Measuring and Extracting Proximity in Networks

Yehuda Koren, Stephen C. North and Chris Volinsky
AT&T Labs – Research
180 Park Ave, Florham Park, NJ 07932
{yehuda,north,volinsky}@research.att.com

ABSTRACT

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). Our proximity measure 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. Also, we report experimental results and show examples for three large network data sets: a telecommunications calling graph, the IMDB actors graph, and an academic co-authorship network.

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

Keywords

proximity, random walks, escape probability, cycle-free escape probability, connection subgraph, proximity subgraph

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 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 be used to predict connections in a social network which have not been made yet [18, 19]. In a network with missing data, proximities could potentially find 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 [10, 12].

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 displaying the network which lies between the two objects. Therefore, explaining network-proximity involves showing significantly more information. Recent work by Faloutsos *et al* [9] 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 three real data sets, 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 related subcommunities of performers and their interrelationships. We have produced a website to allow the reader to query this database and create their own proximity graphs at `http://public.research.att.com/~volinsky/cgi-bin/prox/prox.pl`.

2. THE QUEST FOR PROXIMITY

Proximity is a subtle notion, whose definition can depend on a

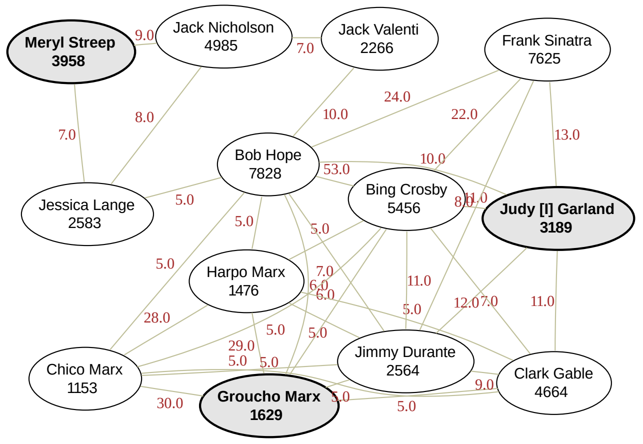

Figure 1: Extraction of proximity graph between Meryl Streep, Judy Garland and Groucho Marx in IMDB.com data. CFEC=4.88e-2. Subgraph captures 97.8% with $maxsize = 12$.

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 potential information exchange between two non-linked 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 co-authorship 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, an idea, or a 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 target individuals 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 one: 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 non-adjacent 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 one 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 the edge weights 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 [6]. 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 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. And 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, and take into account multiple paths between the nodes.

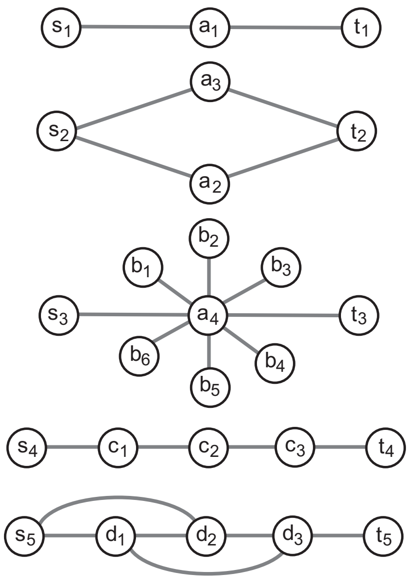

Figure 2: A collection of graphs with all edge weights equal 1

Network flow Consider another concept from graph theory– maximal network flow [6]. 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 increase 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 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 [6]. 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, this 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 the classical graph-theory concepts - modeling the network as an electric circuit by treating the edges as resistors whose conductance is the given edge weights. This way, higher weight edges will conduct more electricity. Descriptions are found in standard references [3, 8]. When dealing with electric 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 was used for such purposes in different occasions, like in the graph-layout algorithm of Cohen [5], or for computing centrality measures in social networks [4]. Faloutsos *et al* [9] also considered 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 the number of alternative paths (more is better, like maximal flow), while avoiding dependence on a single shortest path or a single bottleneck.

