Measuring and Extracting Proximity Graphs in Networks

YEHUDA KOREN, STEPHEN C. NORTH, and CHRIS VOLINSKY
AT&T Labs – Research

Measuring distance or some other form of proximity between objects is a standard data mining tool. Connection subgraphs were recently proposed as a way to demonstrate proximity between nodes in networks. We propose a new way of measuring and extracting proximity in networks called "cycle-free effective conductance" (CFEC). Importantly, the measured proximity is accompanied with a *proximity subgraph* which allows assessing and understanding measured values. Our proximity calculation can handle more than two endpoints, directed edges, is statistically well behaved, and produces an effectiveness score for the computed subgraphs. We provide an efficient algorithm to measure and extract proximity. Also, we report experimental results and show examples for four large network datasets: a telecommunications calling graph, the IMDB actors graph, an academic coauthorship network, and a movie recommendation system.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*Data mining*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*

General Terms: Algorithms, Human Factors

Additional Key Words and Phrases: Connection subgraph, cycle-free escape probability, escape probability, graph mining, proximity, proximity subgraph, random walk

ACM Reference Format: Koren, Y., North, S. C., and Volinsky, C. 2007. Measuring and extracting proximity graphs in networks. ACM Trans. Knowl. Discov. Data 1, 3, Article 12 (December 2007), 30 pages. DOI=10.1145/1297332.1297336 http://10.1145/1297332.1297336

1. INTRODUCTION

Networks convey information about relationships between objects. Consequently, networks successfully model many kinds of information in fields ranging from communications and transportation to organizational and social domains. This has lead to extensive research on searching and analyzing graphs.

Measuring distance, or some other form of proximity or "closeness" between two objects, is a standard data mining tool. It may be applied directly to compare

Authors' addresses: Y. Koren (contact author), S. C. North, and C. Volinsky, AT&T Labs – Research, 180 Park Ave., Florham Park, NJ 07932; email: yehuda@research.att.com.
Permission to make digital or hard copies part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from the Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2007 ACM 1556-4681/2007/12-ART12 $5.00. DOI 10.1145/1297332.1297336 http://doi.acm.org/10.1145/1297332.1297336

similarities between items, or within a more general scheme such as clustering and ordering. Moreover, measuring proximities can help to characterize the global structure of a network by showing how closely coupled it is. However, while proximity measurement is well understood for attribute-based, multivariate data, the situation is not as clear for objects in networks. Measuring proximities between entities or groups of entities in networks is an important and interesting task. Proximities can predict connections in a social network which have not been made yet [Liben-Nowell and Kleinberg 2003; Popescul and Ungar 2003]. In a network with missing data, proximities potentially give evidence of links that have been removed or cannot be observed. Another popular way of using proximities is to find clusters or communities of entities in a network that behave similarly or have some commonality [Flake et al. 2000; Gibson et al. 1998].

Measuring proximity in networks is inherently more difficult and globally oriented than it is with typical multivariate data. To calculate proximity for network objects, we must account for multiple and disparate paths between the objects. Unlike proximity measurement in multivariate data, which usually relies on a direct comparison of two relevant objects, calculating network proximity requires working with the network that lies between the two objects. Therefore, explaining network proximity involves showing significantly more information. Recent work by Faloutsos et al. [2004] introduced *connection subgraphs*: small portions of a network that capture the relationships between two of its objects. Such subgraphs are closely related to the problem of measuring and explaining network proximity.

In this study, which was largely inspired by Faloutsos et al., we introduce a novel way of measuring proximity between network objects that generalizes the previous approaches and provides certain advantages. We develop a new measure of proximity between network objects which has an intuitive interpretation based on random walks. This proximity measure can be readily "extracted" and visualized in the form of a proximity graph, and we describe different ways of visualizing the results. We explain how the proximity measure is easily generalized to find proximities between an arbitrary number of objects. Finally, we apply our methodology to four real datasets, each demonstrating a different strength of the methodology.

As an example of the proximity graphs we produce, consider Figure 1, which represents the proximity graph between Meryl Streep, Judy Garland, and Groucho Marx in the online movie database IMDB. The graph shows subcommunities of performers and their interrelationships. Readers may query this database and create their own proximity graphs on the webpage at `http://public.research.att.com/~volinsky/cgi-bin/prox/prox.pl`.

An earlier version of this article appeared in the *Proceedings of the ACM SIGKDD Conference* [Koren et al. 2006]. The major addition in this version is Section 7 that discusses practical applications. The applications include link prediction, community detection, and estimation of the values associated with nodes in a recommendation system. Additionally, we elaborate on the experimental results that were previously reported.

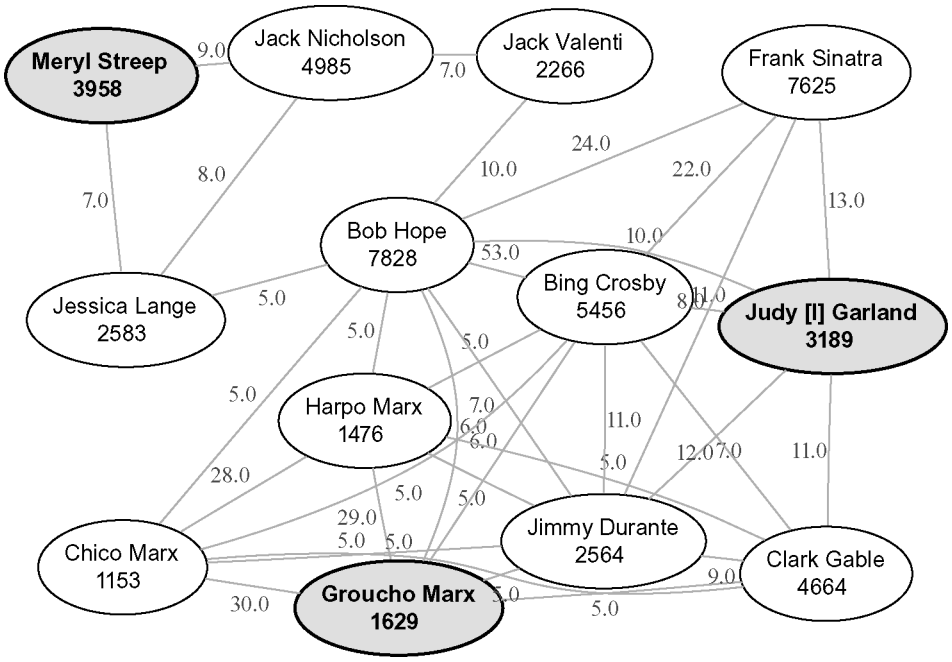

Fig. 1. Extraction of proximity graph between Meryl Streep, Judy Garland, and Groucho Marx in IMDB.com data. Edge weights are the number of cooccurrences in the IMDB data. CFEC = 4.88e-2. Subgraph captures 97.8% with $maxsize = 12$.

2. THE QUEST FOR PROXIMITY

Proximity is a subtle notion whose definition can depend on a specific application. Usually, the notion of proximity is strongly tied to the definition of an edge in the network. In a network where links represent phone or email communication, proximity measures the potential information exchange between two nonlinked objects through intermediaries. Where edges represent physical connections between machines, proximity can represent latency or speed of information exchange. Alternatively, proximity can measure the extent to which the two nodes belong to the same cluster, as in a coauthorship network where authors might publish in the same field and in the same journals. In other cases, proximity estimates the likelihood that a link will exist in the future, or is missing in the data for some reason. For instance, if two people speak on the phone to many common friends, the probability is high that they will talk to each other in the future, or perhaps that they already communicate through some other medium such as email.

There are many uses for good proximity measures. In a social network setting, proximities can be used to track or predict the propagation of a product, idea, or disease. Proximities can help discover unexpected communities in any network. A product marketing strategist could target individuals who are in close proximity to previous purchasers of the product, or those who have many people in close proximity for viral marketing.

In what follows, we formalize the notion of proximity by sequentially refining a series of candidate definitions, starting with the simplest: the shortest path. Notationally, we assume we have a graph $G(V, E)$, where the "network objects" are *nodes* (V) and the links between them are *edges* (E). The *weight* of edge $(i, j) \in E$ is denoted by $w_{ij} > 0$ and reflects the similarity of i and j (higher weights reflect higher similarity). For nonadjacent nodes $(i, j) \notin E$ we assume $w_{ij} = 0$. Each node is associated with a *degree* greater or equal to the sum of the edge weights coming out of that node. Usually, equality applies here such that $\deg_i = \sum_{j:(i,j)\in E} w_{ij}$. Sometimes we will equivalently refer to "distance" measures, which are the opposite of "proximity" measures: Low distance equates to high proximity. Unless stated otherwise, we are measuring proximity between two nodes s and t.

Graph-theoretic distance. The basic definitions we need for network proximity are taken from graph theory. The most basic is the *graph-theoretic distance*, which is the length of the shortest path connecting two nodes, measured either as the number of hops between the two nodes, or the sum of edge lengths along the shortest path. The main rationale for considering graph-theoretic distance is that proximity decays as nodes become farther apart. Intuitively, information following a path can be lost at any link due to the existence of noise or friction. Therefore, two nodes that are not connected by a short path are unlikely to be related. Distance in graphs can be computed very efficiently [Cormen et al. 1990]. However, this measure does not account for the fact that relationships between network entities might be realized by many different paths. In some instances, such as in managed networks, it may be reasonable to assume that information between nodes is propagated only along the most "efficient" routes. However, this assumption is dubious in real-world social networks, where information can be propagated randomly through all possible paths. For example consider Figure 2, which shows several different possible connections between s and t. For $i = 1, 2, 3$, the graph-theoretic distance between nodes s_i and t_i is 2, yet we have different conclusions about their proximity. One would likely conclude that s_2 is closer to t_2 than s_1 is to t_1 because they have more "friends" in common. Moreover, s_3 and t_3 are only connected through a node with high degree, which might indicate that they are not close at all (as in two people who both call a very common toll-free telephone number). Ideally, proximity should be more sensitive to edges between low-degree nodes that show meaningful relationships, as well as take into account multiple paths between the nodes.

