Computational Methods for Dynamic Graphs

Corinna Cortes, Daryl Pregibon and Chris Volinsky [*]

AT&T Shannon Labs

November 24, 2003

[*] All authors are members of research staff at AT&T Labs, 180 Park Avenue, Florham Park, New Jersey 07934.

Abstract

We consider problems that can be characterized by large dynamic graphs. Communication networks provide the prototypical example of such problems where nodes in the graph are network IDs and the edges represent communication between pairs of network IDs. In such graphs, nodes and edges appear and disappear through time so that methods that apply to static graphs are not sufficient. Our definition of a dynamic graph is procedural. We introduce a data structure and an updating scheme that captures, in an approximate sense, the graph and its evolution through time. The data structure arises from a bottom-up representation of the large graph as the union of small subgraphs centered on every node. These subgraphs are interesting in their own right and can be enhanced to form what we call Communities of Interest (COI). We discuss an application in the area of telecommunications fraud detection to help motivate the ideas.

Keywords: transactional data streams, dynamic graphs, approximate subgraphs, exponential averaging, fraud detection.

1 Introduction

Transactional data consists of records of interactions between pairs of entities occurring over time. For example, a sequence of credit card transactions consists of purchases of retail goods by individual consumers from individual merchants. Transactional data can be represented by a graph where the nodes represent the transactors and the edges represent the interactions between pairs of transactors. Viewed in this way, interesting new questions can be posed concerning the connectivity of nodes, the presence of atomic subgraphs, or whether the graph structure leads to the identification and characterization of "interesting" nodes. For example, Kleinberg[9] introduces the notion of "hubs" and "authorities" as interesting nodes on the Internet. The data used by Kleinberg differ significantly from the data we consider in that he uses static links to induce a graph over WWW pages. We use actual network traffic, as captured by interactions between pairs of transactors, to define our graph. Since nodes and edges appear and disappear through time, the graph we consider is dynamic.

There are many challenging issues that arise for dynamic graphs and we have used a specific application to focus our research, namely the graph induced by calls carried on a large telecommunications network. This application is interesting, both because of its size (*i.e.,* hundreds of millions of nodes and edges) and its rate of change (*i.e.,* hundreds of thousands of new nodes and edges each day). Like all networks, it is also diverse in the sense that some nodes are relatively inactive while others are superactive.

The paper is organized as follows. Section 2 illustrates characteristics of large dynamic graphs using network traffic from a specific telecommunications service. Section 3 introduces the definition of a dynamic graph that we adopt and discusses the computational and statistical features of various alternatives. Section 4 describes an approximation to facilitate both interpretation and large scale computations, including updating and maintaining the dynamic graph over time. Section 5 introduces an example that illustrates how these subgraphs are used in practice. In Section 6 we expand on these notions and consider approximate subgraphs centered on nodes that can be further enhanced or pruned to define communities of interest. Section 7 discusses related work in and outside the data mining community. The final section summarizes the findings and discusses future work.

2 Motivation

The large graphs that we are concerned with are defined from records of transactions on large telecommunications networks. We believe that characteristics of such graphs are shared by other large financial and data networks, including the Internet. These characteristics include a large number of nodes and edges, sparse connectivity, and dynamics that have stable macro effects but substantial variation in micro effects. In this section we illustrate these characteristics using telecommunications traffic. Our data consists of hundreds of millions of nodes, each of which represents an account. We observe several billion edges on this network in an average week, presenting themselves in a continuous data stream. This section uses plots and tables to present the salient features of this transaction stream.

2.1 Addition and attrition of nodes

Nodes in dynamic graphs appear and disappear through time. Figure 1 shows the addition and attrition of nodes throughout the study period. By “node addition” we mean the number of new nodes that we see in week i that we haven’t yet seen through week $i-1$. By “node attrition” we mean the number of nodes that we see for the last time in week i. The service we consider has hundreds of millions of nodes and billions of edges. On any given day, tens of millions of these nodes are active and responsible for hundreds of millions of transactions. The exact numbers are considered proprietary and we illustrate the volatility of the service using relative change. Figure 1 attempts to capture this volatility by showing the addition and attrition rates of nodes on the network. The figure illustrates that after a steady state is reached, roughly after 18 weeks of observation, new nodes are observed (for the first time) at a rate of slightly less than 1% per week. Similarly, discounting the end effects of the observation period, old nodes are observed (for the last time) at the same rate. The parallel lines fitted to the end/start of these sequences illustrate that the service is stable despite there being considerable turnover of transactors each and every week.

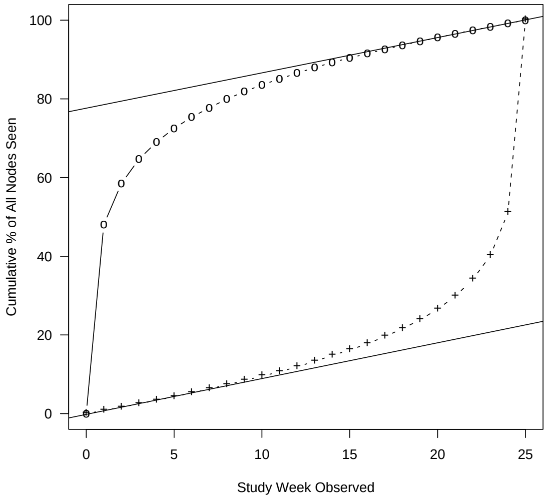


Figure 1: *Node addition and attrition through time. The data correspond to 25 successive weeks of network activity. The upper curve shows the cumulative percent of unique nodes seen each study week that had not been seen before. The lower curve shows the cumulative percent of nodes seen for the last time in each study week. Steady state is reached after about 18 weeks and lines are fitted to the remaining last and first 7 weeks respectively. The slopes correspond to a addition and attritrion rates of just under 1%. This means that 1% of the nodes we observe each week had not been observed previously, and another 1% will never be seen again.*

2.2 Addition and attrition of edges

We illustrate the addition and attrition of edges by following a random sample of 1000 residential accounts over a 180 day period. This period was broken up into six 30-day slices, and the edges from the first month were tracked to see if they appeared in subsequent months.

