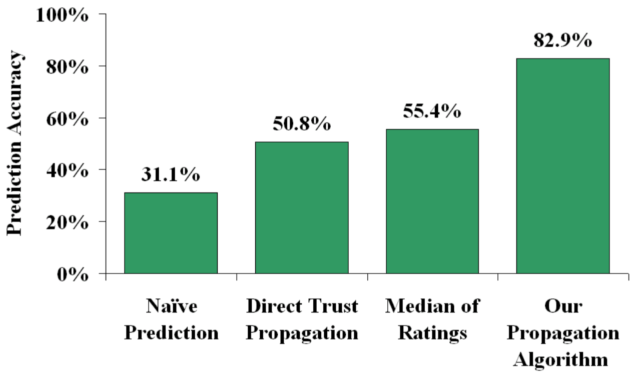


Figure 6. Predictive accuracy of four algorithms.

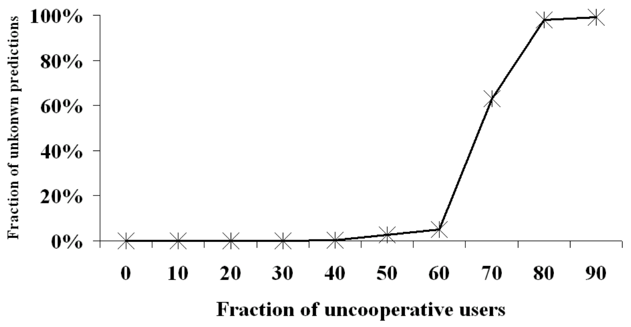


Figure 7. Fraction of unknown predictions as a function of uncooperative users (users who are not willing to make their ratings available).

user has a single (global) trust value computed by composing other users ratings. There are three possible ratings: apprentice, journeyer, and master. Global trust is used to control access to the discussion board: 'apprentices' can only post comments, whereas 'journeyers' and 'masters' are able to post both stories and comments. From this community, we have extracted 55,455 trust relationships.

4.1 Predictive Accuracy

We evaluate the predictive accuracy of our algorithm by using leave-one-out cross validation.

Validation Execution. The cross validation unfolds as follows. We take Advogato's web of trust. We mask one trust relationship and then predict the relationship's rating in four different ways. We repeat this on all relationships. In doing this, we measure the *predictive accuracy*, i.e., the fraction of correct predictions.

The four ways of predicting the rating of a masked relationship $A \rightarrow B$ that we have compared are: *naive prediction* (random guess); *median of ratings* about B; *direct trust propagation* (as described in Section 2.2); and *our algorithm*. Since we cannot assume complete knowledge in mobile settings, we consider that our algorithm does not know the whole web of trust, but predicts $A \rightarrow B$ on input of only the ratings known by A (as described in Section 3.2). Instead, we allow distributed trust propagation to know all paths (and corresponding ratings) between A and B because it is the only way it can be carried out.

Our propagation algorithm has five parameters. We tune each parameter in the range shown in Table 1. This leads to 1250 possible combinations. For each combination, we compute the predictive accuracy.

Validation Results. For our algorithm, we first computed the optimal parameters (parameters for which the accuracy is highest) as described above: they turn out to be $\gamma = 1$, $\alpha =$"confidence interval", $f_r = 0.1$, $f_u = 0.2$, $\beta = 0.1$, and $k = k' = 20$ (a relationship graph should contain at most 20 edges). We expect that these values will apply in other domains; that is because they, informally speaking, specify reasonable and intuitive choices. More specifically, they indicate that to predict the rating of a trust relationship, the learning algorithm has: to consider *relevant* those relationships that include people who perform or rate alike ($\gamma = 1$); to estimate relevance by aggregating ratings using the confidence interval of their average ($\alpha =$"confidence interval"); to consider a small fraction of those relevant relationships ($f_r = 0.1$, $f_u = 0.2$); to weight the *actual* ratings more than the ratings it predicts during the learning process ($\beta = 0.1$).

This preliminary analysis allow us to move on to answer the key question: how would the predictive accuracy of our algorithm compare to that of any of the algorithms previously mentioned. Fig. 6 shows that direct trust propagation performs better than naive prediction, but is comparable to a median of ratings. It also shows that our algorithm's accuracy is as high as 82.9%. In all cases in which our algorithm failed to predict (17.1%), the actual rating and the predicted rating differed by one only (with $L = \{1, 2, 3\}$).

4.2 Prediction Robustness

All trust propagation techniques rely on knowing ratings. In mobile computing, this translates into users making available their ratings. For privacy reasons, some users may well decide not to do so. Being this plausible, we now evaluate how our algorithm would cope if different fractions of users did not disclose their ratings. Again, we measure predictive accuracy by cross validation: we mask one trust relationship and then predict its rating upon the limited knowledge of the nodes subject of the trust relationship; we do so for all relationships that link any pair of cooperative users (users willing to make their ratings available). That is because we consider that users willing to be subject to prediction are also willing to cooperate.