Network flow. Consider another concept from graph theory: maximal network flow [Cormen et al. 1990]. Here we assign a limited capacity to each edge (e.g., one proportional to its weight) and then compute the maximal number of units that can be simultaneously delivered from node s to node t. This maximal flow can be taken as a measure of s-t proximity. It favors high-weight (thus, high-capacity) edges and captures the premise that an increasing number of alternative paths between s and t increases their proximity. Referring again to Figure 2 we can deliver twice as many units between s_2 and t_2 than $s_1 - t_1$, thereby gaining from the alternative paths. However, maximal flow disregards path lengths, so that we get the same maximal flow between s_4 and t_4 as between s_1 and t_1. Also problematic with this definition is that the maximal s-t

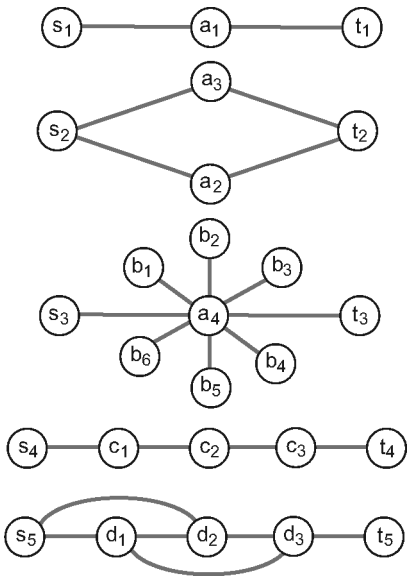

Fig. 2. A collection of graphs with all edge weights equal 1.

flow in a graph equals the minimal *s-t* cut, that is, the minimal edge capacity we need to remove to disconnect s from t [Cormen et al. 1990]. In other words, the maximal flow equals the capacity of the bottleneck, making such a measure less robust. For example, the maximal flow between s_5 and t_5 is 1, equal to the maximal flow between s_4 and t_4 despite the added pathways in $s_5 - t_5$. From the information flow viewpoint, added pathways create more opportunity for the information to pass from s to t, and hence the points should be closer. The network flow measure is too dependent on the bottleneck, and does not account for multiple paths or node degrees.

Effective conductance (EC). A more suitable candidate comes from outside classical graph-theory concepts: modeling the network as an electrical circuit by treating the edges as resistors whose conductance is proportional to the given edge weights. This way, higher-weight edges will conduct more electricity. Descriptions are found in standard references [Bollobas 1998; Doyle and Snell 1984]. When dealing with electrical networks, a natural *s-t* proximity measure is found by setting the voltage of s to 1, while grounding t (so its voltage is 0) and solving a system of linear equations to estimate voltages and currents of the network. The computed delivered current from s to t is also called the *effective conductance*, or EC. Effective conductance appears to be a good candidate for measuring proximity. It has been applied previously in graph layout algorithms [Cohen 1997], and to compute centrality in social networks [Brandes and Fleischer 2005]. Faloutsos et al. [2004] also consider EC in their study of connection subgraphs. An important advantage is that it accounts for both path length (favoring short paths, like graph-theoretic distance) and alternative paths (more are better, like maximal flow), while avoiding dependence on a single shortest path or bottleneck.

An appealing property of EC is that it has an equivalent intuitive definition in terms of random walks. Let's denote the effective conductance between s and t as $EC(s,t)$. Also, let $P_{\text{esc}}(s \to t)$ be the *escape probability*: the probability that a walk starting at s reaches t before returning to s. It is known [Doyle and Snell 1984] that $EC(s,t)$ can be expressed as

$$EC(s,t) = \deg_s \cdot P_{\text{esc}}(s \to t) = \deg_t \cdot P_{\text{esc}}(t \to s). \tag{1}$$

Effective s-t conductance is the expected number of "successful escapes" from s to t, where the number of attempts equals the degree of s. From an information-sharing viewpoint, this interpretation captures the essence of unmanaged, self-organizing networks when information, or general interactions, may pursue random routes rather than following well-planned ones. Consequently, the escape probability decreases if long paths must be followed because when tracking a long path from s to t, there is a increased chance of returning to s before reaching t.

EC also has a monotonicity property, courtesy of *Rayleigh's Monotonicity Law* [Doyle and Snell 1984]. This law states that in an electrical resistor network, increasing the conductance of any resistor or adding a new resistor can only increase the conductance between any two nodes in the network. In our context it means that each additional s-t path contributes to an escape from s to t, increasing EC. Referring back to Figure 2, this implies that $EC(s_5,t_5) > EC(s_4,t_4)$, and that $EC(s_2,t_2) > EC(s_1,t_1)$, which is intuitive.

While monotonicity is desirable in some cases, in others it contradicts our desired notion of proximity. As an example, again in Figure 2, $EC(s_1,t_1) = EC(s_3,t_3)$. In terms of random walks, the nodes of degree 1 emanating from node a_4 to $b_1,\ldots,b_6$ have no effect on escape probability, and therefore EC, because any walk following these edges is going nowhere and will eventually backtrack to a_4. Such backtracking means the random walk can make an unlimited number of attempts to reach s or t through these high-degree nodes without affecting EC. Real-world datasets tend to follow power laws in their degree distribution [Barabasi and Albert 1999] and have many nodes of degree 1, so this shortcoming can be quite significant. In our view, links into degree-1 nodes indicate real relationships, and should not be neglected when measuring proximity.

Sink-augmented effective conductance. Faloutsos et al. [2004] modify the EC model by connecting each node to a *universal sink* carrying 0 voltage. Thus, the universal sink competes with t in attracting the current delivered from s. This has the effect of a "tax" on every node that absorbs a portion of the outgoing current. Consequently, this forces all nodes to have a degree greater than 1, so the issue we mentioned with degree-1 nodes can no longer exist. Considering the universal sink from the random-walk perspective, it gradually overwhelms the walk, as after each step there is a certain probability that the walk will terminate in the universal sink.

The universal-sink model requires a parameterization of the sink edges to determine how much of the flow through each node is taxed. Understanding how such a parameter influences the computed proximities is difficult. A more important shortcoming of the universal sink is that it does not guarantee

monotonicity. As the network becomes larger, adding paths between s and t, the proximity between s and t typically *decreases*, approaching zero for large networks. The reason is that every new node must provide a direct link to the sink, while usually providing no direct link to the destination t. Therefore, an increasing number of nodes strengthens the sink to the point that t is not able to compete for delivered current. This introduces a counterintuitive *size bias*: The proximity between s and t calculated from a graph will typically be significantly *increased* when looking at a small subgraph of that graph. One immediate implication of this size-bias artifact is that it thwarts our goal of proving (or explaining) proximity using a small representative subgraph; since proximity value strongly depends on graph size, each selected subgraph will convey a different proximity value, usually much larger for smaller subgraphs. Computing optimal subgraph size or understanding how to normalize for graph size is unclear. Moreover, this dependency on size makes proximity comparisons across different pairs uninterpretable. Therefore, we assert that while sink-augmented current delivery solves many of the problems previously mentioned, its nonmonotonic nature is unsuitable for measuring proximity.

In the following section, we introduce a proximity measure, cycle-free effective conductance, which provides advantages over the aforementioned methods and has an intuitive random-walk interpretation.

3. CYCLE-FREE EFFECTIVE CONDUCTANCE (CFEC)

Our approach for measuring proximity is based on improving the effective conductance measure. To this end, we consider the random-walk interpretation. Before we give exact definitions, some notation is necessary: In the random walk, a probability of transition from node i to node j is $p_{ij} = \frac{w_{ij}}{\deg_i}$. Thus, given a *path* $P = v_1 - v_2 - \cdots - v_r$, the probability that a random walk starting at v_1 will follow this path is given by

$$\text{Prob}(P) = \prod_{i=1}^{r-1} \frac{w_{v_i v_{i+1}}}{\deg_{v_i}}. \tag{2}$$

At times we will refer to the *weight of a path* P, which we will define as

$$\text{Wgt}(P) = \deg_{v_1} \cdot \text{Prob}(P). \tag{3}$$

These definitions also apply to directed graphs. Recall that Eq. (1). relates effective conductance to the escape probability. Escape probabilities are characterized by a walk that will make an unlimited number of trials to reach an endpoint: s or t. Meanwhile, it might backtrack and visit the same nodes many times. We argued earlier that this is problematic when measuring proximity, as it does not consider any such paths that lower s-t proximity to be distracting. We fix this problem by considering *cycle-free escape probabilities* which disallow backtracking and revisiting nodes.

Definition 3.1. The cycle-free escape probability ($P_{\text{cf.esc}}(s \to t)$) from s to t is the probability that a random walk originating at s will reach t without visiting any node more than once.

Fig. 3. A family of s-t paths.

From the information-sharing viewpoint, this means that information flows randomly in the network, while already-known information has no contribution to the overall flow. This way, for example, high-degree nodes will distribute their information among all their neighbors, but sending information in the "wrong" direction (not leading to t) is a wasted effort which cannot be fixed by a later backtracking.

Consequently, we replace effective conductance by *cycle-free effective conductance*, which is defined, similarly to Eq. (1), as

$$EC_{\text{cf}}(s,t) = \deg_s \cdot P_{\text{cf.esc}}(s \to t). \tag{4}$$

As previously, we can take $EC_{\text{cf}}(s,t)$ as the expected number of walks completing a cycle-free escape from s to t, given that $\deg_s$ such walks were initiated. Henceforth, we abbreviate cycle-free effective conductance as *CFEC*.

In undirected graphs, random walks have the reversibility property [Doyle and Snell 1984], resulting in the equality

$$\text{CFEC}(s,t) = \text{CFEC}(t,s).$$

How useful is CFEC proximity? We assert that CFEC serves well as a proximity measure because it integrates the advantages of previous candidates while solving their shortcomings. Before we test CFEC against the previously proposed criteria, we note the following equalities. Let $\mathcal{R}$ be the set of simple paths from s to t (simple paths are those that never visit the same node twice).

$$P_{\text{cf.esc}}(s \to t) = \sum_{R \in \mathcal{R}} \text{Prob}(R) \tag{5}$$

Similarly, by multiplying by the degree, we have

$$\text{CFEC}(s,t) = \sum_{R \in \mathcal{R}} \text{Wgt}(R). \tag{6}$$

It is clear that a cycle-free escape must proceed along a simple path from s to t, so Eq. (5) merely sums the probabilities of all possible disjoint events. We will return to these equations later, when dealing with how to compute CFEC. However, for now they will aid us in studying CFEC's characteristics.

First, we require a proximity measure that favors short paths, as two remote nodes are unlikely to be in proximity. As evident from Eq. (5), CFEC strongly discourages long paths, as the probability of following a path decays exponentially with its length. For example, for the family of graphs shown in Figure 3, s-t proximity decays exponentially with their distance, $\text{CFEC}(s,t) = 2^{-k}$.