The results in Table 1 show that edges seen in one month often do not show up again. Of all the edges observed in the reference month, only 37.9% of them are seen the following month, and the number seen steadily decreases for subsequent months. The cumulative results show aggregation of all the months. By the end of the study period, only 53.7% of the edges from the first month have been observed again, and the leveling off of the cumulative numbers indicate that there will be a reasonably large percent of edges that will never be seen again.

Month	Old Edges Seen	Percent	Cumulative Percent
1	5995	100	–
2	2272	37.9	37.9
3	2001	33.4	46.5
4	1734	28.9	50.0
5	1585	26.4	52.3
6	1376	23.0	53.7

Table 1: *Edge attrition for activity on 1000 residential accounts over a 6 month period. Month 1 is a "reference month" for which we look for those edges in subsequent months. For each subsequent month the table shows how many of the edges are observed again. The column marked Cumulative aggregates the subsequent months to show how many of the edges we have seen overall.*

Table 2 shows edge addition effects on the same sample over the same period. This time for each month, we note how many edges are seen for the first time. In addition, we show the total number of unique edges seen up to and including that month. Note that after five months, we have observed nearly 4 times more edges than we observed in the first month.

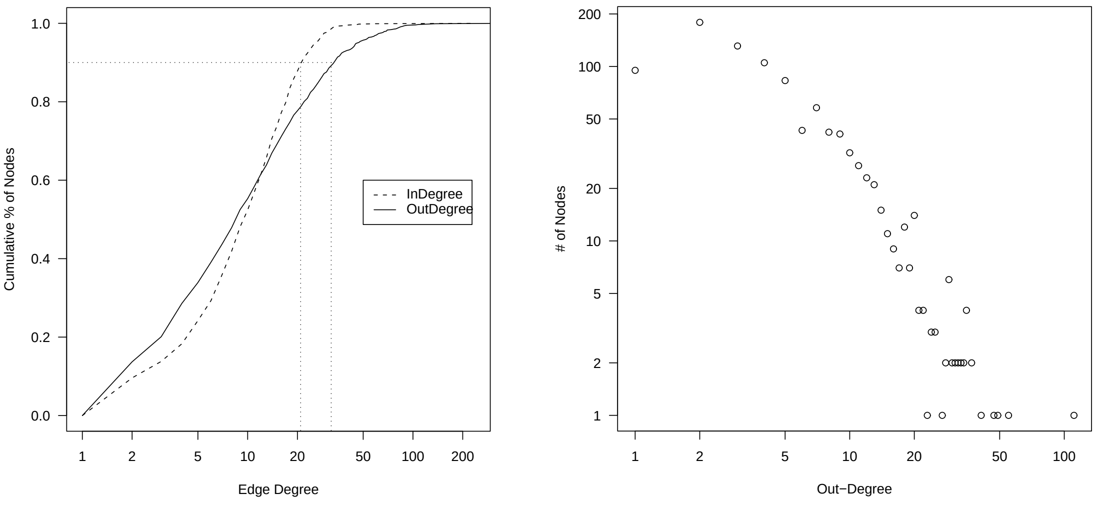
2.3 Connectivity

A fully connected graph with N nodes has $M = N(N-1)$ directed edges. We explore connectivity using the sample of 1000 residential accounts. Figure 2(a) displays the cumulative distribution of in-degree and out-degree for the nodes in this sample. The figure shows that 90% of all nodes have in-degree of 22 or less, and out-degree of 32 or less. This relative sparseness suggests a relationship of the form $M \propto Nlog(N)$, or maybe even $M \propto N$, rather than $M \propto N^2$. In Section 4 we exploit this sparsity with an approximate representation for large network graphs.

Month	New Edges Seen	Percent New	Cumulative Percent
1	5995	100	25.6
2	4461	66.3	44.7
3	4441	59.8	63.6
4	3130	50.8	77.0
5	3102	50.0	90.3
6	2274	44.4	100

Table 2: *Edge addition for activity on 1000 residential accounts over a 6 month period. For each month we show the percentage of the edges that we observed that we had not seen yet. The column marked Cumulative Percent shows the unique edges we have seen through the entire study up until that point as a percentage of the union of all edges.*

Figure 2(b) shows that the distribution for the out-degrees roughly follows a power-law distribution (and hence shows up as linear on a log-log scale). There is a growing literature (e.g. Barabasi[2]) showing that power-law behavior exists in large networks, including the Internet, genetic networks, and social interactions – so it is not surprising to see evidence of this property in our data.



(a)

(b)

Figure 2: *Plots of the (in-) out-degree of 1000 residential accounts over a 180 day period. Panel (a) shows the cumulative percent of accounts having node degree less than or equal to k. Panel (b) shows the number of nodes having a specific out-degree.*

2.4 Activity

While nodes and edges arrive and depart in fairly large numbers, it is also interesting to consider how often they were observed. Figure 3 is a histogram of the number of days that the given nodes were active during the entire 180 day period. The median of this distribution is 29 days, indicating that a typical node was only active on one out of six days during the study.

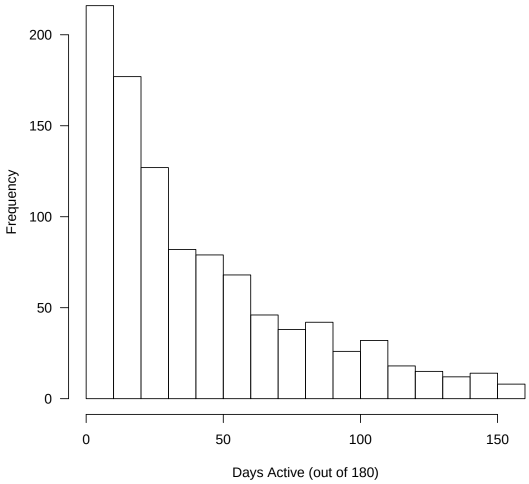


Figure 3: *Histogram of the number of days with activity for 1000 residential accounts over a 180 day period.*

2.5 Implications

The plots and tables in this section illustrate that large numbers of nodes and edges appear or fail to appear on a daily basis. In some cases, their disappearance is temporary and in others it is permanent. Any procedure that attempts to capture network behavior will have to deal with node/edge addition and attrition in an automated fashion since there is little time to synchronize with databases of account information and yet process the current network activity. The relatively sparse connectivity of the graph is the aspect of the behavior that we attempt to exploit in devising a methodology to define, build, evolve, and maintain an approximate representation of the graph through time.

