cGraph: A Fast Graph-Based Method for Link Analysis and Queries

Jeremy Kubica Andrew Moore David Cohn Jeff Schneider
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
{jkubica,awm,cohn+,schneide}@cs.cmu.edu

Abstract

Many techniques in the social sciences and graph theory deal with the problem of examining and analyzing patterns found in the underlying structure and associations of a group of entities. However, much of this work assumes that this underlying structure is known or can easily be inferred from data, which may often be an unrealistic assumption for many real-world problems. Below we consider the problem of learning and querying a graph-based model of this underlying structure. The model is learned from noisy observations linking sets of entities. We explicitly allow different types of links (representing different types of relations) and temporal information indicating when a link was observed. We quantitatively compare this representation and learning method against other algorithms on the task of predicting future links and new "friendships" in a variety of real world data sets.

1 Introduction

This paper is an extended version of a paper submitted to the 2003 ICML conference.

Many techniques in the social sciences, criminology, and graph theory deal with the problem of examining and analyzing patterns found in the underlying structure and associations of a group of entities. A typical query may examine an entity's "importance" by looking at its relations to others and thus its position within the underlying structure [Wasserman and Faust, 1994]. However, in many real world situations true structure may not be known. Instead the relevant information may be buried within large amounts of noisy data.

In this paper, we examine the problem of learning and querying a graph-based model of the underlying structure. The model is learned from link data, where each link is a single noisy input observation, specifically a set of entities observed to share some relation. As with most sources of real world data, we assume this data is noisy, possibly containing many irrelevant links, as well as links with spurious or missing members. Below, we present cGraph, a learning method that approximates the underlying structure. We account for different types of links, which may vary in their frequency and extent of noise. We also incorporate information about when a link occurred, allowing us to account for changes in the underlying structure over time. By learning such a graph from noisy co-occurrence data our method complements many existing analysis methods that require knowing this underlying structure.

We also introduce the "friends" problem, which attempts to predict co-occurrences in future links. This question also inherently measures how strongly two entities are connected and thus can be used to analyze the connections found in the underlying graph. We use this query both as a possible motivation for learning the underlying structure and also as a novel test of the effectiveness of various algorithms.

The motivation for this research is the increasing number of analytical applications that are trying to exploit massive amounts of noisy transactional and co-occurrence data. This kind of data is important to insurance analysts, intelligence analysts, criminal investigators, epidemiologists, human resource professionals, and marketing analysts. Of the many questions such analysts might like to ask, one general question is "What are the underlying relationships among people in this organization?" and, at a more focused level: "Who are the other people most likely to be directly interacting with this person?" For example, if a key employee leaves a corporation, HR needs to know who is most likely to be impacted. Or if an individual is caught transporting illegal materials, law enforcement may need to rapidly deduce who are possible accomplices.

2 Collaborative Graph Model

Graphs, or social networks, are a common method for representing the relationships among a set of entities. Nodes represent the entities and edges capture the relations between them. For example, a simple graph representing the "friends" relation may consist of a graph with an undirected edge between any two entities that are friends.

We use a directed weighted graph to capture the underlying relations between entities, but restrict the weights to form the coefficients of a multinomial distribution over the outgoing edges. In other words for W_{AB}, the weight of an edge from A to B, we have:

$$0 \leq W_{AB} \leq 1 \qquad \text{and} \qquad \sum_B W_{AB} = 1 \tag{1}$$

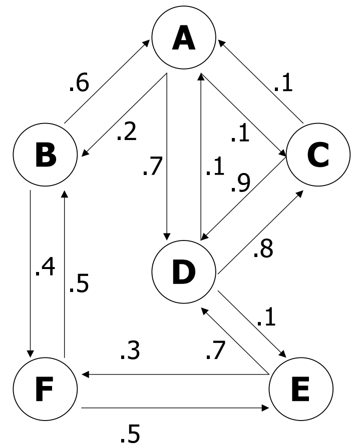

Figure 1: Example collaborative graph.

Thus W can be interpreted as a transition matrix. It should be noted that an **edge** is a weighted ordered pair representing a true underlying connection and is different from the observed input links. An example collaborative graph is shown in Figure 1. Although the weights have a formal probabilistic interpretation that is discussed below, we can intuitively view the weights as capturing the relative strength of an entity's connections. Such a graph-based structure is simple, capturing only pair wise relationships, but is fast to approximate and query and is easy to interpret.

3 Link Data and Generation Models

The model for the N_P entities is learned from a set of N_L observed links. A **link** is a single noisy input observation, an unordered set entities that are linked by some relation.[1] The observations are noisy - some relations that exist in truth may not be observed, and some observations may not be the result of actual relations. A given link may also have spurious or missing members.

The word *link* should not be interpreted too narrowly. A link can be used to capture a range of different relations such as: direct communication, co-occurrence at a given time and place, or even sharing a common attribute. For example, in the domain of food products the link `{flour, sugar, eggs}` may result from a cookie recipe and the link `{water, milk}` may result from the fact both entities are liquids.

3.1 Link Generation

Below we present several models for link generation. These models serve both to provide a model of link generation (for MLE graphs and creating artificial data) and also as motivation for the approximations presented below. It is important to appreciate that each of these models presents only a simple link generation process and that none of these models fully captures the complexities of real world link generation.

[1]For example, this paper links its authors by the *co-author* relation.