Another desired property of a proximity measure is that it favors pairs that are connected by multiple paths. This property, too, is inherent to CFEC, its being a sum of (positive) weights of all alternative paths. For example, for the graph family in Figure 4, the $\text{CFEC}(s,t)$ proximity is $k/2$, growing with the number of alternative paths.

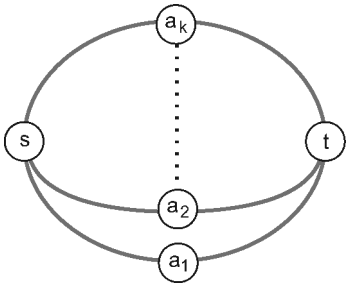

Fig. 4. A family of graphs characterized by the number of s-t paths.

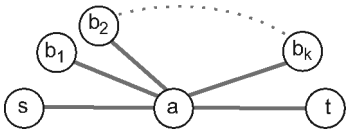

Fig. 5. A family of graphs with varying degree of a.

Moreover, unlike graph-theoretic and maximal flow distances, which are dependent, respectively, on a shortest path and the max cut, CFEC relies on all s-t paths, so each edge on any such path makes a contribution to the measured value.

We now address the issue of degree-1 nodes and, in general, dead-end paths. Recall that such distracting paths are neglected by the other proximity measures because of their monotonicity property. However, with CFEC any simple path leading from s to a node other than t must reduce the probability of escaping to t. For example, in Figure 5, the degree-1 nodes dilute the significance of the $s - a,\ a - t$ links, yielding a $1/(k + 2)$ CFEC proximity for this family of graphs. Of course, this is also tightly related to discounting the effect of high-degree nodes, as any single path passing through a node has a probability which is inversely proportional to the degree of that node.

The correct treatment of dead-end paths means that the CFEC measure is not monotonic with respect to edge addition; for example, adding degree-1 nodes decreases proximity. However, we avoid the aforementioned size bias where subgraphs have smaller proximity values than their corresponding supergraph. We accomplish this by constructing subgraphs in a special way which preserves monotonicity, as we explain now.

It is reasonable to expect the most accurate proximity value to be the one measured on the full network. Therefore, when we measure proximity on increasingly large subgraphs, we expect it to steadily converge towards its more accurate value. To this end, the operation we use for cutting a subgraph from the full graph preserves node degrees. Formally, given a graph $G(V, E)$ and a subset $\hat{V} \subset V$, the subgraph induced by $\hat{V}$ is $\hat{G}(\hat{V}, \hat{E})$, where $\hat{E} = \{(i, j) \in E \mid i, j \in \hat{V}\}$ and degrees are unchanged. In other words, for each $i \in \hat{V}$: $\deg_i^{\hat{G}} \stackrel{\text{def}}{=} \deg_i^G$, where superscripts indicate the respective graph. Here, we took the liberty of using node degrees that are greater than the sum of respective adjacent edge weights (consistent with our definition of degree in Section 2). In essence, each node has

a self-loop whose weight ensures that each degree equals the sum of adjacent edge weights. Preservation of original degrees ensures that the CFEC proximity measured on a graph cannot be smaller than that measured on a subgraph thereof: According to the definition of CFEC in Eq. (6), any simple s-t path that exists in $\hat{G}$ must have the same weight (or probability) as in the full graph G. However, there might be additional s-t paths that exist only in G, which will lower CFEC proximity when measured on the subgraph $\hat{G}$. Therefore CFEC proximity is a monotonic series under this subgraph operation. This series is tightly bounded from above by the proximity measured on the full graph. In other words, when measuring proximity on a subgraph of the full network, the larger the subgraph, the more accurate the measurement: a desirable property.

In practice, we found that measuring proximity on an s-t neighborhood requires only a small fraction of the full graph. Further increasing the neighborhood size yields diminishing improvements. When an accurate proximity measure is needed, one cannot lose accuracy by working with the largest possible network. On the other hand, CFEC proximity can often be explained by small subgraphs in the sense that if we measure a certain proximity value on a very small subgraph, then this value is a provable lower bound of the accurate proximity value. We will show later that experimentally, a very significant portion of the proximity can usually be explained by subgraphs with fewer than 30 nodes. This strongly suggests that both the connection subgraphs of Faloutsos et al. [2004] and our *proximity graphs* are very effective in capturing the relationships that determine CFEC proximity. Later (Section 5.1) we will see that this allows us to assign confidence scores to the proximity graphs, reflecting how well they describe full proximity.

Directed graphs differ from undirected graphs in having asymmetric connections, so an edge $i \rightarrow j$ can be used only for moving from i to j, but not vice versa. Interestingly, in contrast with electrical resistor network models, cycle-free effective conductance naturally accommodates directed edges, as its underlying notions remain well defined for directed graphs. An interesting issue is the emerging asymmetry $\mathrm{CFEC}(s,t) \neq \mathrm{CFEC}(t,s)$. In fact, one can view it as a mere reflection of the inherent asymmetry of directed edges, which also exist in the graph-theoretic distance, the other proximity measure that deals with edge directions. If needed, one can remove this symmetry by taking the sum $\mathrm{CFEC}(s,t) + \mathrm{CFEC}(t,s)$ or the product $\mathrm{CFEC}(s,t) \cdot \mathrm{CFEC}(t,s)$ as the s-t proximity.

4. COMPUTING CYCLE-FREE ESCAPE PROBABILITY

We are not aware of any efficient algorithm for computing CFEC proximity exactly, however, we can efficiently find accurate approximations. Fortunately, we may not even need much accuracy: Experiments show that even merely estimating the order of magnitude of proximity may suffice, since proximity values appear to be lognormal distributed (see Figure 10). The basis for our approximation is Eq. (5).

$$P_{\mathrm{cf.esc}}(s \rightarrow t) = \sum_{R \in \mathcal{R}} \mathrm{Prob}(R)$$

We can estimate this value using only the most probable s-t paths. In all our experimental datasets, we consistently found that path probability falls off exponentially. If we order simple s-t paths by probability, the 100^{th} path is typically a million times less probable than the first. Based on empirical results, the probability of paths drops fast enough that the low-probability paths cannot sum to any significant value. Therefore, we estimate $\text{CFEC}(s,t)$ by restricting the sum in Eq. (5) to the k most probable simple paths $\mathcal{R}_k$, where this set is determined by some threshold. Usually, we stop when the probability of the unscanned paths drops significantly below that of the most probable path (e.g., below a factor of 10^{-6}).

Thus we need an algorithm to find simple s-t paths in order of decreasing probability. To this end, we first transform edge weights into edge lengths, establishing a 1-to-1 correspondence between path probability and path length. For each edge (i, j) we define its *length* l_{ij} to be

$$l_{ij} = -\log\left(\frac{w_{ij}}{\sqrt{\deg_i \cdot \deg_j}}\right). \tag{7}$$

Using these edge lengths, path lengths correspond to path probabilities via the equation

$$\text{Prob}(R) = C \cdot e^{-\text{length}(R)},$$

where R is some s-t path, and the positive constant C is $\sqrt{\frac{\deg_t}{\deg_s}}$. Therefore the shortest path is the most probable path; the second shortest path is the second most probable, and so on.

Our solution involves the *k-shortest simple paths problem* a natural generalization of the shortest path problem in which not one, but several paths in order of increasing length are sought. Interestingly, Katoh et al. [1982] describe an efficient algorithm for undirected graphs, with running time linear in k and log-linear in the graph size $O(k(|E| + |V| \log |V|))$; Hadjiconstantinou and Christofides [1999] present additional implementation details. We employ this algorithm to find the shortest (most probable) simple paths. Paths of monotonically increasing length are generated successively. (As mentioned, we stop the search when the path probability drops below a certain threshold. In our implementation this is $\epsilon = 10^{-6}$ of the probability of the first, most probable path.) The number k of such computed paths is usually a few hundred. Then, the estimated CFEC proximity is

$$\text{CFEC}(s,t) = \sum_{R \in \mathcal{R}_k} \text{Wgt}(R). \tag{8}$$

5. EXTRACTING PROXIMITY THROUGH PROXIMITY GRAPHS

The computed CFEC proximity value depends on the full network. Therefore, explaining the proximity value or convincing a user of its validity can be complex. Nonetheless, frequently our ability to use proximity values and act upon them depends on our ability to understand the structure that drives them. So, one of our main goals is to find ways of extracting and explaining proximity

using small, easily visualized subgraphs, which we call proximity graphs. Practically, we are looking for small subgraphs of the original network such that the s-t CFEC proximity measured on this subgraph is close to that measured using the full network. This is related to the connection subgraphs structure proposed by Faloutsos et al. [2004], which deals with extracting a small subgraph that describes the relationship between two nodes. A distinction from Faloutsos et al. [2004], however, is that we work here in the context of explaining a specific CFEC proximity value.

Fortunately, the cumulative nature of CFEC proximity, as expressed in Eq. (8), eases the task. We store the k paths that were used for computing the proximity value, so these paths serve as the building blocks of the connection subgraph. Formally, we have a sorted series of paths $\mathcal{R}_k = P_1, P_2, \ldots, P_k$ such that $\mathrm{Wgt}(P_i) \geqslant \mathrm{Wgt}(P_{i+1})$. We form the connection subgraph by merging a subset of $\mathcal{R}_k$. The easiest solution would be to merge the most probable paths, without exceeding some fixed bound on the number of nodes $B > 0$. In other words, we compute the maximal r for which $|\bigcup_{i=1}^{r} P_i| \leqslant B$, and then define the connection subgraph as the subgraph induced by the union of the first r paths $\bigcup_{i=1}^{r} P_i$. This simple approach is plausible, as we have found experimentally in our datasets. However, it fails to account for overlaps between paths. Overlapping paths, which share common nodes, have the advantage of increasing the captured proximity while using fewer nodes. Therefore, we can do better by looking at general subsets of $\mathcal{R}_k$ rather than considering only prefixes thereof.

In which subset of $\mathcal{R}_k$ we are interested? One possibility is to find a subset that solves the following problem.

(A1) Extract a subgraph with at most B nodes that captures maximal s-t proximity.

Dually, instead of fixing size, we can fix the desired amount of captured proximity, so the subset will solve the problem next posed.

(A2) Extract a minimal-sized subgraph that captures an s-t proximity value of at least C.