3 Definition of a Dynamic Graph

In this section we consider the definition of $\mathcal{G}_t$, a dynamic graph $\mathcal{G}$ at time t. We consider discrete time applications where new sets of nodes and edges corresponding to the transactions from time step t to $t+1$ only become available at the end of the time step, for example once a day. Associated with every edge is a weight, $\mathrm{w}(\cdot) \geq 0$, that is derived from an aggregation function applied to all (directed) transactions between a pair of nodes at time step t. For example, the aggregation function can be the "total duration of calls" or the "number of calls" from one node to another.

We first define the sum of two graphs g and h

$$G = \alpha g \oplus \beta h$$

where α and β are non-negative scalars. The nodes and edges in G are obtained from the union of the nodes and edges in g and h. The weight of an edge in G is

$$\mathrm{w}(G) = \alpha \mathrm{w}(g) + \beta \mathrm{w}(h),$$

where the weight of an edge is set to zero if the edge is absent from the graph.

Let the graph corresponding to the transactions during time step t be g_t. We can define $\mathcal{G}_t$ from g_i where $i = 1, \ldots, t$ in several ways, depending on the purpose to which the graph is intended to be used.

3.1 Summarizing historical behavior

The cumulative behavior of the graph through time can be defined as

$$\mathcal{G}_t = g_1 \oplus g_2 \oplus \ldots \oplus g_t = \bigoplus_{i=1}^{t} g_i = \mathcal{G}_{t-1} \oplus g_t, \tag{1}$$

This definition of $\mathcal{G}_t$ includes all historic transactions from the beginning of time. The last expression on the right hand side illustrates that for computational purposes, the cumulative summary at time t is the sum of the cumulative summary at time $t-1$ and the network activity at time step t. An alternative definition that considers only recent network activity, say the last k time steps, is the moving window definition

$$\mathcal{G}_t = g_{t-k} \oplus g_{t-k+1} \oplus \ldots \oplus g_t = \bigoplus_{i=t-k}^{t} g_i \tag{2}$$

This definition of $\mathcal{G}_t$ tracks the dynamics of the transactional data stream, and can be thought of as a characterization of network behavior at time $t - k/2$, the center of the moving window. In contrast to Eq. (1), this definition requires storage and retrieval of graphs characterizing network activity at each time step.

3.2 Predicting future behavior

$\mathcal{G}_t$ can be thought of as a predictor of network activity at time $t+1$. The simplest such prediction is $\mathcal{G}_t = g_t$, the network graph corresponding to the transactions at time step t. The stability of this simple predictor can be improved by blending in network activity in a way that discounts the past in favor of recent behavior:

$$\mathcal{G}_t = \omega_1 g_1 \oplus \omega_2 g_2 \oplus \ldots \oplus \omega_t g_t = \bigoplus_{i=1}^{t} \omega_i g_i \tag{3}$$

where the weights ω_i satisfy $\sum \omega_i = 1$ and are an increasing function of i. Eq. (2) can be expressed in this form with $\omega_1 = \ldots = \omega_{t-k-1} = 0$ and $\omega_{t-k} = \ldots = \omega_t = 1/k$. A particularly convenient form of the weights

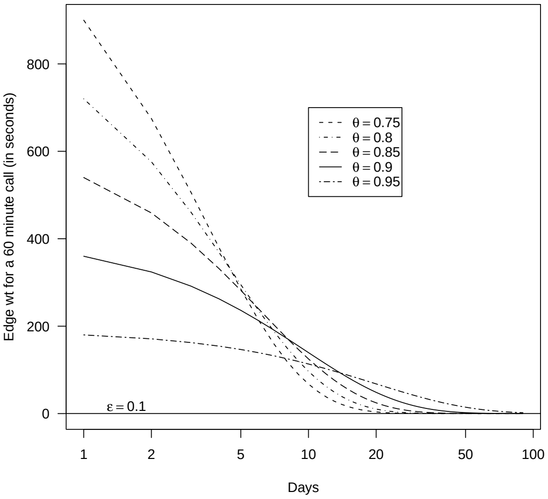


Figure 4: *Contribution of a 60 minute call to edge weights as a function of time steps (days) in the recursive expansion (3) of $\mathcal{G}_t$. The horizontal line at $\epsilon = 0.1$ denotes an adjustable threshold whereby edges with weights less than this value are deleted.*

is $\omega_i = \theta^{t-i}(1-\theta)$ where $0 \leq \theta \leq 1$ is a (scalar) parameter that allows more (θ near 1) or less (θ near 0) history to influence the current graph. Figure 4 displays this graphically. If processing occurs daily, then with a value of $\theta = 0.85$, the edge weight associated with a 60 minute call will be effectively reduced to that of a 6 second call in about 30 days. This form of weight function is convenient in the sense that Eq. (3) can be expressed in recurrence form:

$$\mathcal{G}_t = \theta\mathcal{G}_{t-1} \oplus (1-\theta)g_t. \tag{4}$$

This form is well-known in statistics as exponential smoothing [17]. It provides a smooth dynamic evolution of $\mathcal{G}_t$ without incurring the management and storage of graphs for many previous time periods. All that is needed is the graph through time period $t-1$ and the new set of transactions defined by g_t.

In the following we adopt Eq. (4) as the definition of a dynamic graph at time t. It's usefulness as a smoothing operator and as a prediction make it suitable for a wide range of applications.

4 Approximating $\mathcal{G}_t$

The graph defined by Eq. (4) is complete in the sense that it captures the entire graph, including edges with infinitesimally small weights. If such edges are maintained, the graph will eventually grow to be highly connected. This connectivity is undesirable as it preserves dated relationships that may be misleading (*e.g.,* behavior 6 months ago may not be representative of current activity) or invalid (*e.g.,* when accounts are closed their node labels are often reassigned, as is the case with telephone numbers and IP addresses). In addition to these considerations, the entire graph of the size we consider can be unwieldy, especially if it is too large to fit into main memory. In this section we propose approximations to the entire graph that enable sophisticated applications without compromising it's integrity.

Section 2 demonstrated that over a 6 month period, the vast majority of nodes exhibit a low degree of connectivity. We propose an approximation to the entire graph that exploits this sparsity. The key to our approximation is the introduction of new aggregator edges to the graph that summarize edges that are eliminated for one of two reasons:

- *Global costs.* Either the weight associated with an edge is too small, in an absolute sense, to justify the overhead of maintaining that edge, or
- *Local costs.* The weight associated with an edge is too small relative to other edges coming in or out of a node to justify the overhead of maintaining that edge.

