Dynamic Social Network Analysis using Latent Space Models

Purnamrita Sarkar, Andrew W. Moore
Center for Automated Learning and Discovery
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
psarkar,awm@cs.cmu.edu

ABSTRACT

This paper explores two aspects of social network modeling. First, we generalize a successful static model of relationships into a dynamic model that accounts for friendships drifting over time. Second, we show how to make it tractable to learn such models from data, even as the number of entities n gets large. The generalized model associates each entity with a point in p-dimensional Euclidean latent space. The points can move as time progresses but large moves in latent space are improbable. Observed links between entities are more likely if the entities are close in latent space. We show how to make such a model tractable (sub-quadratic in the number of entities) by the use of appropriate kernel functions for similarity in latent space; the use of low dimensional KD-trees; a new efficient dynamic adaptation of multidimensional scaling for a first pass of approximate projection of entities into latent space; and an efficient conjugate gradient update rule for non-linear local optimization in which amortized time per entity during an update is $O(\log n)$. We use both synthetic and real-world data on up to 11,000 entities which indicate near-linear scaling in computation time and improved performance over four alternative approaches. We also illustrate the system operating on twelve years of NIPS co-authorship data.

1. INTRODUCTION

Social network analysis is becoming increasingly important in many fields besides sociology, including intelligence analysis [1], marketing [2] and recommender systems [3]. Here we consider learning in systems in which relationships drift over time. In 2002, Raftery et al[4] introduced a model similar to Multidimensional Scaling in which entities are associated with locations in p-dimensional space, and links are more likely if the entities are close in latent space. In this paper we formulate an extension of this static model that allows link prediction and visualization in a dynamic setting. Unlike most of the existing models our work takes the sequential aspect of the data into account.

A friendship graph is one in which the nodes are entities and two entities are linked if and only if they have been observed to collaborate in some way. We try to embed an evolving friendship graph in a p dimensional latent space. We expect that with time entities will come closer together forming new groups, or moving away from one another. These kind of changes in interaction patterns are very common in social networks, where people move in and out of neighborhoods forming new friend circles. This model will also help us predict whether two entities will form a connection at timestep t, given the kind of relation they had over the past timesteps. At first sight a much simpler algorithm might seem preferable: predict x and y are linked at time t if and only if they were linked at time $t-1$. But in many cases even without having been linked at all, two entities can be very close to one another because of common friends or friends of friends. In this paper we suppose that each observed link is associated with a discrete timestep, so each timestep produces its own graph of observed links, and information is preserved between timesteps by two assumptions. First we assume entities can move in latent space between timesteps, but large moves are improbable. Second, we make a standard Markov assumption that latent locations at time $t+1$ are conditionally independent of all previous locations given latent locations at time t and that the observed graph at time t is conditionally independent of all other positions and graphs, given the locations at time t (see Figure 1). This is the same assumption as in HMMs and Kalman Filters.

Let G_t be the graph of observed pairwise links at time t. Assuming n entities, and a p-dimensional latent space, let X_t be an $n \times p$ matrix in which the i^{th} row, called x_i, corresponds to the latent position of entity i at time t. Our conditional independence structure is shown in Figure 1. For most of this paper we treat the problem as a tracking problem in which we estimate X_t at each timestep as a function of the current observed graph G_t and the previously estimated positions X_{t-1}. We want

$$\begin{aligned} X_t &= \arg\max_X P(X|G_t, X_{t-1}) \\ &= \arg\max_X P(G_t|X)P(X|X_{t-1}) \end{aligned} \quad (1)$$

if we put a uniform prior on X_t. In Section 2 we design models of $P(G_t|X_t)$ and $P(X_t|X_{t-1})$ that meet our modeling needs *and* which have learning times that are tractable as n gets large. In Sections 3 and 4 we introduce a two-stage procedure for locally optimizing equation (1). The first stage generalizes linear multidimensional scaling algorithms to the dynamic case while carefully maintaining the ability to computationally exploit sparsity in the graph. This gives an approximate estimate of X_t. The second stage re-

fines this estimate using an augmented conjugate gradient approach in which gradient updates can use KD-trees over latent space to allow $O(n \log n)$ computation per step.

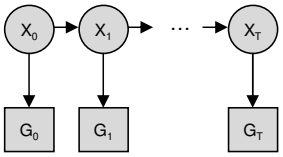


Figure 1: Model through time

2. THE DSNL (DYNAMIC SOCIAL NETWORK IN LATENT SPACE) MODEL

Let $d_{ij} = |x_i - x_j|$ be the Euclidean distance between entities i and j in latent space at time t. We will not use a t subscript on these variables except where it is needed for clarity. We denote linkage at time t by $i \sim j$, and absence of a link by $i \not\sim j$. $p(i \sim j)$ denotes the probability of observing the link. We use $p(i \sim j)$ and p_{ij} interchangeably.

2.1 Observation Model

Following [4] the probability of a link between i and j, denoted as p_{ij}^L is

$$p_{ij}^L = \frac{1}{1 + e^{(d_{ij} - \alpha)}} \tag{2}$$

where α is a constant whose significance is explained shortly. $P(G_t|X_t)$ is then simply

$$p(G_t|X_t) = \prod_{i \sim j} p_{ij} \prod_{i \not\sim j} (1 - p_{ij})$$

The likelihood score function intuitively measures how well the model explains pairs of entities who are actually connected in the training graph as well as those that are not. So far this model is similar to [4]. To extend this model to the dynamic case, we now make two important alterations.

First, we allow entities to vary their sociability. Some entities participate in many links while others are in few. We give each entity a *radius*, which will be used as a sphere of interaction within latent space. We denote entity i's radius as r_i.

We introduce the term r_{ij} to replace α in Equation (2). r_{ij} is the maximum of the radii of i and j. Intuitively, an entity with higher degree will have a larger radius. Thus we define the radius of entity i with degree δ_i as $c(\delta_i + 1)$ so that r_{ij} is $c \times (max(\delta_i, \delta_j) + 1)$, and c will be estimated from the data. In practice, we estimate the constant c by a simple line-search on the score function. The constant 1 ensures a nonzero radius.