Fig. 7 shows the fraction of predictions for which a relationship graph is not defined (percentage of unknown predictions) as a function of the fraction of uncooperative users. If at most 60% of the users are not willing to make their ratings available, the remaining users can still propagate their trust, and they do so with a high predictive accuracy (82.9%). However, as Fig. 7 shows, if the number of uncooperative users reaches a critical point (if it is higher than 60%), the remaining users are abruptly unable to form a relationship graph. In other words, if at least 40% of the users make their ratings available, those users can still effectively propagate trust. For one possible explanation of this result, consider that the web of trust is a social network and that social networks are robust because they are scale-free. Albert *et al.* [2] studied the fraction of nodes that must be removed at random from a scale-free network to break it into pieces: they "removed as many as 80% of all nodes and the remaining 20% still hung together, forming a highly interlinked cluster" [3][6].

4.3 Overheads

Communication and Storage Overheads. Both communication and storage overheads are minimal. As described in Section 3.2, any device stores the ratings of its outgoing relationships plus those of its incoming neighbors in a table. Each tuple of this table corresponds to a trust relationship, i.e., to two identifiers (of the connected persons) and one rating. Hence, say that the size of a tuple is roughly 10B. Even with 50 incoming and 50 outgoing edges (which is pessimistically high), the table size is 30KB. Also, for a single trust propagation, the data to be sent is less than 30KB.

Computational Overhead. We ran a J2ME implementation of our algorithm on a Nokia 3230 mobile phone whose features include: Symbian operating system 7.0, 32 MB of memory, 32-bit RISC CPU (123 MHz). In Section 4.1, we evaluated that a relationship graph should contain 20 nodes at most. We run our algorithm in this worst case scenario. We minimized background activities by shutting down all applications other than our algorithm. The computation overhead, given as the mean of 10 runs, is as low as 2.8 milliseconds.

[6]Our scheme does not work if more than 60% of the nodes are removed. The reason is that, to propagate trust from A to B, our scheme does not simply need a connecting path between A and B (which may likely exist even if 80% of the nodes are removed), but needs the set of links that are necessary for propagating trust (as per Section 3.2). And those links likely disappear after removing more than 60% of the nodes.

5 Discussion

Based on the previous results, we now discuss some open questions.

Privacy Concerns. By exchanging their web of trust, users reveal their social ties (people with whom they have interacted), and some users may not feel comfortable doing so for privacy concerns [14]. Our design alleviates these concerns for two reasons. First, users are identified by anonymous tokens. Second, one inherent property of distributed trust propagation is that users' ratings are not made available on public servers, but each user discloses her ratings whenever she finds convenient to do so. Moreover, during our evaluation, we found that if at least 40% of the users make available their ratings, those users can still propagate their trust without relying on any other user (Section 4.2).

Sybil Users. Given that users' identifiers correspond to anonymous tokens, one may rightly point out that user-token bindings need to be certified to avoid sybil attacks [8], in which a malicious user takes on multiple identities and pretends to be multiple, distinct users. In mobile computing, user-token bindings cannot be certified by a central authority. However, those bindings may be statistically guaranteed by mechanisms similar to SybilGuard [22]. If those mechanisms will prove ineffective, one might be interested in knowing whether our model is robust against sybil attacks. We evaluate the predictive accuracy of our algorithm as follows: we mask one trust relationship $A \rightarrow B$; we create n sybil identities who highly rate B; we then predict $A \rightarrow B$'s rating. We do so for all trust relationships. Regardless of n, the prediction accuracy remains unchanged (82.9%). The reason for this result is that sybil users are not connected in the same way as real users are and, as a consequence, their ratings do not influence trust propagation.

Distrust. Previous work has shown that introducing distrust may be beneficial to trust propagation [11]. Despite not explicitly modeling distrust, we may introduce it by simply adding an additional rating level. For example, if we have 3 possible ratings, we may simply add a fourth rating representing distrust.

Dynamic Ratings. Since ratings may change over time, we have allowed for rating tables to be updated either reactively or proactively and the storage and communication overheads of doing so are minimal (Section 4.3).

6 Conclusion

We proposed a model that makes it possible for mobile users to predict their trust for content producers from whom they have never received content before. The model scales (it entails minimal storage and communication overhead) and is effective (its predictive accuracy on a large data set is as high as 82.9%). That accuracy remains unchanged even if most of the users were not to make available their ratings. The model also runs on portable devices (a J2ME implementation spends at most 2.8ms for one propagation on a Nokia phone). To further evaluate our model, we are currently designing controlled experiments to be run in a large-scale deployment.