The new aggregator edges are introduced at the subgraph level. For each node in $\mathcal{G}_t$, consider the subgraph consisting of that node and the directed edges to its immediate neighbors. A new outbound aggregator edge effectively replaces a subset of outbound edges of this subgraph such that it contains the same total weight of the edge subset. The node label on the terminating side of this edge is simply called `other`. A new inbound aggregator edge applies to a subset of inbound edges. The subsets of edges that are removed can be parameterized by a pair of thresholding functions, one applying to global thresholding of edge weights, and the other to local thresholding of edge weights.

We first describe the global thresholding function. In practice, edges in $\mathcal{G}_t$ with extremely small weights carry a disproportionate amount of overhead in maintenance and storage of the graphs relative to the information that they contain. Such edges come about from new calls with unusually small weights (*e.g.,* realizing that one dialed a fax number instead of a voice number) or from old calls that had meaningfully large weights initially, but have decayed through time by exponential weighting. We apply a thresholding function to each edge in $\mathcal{G}_t$ such that all edges with weights less than a constant ϵ are eliminated prior to storing the updated graph. In our applications we use $\epsilon = 0.1$. If edge weights reflect call durations (in seconds) then $\epsilon = 0.1$ coupled with $\theta = 0.9$ means that a one second call lasts one day in the updated graph (since w(e)=$0.9 \times 0 + 0.1 \times 1 ses$). Alternatively a 60 minute call persists in the graph for 78 days (since w(e)=$0.9^{78} \times 0.1 \times 3600 sec < 0.1$).

The local thresholding function that we use applies indirectly to the value of the edge weights. For a node n with outbound (inbound) edgeset $\{e : e_i, i = 1, ...N\}$, we retain only the top-k outbound (inbound) edges where "top" is relative to the value of the weight associated with each edge:

$$\text{top-}k\{e\} = \{e : \omega_j(e) > \omega_{[N-k]}(e)\} \tag{5}$$

where $\omega_{[i]}(e)$ is the ith order statistic of $\omega(e)$. This type of thresholding function leads to possible asymmetry in the sense that an edge $e_{ij} : n_i \rightarrow n_j$ might be retained in the top-k outbound edgeset of n_i but not in the top-k inbound edgeset of n_j. This would be the case for example if n_i corresponds to a "normal" residential

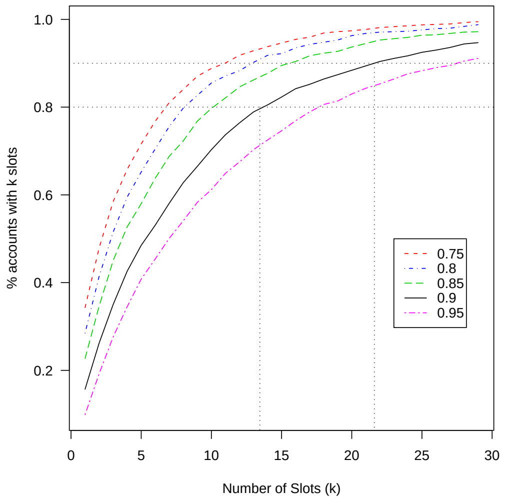


Figure 5: *Cumulative proportion of accounts that had k distinct outbound edges at the end of a 90 day study period.*

account and n_j a toll free number for a large retailer. For all finite k, the only invariance between the complete graph and its top-k approximation is that the sum of the outbound edge weights equals the sum of the inbound edge weights, where the sum includes all nodes labeled as `other`.

Considerations for selecting a value of k are discussed in the next subsection. In our experience we prefer a relatively small value of k that balances computational complexity (*e.g.,* as regards speed and storage) with empirically determined accuracy (see below).

4.1 Tuning the approximation

The definition of $\mathcal{G}_t$ given by Eq. (4) requires a value of θ that governs the degree to which new nodes and edges are blended in with recent activity. For top-k approximations, interplay between θ and k determines both the size of $\hat{\mathcal{G}}_t$ and the degree to which it captures the evolving network graph. In theory, ϵ is also a parameter which affects the graph, however we choose to hold $\epsilon = 0.1$ for its nice interpretation stated above, that a one second call lasts about one day, and any edge weight less than one second will be below threshold. In this section we explore the relationship between θ and k in top-k approximations. In practice, our recommended approach is to first settle on a value of θ that makes sense for the service being modeled, and for that θ, choose k so that interesting detail on most of the nodes is adequately captured.

We explore values of θ in the range $0.75 - 0.95$. Reference to Figure 4 indicates that for this range of values, a one hour call persists in the evolving network graph from several weeks to several months. In our applications, where phone numbers can be reassigned 30 days after service is discontinued, a value of $\theta > 0.95$ would lead to contamination of the new accounts subgraph with activity from the previous account. Smaller

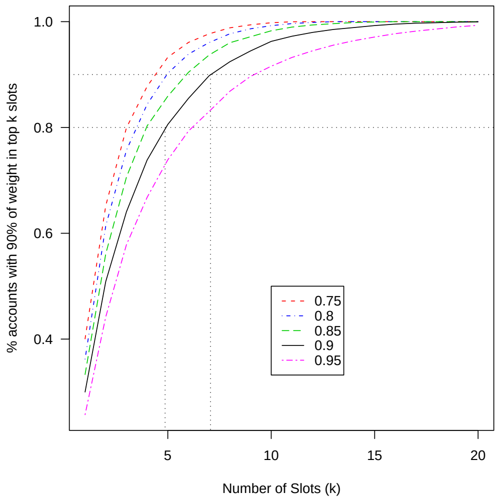


Figure 6: *Cumulative proportion of accounts that have at least 90% of their edge weights captured by k edges.*

values of θ force more dynamics, with the result being that for many accounts with infrequent or sporadic usage, their subgraph is not adequately captured. We illustrate these points with several plots derived from the sample of 1,000 accounts introduced in Section 2. The graphs for these accounts were evolved using a range of values of θ, and the status of the accounts at the end of the period were used.

