Fast Discovery of Connection Subgraphs

Christos Faloutsos*, Kevin S. McCurley, and Andrew Tomkins
IBM Almaden Research Center

ABSTRACT

We define a *connection subgraph* as a small subgraph of a large graph that best captures the relationship between two nodes. The primary motivation for this work is to provide a paradigm for exploration and knowledge discovery in large social networks graphs. We present a formal definition of this problem, and an ideal solution based on electricity analogues. We then show how to accelerate the computations, to produce approximate, but high-quality connection subgraphs in real time on very large (disk resident) graphs.

We describe our operational prototype, and we demonstrate results on a social network graph derived from the World Wide Web. Our graph contains 15 million nodes and 96 million edges, and our system still produces quality responses within seconds.

1. INTRODUCTION

Suppose we are given a large graph and we are asked to find the relationship between two nodes 'A' and 'B'. For illustration, we shall use a social network as an example of a graph. In the simplest case, the relationship is manifest as an edge in the graph. However, social network graphs are typically sparse, meaning that a vanishing fraction of node pairs actually have an edge between them. Nonetheless, they may be related due to a composition of simple edges: 'A' is related to 'X', and 'X' is related to 'B'. In this case, the relationship might be encapsulated as a path in the graph. In real life, however, the relationship between two people is often multi-faceted; for example, 'A' and 'B' might have the same manager and the same dentist. Moreover, the paths connecting two people may not be vertex-disjoint; for instance, the dentist may also be the sister of 'A', or may be dating the sister of 'A'. Representing the real-life relationship between two nodes in a graph using a single path is inherently limiting for two reasons: first, any automated mechanism to pick the most important path will make mistakes, and showing a subgraph will increase the probability that the critical path is present; and second, there may not be a critical path, as in the example of two people who have written papers together with a multitude of co-authors, rather than a single co-author. In this paper, we address the problem of extracting from very large graphs, a small (amenable to visual inspection) subgraph that best captures the connections between two nodes of the graph.

*On sabbatical from Carnegie-Mellon University

Connection subgraphs are useful in many domains. In a social network setting, connection subgraphs will help us identify the few people most likely to have been infected with a disease (or heard a rumor, or information-leak, or joke). They can also help us spot whether an individual has unexpected ties to any members of a list of individuals. In other domains, connection subgraphs will help us summarize the connection between two web sites using the hyper-link graph; or the connection between two proteins in a metabolic network; or between two genes in a regulatory network.

More formally, the problem of interest is as follows:

Connection Subgraph Problem

Given: an edge-weighted undirected graph $\mathcal{G}$, vertices s and t from $\mathcal{G}$, and an integer budget b

Find: a connected subgraph $\mathcal{H}$ containing s and t and at most b other vertices[1] that maximizes a "goodness" function $g(\mathcal{H})$.

The constraint on b is motivated by limitations on visualization of graphs (*e.g.*, $b \leq 100$). The function g represents the "goodness" of the solution $\mathcal{H}$. If g is the negative sum of edge weights in $\mathcal{H}$, for instance, then the resulting connection subgraph will be the shortest path from s to t; this is a valid answer, but probably not the most illuminating one. Likewise, on the other extreme, if g is the number of edges of H then the connection subgraph will be the densest set of s-t paths; this is again probably not the best answer.

The Connection Subgraph Problem thus has two sub-problems:

- Sub-problem 1: What function g is an appropriate "goodness"?
- Sub-problem 2: How can the subgraph $\mathcal{H}$ maximizing g be found quickly?

In this paper we propose a particular function g, tailored to produce connection subgraphs that capture salient aspects

[1]In the following, the budget on vertices can be replaced with a budget on edges as required by the problem domain.

of relationships in social network graphs (though it also applies to graphs arising from other applications). We also propose algorithms to compute the solution on very large graphs.

Our formulation and upcoming solutions are domain independent, but we illustrate our techniques on a specific data set that we believe has a great deal of interest in itself. Specifically, we used "named-entity" extraction algorithms to derive a *name graph* from the World Wide Web. In this graph, the nodes represent names of people, and there is an edge of weight w between two names if the names appear in close proximity on w different web pages. Our data set contains roughly 15 million distinct names, and about 96 million distinct edges, drawn from over 500 million web pages. The "name graph" is a valuable resource, because it can help us find patterns, outliers, and connections. In Figure 1 we show three connection subgraphs that were produced by an interactive prototype system (described in Section 5.1) that computes and displays good connection subgraphs in a few seconds.

Although the discussion in this paper focuses on connection sub-graphs between persons, we also envision applying it to graphs that describe relationships between arbitrary pairs of entities, including persons, companies, products, organizations, species, documents, web sites, phone numbers, etc. We expect that connection subgraphs will prove widely useful in interactive data exploration systems.

The structure of the rest of the paper is as follows. In Section 2 we describe some related work, and consider the problem of what constitutes a good objective function g. The details of our approach are presented in Sections 3 and 4, with experimental results on the name graph in Section 5. We describe the interactive system in Section 5.1, and summarize our conclusions and suggest future directions in Section 6.

2. RELATED WORK

The heart of this paper is how to find "good" connection subgraphs. To our surprise, we have *not* found any work that directly addresses this problem in the published literature. There are well studied areas in graph theory and networks that have some overlap on the surface, but none attacks this problem. We review this work next, show why some "reasonable" approaches perform poorly, and we briefly review other related work.

The two most natural measures for choosing "good" paths would be the shortest distance, and the maximum flow criterion, in which the edge weights represent a maximum flow on the edge. Both of these fail to capture a natural notion of "best path" in social networks. Consider the graph of Figure 2 with unit weights. In this case, the shortest paths from s to t go through nodes 3 and 4, and both have length 2. Notice however that node 4 has many edges, as would be the case if the node represented a famous person with many incidental connections. Thus, one would intuitively prefer the path through node 3, but this preference is not captured by the traditional shortest path computation. Other distance functions in graphs[19, 22] match more closely with our intuition of which paths are best for conveying a relationship, but it is important to stress that our measure g prefers subgraphs that contain multiple possibly overlapping paths where possible. Our goal is a good subgraph rather than a path or collection of paths.