The Random Walk Model

The random walk model of the link generation considers a link as a random non-self-intersecting walk on the underlying graph. This process can be visualized as the propagation of a chain letter where each person must pass the letter to exactly one friend who has not yet received it. The chain letter stops after a set number of propagations or when it reaches a person who cannot forward it to a new person. Formally, given a link L_{j-1} that contains $j-1$ entities such that A_{last} was the last entity added to the link, the next entity for the link is chosen as:

$$P(A_i|A_{last} \wedge L_{j-1}) = \begin{cases} \frac{P(A_i|A_{last})}{\sum_{A_k \notin L_{j-1}} P(A_k|A_{last})} & A_i \notin L_{j-1} \\ 0 & A_i \in L_{j-1} \end{cases} \quad (2)$$

and the link is terminated if there is no A_i with $P(A_i|A_{last} \wedge L_{j-1}) > 0$.

While this process can be viewed as a random walk, the transition probabilities are not independent. Since the walk must be non-self-intersecting, the probability of transitioning to the A_i depends not only on A_{last} but also on all of the other entities already in the link.

The Random Tree Model

The random tree model is a more realistic model of link generation, where the jth entity can be added to the link by any of the previous members of the link. This link generation process can be visualized as the forming of a group. A "seed person" starts the group and chooses a second member. Then either the first or second person decides whom to add as the third person. This process continues with a random member of the link deciding which new member to add until the link is complete.

Formally, given a link L_{j-1} that contains $j-1$ entities and a randomly chosen "inviter" from the link, $A_{inv} \in L_{j-1}$, the next entity for the link is chosen as:

$$P(A_i|A_{inv} \wedge L_{j-1}) = \begin{cases} \frac{P(A_i|A_{inv})}{\sum_{A_k \notin L_{j-1}} P(A_k|A_{inv})} & A_i \notin L_{j-1} \\ 0 & A_i \in L_{j-1} \end{cases} \quad (3)$$

and the link is terminated if there is no A_i and A_{inv} such that $P(A_i|A_{inv} \wedge L_{j-1}) > 0$. The resulting link forms a tree with the "seed" entity as the root.

3.2 MLE Graph

It is possible to find the MLE of the underlying graph given the generative model and the set of observed links by just using a simple optimization scheme to search the graph weights. This optimization would include at least N_P^2 real valued parameters, making the search expensive for large numbers of entities. Unfortunately since the ordering of the links is unknown, the step of evaluating the likelihood of the data may also be computationally expensive. Assuming the random walk generative model, simply calculating the probability of a link with N entities requires examining up to $N!$ different orderings.

4 Approximating the Collaborative Graph from Link Data

The first and most important task for cGraph is to learn the underlying graph from the link information. If the learned graph does not reflect the true underlying structure, future queries to it may return incorrect results.

4.1 Collaborative Graph Approximation

The cGraph algorithm learns the underlying graph by using weighted counts of co-occurrences to approximate the edge weights. These counts can be accumulated during a single scan of the data. The resulting edge weights can then be computed directly from these accumulated counts.

We can motivate this approximation by interpreting W_{AB} as $P(B|A)$: the probability that if we randomly chose a link containing A and randomly chose a second entity in the link, this entity would be B. This simple interpretation leads to the approximation:

$$\begin{aligned} W_{AB} = \hat{P}(B|A) &= \sum_L \hat{P}(B|A \wedge L)\hat{P}(L|A) \\ &= \frac{\sum_{L:(A,B)\subset L}\left(\frac{1}{|L|-1}\right)}{\sum_{L:A\in L} 1} \end{aligned} \tag{4}$$

where the second line follows from the assumption that both the link containing A and the second entity were chosen uniformly.

We can extend this sampling interpretation of edge weights further to account for noise. We assume that the links of different types have different and unknown probabilities of being relevant and noise free, denoted by $U(L.type)$. Similarly, links that occur at different times have different probabilities of being relevant to the current graph, denoted by $T(L.time,t)$. Thus the edge weights represent the probability of choosing B from randomly selected relevant (both in type and time) and noise free link containing A.

Given the above probabilistic interpretation, we can derive a closed form approximation of the edge weights as:

$$W_{AB}(t) = \frac{\sum_{L:(A,B)\subset L}\left(\frac{U(L.type)T(L.time,t)}{|L|-1}\right)}{\sum_{L:A\in L} U(L.type)T(L.time,t)} \tag{5}$$

where $|L|$ is the size of the link, $U(L.type)$ is the typical weighting of a link of type $L.type$, and $T(L.time,t)$ is the temporal weighting of a link at time t where the link occurred at time $L.time$. The choice of weighting functions is described in the following two sections. This approach is similar to the methods presented in [Newman, 2001] and [Kautz *et al.*, 1997], but accounts for additional information contained in the links and is normalized to ensure a probabilistic interpretation.

The weighted counting approximation given in (4) can also be justified in the context of the above generative models. In both models the second entity is sampled directly as $P(B_2|A_1)$, where B_i indicates that entity B was the ith entity added to the link. Although we do not know the order that entities were added to the link, if we treat each possible link ordering as equally probable, we can approximate the weight of an edge from A to B as a percentage of possible link orderings in which A_1 and B_2. A combinatorial argument on either model leads to the same approximation:

$$\hat{P}(B_2|A_1) = \begin{cases} \frac{1}{(|L|-1)} & B \in L \\ 0 & B \notin L \end{cases} \tag{6}$$

We can then approximate the overall probability of $A_1 \rightarrow B_2$ as the average probability of $A_1 \rightarrow B_2$ in the links in which A appeared, which results in an approximation identical to that in (4). The difference between the approximate and exact model is that we are no longer accounting for the fact that no entity can be added to a link twice.