The second alteration is to weigh the link probabilities by a kernel function. We alter the simple logistic link probability p_{ij}^L, such that two entities have high probability of linkage only if their latent coordinates are within radius r_{ij} of one another. Beyond this range there is a constant noise probability ρ of linkage. For later optimization we will need the kernelized function to be continuous and differentiable at $d_{ij} = r_{ij}$. Thus we pick the biquadratic kernel.

$$\begin{aligned} K(d_{ij}) &= (1 - (d_{ij}/r_{ij})^2)^2, && \text{when } d_{ij} \le r_{ij} \\ &= 0, && \text{otherwise} \end{aligned} \tag{3}$$

Using this function we redefine our link probability as

$$p_{ij} = \frac{1}{1 + e^{(d_{ij} - r_{ij})}} K(d_{ij}) + \rho(1 - K(d_{ij}))$$

This is equivalent to having,

$$\begin{aligned} p_{ij} &= p_{ij}^L K(d_{ij}) + \rho(1 - K(d_{ij})) && \text{when } d_{ij} \le r_{ij} \\ &= \rho && \text{otherwise} \end{aligned}$$

We plot this function in Figure 2B.

Thus the full expression of the first part of the model log-likelihood is given by,

$$\begin{aligned} &\log P(G_t|X_t) \\ &= \sum_{i \sim j} \log p(i \sim j) + \sum_{i \not\sim j} \log p(i \not\sim j) \\ &= \sum_{i \sim j, d_{ij} <= r_{ij}} \log p(i \sim j) + \sum_{i \not\sim j, d_{ij} <= r_{ij}} \log p(i \not\sim j) \\ &+ \#(i \sim j, d_{ij} > r_{ij}) \log \rho + \#(i \not\sim j, d_{ij} > r_{ij}) \log(1 - \rho) \end{aligned}$$

where #(expression) denotes the number of pairs satisfying the expression.

2.2 Transition Model

The second part of the score penalizes large displacements from the previous time step. We use the most obvious Gaussian model: each coordinate of each latent position is independently subjected to a Gaussian perturbation with mean 0 and variance σ^2. Thus

$$\log P(X_t|X_{t-1}) = -\sum_{i=1}^{n} |X_{i,t} - X_{i,t-1}|^2 / 2\sigma^2 + const \tag{4}$$

Here we are trying to optimize the log-likelihood of the graphs $G_{1:t}$, conditioned on the latent positions $X_{1:t}$, where $t \le T$, T being the total number of timesteps. This is a forward inference, since we constrain the positions on each timestep to be similar to the last time-step only. However we also present a global optimization of all time-steps together in section 5.4.

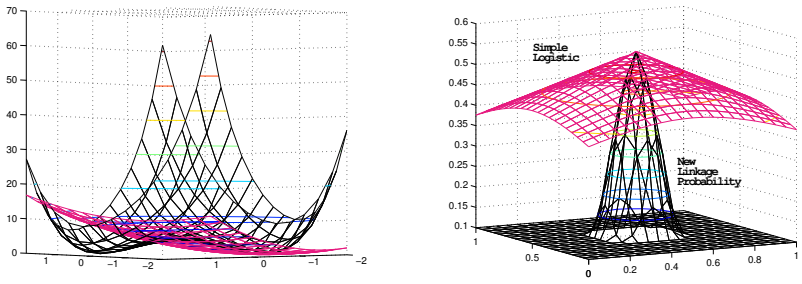
3. LEARNING STAGE ONE: LINEAR APPROXIMATION

We generalize classical multidimensional scaling (MDS) [5] to get an initial estimate of the positions in the latent space. We begin by recapping what MDS does. It takes as input an $n \times n$ matrix of non-negative distances $\mathfrak{D}$ where $\mathfrak{D}_{i,j}$ denotes the target distance between entity i and entity j. It produces an $n \times p$ matrix X where the i^{th} row is the position of entity i in p-dimensional latent space. Let the coordinates of n points in a p dimensional Euclidean space be given by $x_i, (i = 1 : n)$ where $x_i = (x_{i1}, ..., x_{ip})$. Let X denote the unknown coordinate matrix. The Euclidean distance between points i and j is given by

$$d_{ij}^2 = \sum_{k=1}^{p} (x_{ik} - x_{jk})^2 \tag{5}$$

Also let $\tilde{\mathfrak{D}}$ denote XX^T, such that

$$\tilde{d}_{ij} = \sum_{k=1}^{p} x_{ik} x_{jk} = x_i^T x_j \tag{6}$$



(A) (B)

Figure 2: A.The actual (flat, with one minimum), and the modified (steep with two minima) constraint functions, for two dimensions, with X_t varying over a 2-d grid, from $(-2,-2)$ to $(2,2)$, and $X_{t-1} = (1,1)$. B. The actual logistic function, and our kernelized version with $\rho = 0.1$.

$$d_{ij}^2 = x_i^T x_i + x_j^T x_j - 2x_i^T x_j = \tilde{d}_{ii} + \tilde{d}_{jj} - 2\tilde{d}_{ij}$$

If X is centered, then we can write

$$\frac{1}{n}\sum_{i=1}^{n} d_{ij}^2 = \frac{1}{n}\sum_{i=1}^{n} \tilde{d}_{ii} + \tilde{d}_{jj}$$
$$\frac{1}{n}\sum_{j=1}^{n} d_{ij}^2 = \tilde{d}_{ii} + \frac{1}{n}\sum_{j=1}^{n} \tilde{d}_{jj}$$
$$\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}^2 = \frac{2}{n}\sum_{i=1}^{n} \tilde{d}_{ii} \quad (7)$$

From equations 6 and 7 it follows that,

$$\tilde{d}_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{i\cdot}^2 - d_{\cdot j}^2 + d_{\cdot\cdot}^2) \quad (8)$$

Substituting a_{ij} for $-\frac{1}{2}d_{ij}^2$ we have a new matrix A, and

$$\tilde{d}_{ij} = a_{ij} - a_{i\cdot} - a_{\cdot j} + a_{\cdot\cdot}$$

Thus we have ,

$$\tilde{\mathfrak{D}} = XX^T = HAH = H(-\frac{1}{2}\mathfrak{D}^2)H$$

where H is the idempotent centering matrix, such that

$$h_{ij} = -\frac{1}{n} \qquad \text{if i} \neq \text{j}$$
$$= 1 - \frac{1}{n} \qquad \text{otherwise}$$

However we do not know the true distance matrix $\mathfrak{D}$ or the resulting similarity matrix $\tilde{\mathfrak{D}}$. Therefore classical MDS works with the dissimilarity matrix D obtained from the data [6]. $\tilde{D}$ is the similarity matrix obtained from D using

$$\tilde{D} = H(-\frac{1}{2}D^2)H \quad (9)$$

Let Γ be the matrix of the eigenvectors of $\tilde{D}$, and Λ be a diagonal matrix with the corresponding eigenvalues. Denote the matrix of the first p positive eigenvalues by Λ_p and the corresponding columns of Γ by Γ_p. From this follows the expression of classical MDS, i.e. $X = \Gamma_p \Lambda_p^{\frac{1}{2}}$. MDS finds

$$\arg\min_X |\tilde{D} - XX^T|_F$$

where $|\cdot|_F$ denotes the Frobenius norm [6].