If we instead treat connection subgraph generation as a

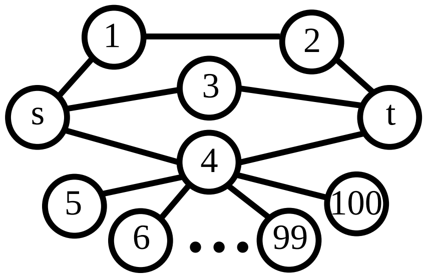


Figure 2: A simple network where both shortest path and network flow fail to adequately model social relationships. With all edges having weight 1, flow fails to distinguish between the paths s,1,2,t and s,3,t, even though the latter is shorter. Total path length fails to distinguish between the paths s,3,t and s,4,t, even though path through 4 is diluted by many extra connections.

maximum flow problem, we find that paths s,1,2,t and s,3,t both carry 1 unit of flow, although we would intuitively prefer the shorter path through node 3, since social relationships tend to blur with distance. Thus both shortest paths and network flow models fail to adequately capture the notion of a "good" path in social networks, although both seem related.

There has been considerable work on community detection [11, 9, 12]. However, reporting the "community" of two remotely related nodes will force us to far exceed our budget b of allowable edges, and in cases where the two people belong to different communities, we are interested in relationships between their communities.

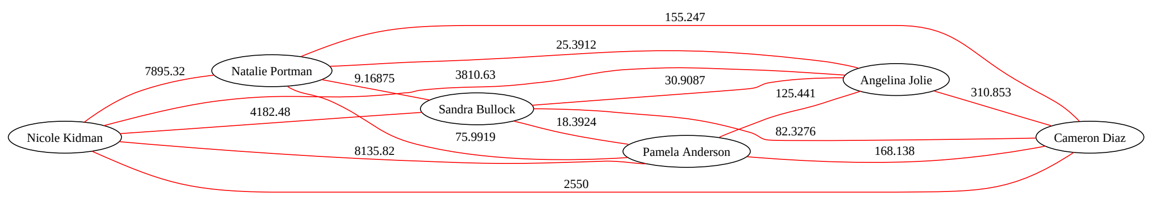
The main problem of this paper is also related to previous work on survivable networks (e.g., see [13]). There, the objective function g is usually expressed as the count of edge-disjoint or vertex-disjoint paths from the source to the destination. This measure also fails to adequately model social relationships, as the two paths in Figure 2 through nodes 3 and 4 have the same survivability: each path 'dies' with the deletion of one node or edge.

Other related work includes the PageRank [21] and personalized PageRank algorithms [15, 16]; graph clustering, partioning, and matrix reordering [2, 4, 17, 23, 24]; electrical circuits and random walks [3, 7, 22]; and influence propagation [18]. "customer value" of a node [5]; and other topics on sparse graphs [1, 6, 8, 20].

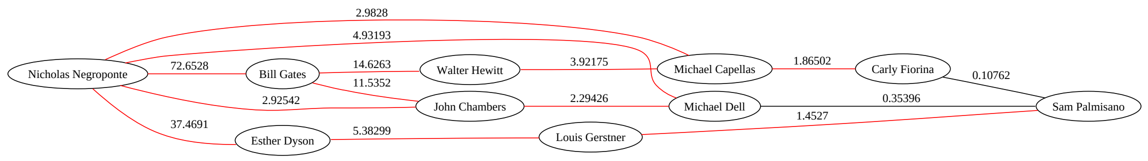
However, again, in all these references, we have not found even the *definition* of the connection subgraph problem, let alone an attempt to solve it.

3. OUR MEASURE OF GOODNESS

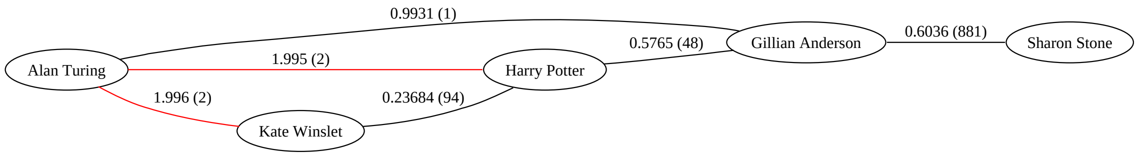
The approach that we propose is related to electrical currents in a network of resistors [7]. Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ denote an undirected weighted graph, and let $C(e)$ denote the weight of the edge e. We then interpret this graph as an electrical network in which each edge e represents a resistor with conductance $C(e)$. In a nutshell, we propose to choose as our connection subgraph the one that can deliver as many units of electrical current as possible. There are a few subtle traps, however, that can lead us astray. Let's start with a review of electricicty laws. Table 1 lists the symbols and



(a) Nicole Kidman to Cameron Diaz



(b) Nicholas Negroponte to Sam Palmisano



(c) Alan Turing to Sharon Stone

Figure 1: Results graphs from the interactive system. Edge weights indicate the relative strength of the connection via our algorithm. In the bottom example, original edge weights are also shown in parentheses.

definitions used throughout the paper.

Symbol	Definition
$\mathcal{G}(\mathcal{V},\mathcal{E})$	an undirected graph
$\mathcal{V}$	set of vertices
$\mathcal{E}$	set of edges
N	Number of nodes
E	Number of edges
$deg(u)$	degree of node u
$V(u)$	Voltage of node u
$I(u,v)$	current on edge $(u\ ,\ v)$
$C(u,v)$	conductance of edge $(u\ ,\ v)$
$C(u)$	$=\sum_v C(u,v)$: conductance of node u
$\hat{I}(\mathcal{P})$	delivered current over "prefix path" $\mathcal{P}$
$CF(\mathcal{H})$	flow captured by subgraph $\mathcal{H}$
s	source node
t	destination node
z	'universal sink' node