It is also interesting to note that by interpreting the edge weights in the above probabilistic manor, our approximation will inherently posses for the following desirable effects:

1. frequency - Entities that co-occur in many links will be strongly connected.
2. size - Links with fewer entities (a smaller link size) will imply a stronger connection among their entities.
3. proximity - More recent links will imply a stronger connection among their entities. We refer to this as the *temporal weighting* of a link.
4. relevancy - More relevant links, or simply links of relevant types, will imply a stronger connection among their entities. We refer to this as the *typical weighting* of a link.

These are all properties that we would intuitively expect to be true of real world linkings between people.

4.2 Temporal Weighting

The temporal weighting of a link, $T(L.time,t)$, determines the extent to which older links are counted as relevant. The intuition behind using a temporal weighting function is that we expect recent links to be more indicative of the current graph than links that occurred some time ago. We choose $T(L,t)$ to be a exponential decay function, specifically:

$$T(L,t) = \begin{cases} e^{-\beta(t-L.time)} & t \geq L.time \\ 0 & t < L.time \end{cases} \tag{7}$$

where β is a parameter that determines the decay rate of the exponential and thus the extent to which older links are discounted. The choice of $\beta = 0$ makes all links that occurred on or before t equally likely.

4.3 Typical Weighting

Similar to the temporal weighting function, the typical weighting of a link type determines how much a link of that type should be counted compared to links of other types. Again, the typical weighting function has the probabilistic interpretation that the link weight is the probability that a link of this type is relevant and noise free. Using this interpretation and the fact that only the relative weights of link types matter, we can restrict the weights to lie in the range of $[0,1]$, where a weight of 0 results in the cGraph algorithm effectively ignoring all links of that type.

This relative weighting is important in cases where input data consists of links of multiple types and links of different types may have different properties. Links of some types, such as linking all people who co-occur in a given city on a given day, could be prone to noise or in general contain little information. In contrast links of other types, such as large monetary transactions, may rarely be noisy and in general contain much information.

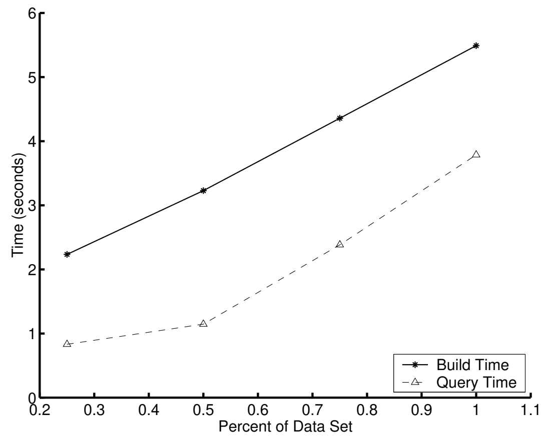

Figure 2: Average time to build the graph and query 25 entities as the size of the data set increases.

4.4 Time Complexity and Scaling

To build the graph we only need to scan through the data set one time and accumulate weighted counts of the number of occurrences and co-occurrences. If $|L_i|$ is the length of link i then there are $\sum_{i=1}^{N_L} |L_i|^2$ co-occurrences to count. Thus if $|L_{max}|$ is the length of the largest link, accumulating the counts has an upper bound of $O(N_L * |L_{max}|^2)$. In addition we need to normalize the outgoing edge weights. If we let B_{max} be the largest number of outgoing edges (branching factor) from a single node, the entire approximation requires time $O(N_L * |L_{max}|^2 + N_P * B_{Max})$.

Table 1 illustrates how this algorithm scales. The entry t_{build} is the time in milliseconds (or 0.0 if less than 0.1 ms) it takes to approximate the graph once. Figure 2 shows how the algorithm scales with the number of links (from a given data set) used to build the model. Even for large numbers of links and entities this approximation is reasonably fast.

5 Learning the Parameters

The approximation presented above contains several parameters, such as the link type weighting and the temporal decay parameter. It may often be possible to set these parameters a-priori using domain specific knowledge. For example, a human operator may be able to guess which link types are most likely to be noisy and weight them accordingly. A second, more appealing approach is to learn these parameters while learning the underlying graph.

The cGraph algorithm uses a combination of these two approaches. If the parameters are known, they can be directly specified. If the parameters are unknown, then the cGraph algorithm tries to learn them using cross validation on the training set. The performance is measured by how closely the learned graph corresponds to the normalized counts from the cross validation set. Unless otherwise noted, the experiments below learn link weightings using cross validation and use $\beta = 0$.

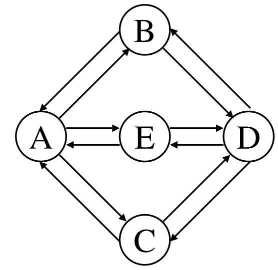

Figure 3: Neighbors on a graph.

6 Friends Identification

One of the primary goals of the cGraph algorithm is to answer such queries as: "List A's top 10 collaborators" and "Who is A most likely to start collaborating with next?" We call these queries the friend identification problem. As implied by the two different questions, there are two versions of the friend identification problem. The *All Friends* problem consists of answering the question: "Given the data who are the K entities with whom A is most likely to appear future links?" The *New Friends* problem consists of answering the question: "Given the data seen so far, who are the K entities with whom A has never appeared in a link and is most likely to appear with in future links?"

Another way to look at the friendship problem is as a distance measure where two entities that are "close" are more likely to collaborate in the future. Thus, in order to answer a friends query, we need to define the distant between two entities. The simplest distance is:

$$dist(A,B) = \frac{1}{W_{AB}} \tag{8}$$

Although this method is computationally cheap, it is unable to generalize past co-occurrences it has already seen. Specifically, if two entities have never co-occurred in a link then they will have an infinite distance from each other. For example in the graph shown in Figure 3 the entities A and D are not directly linked and thus $dist(A,D) = \infty$. If the goal is to look for connections between people that may not have explicitly appeared together in a link, a different distance metric is needed.