Two questions remain. Firstly, what should be our target distance matrix D? Secondly, how should this be extended to account for time? The first answer follows from [4] and defines D_{ij} as length of the shortest path, $nhops_{ij}$ from i to j in graph G. We restrict this length to a maximum of three hops in order to avoid the full n^2 computation of all-shortest paths. Thus D is a dense mostly constant matrix, such that,

$$D_{ij} = nhops_{ij} \qquad \text{if } nhops_{ij} < c$$
$$= c \qquad \text{otherwise}$$

In spite of having this linear approximation step as an initialization to the nonlinear optimization described in Section 4, we want to account for the temporal aspect, so that we begin with a educated guess. Therefore we do not want the positions of entities to change drastically from one time step to another. Hence we try to minimize $|X_t - X_{t-1}|_F$ along with the main objective of MDS. Let $\tilde{D}_t$ denote the $\tilde{D}$ matrix derived from the graph at time t. We formulate the above problem as minimization of $|\tilde{D}_t - X_t X_t^T|_F + \lambda|X_t - X_{t-1}|_F$, where λ is a parameter which controls the importance of the two parts of the objective function. The above does not have a closed form solution. However, by constraining the objective function further, we can obtain a closed form solution for a closely related problem. The idea is to work with the distances and not the positions themselves. Since we are learning the positions from distances, we change our constraint (during this linear stage of learning) to encourage the pairwise distance between all pairs of entities to change little between each time step, instead of encouraging the individual coordinates to change little. Hence the new function we try to minimize is given by

$$|\tilde{D}_t - X_t X_t^T|_F + \lambda|X_t X_t^T - X_{t-1}X_{t-1}^T|_F \quad (10)$$

Minimizing Equation 10 is equivalent to minimizing the trace of

$$\begin{aligned}
&(\tilde{D}_t - X_t X_t^T)^T(\tilde{D}_t - X_t X_t^T) + \\
&\lambda(X_t X_t^T - X_{t-1}X_{t-1}^T)^T(X_t X_t^T - X_{t-1}X_{t-1}^T) \\
&= \tilde{D}_t^2 + \lambda(X_{t-1}X_{t-1}^T)^2 - (\tfrac{\tilde{D}_t + \lambda X_{t-1}X_{t-1}^T}{1+\lambda})^2 + \\
&\quad (1+\lambda)(X_t X_t^T - \tfrac{\tilde{D}_t + \lambda X_{t-1}X_{t-1}^T}{1+\lambda})^2 \\
&= \text{Constant w.r.t } X_t + (1+\lambda)(X_t X_t^T - \tfrac{\tilde{D}_t + \lambda X_{t-1}X_{t-1}^T}{1+\lambda})^2
\end{aligned} \quad (11)$$

The trace of Equation (11) is minimized at an affine combination of the current information from the graph, and the coordinates at the last timestep. Namely, the new solution satisfies,

$$X_t X_t^T = \frac{1}{1+\lambda}\tilde{D}_t + \frac{\lambda}{1+\lambda}X_{t-1}X_{t-1}^T \tag{12}$$

We plot the trace of the two constraint functions in Figure 2A.The steeper surface belongs to our second constraint. It can be seen that this new function has two minima, namely $X_t = \pm X_{t-1}$ whereas the first one is much more flat, and has an unique minima at X_{t-1}. An eigendecomposition of the right hand side of the solution, as in Equation 12, minimizes the objective function. We shall explain the role of λ, by varying it between two extreme values. When λ is zero, $X_t X_t^T$ equals $\tilde{D}_t$, and we ignore all information except the current graph. When $\lambda \to \infty$, $X_t X_t^T$ equals $X_{t-1}X_{t-1}^T$ and we are entirely concerned with keeping entities stationary in the latent space. So λ works like a forgetting factor.

We now have a method which finds latent coordinates for time t that are consistent with G_t and have similar pairwise distances as X_{t-1}. But although all pairwise distances may be similar, the coordinates may be very different. Indeed, even if λ is very large and we only care about preserving distances, the resulting X_t may be any reflection, rotation or translation of the original X_{t-1}. We solve this by applying the *Procrustean* transform to the solution X_t of Equation 12. This transform finds the linear area-preserving transformation of X_t that brings it closest to the previous configuration X_{t-1}. The solution is unique if $X_t^T X_{t-1}$ is nonsingular [7], and for zero centered X_t and X_{t-1}, is given by $X_t^* = X_t U V^T$, where $X_t^T X_{t-1} = USV^T$, using conventional notation for Singular Value Decomposition(SVD).

Before moving onto stage two's nonlinear optimization we must address the scalability of stage one. The naive implementation (SVD of the matrix from equation 12) has a cost of $O(n^3)$, for n nodes, since both $\tilde{D}_t$, and $X_t X_t^T$, are dense $n \times n$ matrices. We use the power method [8], to find the p eigenvectors and values. The naive implementation of this is $O(n^2)$ for n nodes, since both $\tilde{D}_t$ and $X_t X_t^T$ are dense $n \times n$ matrices. However with some care its possible to exploit the dense mostly constant structure of the distance matrix D_t obtained from the graph. The power method involves the multiplication of a matrix with a vector iteratively, till it converges to the first eigenvalue-vector pair. Choosing the next starting vector to be perpendicular to the first eigenvector, we find the second eigenvalue-vector pair too. The beauty of this method is that, in case the underlying matrix is sparse, the matrix-vector multiplication involves only $n^2 \times f$ computation per iteration, where f is the fraction of nonzero entries in the matrix. We briefly sketch the equations to show how we avoid $O(n^2)$ computation despite the fact that our matrices are dense. The heart of power method is computing $v' = \tilde{D}v$, where $\tilde{D}$ is defined in Equation(9). Thus each iteration involves computing n entries of v'.