Table 1: Symbols and Definitions

Suppose that we apply a voltage of +1 volt to the start node s, and ground (0 volts) on the destination node t. Let $I(u,v)$ be the current flow from u to v and let $V(u)$ denote the voltage at u. Then we have Ohm's law:

$$\forall u,v: I(u,v) = C(u,v)(V(u) - V(v)) \tag{1}$$

and Kirchhoff's current law:

$$\forall v \neq s,t: \quad \sum_u I(u,v) = 0 \tag{2}$$

These laws uniquely determine all the voltages and currents, as the solution to a linear system:

$$V(u) = \sum_v V(v)C(u,v)/C(u) \quad \forall u \neq s,t \tag{3}$$

(where $C(u) = \sum_v C(u,v)$ is the total conductance of edges incident to the node u), with boundary conditions:

$$V(s) = 1, \quad V(t) = 0 \tag{4}$$

The voltages and currents of the resulting network have fascinating connections to quantities related to random walks along the graph. For example

Lemma 1. *(See [7, p. 52]) Consider an electrical network defined by (3), (4). Consider also all random walks on the associated graph that (a) start from the destination node t (b) end on the source node s (c) following an edge (u,v) with a probability that is proportional to its conductance ($C(u,v)$) (d) without revisiting the destination node. (Zero or more intermediate visits to the source node are permitted). Then, the electric current $I(u,v)$ is proportional to the net number of times that such walks will traverse the edge (u,v).*

It is tempting to use this formulation of current flow as our measure of goodness for a connection graph, namely the subgraph of a given size that maximizes the total current $\sum_v I(v,t)$ flowing into the destination node. However, that has a flaw: consider the graph of Figure 2, and compare the two paths $s \to 3 \to t$ and $s \to 4 \to t$. In the above setting, they will both carry the same current of 1/2 Amperes each, while we would like the path through node 3 to be more favorable. To compensate for this, we propose to follow [22], and introduce a universal sink node z that is grounded:

$$V(z) = 0 \tag{5}$$

and is connected to every node u of the graph with an edge of conductance

$$C(u, z) = \alpha \sum_{w \neq z} C(u, w) \tag{6}$$

for some parameter $\alpha > 0$. We used α=1, but nearby choices make little difference. The universal sink absorbs a positive proportion of the current that flows into any given node, in a way reminiscent of 'tax'. Thus, it penalizes a node with high degree, because it taxes it not only directly, but multiple times as well, indirectly, through its neighbors. An extra fringe benefit is that it also heavily penalizes long paths, exactly because it taxes them repeatedly for every node that the path contains.

The intuition of Lemma 1 carries through, with just a few slight modifications, namely, that the random walks are also forbidden from reaching the universal sink. In any case, subgraphs that carry much current are exactly the subgraphs we want to include. More accurately, we want a subgraph that, after the 'taxation' by the universal sink z, is responsible for delivering high current to the sink t. This is the concept of *Delivered Current*, which we formalize next in subsection 4.1.

4. OUR ALGORITHMS

The goodness function $g(\mathcal{H})$ that we propose is exactly the total *delivered current* that the chosen subgraph $\mathcal{H}$ carries from source to sink, after the repeated taxations by the universal sink z. We are now faced with the problem of finding good connection subgraphs under that measure. We can reduce the problem to that of calculating the currents on the original graph, followed by a process that extracts a subgraph that carries high current to t. We refer to the latter problem as that of *display generation*, and we discuss it in detail in Section 4.1. Calculating current flows with a universal sink is feasible even for very large graphs, but not in an interactive environment. In order to address this problem, we propose an optional preprocessing step, called *candidate generation*. The idea is to quickly produce a moderate-sized graph, by removing nodes and edges that are too remote from s and t to influence the solution. In our interactive system, this is what allows us to produce good connection subgraphs within a few seconds. We describe the candidate generation process in Section 4.2.

4.1 Display Generation

The display generator takes as input the weighted, undirected graph $\mathcal{G}$ and the flows $I(u, v)$ on all (u, v) edges, and produces as output a small, unweighted, undirected graph $\mathcal{G}_{\text{disp}}$ ($\equiv \mathcal{H}$) suitable for display to the user. Typically, we expect $\mathcal{G}_{\text{disp}}$ to have 20–30 nodes. Results showing how well this algorithm performs are given in Section 5.

As we mentioned earlier, the proposed goodness measure is the "delivered current" that the chosen subgraph $\mathcal{G}_{\text{disp}}$ carries from source s to sink t. Notice that each atomic unit of flow (i.e., each electron) must travel along a single path; it is thus possible to decompose the flow into paths. This will allow us to formalize the notion of current delivered by a subgraph. We require the following sequence of definitions.

Definition 1. *Node v is* downhill *from u ($u \to_d v$), if $I(u, v) > 0$, or, identically, $V(u) > V(v)$.*

We can then define $I_{out}(u)$, the total flow leaving node u:

Definition 2. *Total out-flow from node u: $I_{out}(u) = \sum_{\{v|u \to v\}} I(u, v)$.*

Definition 3 (Prefix path). *A* prefix path *is any downhill path $\mathcal{P}$ that starts from the source s, that is, $\mathcal{P} = (s = u_1, \ldots, u_i)$ where $u_j \to_d u_{j+1}$*

Obviously, a prefix path has no loops, because of the downhill requirement.

Definition 4 (Delivered current). *The* delivered current *$\hat{I}(\mathcal{P})$ over a prefix-path $\mathcal{P} = (s = u_1, \ldots, u_i)$ is the volume of electrons that arrive at u_i from s, strictly through $\mathcal{P}$. Formally, we define $\hat{I}()$ inductively as follows, beginning with a single edge as base case:*

$$\begin{aligned} \hat{I}(s, u) &= I(s, u) \\ \hat{I}(s = u_1, \ldots, u_i) &= \hat{I}(s = u_1, \ldots, u_{i-1}) \frac{I(u_{i-1}, u_i)}{I_{out}(u_{i-1})} \end{aligned}$$

In words, to estimate the delivered current to node u_i through path $\mathcal{P}$, we are pro-rating the delivered current to node u_{i-1} proportionately to the outgoing current $I(u_{i-1}, u_i)$. We are now ready to define the current delivered by a subgraph; notice that this definition is intentionally quite different from the current delivered by applying voltages and computing current flows on the subgraph alone.