We use a slightly more complex distance function, based on a nonself-intersecting walk, that is able to capture the concept of "a friend of a friend." Specifically, we define the distance between A and B to be the probability of a fixed length, random, non-self-intersecting walk from A including the node B. Formally, we define the proximity of two entities as:

$$prox(A,B) = \sum_{v \in V_{(A,B,D)}} \left(\prod_{e_i \in v} P(e_i | A_j \forall j < i) \right) \tag{9}$$

where $V_{(A,B,D)}$ is the set of all nonself-intersecting walks of length less than or equal to D from A to B and $e_i \in v$ is a directed edge in v. We approximate this function by only considering paths of length at most three steps. All nodes

DATA SET	ENTITIES	LINKS	$\lvert L_{max}\rvert$	$\lvert L_{ave}\rvert$	t_{build}	B_{max}	B_{ave}	t_Q
LAB	115	94	16	3.3	0.0	44	7.11	0.0
INSTITUTE	456	1737	49	2.9	40.0	159	26.4	62.3
DRINKS	136	5325	6	2.8	10.0	131	13.5	1.2
MANUAL	4272	5976	14	2.2	100.0	131	4.0	0.1
CITESEER (25%)	105357	45348	122	2.8	2234.2	227	2.6	33.3
CITESEER (50%)	105357	90697	157	2.8	3229.2	333	4.3	45.8
CITESEER (75%)	105357	136046	272	2.8	4357.5	475	6.1	95.3
CITESEER (100%)	105357	181395	385	2.8	5490.0	595	7.7	151.5

Table 1: A summary of data sets: number of entities (N_P), number of links (N_L), maximum link length ($|L_{max}|$), average link length ($|L_{ave}|$), time (in milliseconds) to construct the graph (t_{build}), maximum branching factor of the resulting graph (B_{max}), average branching factor of the resulting graph (B_{ave}), and time (in milliseconds) to perform queries with depth 3 (t_Q).

which are not reachable from A in at most three steps are given an arbitrarily low proximity. Unless otherwise noted, this is the distance function that cGraph used.

This measure uses a depth first search over the graph and thus may be computationally expensive. Formally we can place an upper bound on the running time of $O(min\{B^D_{max}, N_p\})$. As this expression shows, the friendship query is independent of the number of links and scales polynomially with the branching factor of the graph. Table 1 and Figure 2 again illustrate how this operation scales to different size data sets; t_Q indicates the average time of 25 friendship queries on the learned graph.

It is worth noting that it is possible to use other measures of distance. For example consider using the expected length of the shortest nonself-intersecting path between two nodes (abbreviated ESD for expected shortest distance). The distance between two nodes, A and B, is calculated by finding the expected length of a nonself-intersecting random walk starting from A in treating B as a terminal state. Formally, we would define the distance of two entities as:

$$dist(A,B) = \sum_{v \in V_{(A,B)}} \left(Length(v) * \prod_{e_i \in v} P(e_i | A_j \forall j < i) \right) \quad (10)$$

where $V_{(A,B)}$ is the set of all nonself-intersecting walks from A to B and $e_i \in v$ is a directed edge in v.

The ESD distance was also implemented and tested for comparison. Early tests showed it often performed worse than cGraph with distance function given in (9) so it was not included in later experiments. The results for cGraph with the ESD distance are denoted as cGraph(ESD).

7 Data Sets

To test the cGraph algorithm, we used a variety of real world data sets. These data sets are described below and their sizes are summarized in Table 1.

1. The *Institute* data is a set of links of three different types that was collected from publicly available data listed on Carnegie Mellon University Robotic Institute's webpages. The first link type was co-publications with each paper corresponding to a link of its authors. These links were time stamped with their year of publication. The second link type was common research interest, with each of the 91 listed research interests serving as a link of everyone that officially specified it as an interest. Finally, the third link type of information was the advisor/advisee relation with each link consisting of one such pairing. This data set provides a perfect example of multiple link types with different amounts of relevant information. We would expect an advisor/advisee link to be a better predictor of future collaboration than simply a common research interest.
2. The *Citeseer* data is a collection of co-publication links from the Citeseer online library and index of computer science publications. Since we choose to represent entities using the author's first initial and last name, it was the case that a given label could actually corresponded to several authors. As a result, this representation introduced an inherent aliasing problem, adding noise to the data set. In addition subsets consisting of 25%, 50%, and 75% of the Citeseer links were used to demonstrate scaling properties. These links were chosen randomly and the entries in Table 1 are averages over 12 runs.
3. The *Drinks* data set consisted of a series of popular bar tending recipes found on various websites. The intuition behind the data set is that like groupings of people, various ingredients are be more likely to appear together in links. Each link consisted of a list of all ingredients (entities) contained in that drink.
4. The *Manual (Webpages)* Data is a set of links created by a human who manually read a set of public web pages and news stories related to terrorism and subjectively linked entities mentioned in the articles. These pages were often written/hosted by various governments and news organizations. Each link was created by hand from a single relation, such as `memberOf` or `funded`, found a web page and contained the entities for which this relation was mentioned on the page.
5. The *Lab* data consisted of co-publication links for members of our research lab. The entities consisted of all people in the research group and all people whom had published with someone in the research group. The links were again tagged with their year of publication.

8 Friends Identification Results

8.1 Competing Algorithms

The performance of the cGraph algorithm on the friends identification task was tested against a variety of other algorithms, including: competing approaches, simple strawman approaches, and approximate baseline performance approaches. The specific algorithms used, and how they chose which friends to return, are described below.