However, optimizing either (A1) or (A2) can yield inefficient solutions due to the arbitrariness of B or C. For example, when solving (A1), it might be that by replacing B with $B+1$ we could have captured much more proximity. Or, when solving (A2), maybe by taking $C-1$ instead of C we could have significantly reduced the subgraph size. Hence, we prefer working with the following formulation that achieves an efficiently balanced solution by considering CFEC on a per-node basis.

(A3) Extract a subgraph $\hat{G}(\hat{V}, \hat{E})$ that maximizes the ratio

$$\frac{(\mathrm{CFEC}^{\hat{G}}(s,t))^{\alpha}}{\|\hat{V}\|}. \tag{9}$$

Here, the superscript $^{\hat{G}}$ indicates measuring CFEC on $\hat{G}$. The constant $\alpha \geqslant 0$ determines our preference in the size versus proximity tradeoff: The greater is α, the more we favor the numerator, maximizing the captured proximity. Thus,

when $\alpha = 0$ only the subgraph size matters, and an optimal solution will be a shortest path (measured by number of nodes). Similarly, in the other extreme case when $\alpha = \infty$, size becomes irrelevant, so an optimal solution merges all k available paths. As shown experimentally, we find $5 \leqslant \alpha \leqslant 10$ to be effective. Note that by using a ratio, the inefficiencies mentioned before cannot exist; an optimal ratio cannot allow a small increase in the denominator (size) to significantly increase the numerator (captured proximity), or vice versa. In implementation, for numerical reasons we recommend using the logarithm of this ratio as a proxy for its actual value, which is important when α is large. Also, in the denominator, it may be reasonable to replace $\|\hat{V}\|$ with $\|\hat{E}\|$ (or $\|\hat{E}\| + \|\hat{V}\|$), since high edge density interferes with readability of the graph layout (though we have not yet explored this experimentally).

An obvious question is how to find a subset of $\mathcal{R}_k$ that solves the aforesaid problem. Solving (A1)–(A3) is NP-hard, as they are reducible from the closely related knapsack problem [Garey and Johnson 1979] (with the main difference that here we need to account for possible overlaps between the paths in addition to their size and weight). It is very unlikely that we can devise a polynomial-time algorithm for solving these problems. Consequently, we suggest two heuristic approaches. The first is an exact branch-and-bound (B-and-B) algorithm. Since this might take exponential time, we may have to prematurely terminate the search. In that case, we take the best intermediate solution discovered by B-and-B and try to improve it by an agglomerative polynomial-time algorithm.

The branch and bound algorithm for optimizing (A3). Recall that we ordered paths by their weights. Accordingly, we scan all subsets of $\mathcal{R}_k$ in lexicographic order, so we scan all subsets containing P_i before completing the scan of subsets containing P_{i+1}. This way we target more promising subsets (based on total weight) first. A full scan would loop through all 2^k possible subsets and find an optimal one. However, we can significantly reduce running time by pruning subsets that are provably suboptimal. To this end, during the scan we record the best subset found so far. Then, for every new subset, we check whether under the most optimistic scenario it can be a part of an optimal solution. This check is done by adding the weights of all subsequent paths to the weight of the current subset, while assuming that the size of this subset would not be increased at all. If even after this operation the subset's score (according to Eq. (9)) is lower than the current best one, we prune the search, skipping this subset and all subsequent ones containing it. Detailed pseudocode is given in Figure 6.

When the B-and-B algorithm terminates early, we need to switch to the greedy agglomerative optimizer. This algorithm starts with the k sets $\{P_1\}, \{P_3\}, \ldots, \{P_k\}$. To take best advantage of the result of the B-and-B algorithm, we merge all sets whose union is the result returned by the B-and-B algorithm into a single subset. Then, the algorithm iteratively merges the two sets, maximizing Eq. (9) (replacing two sets by their union) until a single set remains. During the process we record the best set found, which is returned as the final result.

In experience, we often find an optimal solution with the branch-and-bound algorithm, and even when the agglomerative optimizer is invoked, usually it

```
Function PathMerger (P_1, P_2, ..., P_k)
% Input: a list of paths sorted by decreasing weight

  Stack ← ∅
  BestSubset ← ∅
  Weights ← Σ_{i=1}^{k} Wgt(P_i)

  for i = 1 to k do
    Weights ← Weights − Wgt(P_i)
    for Subset ∈ Stack ∪ {∅} do
      NewSubset ← Subset ∪ P_i
      NewSubset.wgt ← Subset.wgt + Wgt(P_i)
      if Score(NewSubset.wgt + Weights, ||NewSubset||) >
        Score(BestSubset.wgt, ||BestSubset||) then
        Stack.push(NewSubset)
        if Score(NewSubset.wgt, ||NewSubset||) >
          Score(BestSubset.wgt, ||BestSubset||) then
          BestSubset ← NewSubset
        end if
      end if
    end for
    return BestSubset
  end

Function Score (weight, size)
  return weight^α / size    % See Equation 9
end
```

Fig. 6. The branch-and-bound path-merging algorithm.

cannot improve the B-and-B result. Also, our implementation allows setting an upper bound (*maxsize*) and lower bound (*minsize*) to limit the proximity graph's size.

5.1 Assessing Proximity Graphs

The proximity graphs created by our method are aimed at capturing maximal proximity with few nodes. By definition, the CFEC proximity captured in these subgraphs must be lower than the full CFEC proximity measured on the whole network. In fact, the captured CFEC is easily computed by taking the sum of weights of those paths whose merge created the connection subgraph. Hence, we can use the CFEC proximity to assess the quality of the subgraphs by reporting the percentage of captured proximity. For example, a proximity graph with a 10% score is not very informative, as it captures only a small fraction of the existing relationships. On the other hand, a subgraph with a 90% or higher score shows most of the available proximity. Moreover, the proximity values between different pairs of objects within the same graph are directly comparable. The ability to compute, optimize, and analyze this score is a key contribution of our proposed technique.

5.2 Multiple Endpoints

Another benefit of our method is that it is easily extended to show relationships between more than two endpoints. The only change to the algorithm is the way

we construct the set of paths $\mathcal{R}_k$. When multiple endpoints exist, we are no longer looking for shortest paths among two endpoints, but for the broader set of shortest paths connecting any two of the given endpoints. This is achieved by a straightforward generalization of the k-simple shortest paths algorithm that finds simple shortest paths connecting *any* two members of a given set of nodes. Time complexity is virtually unaffected, being for l endpoints $O((k + l^2)(|E| + |V| \log |V|))$. No other modification to the algorithm is necessary.

There are some important details about handling queries with multiple endpoints. In particular, we do not require that the proximity subgraph connects all endpoints, but only those that make a sufficient contribution to the overall proximity value. Only relevant endpoints appear in the extracted subgraph. Thus, one has the flexibility to query proximity relations among many endpoints, without being concerned about which of them is really relevant. Then our algorithm prunes the unrelated endpoints, allowing the user to concentrate on the most useful relationships. Alternatively, one could enforce that all endpoints be connected. In some applications, users might require such behavior. This is easily achieved by postprocessing to insert each of the unconnected endpoints using the most probable (shortest) path connecting it to the computed proximity subgraph.

5.3 Directed Edges

As mentioned earlier, CFEC proximity is also general enough to handle directed edges. Naturally, this is also valid for the proximity graph construction algorithm, as it deals with entire paths regardless of the direction of their edges. Obviously, if edges are directed, the algorithm that produces the k shortest paths must be changed to account for edge directions. This increases running time. The algorithm we chose (by Katoh et al. [1982]) works only on undirected graphs. An algorithm for computing the k simple shortest paths on directed graphs was described by Hershberger and Suri [2003]. Its time complexity is $O(k|V|(|E| + |V| \log |V|))$, but the authors claim that we rarely encounter this worst-case behavior, and the average-case running time is only $O(k(|E| + |V| \log |V|))$, which is comparable to the undirected case.

6. WORKING WITH LARGE NETWORKS

Some of the databases we work with are huge. The DBLP cocitation graph has over 400,000 authors; the IMDB movie-actors connection graph has about a million nodes and 4 million edges; a telephone call detail database may have hundreds of millions of nodes. The Yahoo Instant Messenger database is even larger, with about 200 million nodes and 800 million edges [Lang 2004]. Computation on such large networks can exceed typical time and space resources. However, in practice, only a tiny fraction of the network makes a significant contribution to any computed proximity value. This is supported by the efficacy of very small connection subgraphs to explain proximity, as we report in Section 7. Therefore, we can work only on that portion of the full network that includes the relevant information. To this end we compute a conservative estimate of this portion, which is usually a tractable subgraph containing a few

thousand nodes. Then, the proximity value is computed on this subgraph and the much smaller connection subgraph is extracted from it. Note that here we are only interested in a fast, coarse approximation of the relevant portion of the full network, unlike the more precise computation that extracts the connection subgraph. Faloutsos et al. [2004] deal with similar issues in computing connection subgraphs. They named the reduced subgraph, on which their connection subgraph is computed, a *candidate graph*. We adopt this nomenclature. It appears that the tight relationship between our candidate graph and measured CFEC proximity allows aggressive pruning of its size, while still maintaining sufficient connectivity structure.

Growing candidate graphs via $\{s, t\}$ *neighborhoods.* Let us assume that we are computing the proximity between two nodes s and t. Cases involving more than two endpoints will be handled analogously. The first stage in producing the candidate graph is to find a subgraph containing the highest-weight paths originating at either s or t. Recall that the weight of a path is defined by Eq. (3) as $\mathrm{Wgt}(P) = \deg_{v_1} \cdot \mathrm{Prob}(P)$. By transforming edge weights into edge lengths as described in Eq. (7), the problem becomes that of finding a subgraph containing shortest paths originating at either s or t. Equivalently, our object is to expand the neighborhoods of s and t. This is done by the standard Dijkstra's algorithm [Cormen et al. 1990] for computing shortest paths on graphs with nonnegative edge lengths. This algorithm finds a series of nodes with increasing distance from $\{s, t\}$. Note that to make the algorithm efficient on a huge graph, we assume an indexing mechanism that allows us to efficiently access the neighborhood of a given node.

Determining neighborhood size. When do we stop growing the $\{s, t\}$ neighborhoods? Assume that the highest-weight, that is, the shortest length, s-t path is P of length L. This path will be discovered by the algorithm when the $\{s, t\}$ neighborhoods grow enough to overlap. Since such an overlap occurs, the path P passes within the neighborhood that was created. Recall that CFEC proximity is computed using the k shortest paths such that the $(k+1)^{\text{th}}$ path is the first whose weight drops below ϵ times the weight of the first path. After transforming weights into lengths, we know that paths longer than $L - \log(\epsilon)$ are not useful. Therefore we should expand the neighborhood until it covers all s-t paths shorter than $L - \log(\epsilon)$. Consequently, we are interested only in nodes whose distance from either s or t is at most $(L - \log(\epsilon))/2$.