Figure 5 shows the cumulative proportion of accounts that had k distinct outbound edges at the end of the 90 day study period. The figure shows that for $\theta = 0.90$, 80% of all accounts have at most 15 distinct edges, and 90% of all accounts have at most 25 distinct edges. If edge preservation was critical in an application, choosing a value of $k = 15(25)$ would lead to approximately 80% (90%) of all nodes having all their edges preserved by the top-k approximation.

The curves displayed in Figure 5 relate to the presence or absence of edges without regard to the weights on those edges. An edge that corresponds to daily one hour calls between a pair of accounts is treated identically to an edge that resulted from a call lasting 1 second on the last day of the study period. The only role that edge weight played was that an edge was deleted if its weight dropped below $\epsilon = .1$. Figure 6 addresses this issue by displaying the cumulative proportion of accounts that had at least 90% of their edge weights captured by k edges. If the weight function represents the number of calls, then the curves represent the cumulative proportion of accounts that had at least 90% of their calls captured by k distinct edges. If the weight function represents the length of calls, then the curves represent the cumulative proportion of accounts that had at least 90% of their "time on network" captured by k distinct edges. The figure shows that for $\theta = 0.90$, 80% of all accounts have 90% of their edge weights captured in 5 distinct edges and 90% of all accounts have 90% of their edge weights captured in 8 distinct edges. If weight preservation was critical in an application, choosing a value of $k = 5(8)$ would lead to approximately 80% (90%) of all nodes having 90% of their edge weights preserved by the top-k approximation.

$$\begin{array}{cccccc} & \begin{array}{c}\text{Old top-}k\text{ edges:}\\ \text{node–labels} \quad \text{wts}\end{array} & & \begin{array}{c}\text{Today's edges:}\\ \text{node–labels} \quad \text{wts}\end{array} & & \begin{array}{c}\text{New top-}k\text{ edges:}\\ \text{node–labels} \quad \text{wts}\end{array} \\ \theta & \left(\begin{array}{cc} X5467 & 5.2 \\ X2656 & 5.0 \\ X4132 & 4.5 \\ X4231 & 2.3 \\ X3142 & 1.9 \\ X4212 & 1.8 \\ X1423 & 0.8 \\ X2312 & 0.5 \\ X4532 & 0.2 \\ \texttt{other} & 0.1 \end{array}\right) & +\,(1-\theta) & \left(\begin{array}{cc} X5467 & 2.0 \\ X2656 & 6.2 \\ X4132 & 0.8 \\ X6547 & 10.0 \\ & \\ & \\ & \\ & \\ & \\ \texttt{other} & 0.0 \end{array}\right) & = & \left(\begin{array}{cc} X2656 & 5.2 \\ X5467 & 4.6 \\ X4132 & 3.9 \\ X4231 & 2.0 \\ X3142 & 1.6 \\ X4212 & 1.5 \\ X6547 & 1.5 \\ X1423 & 0.7 \\ X2312 & 0.4 \\ \texttt{other} & 0.3 \end{array}\right) \end{array}$$

Figure 7: *Computing a new top-k edge set from the old top-k edge set and today's edges. Note how a new edge (X6547) enters the top-k edge set, forcing an old edge (X4532) to be added to* `other`.

4.2 Implementing the approximation

We propose a constructive approach to implementing our approximation to a large time-varying graph. Consider a node in the graph, its associated directed edges, and weights associated with each edge. A data structure that consists of these weighted directed edge sets for each node is a representation of the complete graph. This data structure is redundant since it is indexed by nodes so that edges must be stored twice, once for the originating node and once for the terminating node. In contrast, a data structure that stores each edge once must be doubly indexed by nodes. The cost of edge duplication is often mitigated by gains in processing speed when subgraphs around nodes are expanded (see next subsection). For this reason we have chosen to represent our graphs as a singly-indexed list of nodes, each with an associated array of weighted directed edges.

The singly-indexed node list represents our approximation but the definition in Eq. (6) implies that new activity must be blended into it at fixed time steps. Let $\hat{\mathcal{G}}_{t-1}$ denote the top-k approximation to $\mathcal{G}_{t-1}$ at time $t-1$ and let g_t denote the graph derived from the new transactions at time step t. The approximation to $\mathcal{G}_t$ is formed from $\hat{\mathcal{G}}_{t-1}$ and g_t, node by node, using a top-k approximation to Eq. (4):

$$\hat{\mathcal{G}}_t = \text{top-}k\{\theta\hat{\mathcal{G}}_{t-1} \oplus (1-\theta)g_t\} \qquad (6)$$

Thus, we first calculate the edge weights for all the edges of $\theta\hat{\mathcal{G}}_{t-1} \oplus (1-\theta)g_t$. Then for each node we sort the edges according to their weight. (The overflow node `other` is not given any special treatment in these computations.) The top-k are preserved, and if there are more than k edges in the edge set for that node, the weights of the remaining edges are added to the weight of the edge going from the node to node `other`. These operations are displayed pictorially in Figure 7 using $\theta = .85$. Notice that a new call today with a heavy edge weight (labeled X6547) replaces an old call with a low edge weight (labeled X4532).

Between updating steps, transactions need to be collected and temporarily stored. At the end of that time period, the transactions are aggregated and the subgraph updated. The length of the time period represents a trade-off in accuracy: the longer the time period, the better an estimate of the top-k edge set, but the more outdated the resulting subgraph. In the application discussed in Section 5, we perform daily updates, thereby maintaining reasonable accuracy while requiring temporary disk space for only one day of data. For a related discussion see Cortes and Pregibon[5].

Prior to storing the updated graph, we remove all edges that fall below the ϵ threshold. As argued above, this reaping process results in both a smaller and a more interpretable graph. For our large network graph this thresholding results in a 50% reduction in size, from 14gb (maximum size at saturation of top-9 approximation) to 7gb (stable size with average of 4.5 slots per indexed node).

4.3 Subgraph expansion

Our implementation of the subgraph consisting of the top-k inbound and the top-k outbound edges of a node is ideal for fast extraction of larger subgraphs centered on the node. The data structure containing the top-k approximation can be queried recursively for each node in the top-k edge sets of the center node.