1. The *Random* algorithm provides an approximate baseline performance. When asked for a given entity's top K friends, the algorithm simply returns K random unique entities.

2. The *(Co-occurrence) Counting* algorithm provides a simple strawman algorithm. The top K predicted friends of A are those entities that have most often co-occurred with A.

3. The *Popular (Entity)* algorithm is another simple strawman algorithm. The algorithm simply counts up the number of links in which each entity has occurred and ranks the entities according to this count. The top K predicted friends of any entity are then predicted as the K most "popular" entities (excluding the entity itself) according to this ranking.

4. *Spectral Clustering* algorithms include a broad range of techniques based on eigenvalue analysis of feature vectors that have been shown effective in the field of network analysis and bibliometrics [Garfield *et al.*, 1978; Gibson *et al.*, 1998; Ng *et al.*, 2001]. This analysis can be viewed as a form of factoring: it begins with a matrix A, where each row i corresponds to a link and each column j corresponds to an entity. The value of $A(i,j)$ indicates whether entity j participated in link i. A singular value decomposition (SVD) decomposes A into sets of left and right singular vectors [Press *et al.*, 1993]. The left singular vectors correspond to canonical sets of entities, which may be interpreted as latent groups, while the right singular vectors correspond to the "mixing proportions" in which these groups should be combined to best approximate the observed links. Spectral clustering consists of assigning each observed link to the canonical set that dominates its mixture.

   The mixing proportions may also be used to estimate similarities between entities in one of a number of ways. In this paper, we estimate the "affinity" of two entities as the Euclidean distance between the vectors representing their mixing proportions. Friends are predicted as entities with the highest "affinity" to a given entity.

   In this paper, we examined both traditional SVD-based spectral clustering (labeled as "Spectral-S" in the accompanying tables) and a variation based on non-negative matrix factorization (NNMF) [Lee and Seung, 2001], labeled as "Spectral-N" below. NNMF produces a decomposition similar to the SVD, but adds the constraint that an entity's membership in the canonical sets must be strictly non-negative.

5. The *Group Detection Algorithm (GDA)* is a probabilistic technique for link analysis presented by Kubica *et. al.* [2001]. GDA models link generation by assuming that noisy links are generated from underlying groupings of entities. To this end, GDA learns the most likely groupings given the links it has seen. An entity's proximity to another entity is measured by the weighted sum of the groups to which both entities belong. More formally,

$$prox(A,B) = \sum_{G\, s.t.\, (A,B) \subset G} \frac{1}{size\, of\, G} \tag{11}$$

   Finally, an entity's top K friends are simply the K entities with the highest proximity. Due to running time considerations, GDA was only compared to the other algorithms on the single test set problems.

8.2 Tests and Results

Single Testset Tests

As an initial comparison we examined the relative test set performance of each algorithm on each dataset. Each dataset was divided into equal sized training and test sets. The performance of an algorithm is the percentage of guessed friends with which it actually co-occurred in the test set. The number of friends requested for each entity was equal to the true number of test set co-occurrences. This test provides a quick summary of the relative performance of the different algorithms on the different data sets. The results are shown in Table 2.

As shown in the table no one algorithm out performed the others on all of the tests. However the cGraph algorithm often performed among the best algorithms on a given dataset. It is important to appreciate that these results represent only a single test set/training set split.

From Table 2 the relative performance of the two different cGraph distance functions can be compared. Although the distance measure given in (10) does outperform the distance measure given in (9) on some tests, the difference is small compared to the difference with other algorithms. Further, on many of the tests where the distance measure given in (10) does not outperform the distance measure given in (9) this difference is relatively large. Thus the ESD distance was not included in the later tests.

Cross Validation Results

To more rigorously compare the performance of the algorithms, we performed sequential cross validation tests on the above data sets. Again the performance of an algorithm is the percentage of guessed friends with which it actually co-occurred in the test set. The number of friends requested for each entity was equal to the true number of test set co-occurrences. It is important to appreciate that sequential cross validation is very similar to a real world use of this algorithm. Specifically, given all data up to this time step, we would like to make queries about possible connections in the time step.

Table 3 shows the performance under the different friends metrics of various algorithms on the Institute, Citeseer, Drinks, Lab, and Manual data sets. The performance is given as the mean success rate for 10 fold sequential cross validation. The runs were treated as independent and results marked

Algorithm	Institute		Citeseer		Drinks		Lab		Terror	
	All	New	All	New	All	New	All	New	All	New
cGraph	20.03	0.51	32.77	**0.979**	42.64	8.32	12.67	6.60	**9.39**	**3.163**
cGraph(ESD)	**20.31**	0.56	30.04	0.415	32.45	4.19	12.67	6.60	2.21	0.212
Popular	10.50	**0.80**	0.62	0.109	**43.49**	**11.53**	14.90	8.68	3.78	1.169
Counting	20.26	0.33	**33.30**	0.003	40.75	3.32	15.92	8.92	8.21	0.041
Random	2.19	0.29	0.02	0.004	22.00	5.50	11.30	7.33	0.17	0.035
Spectral-N	9.38	0.48	3.15	0.064	20.46	4.30	17.81	11.86	0.95	0.230
Spectral-S	9.10	0.32	2.56	0.051	19.95	5.33	**19.69**	**12.35**	1.06	0.254
GDA	14.30	0.27	0.23	0.010	32.11	7.83	9.42	5.50	0.82	0.140

Table 2: The test set results (percentage of correctly guessed test set co-occurrences) for the various algorithms on the various data sets.