An appealing property of EC is that it has an equivalent intuitive definition using random walks. Let's denote 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 [8] that $EC(s,t)$ can be expressed as:

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

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 routes. Consequently, the escape probability decreases if long paths must be followed, because when tracking a long path from s to t there is a good chance of returning to s before reaching t.

EC also has a monotonicity property, courtesy of *Rayleigh's Monotonicity Law* [8]. 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 $EC(s_2,t_2) > EC(s_1,t_1)$, which is intuitive.

While monotonicity it 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 [1] 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* [9] 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. It 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 destroys monotonicity. As the network becomes larger, adding paths between s and t, the proximity between s and t usually *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 the proximity value strongly depends on the graph size, each selected subgraph will convey a different proximity value, usually much larger for smaller subgraphs. There is no way to compute optimal subgraph size or to understand how to normalize for graph size. Moreover, this dependency on size makes proximity comparison across different pairs impossible. Therefore, we assert that while sink-augmented current delivery solves many of the problems previously mentioned, its non-monotonic 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:

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

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

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

These definitions also apply to directed graphs. Recall that Equation (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 above that this is problematic when measuring proximity, as it does not consider any such paths to be distracting ones that lower s-t

proximity. We fix this problem is 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.*

From the information sharing viewpoint it 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 in accordance with Equation (1) as:

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

As usual, 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*.

On undirected graph, random walks have the reversibility property [8] 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 advantages of previous candidates while solving 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:

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

It is clear that a cycle-free escape must proceed along a simple path from s to t, so Equation (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 Equation (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, $\text{CFEC}(s,t) = 2^{-k}$.

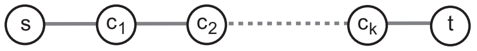

Figure 3: A family of s-t paths

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, being a sum of (positive) weights of all alternative paths. For example, for the graph family in Figure 4, $\text{CFEC}(s,t)$ proximity is $k/2$, growing with the number of alternative paths.

Moreover, unlike graph-theoretic and maximal-flow distances which are dependent respectively on the shortest path and the max cut,

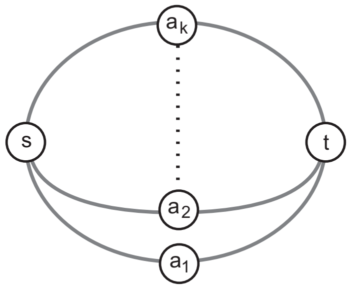

Figure 4: A family of graphs characterized by the number of s-t paths

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 were 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 $1/(k+2)$ 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.

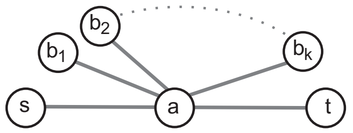

Figure 5: A family of graphs with varying degree of a

The correct treatment of dead end paths means that the CFEC measure is not monotonic with respect to edge addition; e.g., adding degree-1 nodes decreases proximity. However, we avoid the size bias that exists with sink-augmented networks, where subgraphs have smaller proximity values than their corresponding supergraph. To do this, we employ a special operation to create subgraphs, 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 larger 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 $\hat{V} \subset V$, then 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. That is, for each $i \in \hat{V}$: $\deg_i^{\hat{G}} \overset{\text{def}}{=} \deg_i^G$, where superscripts indicate the respective graph. Here, we took the liberty to use 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 CFEC proximity measured on a graph cannot be smaller than the CFEC proximity measured on a subgraph thereof: according to the definition of CFEC in Equation (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. This means that when measuring proximity on a subgraph of the full network, the larger the subgraph, the more accurate the measure becomes, a desirable property.