We first represent $D' = D_t^2$ as a linear combination between a dense fully constant matrix D'_d and a sparse matrix D'_s. All the entries of D'_d have the constant value $c' = c^2$. We shall denote each entry of D'_d ,and D'_s by dd_{ij}, and ds_{ij} respectively. For simplifying the notation we drop the suffix t for the time being. We also denote an element-wise square by D^2 in this section.

$$\tilde{D} = HAH = H(-\frac{1}{2}D^2)H = -\frac{1}{2}HD^2H$$

$$D' = D^2 = D'_d - D'_s$$

$$d'_{ij} = c' - ds_{ij}$$

$$d'_{i\cdot} = c' - ds_{i\cdot}$$

$$d'_{\cdot j} = c' - ds_{\cdot j}$$

$$d'_{\cdot\cdot} = c' - ds_{\cdot\cdot}$$

$$\begin{aligned}\tilde{d}_{ij} &= -\frac{1}{2}(d'_{ij} - d'_{i\cdot} - d'_{\cdot j} + d'_{\cdot\cdot}) \\ &= (-\frac{1}{2})(-(ds_{ij} - ds_{i\cdot} - ds_{\cdot j} + ds_{\cdot\cdot})) \\ &= \frac{1}{2}(ds_{ij} - ds_{i\cdot} - ds_{\cdot j} + ds_{\cdot\cdot})\end{aligned}$$

$$\tilde{D} = \frac{1}{2}HD_sH$$

$$\begin{aligned}\sum_{j=1}^{n}\tilde{d}_{ij}v_j &= \sum_{j=1}^{n}(ds_{ij} - ds_{i\cdot} - ds_{\cdot j} + ds_{\cdot\cdot})v_j \\ &= \sum_{j=1}^{n} ds_{ij}v_j - ds_{i\cdot}\sum_{j=1}^{n} v_j - \sum_{j=1}^{n} ds_{j\cdot}v_j + ds_{\cdot\cdot}\sum_{j=1}^{n} v_j\end{aligned} \tag{13}$$

The complexity of calculating the above, is as follows: the first part requires $O(n^2 f)$ time in one iteration of power method, since D'_s is sparse; The row, column, and overall means i.e. $ds_{\cdot j}$, $ds_{i\cdot}$, and $ds_{\cdot\cdot}$ need to be computed once, with a cost $O(n^2 f)$ overall . Once they are computed, the rest of the terms of Equation(13) take $O(n)$ time per iteration of the power method. Also, we don't create the $X_t X_t^T$ matrix. Since we use the power method, in one iteration the computation comes down to,

$$\begin{aligned}X_t X_t^T v &= X_t(X_t^T v) \\ &= (n \times p)((p \times n)(n \times 1)) = (n \times p)(p \times 1)\end{aligned}$$

The above has a time complexity of $O(pn)$, where p is the number of dimensions of the latent space. Thus the net cost of power-method is $O(n^2 f + n + pn)$ per iteration.

4. STAGE TWO: NONLINEAR SEARCH

Stage one finds reasonably consistent locations for entities which fit our intuition, but it is not tied in any way to the probabilistic model from Section 2. Stage two uses this reasonable initial guess as a starting point and then applies nonlinear optimization directly to the model in Equation 1. We use conjugate gradient (CG) which was the most effective of several alternatives attempted. The most important practical question is how to make these gradient computations tractable, especially when the model likelihood involves a double sum over all entities.

We must compute the partial derivatives of $logP(G_t|X_t) + logP(X_t|X_{t-1})$ with respect to all values $x_{i,k,t}$ for $i \in 1...n$ and $k \in 1..p$. First consider the $P(G_t|X_t)$ term:

$$\frac{\partial \log P(G_t|X_t)}{\partial X_{i,k,t}} = \sum_{j,i\sim j} \frac{\partial \log p_{ij}}{\partial X_{i,k,t}} + \sum_{j,i\not\sim j} \frac{\partial log(1-p_{ij})}{\partial X_{i,k,t}}$$
$$= \sum_{j,i\sim j} \frac{\partial p_{ij}/\partial X_{i,k,t}}{p_{ij}} - \sum_{j,i\not\sim j} \frac{\partial p_{ij}/\partial X_{i,k,t}}{1-p_{ij}} \quad (14)$$

$$\frac{\partial p_{ij}}{\partial X_{i,k,t}} = \frac{\partial (p_{ij}^L K + \rho(1-K))}{\partial X_{i,k,t}}$$
$$= K\frac{\partial p_{ij}^L}{\partial X_{i,k,t}} + p_{ij}^L \frac{\partial K}{\partial X_{i,k,t}} - \rho\frac{\partial K}{\partial X_{i,k,t}} = \psi_{i,j,k,t}$$

However K, the biquadratic kernel introduced in Equation 3, evaluates to zero and has a zero derivative when $d_{ij} > r_{ij}$. Plugging this information in (14), we have,

$$\partial p_{ij}/\partial X_{i,k,t} = \begin{cases} \psi_{i,j,k,t} & \text{when } d_{ij} \le r_{ij}, \\ 0 & \text{otherwise.} \end{cases}$$

Equation (14) now becomes

$$\frac{\partial \log p(G_t|X_t)}{\partial X_{i,k,t}} = \sum_{\substack{j,i\sim j \\ d_{ij}\le r_{ij}}} \frac{\psi_{i,j,k,t}}{p_{ij}} - \sum_{\substack{j,i\not\sim j \\ d_{ij}\le r_{ij}}} \frac{\psi_{i,j,k,t}}{1-p_{ij}}$$

This simplification is very important because we can now use a spatial data structure such as a KD-tree in the low dimensional latent space to retrieve all pairs of entities that lie within each other's radius in time $O(rn + n\log n)$ where r is the average number of in-radius neighbors of an entity [9; 10]. The computation of the gradient involves only those pairs. A slightly more sophisticated trick lets us compute $\log P(G_t|X_t)$, in $O(rn + n\log n)$ time.
From equation(4), we have

$$\frac{\partial \log p(X_t|X_{t-1})}{\partial X_{i,k,t}} = -\frac{X_{i,k,t} - X_{i,k,t-1}}{\sigma^2}$$

In the early stages of Conjugate Gradient, there is a danger of a plateau in our score function in which our first derivative is insensitive to two entities that are connected, but are not within each other's radius. To aid the early steps of CG, we add an additional term to the score function, which penalizes all pairs of connected entities according to the square of their separation in latent space , i.e. $\sum_{i\sim j} d_{ij}^2$.
Weighting this by a constant $pConst$, our final CG gradient is

$$\frac{\partial Score_t}{\partial X_{i,k,t}} = \frac{\partial \log p(G_t|X_t)}{\partial X_{i,k,t}} + \frac{\partial \log p(X_t|X_{t-1})}{\partial X_{i,k,t}}$$
$$- pConst \times 2 \sum_{\substack{j \\ i\sim j}} (X_{i,k,t} - X_{j,k,t})$$