Acknowledgments: We thank Mark Herbster for explaining us different ways of learning over graphs. We also thank Matteo Dell'Amico, Neal Lathia, Ilias Leontiadis, Paolo Garza, Tania Cerquitelli, and the anonymous reviewers for their contributions, and Microsoft Research Cambridge for its financial support.

References

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

[2] R. Albert, H. Jeong, and A. L. Barabasi. Error and attack tolerance of complex networks. *Nature*, July 2000.

[3] A.-L. Barabasi. *Linked: How Everything Is Connected to Everything Else and What It Means*. Penguin, 2003.

[4] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava. Participatory sensing. In *Proc. of ACM WSW*, 2006.

[5] I. Calvino. *Six Memos for the Next Millennium*. Harvad University Press, 1996.

[6] L. Capra. Engineering human trust in mobile system collaborations. In *Proc. of FSE*, 2004.

[7] M. Dell'Amico. Neighbourhood Maps: Decentralised Ranking in Small-World P2P Networks. In *Proc. of HotP2P*, 2006.

[8] J. R. Douceur. The Sybil Attack. In *Proc. of IPTPS*, 2002.

[9] J. Golbeck, B. Parsia, and J. Hendler. Trust Networks on the Semantic Web. In *Proc. of CoopIS*, 2003.

[10] A. Goldberg and X. Zhu. Seeing stars when there aren't many stars: Graph-based semi-supervised learning for sentiment categorization. In *Proc. of Textgraphs*, 2006.

[11] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of trust and distrust. In *Proc. of ACM WWW*, 2004.

[12] M. Herbster, M. Pontil, and L. Wainer. Online learning over graphs. In *Proc. of ICML*, 2005.

[13] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. In *Proc. of ACM WWW*, 2003.

[14] N. Lathia, S. Hailes, and L. Capra. Private Distributed Collaborative Filtering using Estimated Concordance Measures. In *Proc. of ACM RecSys*, 2007.

[15] R. Levien and A. Aiken. Attack-resistant trust metrics for public key certification. In *Proc. of USENIX Security*, 1998.

[16] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford University, 1998.

[17] D. Quercia, S. Hailes, and L. Capra. B-trust: Bayesian Trust Framework for Pervasive Computing. In *Proceedings of iTrust*. LNCS, 2006.

[18] D. Quercia, S. Hailes, and L. Capra. TRULLO - local trust bootstrapping for ubiquitous devices. In *Proc. of IEEE Mobiquitous*, 2007.

[19] H. Rheingold. *Smart Mobs: The Next Social Revolution*. Perseus Books Group, 2002.

[20] M. Richardson, R. Agrawal, and P. Domingos. Trust Management for the Semantic Web. In *Proc. of ISWC*, 2003.

[21] D. Tapscott and A. D. Williams. *Wikinomics: How Mass Collaboration Changes Everything*. Portfolio, Penguin, 2006.

[22] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: defending against sybil attacks via social networks. In *Proc. of ACM SIGCOMM*, 2006.

[23] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proc. of ICML*, 2003.

[24] C.-N. Ziegler and G. Lausen. Spreading activation models for trust propagation. In *Proc. of IEEE EEE*, 2004.

[25] P. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.

Appendix A

Given the graph described in Section 2 and shown in Fig. 4(b), the loss $\mathcal{L}(f)$ over the whole graph is

$$\begin{aligned}\mathcal{L}(f) &= \sum_{j\in R} M\cdot(f(x_j)-y_j)^2 + \\ &+\sum_{i\in U}\sum_{j\in RN(i)} a\cdot w_{ij}\cdot(f(x_i)-f(x_i))^2 + \\ &+\sum_{i\in U}\sum_{h\in UN(i)} b\cdot w_{hi}\cdot(f(x_h)-f(x_i))^2 \quad (3)\end{aligned}$$

where:

R: set of indices of *rated* nodes;
U: set of indices of *unrated* nodes;
$RN(i)$: set of indices of i's *rated* neighbors;
$UN(i)$: set of indices of i's *unrated* neighbors.

Expression (3) can be written as:

$$\mathcal{L}(f) = (\mathbf{f}-\mathbf{y})^\top C(\mathbf{f}-\mathbf{y}) + \mathbf{f}^\top L\mathbf{f} \quad (4)$$

Being C and L symmetric, the gradient is:

$$\frac{\delta\mathcal{L}(f)}{\delta\mathbf{f}} = 2C(\mathbf{f}-\mathbf{y}) + 2L\mathbf{f}. \quad (5)$$

To solve the optimization problem $min_f\mathcal{L}(f)$, we set the gradient to zero, $\frac{\delta\mathcal{L}(f)}{\delta\mathbf{f}} = 0$, and obtain:

$$\mathbf{f} = \Big(C+L\Big)^{-1} C\mathbf{y} \quad (6)$$