In practice we have 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. Hence, when an accurate proximity measure is needed, one cannot lose accuracy by working with the largest possible network. Also, it means that 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. Experimentally, we will show later, that in our datasets a very significant portion of the proximity can usually be explained by a subgraph with fewer than 30 nodes. This strongly suggests that the connection subgraphs of Faloutsos [9], and our *proximity graphs*, are very effective in capturing the relationships that determine CFEC proximity. Later 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 to electrical resistor network models, cycle-free effective conductance can naturally accommodate 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 existing efficient algorithm for computing CFEC proximity. However, we can efficiently find accurate approximations. Fortunately, we may not even need much accuracy–experiments show that even estimating merely the order of magnitude of proximity may suffice since proximity values appear to be lognormal distributed (Figure 10). The basis for our approximation is Equation (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 so that all low probability paths cannot add up to form any significant influence. Therefore, we estimate $\mathrm{CFEC}(s,t)$ by restricting the sum in Equation (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 1-1 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:

$$\mathrm{Prob}(R) = C \cdot e^{-\mathrm{Length}(R)}$$

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* [16] describe a very efficient algorithm for undirected graphs, with running time linear in k and log-linear in the graph size: $O(k(|E| + |V| \log |V|))$; some implementation details can be found in [13]. We employ this algorithm to generate the shortest (most probable) simple paths. Paths of monotonically increasing length are generated successively. We stop execution when the path probability drops below a certain threshold. (As mentioned, 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:

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

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. Therefore, one of our main goals was to find ways of extracting and explaining the proximity using a small subgraph that we can easily visualize, which we call *proximity graphs*. Practically, we are looking for a small, readable subgraph of the original network, such that the s-t CFEC proximity measured on this subgraph will be close to the proximity measured on the full network. This is related to the connection subgraphs structure suggested by Faloutsos *et al* [9], which deals with extracting a small subgraph that describes the relationships between two nodes. A distinction from Faloutsos *et al* [9] 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 Equation (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 $B > 0$. That is, 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

> **(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:

> **(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-per-node:

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

$$\frac{\left(\mathrm{CFEC}^{\hat{G}}(s,t)\right)^{\alpha}}{\|\hat{V}\|} \qquad (9)$$

Here, the superscript $^{\hat{G}}$ indicates measuring CFEC on $\hat{G}$. The constant $\alpha \geqslant 0$ determines our preference in the size vs. proximity tradeoff: The greater α is, the more attention we pay to the numerator, that is, to 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 would be merging all k available paths. As we will show in our examples, we find $5 \leqslant \alpha \leqslant 10$ to be effective. Note that by using a ratio, the inefficiencies mentioned above 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 is reasonable to replace $\|\hat{V}\|$ with $\|\hat{E}\|$ (or $\|\hat{E}\| + \|\hat{V}\|$) since high edge density interferes with the readability of the graph layout (though we have not explored this experimentally yet).

An obvious question is how to find a subset of $\mathcal{R}_k$ that solves the above problem. Solving (A1)–(A3) is NP-hard, as they are reducible from the closely related Knapsack problem [11] (with the main difference that here we need to account for possible overlaps between the paths in addition to their size and weight). Therefore, it is very unlikely 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&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&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 Equation (9)) is lower than the current best one, we prune the search, skipping this subset and all subsequent subsets containing it. Detailed pseudocode is given in 6.

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

  $Stack \leftarrow \emptyset$
  $BestSubset \leftarrow \emptyset$
  $Weights \leftarrow \sum_{i=1}^{k} \mathrm{Wgt}(P_i)$

  for $i = 1$ to $k$ do
    $Weights \leftarrow Weights - \mathrm{Wgt}(P_i)$
    for $Subset \in Stack \cup \{\emptyset\}$ do
      $NewSubset \leftarrow Subset \cup P_i$
      $NewSubset.\mathrm{wgt} \leftarrow Subset.\mathrm{wgt} + \mathrm{Wgt}(P_i)$
      if $\mathrm{Score}(NewSubset.\mathrm{wgt} + Weights, \|NewSubset\|) >$
         $\mathrm{Score}(BestSubset.\mathrm{wgt}, \|BestSubset\|)$ then
        $Stack.\mathrm{push}(NewSubset)$
        if $\mathrm{Score}(NewSubset.\mathrm{wgt}, \|NewSubset\|) >$
           $\mathrm{Score}(BestSubset.\mathrm{wgt}, \|BestSubset\|)$ then
          $BestSubset \leftarrow NewSubset$
        end if
      end if
    end for
    return $BestSubset$
  end