4.1 An example of the different steps of the model

Here we give the figures from the different timesteps through our algorithm, on a simulated dataset consisting of 10 entities. The true model represents the actual spatial positions of the entities, from which the links were generated using our probabilistic model. Figures 3(A) and (C) show the true model at the first and second timestep. Figure 3(B) gives the result of the entire algorithm i.e. our MDS with $\lambda = 0$, i.e. classical MDS on the data in (A) followed by conjugate gradient. Through figures 3(D), (E), and (F) we motivate the time-variant MDS, Procrustean transform and finally the conjugate gradient step in our algorithm.
Figure (D) shows the result of time-variant MDS with $\lambda = 10$, on the data in (C), and the coordinates learned in (B). Note that though entity *ody* is no more connected to *amy* in timestep 2, it is not placed far apart from the latter, since they were connected in the former timestep. This shows how the initial MDS step takes into consideration the coordinates from the last timestep as well. Now look at (E). It is obtained by applying Procrustean transform on the coordinates from (D), so that it aligns as closely as possible to the coordinates in (B). Careful observation reveals that this step in this certain case just has rotated the coordinates in step (D) to have the same orientation as in (B). This demonstrates the necessity of using the Procrustean transform. As MDS deals directly with distances the resulting configuration can be rotated without affecting the mutual distances between the entities. The final figure is (F), which gives the result of applying conjugate gradient on step (E), and also the re-estimated radii from the current time-step. It is evident from the figures that for these small number of entities conjugate gradient after MDS does not yield much improvement. However for larger number of entities MDS helps only as a very educated guess to initialize conjugate gradient, which subsequently makes very significant improvements.

5. RESULTS

We report experiments on synthetic data generated by a model described below and the NIPS co-authorship data [11], and some large subsets of citeseer. We investigate three things: ability of the algorithm to reconstruct the latent space based only on link observations, anecdotal evaluation of what happens to the NIPS data, and scalability.

5.1 Comparing with ground truth

We generate synthetic data for six consecutive timesteps. At each timestep the next set of two-dimensional latent coordinates are generated with the former positions as mean, and a gaussian noise of standard deviation $\sigma = 0.01$. Each entity is assigned a random radius. At each step , each entity is linked with a relatively higher probability to the ones falling within its radius, or containing it within their radii. There is a noise probability of 0.1, by which any two entities i and j outside the maximum pairwise radii r_{ij} are connected. We generate graphs of sizes 20 to 1280, doubling the size every time. Accuracy is measured by drawing a test set from the same model, and determining the ROC curve for predicting whether a pair of entities will be linked in the test set. We experiment with six approaches:

A. The True model that was used to generate the data (this is an upper bound on the performance of any learning algorithm).

B. The DSNL model learned using the above algorithms.

C. A random model, guessing link probabilities randomly (this should have an AUC of 0.5).

D. The *Simple Counting* model (Control Experiment). This ranks the likelihood of being linked in the testset according to the frequency of linkage in the training set.

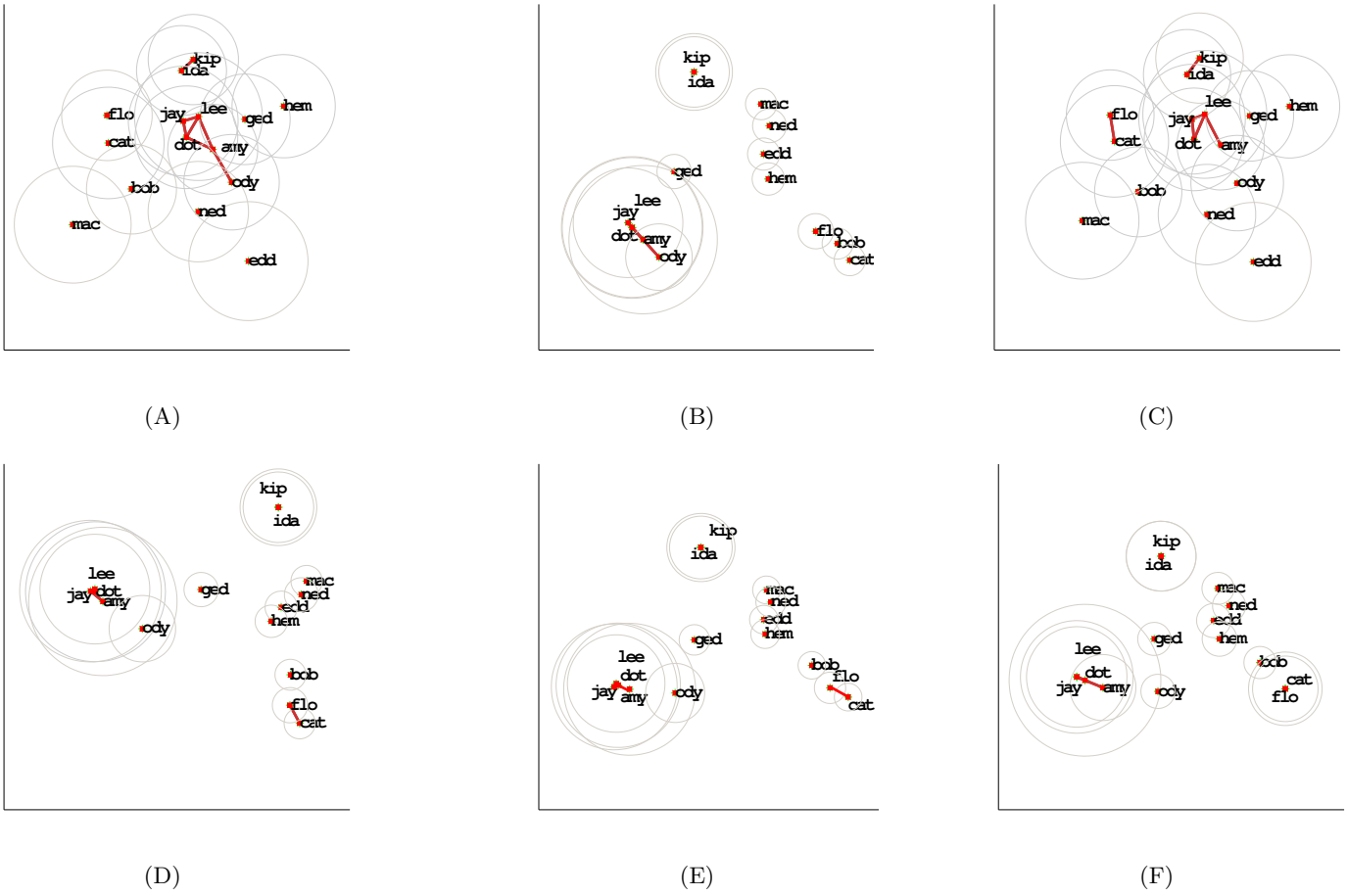


Figure 3: The model through different steps of our algorithm **A**. True model for 1st timestep **B**. Learned model from graph in (A) **C**. True model for 2nd timestep. **D**. Time variant MDS on the graph in (C) accounting for pairwise distances in (B) **E**.Procrustean transform on (D): Note the amount of rotation to bring (D) closest to (B) **D** Conjugate Gradient on (E)

It can be considered as the equivalent of the 1-nearest-neighbor method in classification: it does not generalize, but merely duplicates the training set.