To summarize, $\{s, t\}$ neighborhoods are expanded until they overlap. At this point we learn L: the length of the shortest s-t path. Then, we continue by expanding the $(L - \log(\epsilon))/2$-neighborhoods of both s and t. Every significantly weighted s-t path must go through this neighborhood.

In practice, these $(L - \log(\epsilon))/2$-neighborhoods might be too large. In these cases we stop extending the neighborhood when its size reaches a predetermined limit, typically several thousand.

Pruning the neighborhood. Interestingly, the created neighborhood can be significantly pruned. This is true, regardless of whether we completed generating the full $(L - \log(\epsilon))/2$-neighborhoods or stopped earlier. Let us denote by $\mathrm{dist}(i, j)$ the length of the shortest path between i and j (the graph-theoretic distance). Observe that if $\mathrm{dist}(s, i) + \mathrm{dist}(t, i) > \beta$, then any s-t path going

through i must be longer than β. Therefore, we compute $\text{dist}(s, i)$ and $\text{dist}(t, i)$ for each i in the neighborhood. Then we exclude from the neighborhood any i for which $\text{dist}(s, i) + \text{dist}(t, i) > L - \log(\epsilon)$, as no s-t path of appropriate length can pass through such i. Graphically, this pruning takes the two "circles" centered at s and t and transforms them into an "ellipse" with two foci, namely s and t, defined by the equation $\text{dist}(s, i) + \text{dist}(t, i) \leqslant L - \log(\epsilon)$. Empirically, we have found this pruning to be very effective, usually removing most nodes in the neighborhood. Finally, we take the graph induced by the pruned neighborhood as our candidate graph, calculate CFEC proximity between the objects of interest with respect to this graph, and extract the proximity graph which best captures that proximity.

7. APPLICATIONS

We tested the CFEC proximity measure on several large network datasets. Each application highlights specific features of the proximity measure. The test datasets were as follows.

(1) *Telecommunications Data (Phone)*. The U.S. telephone network has hundreds of millions of phones; the calls between these phones can be viewed as a massive network. AT&T has a record of calls carried on its network which is on the order of hundreds of millions per day, and the number of nodes (telephone numbers) is of the same order of magnitude. Typically, an edge in this graph indicates that there has been at least one call between the two phone numbers in the last time period t, often set to 30 or 90 days. We put a weight on the edges representing the total communication time.

(2) *Coauthorship (DBLP)*. The DBLP database (`http://dblp.uni-trier.de/`) is an online bibliography of computer science journals and proceedings [DBLP 1998]. The data from the bibliography is freely available and can be used to create a network of coauthorship, where the nodes in the network are authors, and the links indicate coauthorship. Link weight indicates how many papers two researchers published together.

(3) *Movie Actors (IMDB)*. The Internet Movie Database (`www.imdb.com`) provides metadata on movies and television shows, including the actors who appeared in these shows. From this data, one can generate a network where the nodes are actors and the links represent actors who appear together in at least one show. Weights on the links indicate multiple appearances.

(4) *Movie Ratings (Netflix)*. In 2006, the Internet movie rental company Netflix released a large dataset as part of a contest to see if anyone could improve upon their movie recommender system. The data consisted of ratings of movies from almost 500,000 Netflix users. We created a network from this data where links represent movies corated by the same user and these links have two attributes: *support*, namely the number of users who corated these movies, and a *similarity score* indicating how alike are the scores when corated. Although proximities could be useful in recommender

Table I. Statistics for Four Experimental Datasets

Value	Phone	DBLP	IMDB	Netflix
Nodes	∼300M	438K	896K	17770
Edges	∼1B	1.2M	47 M	157.5M
Mean Degree	∼40⋆	5	104	17728
Density	∼0⋆	10^{-5}	10^{-4}	1
Link?	90	54	77	100
Median Prox	−6.5	−7.4	−7.6	−3.9
SPath	5.8	4.4	7.7	1.0
GSize	20	28	43	2

(The *Prox* column shows the mean log proximity value. *Link?* reports the percentage of pairs in the sample that had any path between them. *SPath* is the mean shortest path among pairs in the sample where a path was found. *GSize* indicates the median proximity graph size for those pairs (for the Phone data, the ⋆ indicates that the mean degree and density were calculated for a sample of residential data points only, while the other measurements include all data.))

systems, in this article we are simply exploring the utility of this new proximity measure. More comprehensive techniques for building recommender systems for the Netflix dataset are described in Salakhutdinov et al. [2007] and Bell et al. [2007].

Table I shows some basic statistics about the datasets, along with some average features for proximity graphs calculated on a random sample of pairs from each dataset. Some interesting features emerge. In comparing DBLP and IMDB, for instance, we see that DBLP is less dense, with considerably fewer edges per node. This corresponds to a smaller percentage of pairs for which we are able to find connecting paths, but, interestingly, the paths found tend to be much shorter, indicating a network consisting of many small clusters. In contrast, in the much larger telephone call network we find paths between 90% of randomly chosen node pairs, despite an extremely low density. The phone network also has a relatively small median proximity graph size, and high proximity values (although since these values depend on the scale of the edge weights, they are not directly comparable across datasets). So, the phone network, despite being vast and sparse, actually connects most of the communications universe. The Netflix data is an interesting contrast. It is quite close to a complete graph, so measurements like shortest path and graph size are less relevant, yet the density of this graph brings up some different questions about the relationships between nodes, as discussed next.

7.1 Exploring Properties of Proximity Graphs

In our first example, we apply our proximity measure to the phone data. For this exercise, we are interested in proximities between subscribers of a particular communications service. We have all records of communications between members of this group, as well as communications from this group to other members of the larger telecom network. Because there are many calls carried on other networks, there is a potential missing data problem which we ignore

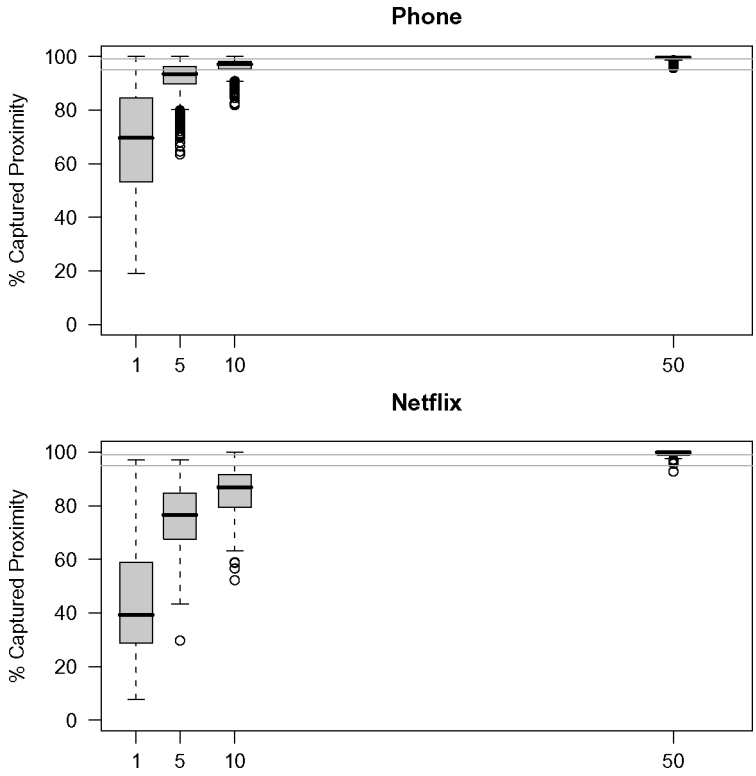

Fig. 7. Boxplots showing percentage of captured proximity as a function of α for the phone and Netflix data. The box marks the space between the 25th and 75th percentiles of the distribution.

for the time being. Note that we have no access to content of the communication, but simply a record that the communication occurred and its duration. In addition, for research purposes we do not access any customer-identifying data such as name or address.

We can calculate CFEC proximity between any two members of this network readily, using the methods described earlier. We extract the candidate graph by growing neighborhoods from the two nodes of interest as described in Section 6, and then find the proximity graph using Eq. (9). For the experiments next described, we selected 2000 random pairs of telephone numbers to calculate the CFEC value. For 1808 (90%) of these pairs, we were able to find paths between them; the others did not have any known path between them, perhaps because they are low-usage numbers, or perhaps due to the missing data problem.

The parameter α (Eq. (9)) allows the user to tradeoff the desire for a small, visualizable subgraph for capturing more of the total proximity in the graph. To investigate this parameter, we calculated the percent of overall CFEC proximity captured in the subgraph. As α increases, so does the size of the subgraph, hence more of the available proximity will be captured. Figure 7 shows the results for our sample. Boxplots plotted at $\alpha = 1, 5, 10,$ and 50 show the percent of captured proximity in the subgraph. A horizontal line is drawn at 95% for

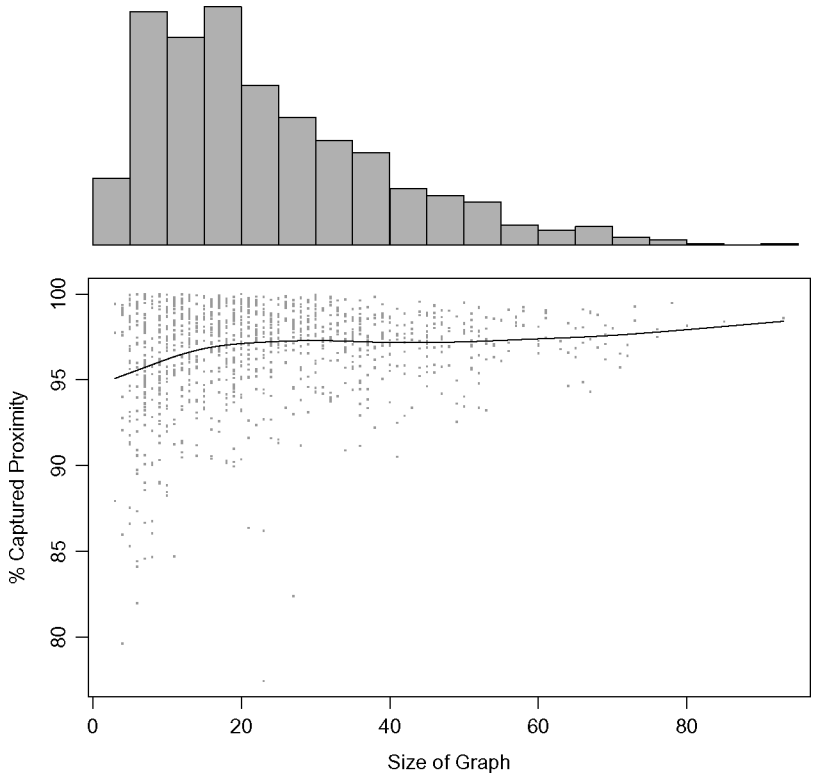

Fig. 8. Plot showing size of proximity graphs and their relation to captured proximity. The top plot shows the distribution of graph sizes for our sample of 2000 pairs. The bottom of the figure plots graph size against the percentage of captured proximity, with a smoothing spline plotted through the data.