Function Score ($weight,\ size$)
  return $\frac{weight^{\alpha}}{size}$   % See Equation (9)
end
```

Figure 6: The branch-and-bound path merging algorithm

When the B&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&B algorithm, we merge all sets whose union is the result returned by the B&B algorithm into a single subset. Then, the algorithm iteratively merges the two sets maximizing (9) (replacing two sets by their union), till 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 cannot improve the B&B result. Also, our implementation allows setting an upper bound ($maxsize$) and a lower bound ($minsize$) to limit the proximity graph's size.

Assessing proximity graphs 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 connection-subgraph 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, 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.

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 is necessary to the algorithm.

A discussion of our handling of multiple endpoints is warranted, as other reasonable strategies could be used. In particular, we do not require the proximity subgraph to connect all endpoints, but only those that show an important contribution to the overall proximity value. Alternatively, one could impose proximity subgraphs to connect all endpoints. Our choice is based on the proximity-based motivation, dictating that only relevant endpoints participate in the revealed subgraph. Accordingly, one has the flexibility to query about proximity relations among many endpoints without being concerned about which of them is really relevant. Then, our algorithm will naturally prune the unrelated endpoints, allowing the user to concentrate on the most useful information. Certainly, in some other applications the user might demand that all endpoints are visible in the proximity subgraph. This is easily achieved by amending the algorithm with a post processing phase, that inserts each of the remaining endpoints using the most probable (i.e. shortest) path connecting it to the computed proximity subgraph.

As mentioned above, CFEC-proximity is also general enough to handle directed edges. Naturally, this is also valid for the connection-subgraph 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 as the algorithm we chose (Katoh *et al* [16]) works only for undirected graphs. An algorithm for computing the k simple shortest paths on a directed graphs was described by Hershberger *et al* in [15]. Its time complexity is $O(k|V|(|E| + |V| \log |V|))$ but Hershberger *et al* claim that we rarely encounter this worst-time complexity and the average case is only $O(k(|E| + |V| \log |V|))$ running time, comparable to the undirected case.

6. WORKING WITH LARGE NETWORKS

Some of the databases we work with are huge. The DBLP co-citation 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 [17]. Computation on such large networks can exceed normal time and space limitations. 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 a 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* [9] 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 relation between our candidate graph and the CFEC proximity measure 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 Equation (3): $\mathrm{Wgt}(P) = \deg_{v_1} \cdot \mathrm{Prob}(P)$. By transforming edge weights into edge lengths as described in Equation (7), the problem becomes: find 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 [6] for computing shortest paths on graphs with non-negative 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 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 path 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 certain limit.