We grow the subgraphs in a breadth-first traversal of the data structure. For notational purposes, we denote the top-k inbound and outbound nodes and edges of node n by $R_1(n)$, the subgraph of radius 1 centered at node n. Similarly let $R_2(n)$ denote the subgraph of radius 2 centered at node n. Note that $R_2(n)$ can be formed from the "union" of $R_1(n)$ and the radius-1 subgraph centered on each node contained in $R_1(n)$. In general, we can define subgraphs of any size using the recursion

$$R_{j+1}(n) = \biguplus_{\text{node} \in R_j(n)} R_1(\text{node})$$

We use the quoted term "union" and the symbol $\uplus$ instead of simply $\cup$ because of the aforementioned potential asymmetry in the edge weights. The top-k approximation may force an edge from/to a high activity node to be partially, or maybe even completely, absorbed by category `other`. However, a low activity node that is connected to a high activity node is likely to preserve the edge in its top-k approximation. In cases where both nodes preserve a common edge in their top-k list, the weights associated with that edge are potentially different for the two nodes. In those cases we use the maximum edge weight in the definition of $R_{j+1}(n)$, since the maximum represents the weight least affected by the top-k approximation.

The index structure of our representation is critical since we often need to compute and compare many subgraphs on a daily basis. We have tuned our algorithms so that the average time for retrieving and rendering $R_2(n)$ subgraphs from our data structure of several hundred million nodes is just under one second (on a single processor).

In our applications, we rarely explore edge sets greater than $R_2(n)$, as the edge sets become unmanageably large and remarkably uninformative. Subgraph expansion also reinforces arguments suggesting that a large value of k (in defining the top-k approximation) is not necessarily desirable when network graphs are used to study relationships between nodes. Spurious edges (*e.g.,* misdialed numbers) can have unintended consequences upon subgraph expansion. We also discuss this further in Section 6 where we suggest additional edge pruning in $R_2(n)$ to further reduce "clutter" in a subgraph.

5 Application

In the telecommunications industry, there are many different types of fraudulent behavior. *Subscription fraud* is a type of fraud that occurs when an account is set up by an individual who has no intention of paying any bills. The enabler in such cases involves either flawed processes for accepting and verifying customer supplied information, or identity-theft where an individual impersonates another person. In either case, if left undetected, the fraud is typically only discovered when the bill is returned to sender, often after thousands of dollars have been lost. Since speed is critical in reducing losses due to fraud, it is essential to essess inherent riskiness as each new account is activated on the network. In this section we explore this possibility by defining a procedure that assesses risk on the basis of a node's connectivity to other nodes.

5.1 Subgraph-based account linkage

Consider the case where we have information on an account that was recently disconnected for fraud. If identity theft was the root cause of the fraudulent account, we might expect this same individual to appear with a new account bearing a different name and address. Basically the fraudster has now assumed the identity of a new victim. We attack this problem with the intuition that while the subscription information is not useful for matching network IDs to the same individual, the calling patterns of the new account, as characterized by its R_2 subgraph, should not change very much from the previous account. Figure 8 provides an illustration where we overlay two R_2 subgraphs apparently belonging to the same individual. The central nodes of the two subgraphs are indicated by the rectangle with two ID numbers. One of these numbers corresponds to a known fraudulent account, the other to a new account on our network. The amount of overlap between the two subgraphs is strong evidence that these numbers are related and increases the fraud risk of the new account.

Subgraph-based account linkage is a non-standard problem that involves defining a distance function to quantify the closeness of two accounts based on subgraphs centered on the two accounts. The distance between two subgraphs depends on both the quantity and the quality of the overlapping nodes. The quantity of the overlap can be measured by the percentage of overlapping nodes. However all overlapping nodes are not equally informative, so a measure of quality is needed as well. Many graphs will intersect at high-use nodes, such as large telemarketing firms or widely advertised customer service numbers. An informative overlapping node is one that has relatively low in- and out-degree, and in the best case, is shared only by the nodes under consideration for a match. We now describe a measure that captures these notions.

Given a new account a and known fraudulent account b, let $O = R_j(a) \cap R_j(b)$ denote the set of all overlapping nodes in the two radius j subgraphs. We define

$$\text{Overlap}(R_j(a), R_j(b)) = \sum_{o \in O} \frac{w_{ao} w_{bo}}{w_o} \frac{1}{d_{ao}} \frac{1}{d_{bo}},$$

where w_o is the overall weight of node o (the sum of all edge weights in $R_1(o)$), w_{ao} is the weight of edges

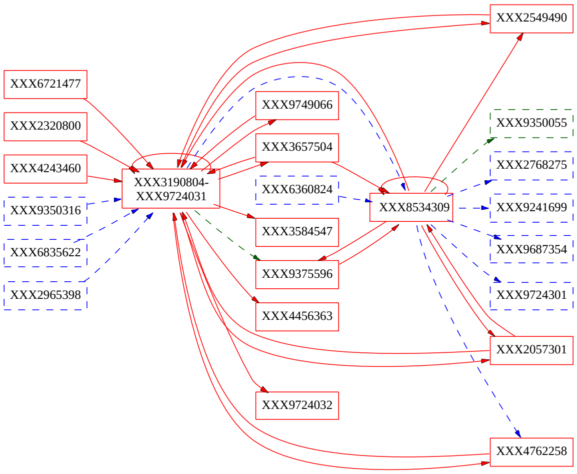


Figure 8: *Subgraph-based account linkage. The two individual subgraphs are superimposed (in the doubly-labeled rectangle) to emphasize their similarity. Solid lines indicate edges common to both subgraphs, while dashed lines indicate edges belonging to only one subgraph.*

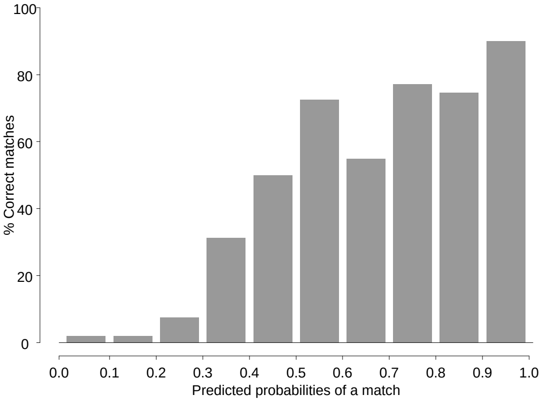


Figure 9: *Success of subgraph matching. Observed percentage of matching node-pairs versus deciles of predicted matching probability.*

between node a and node o in $R_j(a)$, and d_{ao} is the minimal distance from node a to node o in $R_j(a)$. The terms w_{bo} and d_{bo} are defined similarly. [In the case where $d_{ao} > 1$, it is not clear what the weight w_{ao} should be, since there is no direct edge between the two. For this application we elected to assign a small default weight in order to minimize the effect of these overlaps]. Intuitively, the numerator measures the strength of the connection from a and b to the overlapping node, while the denominator corrects for an overlap node which is either common to many nodes or is further in the graph from a or b. This measure is large for overlapping nodes that have strong links to the nodes of interest, but otherwise have low overall volume.