reference. For $\alpha = 10$ about 3/4 of the subgraphs capture 95% of the CFEC proximity, and these graphs are relatively small (median = 24 nodes, standard deviation = 16). If we bump up to $\alpha = 50$, almost 100% of our samples capture 95% of the overall proximity, at the cost of larger subgraphs (mean = 50, standard deviation = 32). We think $\alpha = 10$ gives the best tradeoff, yielding graphs small enough to be visualized, so we use this value in all subsequent analysis for the phone dataset. For contrast, we also show an analogous plot for the Netflix dataset, which is considerably denser. It is clear that $\alpha = 10$ is not sufficient to provide captured proximity of 95% or more, probably because there are simply so many paths between nodes due to the density of the graph.

In Figure 8, we explore how large the subgraphs need to be to capture a meaningful percentage of proximity. Looking at the resulting subgraphs for 2000 pairs using $\alpha = 10$, we plot graph size against the percentage of overall captured proximity in the bottom of the figure. On top we plot a histogram showing proximity graph size. One can see that the majority of the proximity graphs have 50 nodes or less, which is small enough for readable visualization. From the bottom plot, we see that if the algorithm selects a smaller graph size, there is more variance in the amount of captured proximity. However, for all

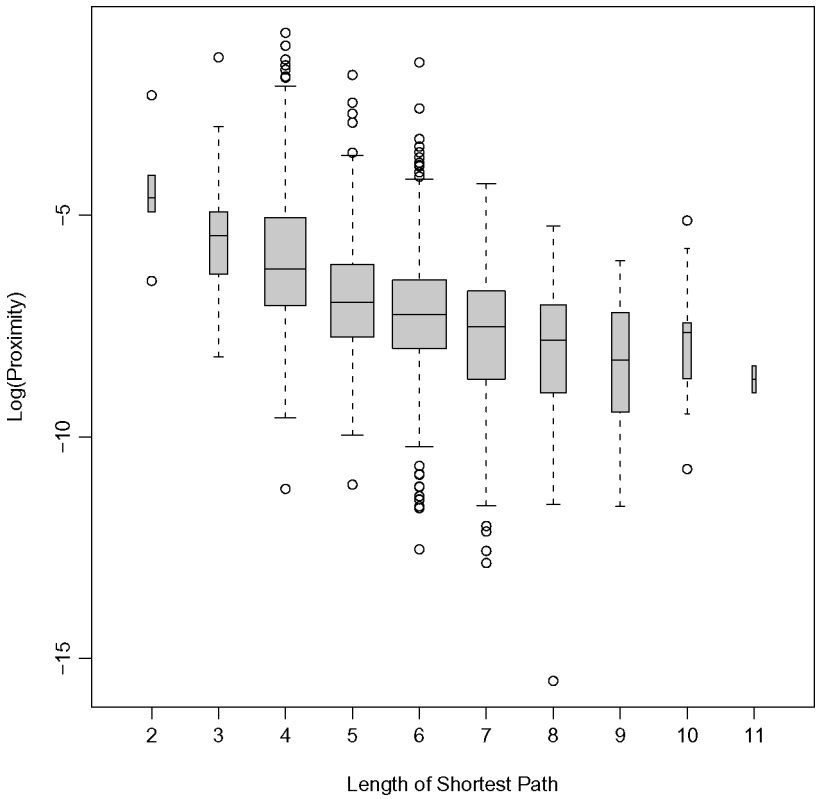

Fig. 9. Boxplots of log proximity for each value of shortest path distance. Boxes represent distance between the 25th and 75th percentile of the distribution.

graph sizes, the mean (represented by the smoothing spline) is consistently around 95%. This indicates that the algorithm does a good job of balancing graph size against captured proximity.

Another aspect worth investigating is the relationship between the CFEC proximity measure and the simple graph-theoretic measure of shortest path. Figure 9 shows this relationship for our sample. We plot the shortest path in the network for each pair in our sample, and compare it to the calculated proximity. The distribution of proximity is shown for each value of shortest path as a boxplot (the width of boxplot is proportional to the number of pairs in that group). The graph clearly shows that our proximity provides different information than shortest path. The medians of the distributions (indicated by the line in the box) show an expected general trend: As the shortest path between two nodes increases, their proximity decreases. However, the amount of boxplot overlap indicates that there may be pairs which are farther apart by shortest path that have quite similar proximities. It is clear that a ranking of closest pairs would be very different using these two metrics. Another interesting fact about the data: The median of the shortest paths across our sample is 6, corresponding (perhaps coincidentally) to Milgram's folkloric "six degrees of separation."

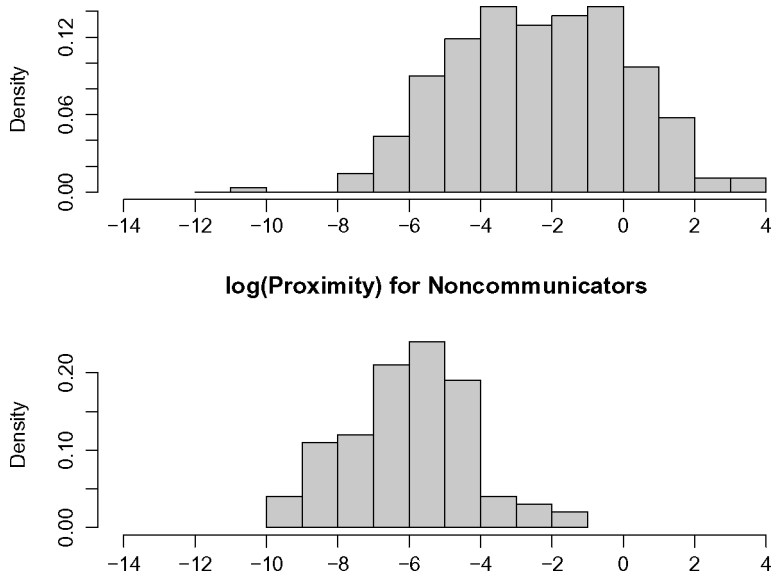

Fig. 10. Histograms comparing distribution of logarithm of proximity score for pairs of nodes that communicated (top) and those that did not (bottom).

7.2 Using Proximities for Link Prediction

Proximities in a social network such as the phone data can be viewed as a measure of social closeness. As such, one might hypothesize that this measure can predict which people will call each other in the next time period. This is the so-called "link prediction" problem [Liben-Nowell and Kleinberg 2003]. Here, we show the utility of CFEC in predicting future links in the network. We performed an observational experiment using the phone data: At time t, collect proximities on a large set of random pairs of phone numbers that have no communication history, wait until time $t + \delta$, then see which of these pairs communicate during $(t, t + \delta)$. We then compare the proximities for those that communicated with those that did not. If proximity has predictive value, we expect those that communicated in the test period to have had closer proximity at time t, before the test period began.

Our experiment shows that proximity indeed seems to have predictive value. Figure 10 shows the distribution of the proximities for the communicators and noncommunicators. Those that ultimately made a phone call had log proximity values averaging -2.4, while those that did not have a call averaged -5.9. Similarly, the data suggests that two nodes with proximity greater than 1 (log proximity > 0) have a high probability to be future communicators.

Liben-Nowell and Kleinberg [2003] describe a thorough comparison of various similarity metrics to predict coauthorship from the arXiv database. The methods range from simple counts of common neighbors to more complex ones based on random walks and clustering. They found that the most successful measures were those that were robust to spurious connections between nodes, such as a one-time collaboration between authors in disparate fields. The CFEC

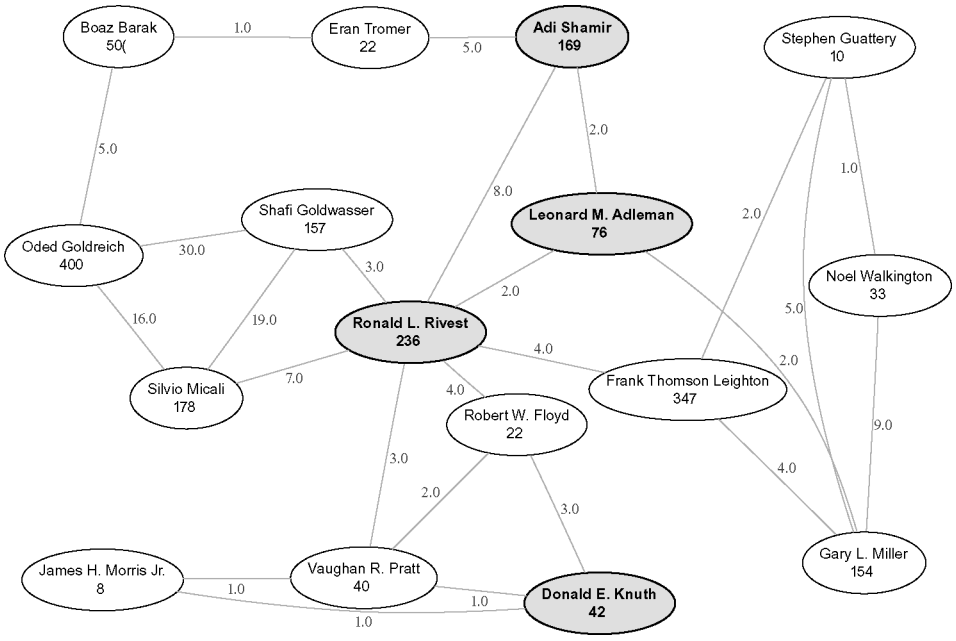

Fig. 11. Proximity graph for computer scientists Rivest, Knuth, Shamir, and Adelman from the DBLP coauthorship graph. The resultant graph shows other well-known colleagues, resulting in an interesting academic community. CFEC $= 1.30e + 1$. Subgraph captures 99.6% with $minsize = 12$.

measure has such robustness: Paths traversing a spurious link will likely not reach the destination, since cycling back down that link is not allowed. Another such method based on random walks [Tong and Faloutsos 2006] uses the idea of occasional restart of the random walk to find "center-piece subgraphs." These subgraphs identify the most central nodes between a given set of nodes, a concept similar to our proximity graphs. We plan to compare our method with these competing methods in future work.