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 $\mathrm{dist}(s, i)$ and $\mathrm{dist}(t, i)$ for each i in the neighborhood. Then we exclude from the neighborhood any i for which $\mathrm{dist}(s,i)+\mathrm{dist}(t,i) > L - \log(\epsilon)$, 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– s and t– defined by the equation $\mathrm{dist}(s,i)+\mathrm{dist}(t,i) \leqslant L - \log(\epsilon)$. 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. EXPERIMENTS

In this section we will apply the CFEC proximity measure to three large network datasets. Each application will highlight specific features of the proximity measure. Our datasets include a telephone call detail dataset, the IMDB movie dataset and the DBLP co-authorship network. Table 1 show some basic statistics about the three datasets, along with some average features for proximity graphs calculated on a random sample of pairs from each dataset. Some interesting features emerge. Although the DBLP network has the smallest average shortest path between two randomly selected members, we were only able to find paths between 54% of the pairs in our sample. Contrast that with the much larger phone network, where we find paths between 90% of our pairs. The phone network also has the smallest median proximity graph size, and the highest proximity values (although since these values depend on the scale of the edge weights, they are not directly comparable across datasets).

Data	Nodes	Edges	Link?	Prox	SPath	GSize
Phone	∼300M	∼1B	90	-6.5	5.8	20
DBLP	438K	1.1M	54	-7.4	4.4	28
IMDB	896K	31M	77	-7.6	7.7	43

Table 1: Statistics for the three experimental data sets. The "Prox" column shows the mean log proximity value. "Link?" reports the percentage of pairs in the sample that had any path between them. The "SPath" column lists the mean shortest path distance from our sample and "GSize" indicates the median proximity graph size for our sample .

Example 1: Telecommunications Graph. In our first example, we apply our proximity measure to a network of telecommunications data. 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 only a record of calls carried on its network, so there is a potential missing data problem which we ignore for the time being. 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. Note that we have no access to content of the communication, 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 quite easily, 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 Equation (9). For the experiments below, 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 mentioned above.