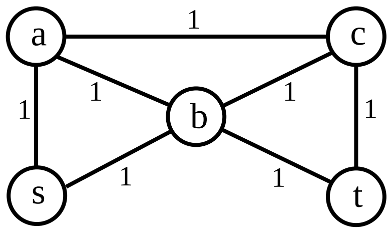
Definition 5 (Captured flow). *We say the* captured flow *$CF(\mathcal{H})$ of a subgraph $\mathcal{H}$ of $\mathcal{G}$ is the total delivered current, summed over all source-sink prefix paths that belong to $\mathcal{H}$.*

$$CF(\mathcal{H}) \equiv g(\mathcal{H}) = \sum_{\mathcal{P}=(s,\ldots,t) \in \mathcal{H}} \hat{I}(\mathcal{P}) \tag{7}$$

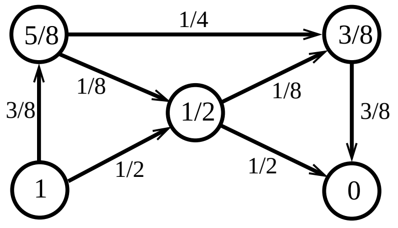
4.1.0.1 Example.

Consider the graph shown in Figure 3. For simplicity of exposition, and without loss of generality, we do not have a universal sink z (that is, we set α=0). After the voltages of the source and sink have been fixed to 1 and 0 respectively, the resulting voltages are shown for each other vertex. These voltages induce currents along each edge as shown. There are five *downhill* source-to-sink paths in the graph. These paths, with their delivered current are shown in Table 2. The path that delivers the most current (and the most current per vertex) is $s \to b \to t$. We can compute the 2/5A delivered by this path by observing that, of the 0.5A that arrive at vertex b on the $s \to b$ edge, 1/5 depart towards vertex c, while 4/5 departs towards vertex t. 4/5 $\times$ 0.5A gives the 2/5A we seek.

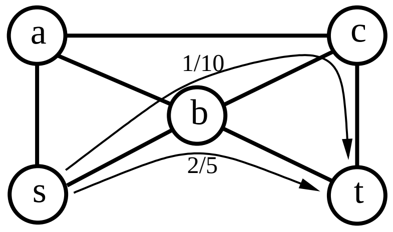
Consider the $\{s, b, c, t\}$ subgraph. We can compute its captured flow by adding the delivered current of all paths that travel exclusively through the subgraph; namely, $s \to b \to c \to t$ and $s \to b \to t$; these paths together capture 2/5 + 1/10 = 0.5A of total current. We observe that this is one of two optimal 4-vertex subgraphs that could be produced.



(a) original network



(b) voltages and amperages



(c) paths with delivered current

Figure 3: A sample network, showing voltages, current, and paths with delivered current.

$s \to b \to t$	2/5
$s \to a \to c \to t$	1/4
$s \to b \to c \to t$	1/10
$s \to a \to b \to t$	1/10
$s \to a \to b \to c \to t$	1/40

Table 2: Current flow along paths in Figure 3

4.1.0.2 Algorithm.

Our optimization problem is now to find a subgraph that maximizes the captured flow over all subgraphs of its size. For this we apply a greedy heuristic, as follows. First, it initializes an output graph to be empty. Next, it iteratively adds end-to-end paths (i.e., from source s to sink t) to the output graph. Since the output graph is growing, a new path may include vertices that are already present in the output graph; the algorithm will favor such paths. Formally, at each step the algorithm adds the path with the highest marginal flow per capita. That is, it chooses the path $\mathcal{P}$ that maximizes the ratio of flow along the path, divided by the number of new vertices that must be added to the output graph.

Notice that the inductive definition of delivered current given above could easily be computed using dynamic programming. We will modify this computation in order to compute the path that maximizes our measure.

We begin with a definition of entries in our dynamic programming table $D_{v,k}$ (for "delivery matrix"), to be interpreted in the context of a partially built output graph.

Definition 6. *$D_{v,k}$ is the current delivered from s to v along the prefix path $\mathcal{P} = (s = u_1, \ldots, u_\ell = v)$ such that:*

1. *$\mathcal{P}$ has exactly k vertices not in the present output graph*
2. *$\mathcal{P}$ delivers the highest current to v among all such paths that end at v.*

To compute D we exploit the fact that the electric current flows $I(*,*)$ form an acyclic graph. Formally, we arrange the vertices into a sequence $u_1 = s, u_2, u_3, \ldots, t = u_n$ such that if node u_j is downhill from u_i ($u_i \to_d u_j$) then u_j follows u_i in our ordering ($i < j$). That is, the nodes are sorted in descending order of voltage, and so electric current always flows from left to right in the ordering. We will fill in the table D in the order given by the topological sort above, guaranteeing that when we compute $D_{v,k}$, we will already have computed $D_{u,*}$ for all $u \to_d v$. The entries of D are computed as follows:

Algorithm 1 (Display Graph Generation).

```
Initialize output graph $\mathcal{G}_{disp}$ to be empty
Let $P$ be the maximum allowable path length
  (trivially, the target size of the display graph)
While output graph is not big enough:
  For $i \leftarrow [1..|\mathcal{G}|]$:
    Let $v = u_i$
    For $k \leftarrow [2..P]$:
      If $v$ is already in the output graph
        $k' = k$
      else $k' = k - 1$
      Let $D_{v,k} = \max_{u|u \to_d v}(D_{u,k'} I(u,v)/I_{out}(u))$
  Add the path maximizing $D_{t,k}/k, k \neq 0$
```

Intuitively, $I(u,v)/I_{out}(u)$ represents the fraction of flow arriving at u that continues to v. Multiplying this by $D_{u,k'}$ gives the total flow that can be delivered to v through a simple path. The path maximizing our measure is then the path that maximizes $D_{t,k}/k$ over all $k \neq 0$; it can be computed by tracing back the maximal value of D from t to s.

4.2 Candidate Generation

As mentioned earlier, computing the voltages and currents on a huge graph can be very expensive, and thus real-time responses are impossible. To create a real-time variant of the system, we propose an optional precursor step which we call *candidate generation*. This step extracts a subgraph of the original graph which we call the *candidate graph*. The extraction algorithm must quickly produce from the original graph a subgraph that contains the most important paths. This subgraph is then treated as the full graph for the remainder of the algorithm: current flows are computed as usual but on the candidate graph, and the display generator is applied to the result.