Algorithm	Institute		Citeseer		Drinks		Lab		Manual	
	All	New	All	New	All	New	All	New	All	New
cGraph	*__38.65__	*1.54	*29.06	*__0.200__	*46.10	**1.51**	**19.74**	1.86	**1.38**	0.21
Popular	10.98	*__1.65__	0.39	0.069	*__47.49__	0.84	14.50	**2.27**	1.37	**0.41**
Counting	34.56	0.18	*__29.49__	0.001	*47.37	0.88	17.27	1.13	1.29	0.03
Random	1.37	0.29	0.03	0.017	13.07	0.95	3.94	1.27	0.10	0.01
Spectral-N	12.52	*0.72	2.90	0.016	18.15	0.74	13.85	2.02	0.44	0.09
Spectral-S	13.16	*0.63	2.31	0.024	14.26	0.66	14.17	1.95	0.52	0.17

Table 3: The average 10 fold sequential cross validation results (percentage of correctly guessed test set co-occurrences) for the various algorithms on the various data sets.

with a * are significantly different (using $\alpha = 0.05$) from those that are not marked by stars. If no entry in the column is marked with a * then the performance of most algorithms on that data set and metric were not significantly different. Thus the *'s are left out for clarity.

Like Table 2, Table 3 illustrates that there is no one algorithm that always outperforms the other algorithms. It is important to appreciate though that on all of the data sets and metrics cGraph is among the best performers. On the Institute, Citeseer, and Drinks data it is significantly better than many of the other algorithms. Further, even when it is not the single best performer, its performance is very often practically close to that of the best algorithm. Another interesting trend shown in the table is the good performance of the Popular algorithm. This success suggests a possible extension to the cGraph algorithm by incorporating priors for the entities.

Effect of Beta Term

The above results do not address whether temporal weighting improves performance. To answer this, the effect of temporal weighting was tested using sequential cross validation on the 1606 publication links in the Institute data set. These links were time stamped with the year of the publication. Under both test metrics, the goal was roughly to predict collaborations in the Kth subset given the $K-1$ previous subsets. Results for several values of β are shown in Table 4. For both metrics, the performance of cGraph with $\beta = 0.5$ was significantly better than with $\beta = 0.0$ with a significance level of $\alpha = 0.05$.

Initially increasing β can lead to increased performance at predicting future collaborations by placing more weight on recent collaborations. But as β continues to increase, the performance decreases, because even relatively recent information is weighted lightly. In fact when β grows too large all information before the current time is effectively ignored. This trend is reflected in Figures 4 and 5, which shows cGraph's All and New Friends performance respectively for the Institute data set with several values of β.

β	All Friends	New Friends
0.0	35.86	0.66
0.5	40.99	0.92
2.0	41.25	0.79

Table 4: The average 10 fold sequential cross validation results (percentage of correctly guessed test set co-occurrences) on the Institute data set for cGraph with different temporal decay rates.

9 Experiments with Known Graphs

Another important test of cGraph is whether it actually learns the "correct" graph or a good approximation. Queries to an incorrect graph are at best unreliable. As a test, we compared learned graphs to known graphs that generated the link data.

In addition to evaluating the performance of the cGraph algorithm, we compared its performance to that of several other learning algorithms. The first algorithm was a simplified version of the cGraph algorithm using fixed, equal link

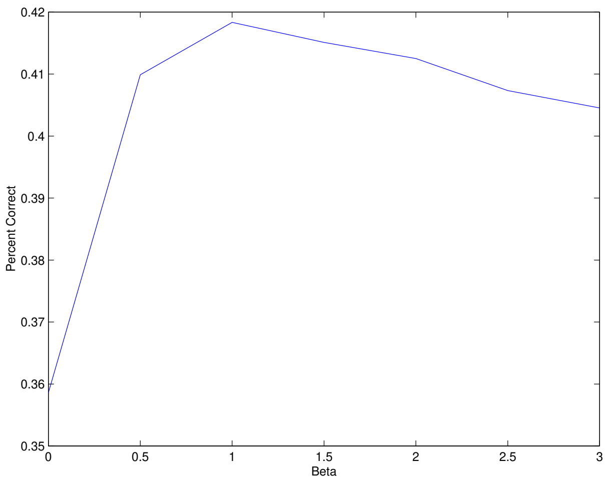

Figure 4: Performance of cGraph on All Friends metric with various settings of β.

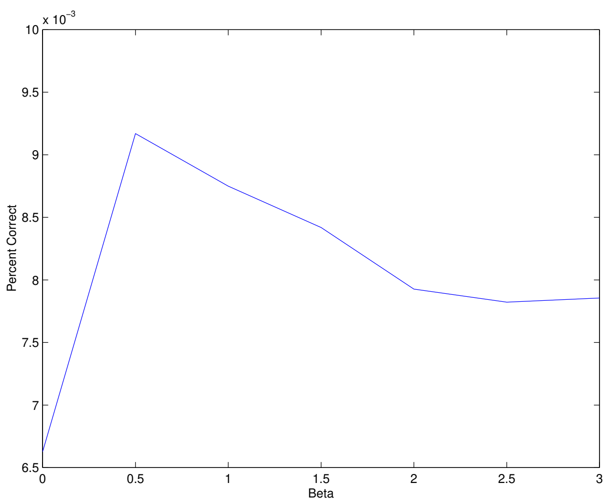

Figure 5: Performance of cGraph on New Friends metric with various settings of β.

type weights. This algorithm counts links weighted solely by the size of the link and thus is similar to the approaches presented in [Newman, 2001] and [Kautz *et al.*, 1997]. The second algorithm was an unweighted link counting approach, which did not even take into account link size. The third algorithm used a hill-climbing search to find the MLE graph and weights as described in Section 3.2. The underlying generative model was assumed to be the random walk model on all of the tests. It is important to appreciate that we do not expect MLE to necessarily perform better, because the structure with the highest likelihood may not be the true underlying structure. The fourth algorithm also used a hill-climbing search to find the MLE graph for the random walk model, but assumed fixed, equal link type weights. Finally, in order to provide an approximate baseline comparison, the fifth algorithm simply generated a random graph.