7.3 Proximities for Community Detection

Measuring proximity between known authors in a particular field can identify interesting clusters of researchers in that field. The DBLP database allows us to find such clusters in computer science. Figure 11 demonstrates our method's ability to find communities and proximity graphs for more than two seed nodes. In this case, we search in the DBLP database for the well-known computer scientists Rivest, Adelman, Shamir, and Knuth. The result shows the close relationship between the first three authors, and the connections with subcommunities of Israeli cryptographers and noted algorithm experts.

Using proximities to find communities in databases like DBLP may also be useful for unaliasing, or correcting errors introduced when the same person shows up with two different labels [Bhattacharya and Getoor 2005]. By combining our work with string matching and other record linkage techniques, proximity graphs could contribute to a wide body of work on deduplication in databases [Winkler 1999].

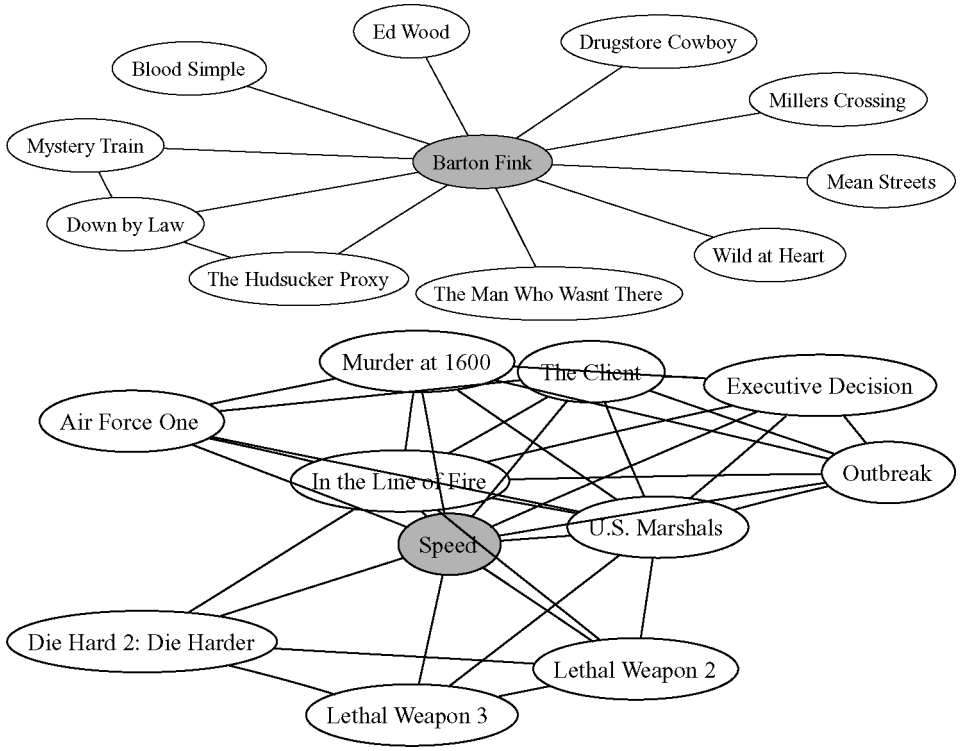

Fig. 12. Proximity graph showing communities around *Barton Fink* and *Speed* for the Netflix data.

Another way of approaching community detection is by defining and characterizing the community around a single node. To demonstrate, we use the Netflix data and build a community around a single movie m by calculating the proximities between that movie and all other movies in the network. By looking at the collection of movies that are closest to m, we can get a sense of whether it is part of a well-defined cluster of movies or is somewhat isolated in movie space.

To show this, we selected two movies from the Netflix database: *Barton Fink* and *Speed*. These movies had near-identical popularity, each scoring an average rating of 3.5 (out of 5) with over 75,000 votes cast. However, *Barton Fink* is a quirkier movie, harder to characterize, and less accessible to a vast audience, while *Speed* belongs to the well-defined genre of action thrillers. We wanted to see if the proximity graphs built on these two movies would show that one was part of a community of movies while the other wasn't.

In fact, this was true. After calculating all proximities, *Barton Fink*'s closest movie was *Miller's Crossing*, with a proximity of 0.765. In contrast, *Speed*'s closest neighbor was *Air Force One*, with proximity 1.148, and had a total of 44 films with proximity greater than 0.765. We show the difference between the two movies in a different way in Figure 12. This graph shows each movie with its 10 closest neighbors, and links between movies that have high proximity

to each other. We see that not only does *Barton Fink* have lower proximity values to its neighbors, but further that those neighbors also are not closely connected to each other. In fact, only two of its neighbors are close in proximity to each other. On the other hand, *Speed*'s neighbors are often closely related to each other, and make up an easily recognizable genre of tense action movies. In addition, this cluster has two subclusters: political intrigue thrillers (e.g., *Air Force One, Murder at 1600, Executive Decision*, etc.) and the rogue-cop-saves-the-day-again action movie subcluster (e.g., *Die Hard 2, Lethal Weapon 2, Lethal Weapon 3*).

7.4 Showing Proximities via Directed Graphs to Show Temporal Effects

Another example uses data from the Internet Movie Database, or IMDB [IMDB 1990]. Figure 1 (shown in the Introduction) demonstrates how proximity graphs can reveal a community of actors from this dataset, with noticeable subcommunities: the Marx brothers, late 20th century film stars, and singers in movies from an earlier era. Figure 13 shows how drawing a proximity graph as a directed graph can reveal interesting temporal properties. A query connecting Doris Day with Halle Berry is displayed as a directed k-layered graph, where actors are placed as closer to one endpoint or another based on their proximity to those endpoints. Direction on the edges represents increasing distance from the source node (i.e., Doris Day). Our measures are not only able to extract an interesting path from Day to Berry, but also are capable of "layering" the intermediating actors, showing temporal progression from one era to another.

7.5 Proximities For Dense Graphs: Making a Similarity Measure More Reliable by Sparsification

The usual targets of the CFEC proximity measure are datasets associated with sparse similarity functions which are readily modeled as weighted graphs. In other words, we directly measure similarity for only a relatively small number of objects, and CFEC proximity measurement fills in the missing similarity values. Interestingly, a further application of CFEC proximity is for datasets with dense similarity measures, where we are given an explicit similarity value between many or all object pairs. Apparently, a network-based proximity measure such as CFEC should be superfluous, as all similarities are already given. However, often we have confidence only in the highest similarities, whereas most of the given ones are not considered reliable. One example is the *bioPIXIE* dataset [Myers et al. 2005], where pairwise functional relationships between proteins are compiled by combining information from multiple heterogeneous sources. While some evidence for pairwise relationships can be found for almost any two proteins, substantial evidence for functional relationships exists for far fewer protein pairs. Consequently, it was suggested to use only the similarities for which we have high confidence. This way, relationships between protein pairs are defined by a network of reliable relationships.

In such cases, when given a dense similarity measure, we would like to assess the competitiveness of the CFEC proximity measure—based on combining many reliable strong similarities—against a direct, but possibly less reliable,

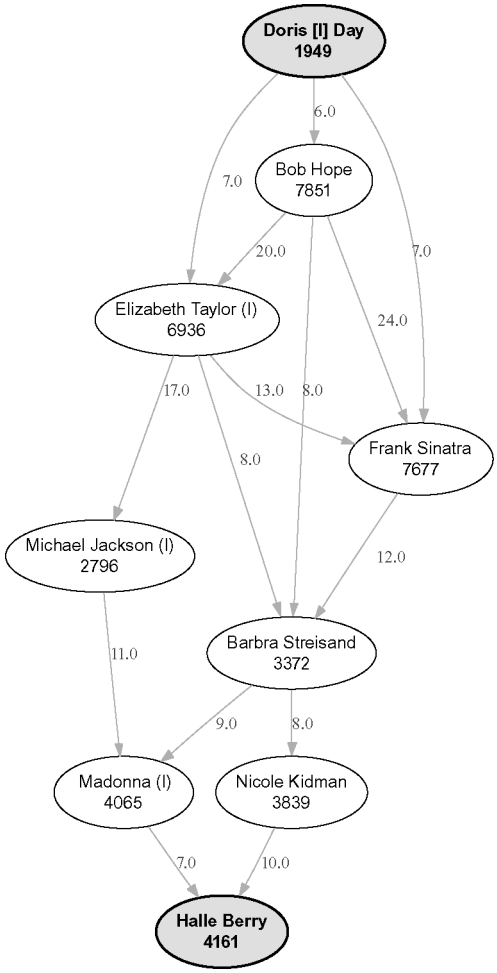

Fig. 13. Proximity graph between Halle Berry and Doris Day, showing the community connecting them as a directed graph. CFEC = 4.97e-7. Subgraph captures 78.3% with $minsize = 8$, $maxsize = 10$.

similarity measure. (This is similar in spirit to local linear embedding algorithms such as Isomap [Tenenbaum et al. 2000], where geodesic distances, computed by summing multiple short, "reliable" distances, are preferred over direct distances.)

Based on the preceding observations, we conducted a study with the Netflix dataset. This dataset enables us to evaluate the performance of the CFEC similarity measure from a unique perspective, by comparing how it fares against direct similarities. Moreover, the Netflix dataset suggests a new application for CFEC proximity: as a way of estimating the values associated with nodes. The Netflix dataset was gathered by the online video rental firm Netflix, and published as part of a contest [Netflix 2007]. The goal of the contest is to accurately predict user ratings on movies. The dataset contains over 100 million ratings

from 480,000 randomly chosen, anonymous Netflix customers on 17,770 movie titles. Each rating is an integer between 1 and 5. We can directly measure the similarity between two movies, as long as they share ratings by the same users. Let us denote the *common support* of two movies i and j as $N(i, j)$, which is the set of users that rated both movies. A suitable measure of distance would be the mean squared error (MSE) between the ratings of the two movies, defined as

$$\mathrm{MSE}(i, j) = \frac{\sum_{u \in N(i,j)} (r_u(i) - r_u(j))^2}{|N(i, j)|},$$

where $r_u(i) \in \{1, \ldots, 5\}$ is the rating of movie i by user u.