The parameter α (Equation (9)) allows the user to trade-off 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, and hence more of the possible 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 delivered current in the subgraph. A horizontal line is drawn at 95% for 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, sd=16). If we bump up to $\alpha = 50$, almost 100% of our sample capture 95% of the overall proximity, at the cost of larger subgraphs (mean=50, sd=32). We think $\alpha = 10$ gives the best trade-off, yielding graphs small enough to be visualized, so we use this value in all subsequent analysis.

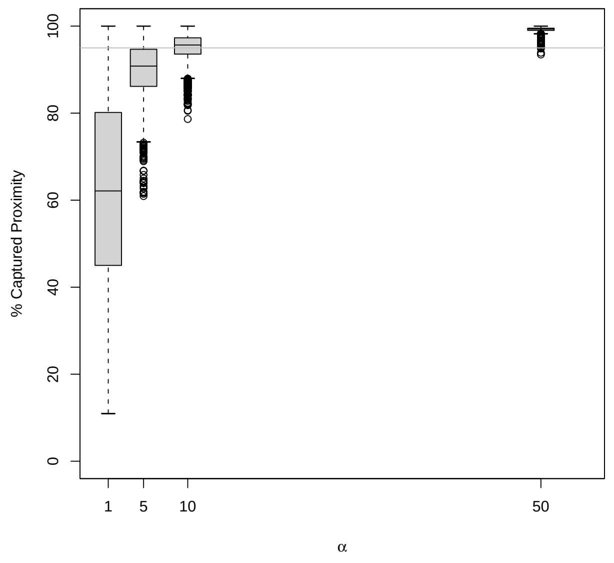

Figure 7: Boxplots showing % delivered proximity as a function of α for a sample of telephone numbers. The box marks the space between the 25th and 75th percentiles of the distribution.

In Figure 8, we explore how large the subgraphs need to be to capture a meaningful percentage of proximity in the subgraph. Looking at the resulting subgraphs for 2000 pairs using $\alpha = 10$, we plot graph size against the percent overall captured proximity in the bottom figure. On top we plot a histogram showing proximity graph size. One can see that the majority of the proximity graphs are 50 nodes or less, which is small enough to allow a good 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 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 our proximity measure and the simple graph-theoretic measure of

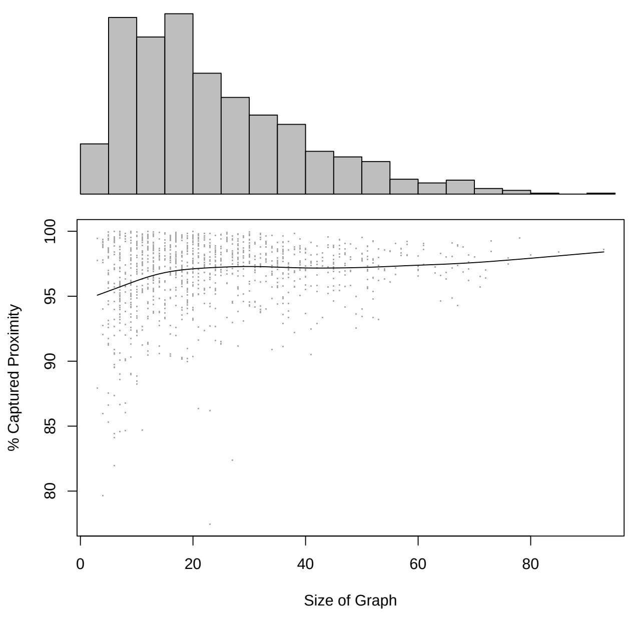

Figure 8: Plot showing size of proximity graphs and their relation to captured proximity. The top plot shows the distribution of graph sizes for our sample of 2000 pairs. The bottom figure plots graph size against the percent captured proximity, with a smoothing spline plotted through the data.

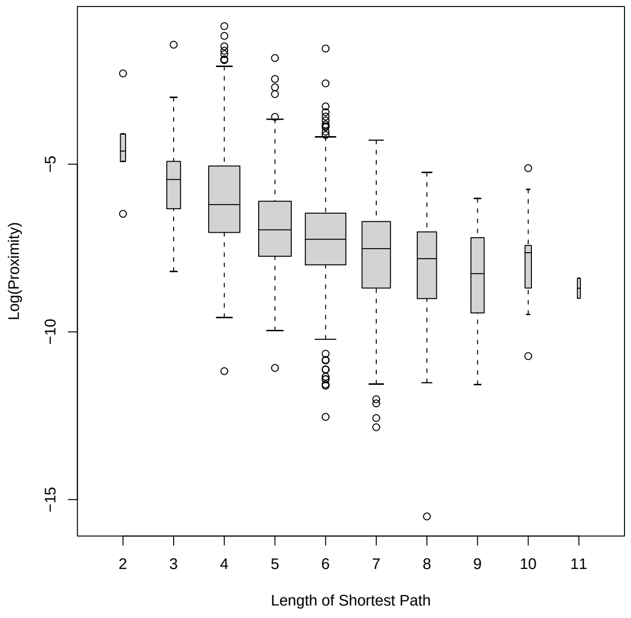

Figure 9: Boxplots of log proximity for each value of shortest path distance. Boxes represent distance between the 25th and 75th percentile of the distribution.

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 the 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 overlap of the boxplots indicate that there may be pairs that 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 the widely held belief that all people are separated by six links, as popularized in the 1993 movie “Six Degrees of Separation”.

Finally, we show the utility of CFEC as a proximity measure in predicting future links in the network. We performed an experiment where we calculated proximity scores for pairs on the network that had never had any direct communication, and then followed these pairs to see which ones communicated during a short time period thereafter. We then compared the proximities for those that communicated with those that did not. If we observe that the proximities for those that ultimately communicate show a closer relationship than for those that did not, we might conclude that our proximity is measuring a commonality, or mutuality, that can predict whether or not these two entities will communicate in the near future.

Our experiment shows that proximity indeed seems to have predictive value (Figure 10). 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 suggest that two nodes with proximity greater than 1 (log proximity > 0) are extremely likely to be future communicators.

Example 2: Co-authorship Graph Our second example applies CFEC proximity to the DBLP database of computer science journals and proceedings [7]. Links in this network refer to co-authors of a paper. Using proximity between known authors in a particular field can identify interesting clusters of researchers in that field. Fig. 11 shows 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 algorithms experts.