E. Time-varying MDS: The model that results from running stage one only.

F. MDS with no time: The model that results from ignoring time information and running independent MDS on each timestep.

Figure 4 shows the ROC curves for the third timestep on a test set of size 160. Table 1 shows the AUC scores of our approach and the five alternatives for 3 different sizes of the dataset over the first, third, and last time steps.

As we can see, from the figures in all the cases, the true model has the highest AUC score, followed by the model learned by DSNL. The ROC curve of the simple counting model goes up very steeply, since it rightly guesses some of the links in the test graph from the training graph. However it also predicts the noise as links, and ends up being beaten by the model we learn. The results show that it is not sufficient to only perform Stage One. When the number of links is small, MDS without time does poorly compared to our temporal version. However as the number of links grows quadratically with the number of entities, regular MDS does almost as well as the temporal version: this is not a surprise because the generalization benefit from the previous timestep becomes unnecessary with sufficient data on the current timestep.

5.2 The best value of λ

In this section we show how we selected the value of λ. We vary λ from 0.0 to 90.0, in steps of 10.0 and plot the AUC scores of the models resulting from time-varying MDS in Figure 5.We repeat this experiment for model sizes 40, 80, 160, and 320. All the experiments show that AUC score of the resulting model from the MDS step grows better as λ increases from 0 to around 10 or 20. However after that the AUC scores start decreasing. This shows the two extreme simple predictive models which could be used. One is the time invariant classical MDS, when $\lambda = 0$. The other extreme is when λ is infinite, i.e. we only use the coordinates learned from the former timestep for predicting the behavior of the current timestep, ignoring information from current time step.We choose $\lambda = 10$ from the empirical results presented in Figure 5 .We choose a bigger value of $\lambda = 30$ for

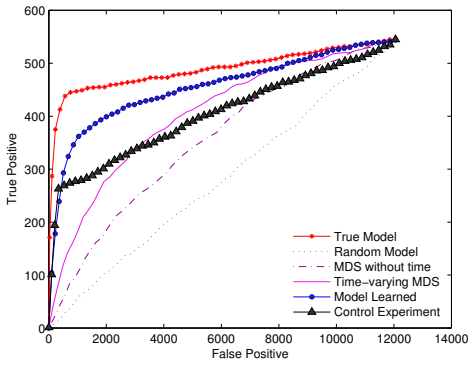


Figure 4: ROC curves of the six different models described earlier for test set of size 160 at timestep 3, in simulated data.

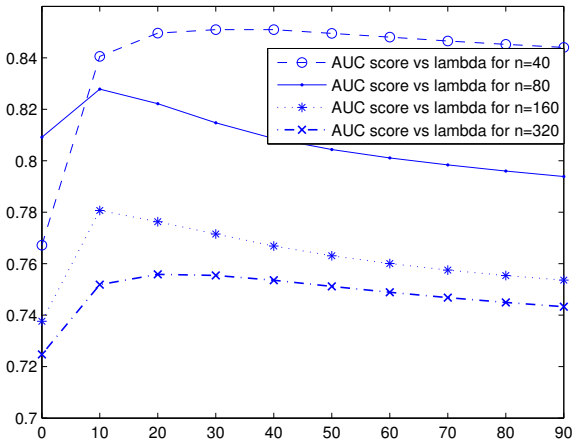


Figure 5: AUC scores vs λ.

the massive datasets.

5.3 How effective is MDS ?

In this section we try to see how effective our initialization step is. We examine the AUC scores of two different models. One starts with the MDS step , and the other starts with the random initialization. We also give the total time taken for both the variations to converge.

First we look at the starting log likelihood of the two probabilistic models with different initializations. We will refer to the method with MDS initialization as method 1, and the one with random initialization as method 2. From the tables 2 and 3 we see that method 1 starts with a much higher log-likelihood. Thus MDS is a much more educated guess than the random initialization. It ends with a better log-likelihood also. This difference between the log-likelihoods of the final models increases with the complexity of the model, i.e. the number of entities. However the AUC scores on the test set generated from the true generative model is not very different for the models learned by methods 1 and 2.

Table 1: AUC score on graphs of size n for six different models (A) True (B) Model learned using DSNL, (C) Random Model, (D) Simple Counting Model, (E) Multidimensional Scaling with time, and (F) MDS without time.

Time	A	B	C	D	E	F
	n=80					
1	0.95	0.88	0.51	0.81	0.80	0.82
3	0.94	0.87	0.49	0.76	0.88	0.81
6	0.93	0.86	0.48	0.75	0.85	0.79
	n=160					
1	0.90	0.85	0.51	0.73	0.73	0.75
3	0.90	0.85	0.50	0.75	0.85	0.73
6	0.90	0.85	0.50	0.73	0.83	0.72
	n=320					
1	0.86	0.82	0.50	0.71	0.72	0.72
3	0.85	0.83	0.49	0.70	0.80	0.71
6	0.86	0.83	0.50	0.72	0.79	0.72
	n=640					
1	0.84	0.81	0.50	0.70	0.68	0.71
3	0.83	0.81	0.50	0.70	0.78	0.70
6	0.83	0.80	0.50	0.71	0.78	0.70
	n=1280					
1	0.81	0.79	0.50	0.68	0.61	0.69
3	0.81	0.79	0.50	0.69	0.77	0.71
6	0.81	0.79	0.50	0.68	0.76	0.70

Another very striking observation is the running time(in seconds) of these two methods per time-step. Method 1 takes almost half as much time as method 2. So we conclude that conjugate gradient followed by MDS converges almost twice as fast as the method with random initialization , and it converges to a solution with higher log-likelihood.

5.4 Forward vs Forward-backward

The transition model presented in section 2.2 was a forward method of finding positions, given the past coordinates, and the current coordinates. Now we also compare the performance of a model with a forward backward method with the forward one. In this new method, we try to maximize the posterior probabilities of data of all time-steps.

$$X_t = \arg\max_X P(X|G_t, X_{t-1}, X_{t+1})$$
$$= \arg\max_X P(G_t|X)P(X|X_{t-1}, X_{t+1})$$

The $P(G_t|X_t)$ part remains the same, though the second part incurs an extra term, which penalizes any shift of the current coordinates w.r.t. the next time-step as well. The additional term is $-\sum_{i=1}^{n} |X_{i,t} - X_{i,t+1}|^2/2\sigma^2$. We first find the positions for each timestep using our time variant extension of classical MDS. After this batch initialization of all timesteps, we try to learn the coordinates at time t so that the resulting model explains the current graph, and is constrained to be very close to the coordinates of the models of the former and later timesteps.