Certainly, as $|N(i, j)|$ grows, the resulting $\mathrm{MSE}(i, j)$ is more reliable because it is based on more information. In other words, a distance measured for two movies corated by 1000 users is more significant than if the two had been corated by only 100 users. Consequently, it is crucial to shrink the movie-movie distances towards the global mean as the size of their support set decreases. In practice, this way we can measure the similarity of almost any two movies.

Similarity values directly lead to a rating prediction rule. Given a user u and a movie m such that we want to predict the corresponding rating $r_u(m)$, we can rely on all the movies rated by user u, denoted by $N(u)$. The estimate would be

$$r_u(m) \sim \frac{\sum_{i \in N(u)} \mathrm{sim}(m, i) \cdot r_u(i)}{\sum_{i \in N(u)} \mathrm{sim}(m, i)}, \tag{10}$$

where $\mathrm{sim}(m, i)$ is a nonnegative similarity measure between the movies m and i.

We measured the quality of the prediction rule with a Netflix-supplied probe dataset containing pairs for which we know the true ratings. We selected 100,000 representative pairs out of this probe dataset. Prediction quality is measured as the root mean squared error (RMSE) between predicted and true ratings. Direct similarities can be taken as a decreasing function of the shrunken MSE values. Accordingly, we define $\mathrm{sim}(m, i) = \exp(-\mathrm{MSE}(m, i))$. By applying the prediction rule (10), we predicted scores for the 100,000 probe pairs with a resulting RMSE of **1.01**.

Alternatively, we could rely only on the significant relations, namely those involving movie pairs with low shrunken MSE values, and then compute the similarity between movie pairs using our proximity measure. More specifically, our graph contained 17,770 nodes (corresponding to movies), and each movie was linked to its 50 most similar movies by an edge of weight $\exp(-\mathrm{MSE}(m, i))$. As before, we employed the prediction rule (10) but this time using the CFEC proximity function $\mathrm{sim}(m, i) = \mathrm{CFEC}(m, i)$. This yields better predictions, reflected by a lower RMSE of **0.975**. For comparison, Cinematch, Netflix' proprietary rating system, produced scores for a comparable dataset with an RMSE of **0.951**. It appears that replacing direct with CFEC proximities goes a long way in reducing the RMSE towards the result of the well-tuned Cinematch system, even before applying other potential improvements. Besides being an encouraging indication of the validity of the CFEC proximity measure, the Netflix dataset also provides a new application for network proximity measurement, which we hope to pursue further.

There are some details concerning efficient implementation here. The average number of ratings per user is about 210. Thus, a naive calculation of prediction rule (10) based on CFEC proximity requires, on average, the computation of 210 CFEC proximities, which is excessively expensive. However, since we have at most 5 distinct rating values by a user, we can significantly speed-up the computation by merging all movies that were rated equally by the same user. This way, when we need to estimate the rating of movie m by user u, we merge all nodes corresponding to movies rated 1 by u into a single supernode v_1; similarly, all movies in $\{i|i \in N(u), r_u(i) = \alpha\}$ are merged into a supernode v_α for $\alpha = 1, \ldots, 5$. When merging nodes, the weights of duplicated edges are accumulated. Thus, to estimate $r_u(m)$ we need to compute only 5 CFEC values, which are $\mathrm{CFEC}(m, v_1)$, $\mathrm{CFEC}(m, v_2)$, $\ldots$, $\mathrm{CFEC}(m, v_5)$, and prediction rule (10) becomes

$$r_u(m) \sim \frac{\sum_{\alpha=1}^{5} \mathrm{CFEC}(m, v_\alpha) \cdot \alpha}{\sum_{\alpha=1}^{5} \mathrm{CFEC}(m, v_\alpha)}.$$

As a result, we achieved an average computation time of 3 seconds per estimate on a 2.6 GHz Pentium-4 PC.

8. CONCLUSIONS

Networks are a ubiquitous data representation, motivating the search for good methods of measuring proximities among network objects. We found that obvious, simplistic approaches have limitations that make them inappropriate for many kinds of real-life, ad hoc networks. Consequently, we introduced a proximity measure based on the notion of cycle-free effective conductance (CFEC). CFEC proximity was shown to exhibit many desired characteristics. While no proximity measure can be suitable for all kinds of data, the methodical examination that led to the choice of CFEC proximity suggests that it will be useful for a wide range of networked data. In particular, we observed good results with several practical datasets.

A proximity measure for networked objects can be much more useful in data analysis when accompanied by a way to explain and demonstrate measured proximity values. CFEC proximity allows us to compute proximity graphs, which are small portions of the network that are aimed at capturing proximity values. A significant benefit is that an analyst studying relationships between nodes in a large network can focus on only those small portions of the network that are relevant to the measured proximity. This is far more practical than displaying and manually exploring a network with thousands, or even millions of nodes. Moreover, proximity graphs are of interest beyond their application to explaining measured proximity values. They can be regarded as an extension of the "connection subgraphs" introduced by Faloutsos et al. [2004]. Connection subgraphs provide a compact representation of the relationships between two network objects. Proximity graphs can serve the same purpose, while explicitly capturing CFEC proximity. In this application, proximity graphs have the advantages of showing relationships between more than two endpoints, accommodating paths in directed graphs, and efficient approximation by solving an optimization problem with tunable parameters.

We experimented with four datasets that shared certain properties, such as apparent log-normal distribution of CFEC proximities, small-world phenomena, and the ability to extract proximities using small subgraphs. In the future, we would like to understand what properties would make network datasets unsuitable for CFEC proximity. Also, we would like to study whether "natural" self-organizing networks tend to allow extracting proximity using relatively small subgraphs, as we observed experimentally.

We invite the reader to explore a website that demonstrates proximity graphs at `http://public.research.att.com/~volinsky/cgi-bin/prox/prox.pl`. On this page the user can create proximity graphs from the IMDB or DBLP databases. Queries take only a few seconds to execute, including generation of the candidate graph and creating and rendering the proximity graph. The user can also experiment with different values of the parameters α, $minsize$, and *maxsize*.

REFERENCES

BARABASI, A.-L. AND ALBERT, R. 1999. Emergence of scaling in random networks. *Sci. 286*, 509–512.

BELL, R. M., KOREN, Y., AND VOLINSKY, C. 2007. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

BHATTACHARYA, I. AND GETOOR, L. 2005. Relational clustering for multi-type entity resolution. In *Proceedings of the 11th ACM SIGKDD Workshop on Multi Relational Data Mining*, 3–11.

BOLLOBAS, B. 1998. *Modern Graph Theory*. Springer.

BRANDES, U. AND FLEISCHER, D. 2005. Centrality measures based on current flow. In *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS)*. Lecture Notes in Computer Science, vol. 3404, Springer, 533–544.

COHEN, J. D. 1997. Drawing graphs to convey proximity: An incremental arrangement method. *ACM Trans. Comput.-Hum. Interact. 4,* 3, 197–229.

CORMEN, T. H., LEISERSON, C. L., AND RIVEST, R. L. 1990. *Introduction to Algorithms*. McGraw-Hill/MIT Press.

DBLP. 1998. DBLP computer science bibliography. dblp.uni-trier.de.

DOYLE, P. G. AND SNELL, J. L. 1984. *Random Walks and Electrical Networks*. Mathematical Association of America. http://arxiv.org/abs/math.PR/0001057.

FALOUTSOS, C., MCCURLEY, K. S., AND TOMKINS, A. 2004. Fast discovery of connection subgraphs. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 118–127.

FLAKE, G., LAWRENCE, S., AND GILES, C. L. 2000. Efficient identification of web communities. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Boston, MA, 150–160.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York.

GIBSON, D., KLEINBERG, J. M., AND RAGHAVAN, P. 1998. Inferring Web communities from link topology. In *Proceedings of the 9th ACM Conference on Hypertext and Hypermedia*, Pittsburgh, Pennsylvania, 225–234.

HADJICONSTANTINOU, E. AND CHRISTOFIDES, N. 1999. An efficient implementation of an algorithm for finding k shortest simple paths. *Netw. 34*, 88–101.

IMDB. 2007. Internet Movie Database. www.imdb.com.

JOHN HERSHBERGER, J. M. M. AND SURI, S. 2003. Finding the k shortest simple paths: A new algorithm and its implementation. In *Proceedings of the 5th Workshop on Algorithm Engineering and Experimentation (ALENEX)*. SIAM, 26–36.

KATOH, N., IBARAKI, T., AND MINE, H. 1982. An efficient algorithm for k shortest simple paths. *Netw. 12*, 411–427.

KOREN, Y., NORTH, S. C., AND VOLINSKY, C. 2006. Measuring and extracting proximity in networks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 245–255.

LANG, K. 2004. Finding good nearly balanced cuts in power law graphs. Tech. Rep., Yahoo Research Labs.

LIBEN-NOWELL, D. AND KLEINBERG, J. M. 2003. The link prediction problem for social networks. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, ACM, 556–559.

MYERS, C. L., ROBSON, D., WIBLE, A., HIBBS, M. A., CHIRIAC, C., THEESFELD, C. L., DOLINSKI, K., AND TROYANSKAYA, O. G. 2005. Discovery of biological networks from diverse functional genomic data. *Genome Biol. 6*, R114.

NETFLIX. 2007. Netflix prize. www.netflixprize.com.

POPESCUL, A. AND UNGAR, L. H. 2003. Statistical relational learning for link prediction. In *Proceedings of the Workshop on Learning Statistical Models from Relational Data (IJCAI)*.

SALAKHUTDINOV, R., MNIH, A., AND HINTON, G. 2007. Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, 791–798.

TENENBAUM, J. B., DE SILVA, V., AND LANGFORD, J. C. 2000. A global geometric framework for nonlinear dimensionality reduction. *Sci. 290*, 2319–2323.

TONG, H. AND FALOUTSOS, C. 2006. Center-Piece subgraphs: Problem definition and fast solutions. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 404–413.

WINKLER, W. E. 1999. The state of record linkage and current research problems. Tech. Rep., Statistical Research Division, U.S. Bureau of the Census.

Received December 2006; revised September 2007; accepted September 2007