Formally, the candidate generation process takes a s and t vertex in the original graph $\mathcal{G}$, and produces a much smaller graph ($\mathcal{G}_{\text{cand}}$) by carefully growing neighborhoods around s and t. The focus of the expansion is on recall rather than precision; during display generation we will remove any spurious regions of the graph. As we will show, using candidate generation it is possible to attain performance close to optimal with a latency that is orders of magnitude smaller.

4.2.0.3 *The Algorithm.*

In the framework, candidate generation algorithms strategically expand the neighborhoods of s and t until there is a significant overlap. As the algorithm proceeds, it will expand s, discovering other candidate vertices that it may choose to expand later. Our underlying assumption is that the graph is stored, say, in edge-list format, which makes node expansions inexpensive.

Let $D(s)$ be the set of vertices first discovered through a series of expansions beginning at s; we say that s is the *root* of all vertices in $D(s)$. We define $E(s)$ as the set of *expanded* vertices within $D(s)$; that is, they have been accessed in a data structure, and their neighbors are now known. Likewise, let $P(s)$ be the set of *pending* vertices within $D(s)$ that have not yet been expanded. Similarly, define $D(t), E(t)$, and $P(t)$. Note that $D(s)$ is disjoint from $D(t)$ since each vertex is discovered only once, by expanding a vertex whose root is either s or t. Recall that for weighted graphs, we use $C(u, v)$ as the weight of the edge from u to v. We define deg(u) to be the degree (number of neighbors) of u. Algorithm 2 gives the high level pseudocode.

ALGORITHM 2 (CANDIDATE GENERATION). *Given a graph $\mathcal{G}$ that is weighted and undirected, and two vertices s and t, find $\mathcal{G}_{cand} \subset \mathcal{G}$ which is much smaller than $\mathcal{G}$ but contains most of the interesting connections between s and t.*

```
Set $P(s) = \{s\}$ and $P(t) = \{t\}$.
While not stoppingCondition():
    // pick v, the most promising node of $P(s) \cup P(t)$
    $v \leftarrow$ pickHeuristic()
    // and expand it
    Let $r$ be the root of $v$
    Expand $v$, moving it from $P(r)$ to $E(r)$
    Add all new neighbors of $v$ to $P(r)$
```

Thus, the details of the algorithm lie in the process of deciding which node to expand next, and when to terminate expansion. Our algorithm repeatedly expands carefully selected unexpanded nodes, chosen by the *pickHeuristic()*, until a stopping condition *stoppingCondition()* is reached. These are the two major routines, and we describe a specific heuristic for them in the Appendix. In effect, *pickHeuristic()* strives to suggest a node for expansion, estimating how much delivered current this node will carry. Thus, the heuristic favors nodes that are (a) close to the source s or the sink t (b) with strong connections (high conductance) and (c) low degree, to avoid the 'famous-node' effect (recall node 4 of Figure 2).

The *stoppingCondition()* puts limits on the size of the output graph $\mathcal{G}_{\text{cand}}$ (count of expansions, count of distinct nodes discovered, etc).

4.3 Computational Complexity

The calculation of currents on a network with a universal sink requires the solution of the linear system (3) and (4). For a graph with N nodes and E edges, this can be done by direct methods in $O(N^3)$ operations, but iterative methods will often perform much better on sparse graphs. For a graph with E edges, we would expect to perform $O(E)$ operations per iteration, and the number of iterations depends on the gap between the largest eigenvalue and the second largest eigenvalue. In the case of the names graph we observed a fairly rapid rate of convergence, and this can be expected for many other social network graphs as well.

The display generator takes $O(ekb)$ time, and $O(vk)$ space, where v is the number of vertices in the input graph, e is the number of edges, k is the maximum length of any allowed s-t path, and b is the budget, or desired number of vertices in the display graph.

The candidate generator runs until its termination conditions are met, performing a single disk seek per expansion. Timings are provided in the appendix. Typically in our interactive system, the candidate generator requires 1-1.5 seconds to run, on a graph with $\approx 10^5$ edges.

5. EXPERIMENTAL RESULTS

In this section we describe components of our prototype system and evaluate the performance of the various components. First, we give the experimental set up (data and queries), and then we describe our results. Our experiments were designed to answer the following questions:

- How good is the proposed "goodness" function $g(\mathcal{H})$?
- Does our display generator algorithm capture most of the delivered flow?
- How well does the candidate generation algorithm perform, and which settings of parameters work best?

5.1 Experimental setup

We started from a text analytics system called WebFountain[14] that has been under development at IBM Almaden Research Center. This system routinely crawls the web and collects documents, which are further processed ("named-entity" extraction modules, cleanup of homonyms, synonyms etc.). Our "name graph" was derived from approximately 500 million web pages, by spotting names, and adding an edge between them whenever two names occurred within a window of approximately ten words of one another. If the pair of names generates edges from w distinct webpages, then this gets collapsed to a single edge of weight w. The resulting graph contains N=15,020,632 names, with E=96,689,078 unique edges between them.

Software architecture. Our system was implemented with a web-based interface, in perl, python, C++, Apache, and php, and it runs on a Pentium class machine with a 2GHz processor running Linux 2.4. Users can submit pairs of names from our web interface. The system then runs the 'candidate generation', the voltage computations and the 'display generation', and displays the resulting connection graphs. We used the GraphViz system [10] for the graph-layouts. Moreover, our system provides click-able nodes and edges: on a click our system displays additional information (names of neighbors, web site for this person, etc)

Query pairs. In order to test our algorithms, we selected a set of ten computer scientists and mathematicians, and a set of seven actors and actresses. We defined the query pairs shown in Table 3 for use in the experiments. We also considered three termination conditions for the candidate generator, *C-small*, *C-medium*, and *C-large*, which result in small, medium and large candidate graphs $\mathcal{G}_{\text{cand}}$. Their exact details are in the Appendix.

5.2 Goodness of g — case studies

Name	Description	# pairs
AA	Both nodes are actors/actresses	45
CSM	Both nodes are CS/mathematicians	21
Cross	One node from each set	70

Table 3: Query pairs for experimental evaluation.

Figure 1 shows the graphs output by our system for three representative test cases. Figure 1(a) shows a small display graph linking two movie actresses, Nicole Kidman and Cameron Diaz. In this example there are strong links, with high current, as expected.