Algorithm	20 Entities Walk	20 Entities Tree	50 Entities Walk	50 Entities Tree
cGraph (Learned)	1.47	1.52	*3.76	*3.78
cGraph (Fixed)	1.96	2.01	5.01	4.96
MLE (Learned)	*1.23	*1.35	10.05	10.12
MLE (Fixed)	1.34	1.46	12.46	12.40
Counting	2.40	2.44	6.04	5.94
Random Graph	11.74	11.34	29.83	30.33

Table 5: Mean error (squared distance between the learned and known graphs) in reconstructing a graph.

9.1 Data Generation for "Known Graph" Tests

At the start of each test, a ground truth graph was randomly generated with a fixed number of entities and fixed link type weighting. Edges added between two nodes were added in both directions, but often had different weights. A set of 2,000 links was then generated using one of the generative models. In addition noise links, consisting of K random entities, were also added to the data.

Whether a link was noisy or not was determined by first randomly choosing a link type then choosing to make the link a noise link with probability equal to $1 - weight(linktype)$. All tests used three different link types with fixed weights of 0.9, 0.9, and 0.5. Thus while one link type contained significantly more noisy links than the others, none of the link types were completely noise free.

9.2 Results on Known Graphs

Table 5 summarizes the resulting performance of the algorithms. Each column consists of the average error over 50 tests using the indicated generative model and number of entities. The error metric itself is the squared distance between the learned and known graphs:

$$error = \sum_A \sum_B (W_{AB}^{true} - W_{AB}^{learned})^2 \qquad (12)$$

where W_{AB}^i is the weight of the edge from A to B in graph i. This metric provides a reasonable approximation of the distance between two graphs and is computationally efficient to calculate. The entries in Table 5 marked with a * are significantly better than all unmarked entities in their column with $\alpha = 0.05$. In fact the difference in performance of almost all algorithms in a given column is statistically significant at $\alpha = 0.05$.

The results show a clear advantage to weighting the different link types and an advantage to using the cGraph algorithm over the MLE as the number of entities grows. Although the MLE algorithm outperformed the cGraph algorithm for a small number of entities, this difference is comparatively small compared to the difference between cGraph and the other algorithms. More significantly, the cGraph algorithm has a distinct advantage on data sets with more links and larger links. The link sizes above were limited to at most 5 entities. For more links or links of larger sizes, simply evaluating the probability for MLE may not be computationally feasible.

10 Qualitative Effects

In addition to the quantitative results present above, the cGraph algorithm also provided qualitative results that agreed with intuition. Below we describe several such instances.

10.1 Effect of Beta Term

Interesting qualitative results could be seen when changing the temporal weighting function while analyzing the Lab dataset. For example, until recently two of the authors on this paper, Kubica and Moore, were at different institutions and had never collaborated with each other. Learning the graph with $\beta = 0$ reflected this fact giving a relatively weak connection between them. But during the past year, they have collaborated on several pieces of work. Thus we would expect the two of the authors to be more likely to collaborate on future projects together. Increasing β made the underlying graph reflect this. As β increased the strength of the connections between the two authors increased.

10.2 Weights and Known Types

Another interesting qualitative result is the result of learning the link type weights from the Robotics Institute data set. Of the three types of links, prior knowledge would lead us to expect common research interest links to be weighted the lowest and advisor/advisee links to be weighted the highest. The cGraph algorithm did exactly that, learning the following weightings: co-publication = 0.45, common research interest = 0.0, and advisor/advisee = 1.0. Thus research interest is effectively ignored when building the graph.

11 Related Work

As illustrated by several of our data sets and examples above, our work is similar to the analysis of co-citations, referral webs, and web structure. Within the areas of bibliometrics and web analysis eigenvector methods, such as spectral clustering, have been shown effective [Garfield *et al.*, 1978; Gibson *et al.*, 1998; Ng *et al.*, 2001]. These techniques presented a promising complement/alternative to our research and thus were considered in the above tests. It is important to appreciate that while our work is similar to the areas above, we are attempting to infer different information. Specifically, we wish to infer the underlying direct connections between entities. Co-citations and linked websites imply only relatively weak and indirect connections between the entities, such as common research topic.

The use of probabilistic models is another recent approach to the above problems [Cohn and Hofmann, 2001; Friedman *et al.*, 1999; Getoor *et al.*, 2001; Kubica *et al.*, 2002]. Cohn and Hofmann [2001] presented a model of document generation where they assume that documents are generated from a mixture of sources and that the contents of the document (citations and terms) are probabilistically determined from these sources. Similarly Kubica *et. al.* [2001] presented a model of link generation where links are generated from a single underlying group and then have noise added. These models differ significantly from ours in that we do not assume that a mixture of sources, but rather probabilistically determine an entity's membership in the link by which entities are already members of the link. Getoor *et. al.* [2001] present a technique that uses a probabilistic relational model to classify web pages. Here again there is an important difference in the type of information we are attempting to infer. We are not attempting to classify the entities or to create a full probabilistic model, but rather to predict the underlying direct connections between entities.

Our approach is similar to techniques described in work by Newman and Kautz *et. al.* [Kautz *et al.*, 1997; Newman, 2001]. Both Newman and Kautz *et. al.* describe computational methods for approximating and querying a social network given link data. Although both present estimation approaches based on a weighted count of co-occurrences, these approaches differ from our own. Both works use a heuristic guide to choose link weighting and neither work considers link reliability factors such as type or when the link occurred. Further, both works attempt answer questions different from the friendship problem. For example, Kautz *et. al.* attempt to make referrals by finding chains between two entities. While this is similar to our problem it lacks the important characteristic that multiple paths imply a stronger, although indirect, connection.