We compare these two methods by presenting their respective training, and test log-likelihoods, along with the AUC scores in table 4. We see that the log-likelihood of the

Table 2: Log-likelihood, and time taken per iteration of Conjugate gradient with MDS initialization

	Conjugate gradient with MDS			
Time	log-likelihood after initialization	log-likelihood CG	AUC	Total time
	n=80			
1	-1078.76	-708.749	.87	19.76
2	-1047.57	-743.59	.87	11.23
3	-999.334	-720.714	.86	11.46
	n=160			
1	-3492.85	-2564.16	.83	49.2
2	-3121.25	-2587.59	.84	36.37
3	-3151.77	-2601.29	.84	39.36
	n=320			
1	-11808.7	-9477.46	.82	105.47
2	-11147.5	-9507.91	.83	66.07
3	-11126.9	-9419.03	.82	86.38
	n=640			
1	-62543.7	-44305.9	.81	449.71
2	-46268.7	-44227.2	.81	410.0
3	-46713.8	-44609.3	.81	209.32

Table 3: Log-likelihood, and time per iteration of Conjugate gradient with random initialization

	Conjugate gradient with random initialization			
Time step	log-likelihood after initialization	log-likelihood after CG	AUC	Total time per timestep
	n=80			
1	-2425.81	-710.383	.86	23.77
2	-2569.34	-785.118	.85	40.01
3	-2415.42	-758.374	.83	29.40
	n=160			
1	-6569.84	-2582.44	.84	62.69
2	-7159.99	-2698.56	.84	84.73
3	-7073.78	-2648.44	.83	80.33
	n=320			
1	-22124.2	-9622.56	.80	215.65
2	-21333.4	-9720.77	.81	145.15
3	-21171	-9718.47	.80	212.61
	n=640			
1	-79211.4	-44519.6	.80	671.04
2	-81051.8	-45050.8	.81	1084.6
3	-79280.1	-45763.5	.80	731.33

model from the forward-backward method gets better than the forward-only method as we increase the number of entities. However the AUC score on unseen data is almost comparable for these two methods.

5.5 Visualizing the NIPS coauthorship data over time

In this section we present a subset of the NIPS dataset, obtained by choosing a well-connected author, and including all authors and links within a few hops. We dropped authors who appeared only once and we merged the timesteps into three groups: 1987-1990 (Figure 6A), 1991-1994(Figure 6B), and 1995-1998(Figure 6C). In each picture we show the links for that timestep and highlight a few well connected people with their radii. These radii are learned from the model. Remember that the distance between two people is related to the radii. Two people with very small radii are considered far apart in the model even if they are physically close. To give some intuition of the movement of the rest of the points, we divided the area in the first timestep in 4 parts and used different colors and shapes for the points in each. This coloring is preserved throughout all the timesteps.

In this paper we limit ourselves to anecdotal examination of the latent positions. For example, with $Burges_C$ and $Vapnik_V$ we see that they had very small radii in the first four years, and were further apart from one another, since there was no co-publication. However in the second timestep they move closer, though there are no direct links. This is because of the fact that they both had co-published with neighbors of one another. On the third time step they make a connection, and are assigned almost identical coordinates, since they have a very overlapping set of neighbors.

We end the discussion with entities $Hinton_G$, $Ghahramani_Z$, and $Jordan_M$. In the first timestep they did not coauthor with one another, and were placed outside one-another's radii. In the second timestep $Ghahramani_Z$, and $Hinton_G$ coauthor with $Jordan_M$. However since $Hinton_G$ had a large radius and more links than the former, it is harder for him to meet all the constraints, and he doesn't move very close to $Jordan_M$. In the next timestep however $Ghahramani_Z$ has a link with both of the others, and they move substantially closer to one another.

5.5.1 *Varying the number of latent space dimensions*

It is possible and often useful to have more than two latent dimensions. For visualization we can only utilize up to three dimensions. But it might be necessary to have the entities in a higher dimensional latent space for improved link prediction. For the NIPS corpus we experimented on learning models with 1 to 3 dimensional latent space. Based on the results presented here, we decided to use 2 dimensions. We split the NIPS dataset randomly in different sized test, and training sets. We trained 3 different models with 1,2 and 3 latent space dimensions and computed the AUC scores for predicting the links in the test data.

We present the AUC scores of the datasets for the 4 : 1, and 3 : 2 ratio cuts of training and test set in table 5. As the ratio goes up the AUC scores go down in general. This is expected as we are increasing the test set size w.r.t the training set size, thus constraining the training phase to less evidence of connections. We see that in both cases the 2-dimensional model performs at least as good as the 3 dimensional model in terms of prediction. This is why we chose 2 dimensions for visualizing the NIPS dataset.

5.6 Scaling to Very Large Networks

We successfully applied our algorithms to networks of sizes up to 11,000. These graphs were extracted from the citeseer dataset. We will only mention the performance on two datasets. The first one contained 9364 entities from citeseer, publication years spanning over 6 years : 1998 to 2003. We split up each year's links in a 9:1 train:test set partition. The computation took around three and half hours. The AUC scores for the six timesteps ranged from 0.78 to 0.86. The second dataset with 11,218 entities was more densely con-