Figure 1(b) shows a 10-node display graph for the connections from MIT Professor Nicholas Negroponte to IBM CEO Sam Palmisano. In this case the strongest connection to Negroponte is through Esther Dyson, as evidenced by the fact that they have both published articles in Wired Magazine, have been mentioned together in the New York Times, and have authored books that are compared to each other in Amazon. The second node in the strongest path is Louis Gerstner, the IBM CEO that Palmisano replaced. The rest of the paths are weaker, involving the CEOs of most major computer companies: HP (Fiorina), Microsoft (Gates), Cisco (Chambers), and Dell (Dell). Notice that our goodness function led to results that make intuitive sense: the co-occurrences of Palmisano with the other CEOs is expected, but not as strong and informative as the connection with Dyson.

The final example 1(c) shows the network connecting Alan Turing to Sharon Stone, who are two people from largely disjoint communities. Without looking at the graph, one would expect to find weak paths, if any at all. The result is surprising: there *is* a connection, and there is even a fairly strong connection through the actress Kate Winslet, because she starred in a movie about the Engima cipher machine, in which Alan Turing played a part durign his lifetime. We also see a connection through Gillian Anderson because she stars in a science fiction television show that is popular among a technical audience. We note also that Alan Turing has direct connections to Alan Thicke, Alan Alda, and Bruce Lee (all of whom have direct connections to Sharon Stone), but these edges were discarded as carrying too little current. This example is also interesting in that they come from distinct communities, and both have high degree (Turing has 1,249 neighbors and Stone has 6,014 neighbors). Again, our goodness function $g()$ led to results that are sensible and revealing.

5.3 Evaluation of Display Generation

In this section, we evaluate the performance of the display generator by measuring the delivered current as a function of the budget b of allowable nodes.

Figure 4 shows the fraction of delivered current as a function of the size of the display graph for four representative examples. The candidate generator for these examples was run using a stopping condition that resulting in 15–25K total edges in the candidate graph. As the figure shows, the curve quickly flattens, and a reasonably-sized display graph delivers the vast majority of the total current.

Table 4 shows aggregate results over a larger set of experiments and candidate generator stopping conditions. Table 4(b) shows total current delivered in the candidate graph, and then in parentheses the fraction of that current delivered in a 20-node display graph. As stopping conditions change and the expansion algorithm is allowed to proceed further, the display generator captures a smaller fraction of the current. A more surprising bias arises from the choice of dataset. For Actors and Actresses, the display generator captures the vast majority of the flow, suggesting that while this neighborhood is extremely dense, nonetheless there are a few nodes that are responsible for most of the flow. Computer Scientists and Mathematicians fare slightly worse; for condition *C-medium*, 87% of the flow is captured on average, down from 99%. The Cross case is again slightly worse, validating the intuition that relationships between nodes that are not naturally related come about due to a larger number of serendipitous low-flow paths. Nonetheless, overall, the display generator is able to capture the vast majority of current with a relatively small output graph.

Over all these experiments, the first b=20 nodes routinely carry most of the current. To conclude, the display generator seems to be doing a good job of capturing flow from the much larger candidate graph.

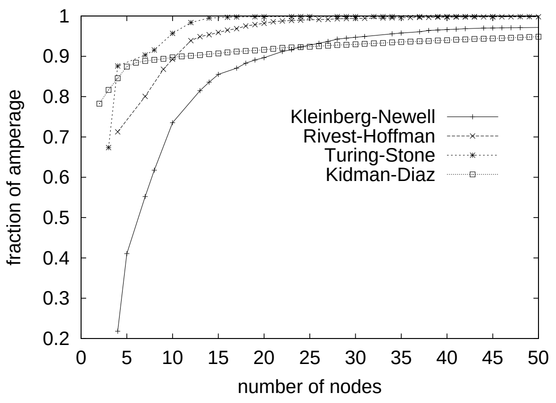


Figure 4: Fraction of amperage captured by display generator as a function of b (budget of allowed count of nodes). Notice the sharp rise, and the diminishing returns, in all cases.

	AA	CSM	Cross
C-small	3.97	2.71	3.32
C-medium	4.59	6.4	5.08
C-large	28.68	24.23	24.32

(a) Total Elapsed Time (sec)

	AA	CSM	Cross
C-small	618.47(99)	1.31(91)	0.76(86)
C-medium	625.52(99)	1.27(87)	0.78(83)
C-large	727.26(94)	1.23(83)	0.64(79)

(b) Current Measurements (Amperes)

Table 4: Comparison of times and currents across three datasets and three stopping conditions. Values in parentheses represent percentage of total current delivered by display graph of size 20.

5.4 Evaluation of Candidate Generation Heuristics

In this section we summarize the results of experiments on different distance measures and stopping conditions for the candidate graph. Details are given in the appendix. The *stoppingCondition()* shows exactly the diminishing returns that we observed above; the heuristics for node expansion (*pickHeuristic()*) usually perform about the same, with the surprising exception that one natural-sounding heuristic performs quite poorly—see Section A.2 for details.

Stopping Conditions. Table 4 shows the average runtime of the algorithm for each data set and each stopping condition. Most interestingly, more resources help for Actors & Actresses, but not Computer Scientists & Mathematicians or for relationships between the two groups ("Cross"). Nodes in the AA region of the graph tend to have very high degrees, and may therefore require significantly more expansion to find the good paths. For the other regions, it is possible to find the most important paths with significantly less computational effort. The timings given in the figure are for all three stages of the algorithm in aggregate. A deeper exploration of the timing details shows that candidate generation typically requires more than 50% of the overall effort, with the remainder roughly split between voltage computation and display generation. As a takeaway, it is quite feasible to find "good" graphs in the most important CMS case (representing well-connected individuals without a massive media presence) over a 100M edge graph in under three seconds on a single machine without careful code tuning or optimization.