Even armed with this definition of distance, subgraph-based matching is computationally difficult because of the dynamic nature of our network data as described in Section 2 – each day we see tens of thousands of new accounts. For each of the new accounts, we need to compute it's subgraph, and then the distance from it to the subgraph of all recently confirmed fraudulent accounts. Assuming for these purposes that we maintain a library of the most recent 1000 fraudulent accounts, tens of millions of pairwise distances need to be computed daily. We have harnessed the computations by maintaining R_2 subgraphs for all accounts in our "fraud library" and computing R_1 subgraphs for all new accounts.

We obtained a training set of paired accounts where investigators were able to determine whether the old and new accounts belonged to the same individual. We built a decision tree using the overlap score defined above as a predictor along with several covariates obtained from the information provided by the subscriber. The decision tree produces a "matching" probability for any set of node pairs. Figure 9 shows the performance of the decision tree on an independent test set of paired accounts. For the sample of 1537 pairs that we validated, the figure shows the observed proportion of matching node-pairs for each decile of predicted matching probability. As the plot shows, account pairs with a high predicted probability of matching based on our methodology were indeed usually associated with the same individual.

6 Communities of Interest

As stated earlier, edge sets larger than $R_2(n)$ can be unmanageably large and remarkably uninformative. To help reduce this "clutter" we often apply a thresholding function to the edge weights in the expansion of subgraphs, so that any edge whose weight is below the threshold need not be expanded.

This threshold function is the simplest operator in a series of functions one can apply to an edge set to bring out what we call the Community of Interest, or COI, for a given node. In telecommunications, a COI operationalizes the notion of a "calling circle", the group of accounts around a specified account, where there is some unobservable relationship between them (i.e., personal/professional/familial interests) that motivates accounts to communicate with each other. Intuition suggests that when such a relationship exists, that nodes involved in the relationship will be linked and that the weights along these links will be larger than weights along links to nodes not sharing in the relationship. There is also the notion of diameter of a calling circle since one can discuss "immediate calling circles" as well as "extended calling circles". Our subgraphs captures these notions in a primitive fashion through $R_1(n)$ and $R_j(n), j > 1$ edge sets respectively, but in applications, the raw edge sets are often treated as the starting point in deriving a COI.

There are several reasons why the raw edge sets are often not sufficient for capturing COIs. They can be summarized into two main categories:

- spurious edges in $R_j(n)$
- missing edges in $R_j(n)$

Spurious edges arise from the fact that while we posit an unobservable relationship between accounts that encourages communication between them, additional calls are captured in the data that can be totally outside the relationship. For example, misdialed calls are captured by the edge sets, as well as unwanted telemarketing calls. The effect of such calls on the edge sets can be enormous, since expanding the edge set to the next diameter, brings in all the accounts linked to this spurious node, and arguably, these are conceptually far removed from an accounts calling circle.

Missing edges arise in several ways reflecting realities associated with large graphs arising from transactinoal data. One important way that edges are missing relates to the fact that in many applications there are numerous service providers so that any single network carries only a fraction of the transactions. A related way that edges are missing concern the locations of the devices in the network topology that records transactions. By this we mean that transactions are recorded when they cross certain network elements where recording equipment is located, and the corresponding records are then sent to a central repository for analysis. But many transactions could occur "below" the recording point and subsequent analysis is blind to these transactions. An example is the seperation between local and long distance calls in telecommunications whereby the long distance carrier is blind to any calls on the local level. Similarly for internet traffic monitoring, data collection equipment at internet gateway routers is blind to TCP/IP traffic between computers behind that network gateway.

In our applications of COI for large dynamic graphs we address these deficiencies by introducing aspects of the problem not captured in the available data. For example, we would like to discount edges that might have large weight due simply to a single long call, for example, to a customer support center. One way we deal with such nuances is to build a large edge set, and then find the strongly connected component containing the node of interest. This creates a subgraph where every node can be reached from every other node. Another way of removing spurious edges is to apply a high threshold ϵ, which will mitigate the effects of one-time non-representative calls.

Application of a thresholding function and applying a strongly connected component algorithm are both examples of operators we apply to prune edges and nodes from $R_j(n)$. Alternatively, if we believe that edges may be missing from the observed set of transactional records, we may want to insert *pseudo edges* between certain nodes in $R_j(n)$. For example, local phone calls between accounts would not appear in traffic collected from a long distance network. Similarly since most networks, telephony or otherwise, exist in competitive markets, the observed edge set collected from a single network is blind to traffic carried on a competitors network. If the notion of COI is meant to capture the existence of implicit underlying relationships, adding certain pseudo edges to competitor nodes is a reasonable approach to uncovering such relationships.

As is clear from this discussion, the transformation of an edge set into a COI is not a science. One normally lacks a reference for calibrating the process so that feed-back from COI-based applications is often the only guidance.

7 Related Work

The analysis of directed graphs goes by many names and has been studied in many fields, including sociology [16], epidemiology [10], information retrieval [15], statistics [3], operations research [13], and software engineering [12]. Clustering is a common goal in these fields, and is similar in spirit to our approach of defining communities of interest.

Arguably the oldest research field in this area is the field of *social networks*. Social networks model the interdependencies between "actors" or "agents" in a data set by analyzing the relationships between them, represented as edges in a graph. This type of analysis has grown to study such diverse topics as disease transmission, international trade, and computer networking. Social network theory can incorporate complex stochastic models, explanatory variables for both nodes and edges, and time dependent graphs. However, the field has always focused on the study of small graphs. A popular textbook in the field, Wasserman and Faust[16], contains five datasets used to illustrate the methodology, the largest of which contains 32 nodes. The mathematically complex and computationally intensive methods generally do not scale, and to date, we have not used them in our research.