Using proximities to find communities in databases like DBLP can also be useful for unaliasing, or correcting errors introduced when the same person shows up with two different labels [2]. 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 [20].

Example 3: IMDB Movie/TV Actor Graph Our final example uses data from the Internet Movie Database, or IMDB [14]. IMDB provides data on movies and television shows and the actors who appeared in those shows. Links in the IMDB graph represent actors who appear together in at least one show. Figure 1 (shown in the Introduction) demonstrates how proximity graphs can extract 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 12 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

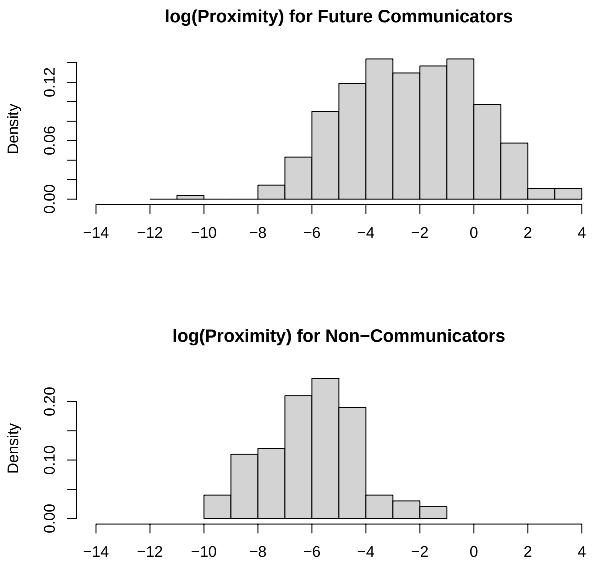

Figure 10: Histograms comparing distribution of logarithm of proximity score for pairs of nodes that communicated (top) and those that did not (bottom).

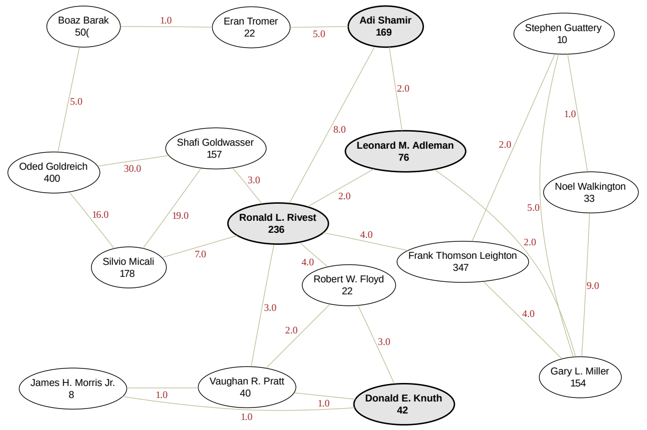

Figure 11: Proximity graph for computer scientists Rivest, Knuth, Shamir and Adelman from the DBLP co-authorship graph. The resultant graph shows other well-known colleagues, resulting in an interesting academic community. CFEC=1.30e+1. Subgraph captures 99.6% with $minsize = 12$.

those endpoints. Direction on the edges represents increasing distance from the source node (Day). Our measures are not only able to extract an interesting path from Day to Berry, but also are able to "layer" the intermediating actors showing temporal progression from one era to another.

8. CONCLUSIONS

Networks are a ubiquitous data representation, motivating suitable methods for measuring proximities among network objects. We have found that obvious approaches have limitations that render 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 rigorous examination that led to our choice of CFEC proximity raises hope that it will be useful for a wide range of networked data. In particular, we observed very encouraging results with three different practical datasets.

A proximity measure of networked objects can be fully utilized only when accompanied by a means for explaining and demonstrating the measured proximity. CFEC proximity allows us to readily compute proximity graphs, which are small portions of the network that are aimed at capturing a related proximity value. This way an analyst studying proximity in a network can focus on only the most relevant portions to the measured proximity value. In fact, proximity graphs are of interest in themselves, beyond their use for explaining a measured proximity value. They can be regarded as an extension of "connection subgraphs" introduced by Faloutsos *et al* [9]. Such connection subgraphs provide a compact representation of the relationships between two network objects. Our proximity graphs can serve the same purpose, while explicitly aiming to capture CFEC proximity. Beside this, other useful properties of our proximity graphs include the ability to show relationships between more than two endpoints, the flexibility to handle edge direction, and the fact that they are obtained by solving an intuitively tunable optimization problem.