6. CONCLUSIONS

In this work, we defined and addressed the problem of "Connection Graphs", small graphs that convey much information about the relationship of a pair of nodes. In addition to posing the problem, additional contributions are the following:

- We proposed the "delivered current", a novel, intuitive way to measure the goodness of a "Connection Graph". We showed that straightforward methods like network flow and traditional shortest paths lead to poor, counterintuitive answers while our measure naturally gives high preference to paths that are more likely to occur in a random walk from the source to the sink (with the very careful addition of a "universal sink" node).
- We provide the *display graph generation* algorithm (Algorithm 1), a dynamic-programming algorithm that attempts to find the best "Connection Graph" with $\leq b$ nodes
- We provide the *candidate graph generation* algorithm (Algorithm 2), with fast heuristics that can handle huge, disk-resident graphs, in near-real time, while still maintaining high accuracy.

Moreover, we implemented our algorithms in a working prototype, complete with an interactive web-based interface, on a real graph that we derived from the Web. The graph has 15 Million nodes and 96 Million edges.

Directions for future research include generalization to graphs that contain more than one type of entity (eg., 'people', 'companies' and 'products'); and generalization to connection subgraphs between more than two nodes.

Acknowledgements: We would like to thank Madhukar Korupolu for his help with the *name graph* dataset and his observations on the Display Generator, and Chris Palmer for his feedback on the voltage computation.

7. REFERENCES

[1] R. Albert, H. Jeong, and A.-L. Barabasi. Diameter of the world wide web. *Nature*, (401):130–131, 1999.

[2] U. Brandes, M. Gaertler, and D. Wagner. Experiments on graph clustering algorithms. In *Proc. 11th Europ. Symp. Algorithms (ESA '03), LNCS 2832*, pages 568–579. Springer-Verlag, 2003.

[3] A. K. Chandra, P. Raghavan, W. L. Ruzzo, R. Smolensky, and P. Tiwari. The electrical resistance of a graph captures its commute and cover times. *STOC*, pages 574–586, 1989.

[4] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proc. ACM KDD*, Washington, DC, August 24-27 2003.

[5] P. Domingos and M. Richardson. Mining the network value of customers. *Proc. ACM KDD*, pages 57–66, 2001.

[6] S. Dorogovtsev and J. Mendes. Evolution of networks. *Advances in Physics*, 51:1079–1187, 2002.

[7] P. Doyle and J. Snell. *Random walks and electric networks*, volume 22. Mathematical Association America, New York, 1984.

[8] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *SIGCOMM*, pages 251–262, Aug-Sept. 1999.

[9] G. Flake, S. Lawrence, C. L. Giles, and F. Coetzee. Self-organization and identification of web communities. *IEEE Computer*, 35(3), Mar. 2002.

[10] E. . Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience*, 30:1203–1233, 1999.

[11] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *Ninth ACM Conference on Hypertext and Hypermedia*, pages 225–234, New York, 1998.

[12] M. Girvan and M. E. J. Newman. Community structure is social and biological networks.

[13] M. Grötschel, C. L. Monma, and M. Stoer. Design of survivable networks. In *Handbooks in Operations Research and Management Science 7: Network Models*. North Holland, 1993.

[14] D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Zien. How to build a WebFountain: An architecture for very large-scale text analytics. *IBM Systems Journal*, 43(1):64–77, 2004.

[15] T. H. Haveliwala. Topic-sensitive pagerank. *Proc. WWW*, pages 517–526, 2002.

[16] G. Jeh and J. Widom. Scaling personalized web search. *WWW*, pages 271–279, 2003.

[17] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning for irregular graphs. *SIAM Review*, 41(2):278–300, 1999.

[18] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing

the spread of influence through a social network. In *Proc. ACM KDD*, Washington, DC, 2003.

[19] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proc. CIKM*, 2003.

[20] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[21] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[22] C. R. Palmer and C. Faloutsos. Electricity based external similarity of categorical attributes. Seoul, South Korea, April-May 2003.

[23] S. van Dongen. *Graph Clustering by Flow Simulation.* Ph.D. thesis, University of Utrecht, May 2000.

[24] S. Virtanen. Clustering the chilean web. *LA-WEB 2003*, Nov. 2003.

APPENDIX

A. DETAILS ON CANDIDATE GENERATION

We begin by describing the particular heuristics used in the candidate generator, after which we then report on some experiments comparing different parameter settings, and draw conclusions regarding the appropriate choice.

A.1 Heuristics and justifications

A.1.1 pickHeuristic()

Recall that the *pickHeuristic()* function chooses the next node to expand during candidate generation. We do this within a framework based on a distance function on the in-process candidate graph. Among the pending nodes, we always choose for expansion the one that is closest to its root, in some sense. There are several reasonable ways to define closeness. We introduce a (possibly asymmetric) length on edges, and define the distance between two nodes u and v as the minimum over all paths from u to v of the sum of the lengths of the edges along the path. Thus, the decision about what to expand next is encoded as a weighted, directed, graph distance. This formulation tends to focus on particular types of expansions, and does not allow others, but has the advantage that we know the exact shortest path from any expanded node to its root.

We considered eight definitions of the length of an edge from u to v, based on three flags that can each be set two ways. Generally, the distance is given by $f(n/d)$, where the three flags control the values of f, n, and d, as follows:

Numerator: If the distance is *degree-weighted* then $n = \deg^2(u)$, otherwise $n = \deg(u)$.

Denominator: If the distance is *count-weighted* then $d = C(u,v)^2$, otherwise $d = C(u,v)$

Multiplicative: If the distance is *multiplicative* then $f(x) = \log(x)$, else $f(x) = x$.

Thus, the basic distance function is $d(u)/C(u,v)$, and the degree-weighted, count-weighted, multiplicative distance function is $\log(\deg^2(u)/C(u,v)^2)$.

We give a brief intuition for these definitions. If the graph were not weighted then the basic distance function would be $\deg(u)$. A more natural measure might be $\deg(u) + \deg(v)$, but we haven't yet expanded v so we don't have access to its degree; thus, we cannot include the $\deg(v)$ term in our definition, and we must instead employ an asymmetric distance. The distance function as given treats lower-degree nodes as closer, so the expansion is designed to discover longer paths through low-degree nodes rather than shorter paths through high-degree nodes. Recall however that our graph is weighted, and that nodes with high weight edges should be considered close together because they have a relatively strong connection. This explains the presence of the term $C(u,v)$, corresponding to the weight of the edge.