12 Complementary Techniques and Fields

Many techniques and fields require that the underlying structure of a group of entities is known or can easily be inferred from data. Our research complements these fields by providing an estimate of an underlying graph structure relating the entities. This approximation may be able to be used directly or as a motivation for designing more thorough experiments to uncover the true structure such as the use of targeted surveys.

One immediate such technique is that of social network analysis [Anderson *et al.*, 1999; Sparrow, 1991; Wasserman and Pattison, 1996]. For example p* creates predictive models for relational variables (such as our friendship question) from a collection of explanatory variables extracted from demographics and a known social network [Anderson *et al.*, 1999; Wasserman and Pattison, 1996]. Another example is the work of Schwartz and Wood where a graph is used to find subgraphs of people with related interests [Schwartz and Wood, 1992]. While Schwartz and Wood estimate the graph using link information, they limit this estimation to Boolean edges indicating the presence of any communication.

Further many other techniques, such as that of viral marketing, also assume that the social network is known or can be easily learned from such information as survey data. For example Richardson and Domingos mine knowledge sharing sites to create an underlying trust network by looking for "trusts" and "trusted by" links between entities [Richardson and Domingos, 2002]. While our technique does not explicitly look at graphs that model influence, it is possible that our ideas can be extended to complement this area of research.

13 Conclusion

We examined the problem of quickly inferring an underlying graph structure to capture relationships between entities. This structure is learned from a collection of noisy link data, which for many problems may be easier to obtain than the true underlying graph. In conjunction, we introduced the "friends" problem, which examines how strongly two entities are connected even if they have never appeared in a link together. These problems have promise for applications in such areas as: sociology, marketing, security, and the study of artificial networks.

There are several natural extensions that we leave as future work. First is the incorporation of the concept of "popularity of entities" into the model. Motivated by both a natural extension to the generative models and the success of the Popular algorithm on the new friends problem, the incorporation of priors into cGraph algorithm may boost performance. Second is an extension beyond positive pairwise relationships to include such relationships as A, B and C are all likely to appear together in pairs, but never as a triple.

Acknowledgements

Jeremy Kubica is supported by a grant from the Fannie and John Hertz Foundation. This research is supported by DARPA under award number F30602-01-2-0569. The authors would also like to thank Steve Lawrence for making the Citeseer data available.

References

[Anderson *et al.*, 1999] C. J. Anderson, S. Wasserman, and B. Crouch. A p* primer: Logit models for social networks. *Social Networks*, 21:37–66, 1999.

[Cohn and Hofmann, 2001] David Cohn and Thomas Hofmann. The missing link - a probabilistic model of document content and hypertext connectivity. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 430–436. MIT Press, 2001.

[Friedman *et al.*, 1999] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309. Morgan Kaufmann Publishers, 1999.

[Garfield *et al.*, 1978] Eugene Garfield, Morton V. Malin, and Henry Small. Citation data as science indicators. In Yehuda Elkana, Joshua Lederberg, Robert K. Merton, Arnold Thackray, and Harriet Zuckerman, editors, *Toward a Metric of Science: The Advent of Science Indicators*. John Wiley and Sons, 1978.

[Getoor *et al.*, 2001] Lise Getoor, Eran Segal, Ben Taskar, and Daphne Koller. Probabilistic models of text and link structure for hypertext classification. In *IJCAI01 Workshop on Text Learning: Beyond Supervision*, August 2001.

[Gibson *et al.*, 1998] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *Proc. 9th ACM Conference on Hypertext and Hypermedia*. ACM, 1998.

[Kautz *et al.*, 1997] Henry A. Kautz, Bart Selman, and Mehul A. Shah. The hidden web. *AI Magazine*, 18(2):27–36, 1997.

[Kubica *et al.*, 2002] Jeremy Kubica, Andrew Moore, Jeff Schneider, and Yiming Yang. Stochastic link and group detection. In *The Eighteenth National Conference on Artificial Intelligence*, pages 798–804. ACM Press, Jul 2002.

[Lee and Seung, 2001] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In Todd Leen, Thomas Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 556–562. MIT Press, 2001.

[Newman, 2001] M. E. J. Newman. Who is the best connected scientist? a study of scientific coauthorship networks. *Phys. Rev.*, 64(016131; 016132), 2001.

[Ng *et al.*, 2001] Andrew Y. Ng, Alice X. Zheng, and Michael I. Jordan. Link analysis, eigenvectors and stability. In Bernhard Nebel, editor, *IJCAI*, pages 903–910. Morgan Kaufmann Publishers, 2001.

[Press *et al.*, 1993] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, second edition, 1993.

[Richardson and Domingos, 2002] Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *The Eighteenth National Conference on Artificial Intelligence*, pages 61–70. ACM Press, Jul 2002.

[Schwartz and Wood, 1992] M. Schwartz and D. Wood. Discovering shared interests among people using graph analysis of global electronic mail traffic. *Communications of the ACM*, 36(8):78–89, 1992.

[Sparrow, 1991] M. Sparrow. The application of network analysis to criminal intelligence: an assessment of prospects. *Social Networks*, 13, 1991.

[Wasserman and Faust, 1994] Stanley Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[Wasserman and Pattison, 1996] Stanley Wasserman and Philippa Pattison. Logit models and logistic regression for social networks: I. an introduction to markov graphs and p*. *Psychometrika*, 60:401–425, 1996.