Flake, Lawrence, and Giles[7] provide a definition of communities of interest in terms of number of edges connecting a set of nodes that has the nice property that the COI can be efficiently enumerated by applying a maximum flow algorithm. Citation analysis of scientific papers also aims at finding communities in large graphs. A distance between two documents is defined using *co-citation* (the number of citations in common) or *bibliographic coupling* (the number of times both works are cited in other papers). Using this distance measure, clusters in the database can be found. A successful example of this work is the NEC Research Index [11] (`http://citeseer.nj.nec.com/cs`), which currently documents and cross references 7 million scientific works, and for each of those works, lists the most similar books by several different metrics.

The Internet is natural to treat as a massive graph, where web sites are nodes, and the links between them are edges. Current research [8, 9] uses "hubs" (sites that link to many others) and "authorities" (sites that are linked to by others) in order to identify clusters in the web that deal with a particular topic. Extensions of this work use network flow algorithms [7] and these are quite effective in finding small subject clusters.

Marketing has also inspired analysis of large graphs. The active research topics of market basket analysis [1], viral marketing [6], and collaborative filtering [14] all use graph algorithms to discover communities of consumers with similar behavior. These popular methods have been used successfully at sites like Amazon.com, which suggests items to purchase based on purchases of other customers who recently purchased the same item.

Despite the wealth of research into large network graphs, our research is unique in combining the following attributes:

- *Scale.* Our network graphs contain hundreds of millions of nodes, and we are potentially interested in retrieving local subgraphs for any one of them.
- *Speed.* Our data structure, accessed recursively, along with off-line processing, allows us to compute subgraphs centered on any node of the graph in fractions of a second.
- *Dynamic updating.* The graph incorporates the continuous stream of incoming data, so that any analysis is as recent as the most recent data. Time is a crucial element, since today's network may contain tens of thousands of new nodes and edges than yesterday's graph did. Our exponential updating creates a smoothed view of network behavior, with the largest weights on the most recent events and the smallest weights on the oldest events.
- *Condensed representation of the graph.* Conceptually, we view a massive graph as the union of a massive number of small graphs $R_1(n)$. The approximation we employ that limits node degree to the top-k is effective because large dynamic graphs seem to be sparse as well.

Another appealing aspect of this work is that our applications measure *direct* interaction between the nodes. Accounts form a community by actually communicating, creating a richer basis on which to define clusters. In collaborative filtering or market basket analysis, the goal is to find *indirect* links between people. Two people are similar not because of a direct interaction, but because they both purchased similar items. Similarly, large scale web mining explores static links between pages, but not user traffic along those links.

8 Conclusions

In this paper we introduced the concept of a dynamic graph and motivated the concept with network data from a sample of AT&T residential accounts. This led to our definition of a dynamic graph $\mathcal{G}_t$ (at time t) as an exponentially weighted average of the previous graph (at time $t-1$) and a new network activity. We introduced a data structure that can be used to capture the evolution of a graph through time that was amenable to the exponential weighting scheme. This data structure allows the subgraph around any particular node to be quickly and efficiently expanded to an arbitrary diameter. An application was introduced that capitalized on this feature.

We have concentrated on the computational aspects of building and evolving the data structure for real applications. We have not explored the statistical aspects of treating our data structure and the associated algorithm for traversal as an approximation $\hat{\mathcal{G}}_t(k)$ to the true graph $\mathcal{G}_t$ where k denotes the size of the top-k edge set maintained in the data structure. Similarly models and methods in social networks, while not applicable to massive graphs, are applicable to $R_2(n)$ edge sets. These models might provide the rigorous justification for transforming edge sets into COI that we currently lack. We hope to explore these ideas in the near future.

Another topic for further research is how to prune a subgraph so that only informative edges and nodes are retained. A common approach from (static) graph theory is to extract the strongly connected component. The strongly connected component algorithm has the advantage that it scales linearly in the order of nodes, and we have used it in our computationally intensive applications. However, we feel that certain features inherent to telecommunication networks such as asymmetric edges (due to some customers subscribing to a competitor), sinks (toll-free calling) and sources (large corporations), makes strongly connected components a less than ideal choice for pruning subgraphs $R_j(n)$.

Initializing, storing, and updating the data structures that we employ are facilitated by the programming language Hancock, [4]. Hancock is a domain-specific C-based language for efficient and reliable programming with transactional data. Hancock is publicly available for non-commercial use at

`http://www.research.att.com/~kfisher/hancock/`

.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 1994.

[2] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[3] David Maxwell Chickering and David Heckerman. Efficient approximations for the marginal likelihood of bayesian networks with hidden variables. *Machine Learning*, 29(2-3):181–212, 1997.

[4] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, and F. Smith. The Hancock language for signature processing. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*, 2000.

[5] C. Cortes and D. Pregibon. An information-mining platform. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, 1999.

[6] Pedro Domingos and Matt Richardson. Mining the network value of customers. In *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, pages 57–66. ACM Press, 2001.

[7] Gary Flake, Steve Lawrence, and C. Lee Giles. Efficient identification of web communities. In *Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 150–160, 2000.

[8] David Gibson, Jon M. Kleinberg, and Prabhakar Raghavan. Inferring web communities from link topology. In *UK Conference on Hypertext*, pages 225–234, 1998.

[9] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[10] Alden S Klovdahl. Social networks and the spread of infectious diseases, the aids example. *Social Science and Medicine*, 21:1203 – 1216, 1985.

[11] Steve Lawrence, Kurt Bollacker, and C. Lee Giles. Indexing and retrieval of scientific literature. In *Eighth International Conference on Information and Knowledge Management, CIKM 99*, pages 139–146, Kansas City, Missouri, November 1999.

[12] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner. Using automatic clustering to produce high-level system organizations of source code. In *IEEE Proceedings of the 1998 Int. Workshop on Program Understanding (IWPC'98)*, 1998.

[13] A. Ravindran, D. Phillips, and J. Solberg. *Operations Research-Principle and Practice.* John Wiley and Sons, New York, USA, 1987.

[14] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, 1994. ACM.

[15] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, NY, 1983.

[16] Stanley Wasserman and Katherine Faust. *Social Network Analysis.* Cambridge University Press, 1994.

[17] P. R. Winters. Forecasting sales by exponentially weighted moving averages. *Management Science*, 6:324–342, 1960.