Finally, we motivate the notion of multiplicative distance rather than traditional additive distance. By taking the logarithm of the edge weight and adding these values along a path, we compute the logarithm of the product; since the logarithm is monotonically increasing, comparisons of path lengths result as they would for multiplication of edge weights. Multiplication is a reasonable model here for the following reason. Consider a path in which all edges have weight 1. If the degrees of vertices along the path are $d_1, d_2, \ldots, d_k$ then the number of vertices reachable by expanding all paths of the given length in a tree with branching factor d_i at level i would be $R = \prod_i d_i$. If the sink were uniformly located among all such nodes, the probabilty of reaching the sink would be proportional to R. Thus, in an idealized model, lower multiplicative distance represents nodes that are "closer" to the root in the sense that a sequence of expansions with the given degree would reach a smaller set of vertices.

Results for these various distance functions are shown in Section A.2.

A.1.2 Termination condition - stoppingCondition()

Finally, we must discuss termination conditions. We define three thresholds for termination; the algorithm will stop as soon as any threshold is exceeded. First, we adopt a threshold on total expansions, to limit the total number of disk accesses. Second, we adopt a larger threshold on discovered vertices, even if those vertices have not yet been expanded, to limit memory usage. And finally, we adopt a threshold on number of cut edges (edges between $D(s)$ and $D(t)$), as a measure of the connectedness of the set of nodes with the src as a root with the set of nodes with the sink as a root.

This completely characterizes the candidate generation algorithm.

A.2 Evaluation of distance functions

As mentioned, we considered three termination conditions, *C-small*, *C-medium*, and *C-large*, which result in small, medium and large candidate graphs $\mathcal{G}_{\text{cand}}$. Recall that the algorithm terminates when any of three thresholds is exceeded: the number of cut edges, the number of expanded vertices, and the number of discovered vertices. The termination conditions we considered are described in Table 6. Finally, we considered all eight distance measures; the eight different measures are derived by turning on or off each of three different settings: degree-weighted, count-weighted, and multiplicative.

The number of cases in the overall experimental design is therefore $8 \times 3 \times (45 + 21 + 70) = 3264$. For each case, we ran the candidate generator and measured wallclock time and number of edges in the resulting graph. We then ran the voltage computation on the candidate graph and measured

	Additive				Multiplicative			
	deg		deg^2		deg		deg^2	
Case	w_{uv}	w_{uv}^2	w_{uv}	w_{uv}^2	w_{uv}	w_{uv}^2	w_{uv}	w_{uv}^2
C-small(AA)	620.97(99)	620.97(99)	620.97(99)	620.97(99)	630.29(100)	582.32(99)	620.97(99)	630.29(100)
C-small(CSM)	1.39(89)	1.28(95)	1.42(91)	1.42(91)	1.4(88)	0.68(98)	1.49(84)	1.4(88)
C-small(Cross)	0.84(85)	0.83(86)	0.56(91)	0.84(85)	0.91(83)	0.52(89)	0.69(88)	0.91(83)
C-medium(AA)	620.97(99)	620.97(99)	620.97(99)	620.97(99)	642.71(99)	613.88(99)	620.97(99)	642.71(99)
C-medium(CSM)	1.32(87)	1.23(94)	1.37(89)	1.35(87)	1.36(84)	0.78(98)	1.37(76)	1.36(84)
C-medium(Cross)	0.78(85)	0.76(86)	0.52(85)	0.78(85)	0.85(83)	1.06(72)	0.64(81)	0.85(83)
C-large(AA)	745.47(93)	742.35(93)	736.63(93)	741.8(94)	732.1(94)	634.52(99)	753.11(92)	732.1(94)
C-large(CSM)	1.21(83)	1.2(91)	1.28(88)	1.25(87)	1.23(80)	1.21(86)	1.21(66)	1.23(80)
C-large(Cross)	0.59(86)	0.68(85)	0.39(79)	0.58(85)	0.48(85)	1.56(57)	0.35(72)	0.48(85)

Table 5: Comparison of distance measures. Columns represent distance measures, as defined in Section A.1.1. Rows represent stopping conditions and datasets. Values in each cell are the current delivered through the candidate graph, with the value in parentheses representing percentage of this current captured in the display graph.

Condition	Cut edges	Expanded	Known
C-small	500	500	10000
C-medium	2000	2000	20000
C-large	10000	50000	1000000

Table 6: Candidate Generation termination conditions

wallclock time and total current. Finally, we ran the display generator and measured wallclock time and total captured current in the display graph.

A.2.1 Distance Measures

Table 5 compares total current delivered across the eight different distance measures we employed.

First, we consider distance measures for candidate generation, show in Table 5. We observe that as the algorithm is given more resources (ie, as the stopping condition changes), the best distance measure also changes. In fact, there are cases where normal or count-weighted, normal or degree-weighted, and additive or multiplicative distance measures are preferred. However, there are a few specific recommendations that we can make.

First, we consider our three data cases, which represent three common types of queries:

CSM: Source and sink are connected, and live in a network of medium to sparse connectivity. This is the most common case for applications of the algorithm. In this case, degree-weighted additive distance performs best, but in fact all measures perform comparably with one exception: count-weighted multiplicative distance performs horribly in stopping conditions *C-small* and *C-medium*. This suggests that when resources are limited, a few very strong edges may initially bias the search in inappropriate directions.

AA: Individuals are connected, and live in a very dense network. In all stopping conditions, the eight measures perform within 5% of one another. The very low-resource *C-small* performs as well as *C-medium*, but the much higher resource thresholds of *C-large* finds graphs that are roughly 15% better in terms of delivered current. Offsetting this improved current, however, is an average running time that increases by a factor of 3-10.

Cross: Source and sink have no "natural" connections. In this case, count-weighted multiplicative distance dramatically outperforms all other measures in stopping conditions *C-medium* and *C-large*, suggesting that focusing heavily on stronger paths may be the best way to find connections between a source and sink that are simply not well-connected.

Overall, we observe that the simplest distance measure ("normal" distance) never performs the best, and normal multiplicative distance performs well in all cases except for the somewhat unusual high-resource Cross condition in which count-weighting should be introduced.