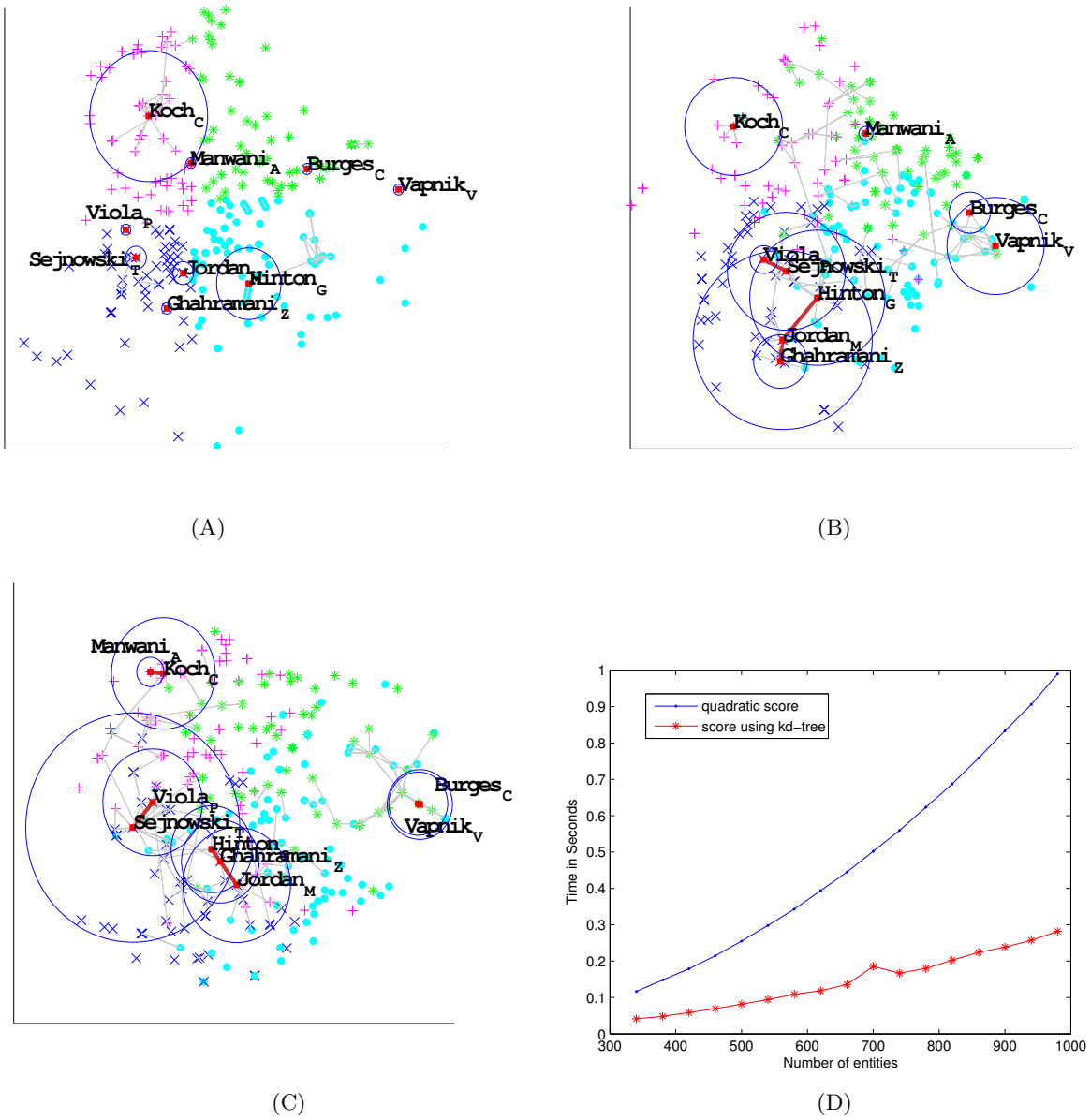


Figure 6: NIPS coauthorship data at **A**. timestep 1: green stars in upper-left corner, magenta pluses in top right, cyan spots in lower right, and blue crosses in the bottom-left, and **B**. Time step 2. **C**. NIPS coauthorship data at timestep 3. **D**. Time taken for score calculation vs number of entities.

nected. It had four years of information (1998-2001). This took around four and half hours, and the AUC scores under similar setting as before ranged from 0.80 to 0.92.

5.7 Performance Issues

Figure 6D shows the performance against the number of entities. When KD-trees are used and the graphs are sparse scaling is clearly sub-quadratic and nearly linear in the number of entities, meeting our expectation of $O(n \log n)$ performance. Similar experiments for gradient computation also show that its complexity is $O(n \log n)$.

6. CONCLUSIONS AND FUTURE WORK

This paper has described a method for modeling relationships that change over time. We believe it is useful both for understanding relationships in a large mass of historical data and also as a tool for predicting future interactions, and we plan to explore both directions further. This paper presented both a forward pass and a forward-backward algorithm on the Markov chain over timesteps. In the second technique we optimize the global likelihood instead of treating the model as a tracking model. We also plan to extend this in order to find the posterior distributions of the coordinates following the approach used in the static case by [4].

Table 4: Training and Test log-likelihood, and AUC scores for forward-backward vs forward method

	Forward-Backward			Forward		
Time step	training LogLike	test LogLike	AUC	training LogLike	test LogLike	AUC
	n=80					
1	-622.30	-688.91	.87	-694.978	-790.62	.87
2	-633.46	-705.31	.87	-691.775	-763.30	.88
	n=160					
1	-2257.9	-2497.05	.85	-2547.31	-3047.48	.84
2	-2322.26	-2459.13	.85	-2710.72	-2881.05	.84
	n=320					
1	-9384.79	-9850.77	.84	-11592.2	-12623.2	.82
2	-9473.83	-9985.53	.83	-11849.7	-12795.7	.83
	n=640					
1	-45189.6	-46462.7	.81	-64153.1	-68754	.79
2	-44126.3	-46756.5	.81	-62573.3	-67271.8	.79

Table 5: AUC scores on the NIPS dataset over 3 timesteps for different number of dimensions

AUC scores over 3 timesteps						
Number of dimensions	Training to test set ratio 4:1			Training to test set ratio 3:2		
1	0.85	0.88	0.83	0.84	0.79	0.80
2	0.94	0.94	0.88	0.88	0.85	0.85
3	0.90	0.91	0.83	0.89	0.84	0.85

7. ACKNOWLEDGMENTS

We are very grateful to Anna Goldenberg for her valuable insights. We also thank Paul Komarek for some helpful discussions.

8. REFERENCES

[1] J. Schroeder, J. J. Xu, and H. Chen. Crimelink explorer: Using domain knowledge to facilitate automated crime association analysis. In *ISI*, pages 168–180, 2003.

[2] J. J. Carrasco, D. C. Fain, K. J. Lang, and L. Zhukov. Clustering of bipartite advertiser-keyword graph. In *ICDM*, 2003.

[3] J. Palau, M. Montaner, and B. López. Collaboration analysis in recommender systems using social networks. In *Eighth Intl. Workshop on Cooperative Info. Agents (CIA'04)*, 2004.

[4] A. E. Raftery, M. S. Handcock, and P. D. Hoff. Latent space approaches to social network analysis. *J. Amer. Stat. Assoc.*, 15:460, 2002.

[5] R. L. Breiger, S. A. Boorman, and P. Arabie. An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling. *J. of Math. Psych.*, 12:328–383, 1975.

[6] I. Borg and P. Groenen. *Modern Multidimensional Scaling*. Springer-Verlag, 1997.

[7] R. Sibson. Studies in the robustness of multidimensional scaling : Perturbational analysis of classical scaling. *J. Royal Stat. Soc. B, Methodological*, 41:217–229, 1979.

[8] David S. Watkins. *Fundamentals of Matrix Computations*. John Wiley & Sons, 1991.

[9] F. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[10] A. G. Gray and A. W. Moore. N-body problems in statistical learning. In *NIPS*, 2001.

[11] R. Salakhutdinov, S. T. Roweis, and Z. Ghahramani. Optimization with em and expectation-conjugate-gradient. In *ICML*, volume 20, pages 672–679, 2003.