We experimented with three 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 might make 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 in our three datasets.

We invite the reader to explore a web site we made to demonstrate 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 and creating and rendering the proximity graph. The user can also experiment with different values of the parameters α, $minsize$, and $maxsize$.

9. REFERENCES

[1] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, October 1999.

[2] I. Bhattacharya and L. Getoor. Relational clustering for multi-type entity resolution. In *Proc. 11th ACM SIGKDD Workshop on Multi Relational Data Mining*, pages 3–11, 2005.

[3] B. Bollobas. *Modern Graph Theory*. Springer-Verlag, 1998.

[4] U. Brandes and D. Fleischer. Centrality measures based on current flow. In *Proc. 22nd Symp. Theoretical Aspects of Computer Science (STACS '05)*, pages 533–544. LNCS 3404, Springer-Verlag, 2005.

[5] J. D. Cohen. Drawing graphs to convey proximity: an incremental arrangement method. *ACM Transactions on Computer-Human Interaction*, 4(3):197–229, 1997.

[6] T. H. Cormen, C. L. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill and the MIT Press, 1990.

[7] DBLP computer science bibliography. dblp.uni-trier.de.

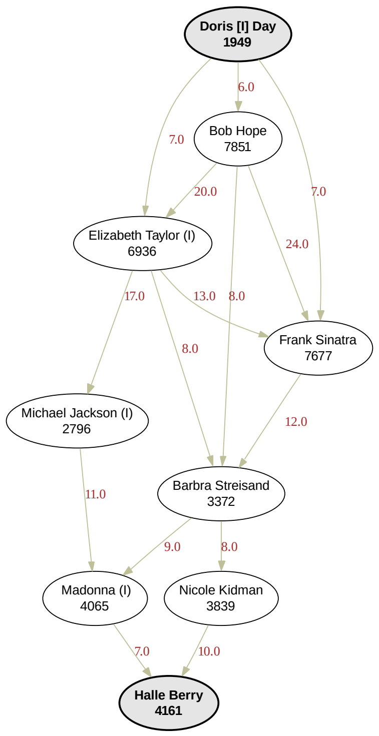

Figure 12: Proximity graph between Halle Berry and Doris Day showing the community connecting them as a directed graph. CFEC=4.97e-7. Subgraph captures 78.3% with $minsize = 8, maxsize = 10$**.**

[8] P. G. Doyle and J. L. Snell. *Random Walks and Electrical Networks*. The Mathematical Association of America, 1984. http://arxiv.org/abs/math.PR/0001057.

[9] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *Proc. 10th ACM SIGKDD conference*, pages 118–127, 2004.

[10] G. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 150–160, Boston, MA, August 20–23 2000.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[12] D. Gibson, J. M. Kleinberg, and P. Raghavan. Inferring Web Communities from Link Topology. In *Proceedings of the 9th ACM Conference on Hypertext and Hypermedia*, pages 225–234, Pittsburgh, Pennsylvania, June 1998.

[13] E. Hadjiconstantinou and N. Christofides. An efficient implementation of an algorithm for finding k shortest simple paths. *Networks*, 34:88–101, 1999.

[14] Internet Movie Database. www.imdb.com.

[15] M. M. John Hershberger and S. Suri. Finding the k shortest simple paths: A new algorithm and its implementation. In *Proc. 5th Workshop on Algorithm Engineering and Experimentation (ALENEX '05)*, pages 26–36. SIAM, 2003.

[16] N. Katoh, T. Ibaraki, and H. Mine. An efficient algorithm for k shortest simple paths. *Networks*, 12:411–427, 1982.

[17] K. Lang. Finding good nearly balanced cuts in power law graphs. Technical report, Yahoo Research Labs, 2004.

[18] D. Liben-Nowell and J. M. Kleinberg. The link prediction problem for social networks. In *CIKM*, pages 556–559. ACM, 2003.

[19] A. Popescul and L. H. Ungar. Statistical relational learning for link prediction. In *Proc. Workshop on Learning Statistical Models from Relational Data (IJCAI '03)*, 2003.

[20] W. E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Bureau of the Census, 1999.