Parameter Learning for Latent Network Diffusion

Xiaojian Wu[†] **Akshat Kumar**[‡] **Daniel Sheldon**[†] **Shlomo Zilberstein**[†]

[†] School of Computer Science, University of Massachusetts, Amherst, MA 01003, USA
[‡] IBM Research, New Delhi 110070, India

Abstract

Diffusion processes in networks are increasingly used to model dynamic phenomena such as the spread of information, wildlife, or social influence. Our work addresses the problem of learning the underlying parameters that govern such a diffusion process by observing the time at which nodes become active. A key advantage of our approach is that, unlike previous work, it can tolerate missing observations for some nodes in the diffusion process. Having incomplete observations is characteristic of offline networks used to model the spread of wildlife. We develop an EM algorithm to address parameter learning in such settings. Since both the E and M steps are computationally challenging, we employ a number of optimization methods such as nonlinear and difference-of-convex programming to address these challenges. Evaluation of the approach on the Red-cockaded Woodpecker conservation problem shows that it is highly robust and accurately learns parameters in various settings, even with more than 80% missing data.

1 Introduction

Dynamic phenomena such as the spread of information, ideas, and opinions [Domingos and Richardson, 2001; Kempe *et al.*, 2003; Leskovec *et al.*, 2007], and infectious disease propagation among humans [Anderson and May, 2002] can be described as a *diffusion* process or *cascade* over an underlying network. Similar diffusion processes have also been used for *metapopulation* modeling in the ecology literature to describe how wildlife spreads over a fragmented landscape [Hanski, 1999]. Such models are crucial for several decision problems in spatial conservation planning such as how to allocate resources to maximize the population spread of an endangered species over a period of time [Sheldon *et al.*, 2010; Ahmadizadeh *et al.*, 2010; Golovin *et al.*, 2011; Kumar *et al.*, 2012].

A fundamental problem in using diffusion-based models for spatial conservation planning is the estimation of the parameters that govern the spread of a species using limited observed data. This problem shares some similarities with recent work in social network structure learning, in which the goal is to learn which edges are present in a social network as well as their transmission probabilities [Myers and Leskovec, 2010; Gomez-Rodriguez *et al.*, 2012; Netrapalli and Sanghavi, 2012; Wang *et al.*, 2012]. Myers and Leskovec [2010] formulate the problem of structure learning as a separable convex program. Gomez-Rodriguez *et al.* [2012] address the problem using submodular optimization. Netrapalli and Sanghavi [2012] address the complementary question of how many observed cascades are necessary to correctly learn the structure of a network. Wang *et al.* [2012] enrich the structure learning problem using additional features from Twitter data.

An implicit common assumption in structure learning approaches is that the status of each node is observed. That is, one observes exactly which nodes in the network are infected and the time of their infection. However, *how* they are infected—which edge is responsible for infection transmission—is not observed. Furthermore, the transmission probabilities that govern the spread of infection through the network are treated as being separate parameters to be learned for each edge. Our problem setting differs substantially in these aspects, making direct application of previous approaches infeasible.

In offline networks, such as the ones used for metapopulation modeling, it is not realistic to observe the status of each node in the cascade due to the sheer amount of instrumentation/manual effort involved. One may only hope to observe the status of a fraction of the nodes in the network as to whether they are infected or not. Furthermore, such networks are often geospatial in nature and therefore the connectivity structure of the network is known, in contrast to the social network setting in which the presence or absence of particular edges is not known in advance. Metapopulation models are typically parameterized so that the transmission probabilities between pairs of nodes are determined by a few key physical properties that describe the biological phenomenon, such as the distance between nodes, the amount of habitat present, and the quality of habitat [Hanski, 1994]. This has the effect of coupling the transmission probabilities of the network edges to depend on common parameters, in contrast with social network models which attempt to learn a separate parameter for each edge. Our work addresses all these unique characteristics of metapopulation modeling and more broadly, computational sustainability.

We formulate the parameter learning problem as a maximum likelihood problem with hidden variables. We use the EM algorithm to maximize the likelihood [Dempster *et al.*, 1977]. However, both the E and M steps are challenging. To overcome the intractability of the exact E-step, we use the Monte Carlo version of EM (MCEM) [Wei and Tanner, 1990]. The M-step entails solving a non-convex optimization problem. We present a number of techniques from the optimization literature such as nonlinear programming and difference-of-convex functions (DC) programming to address this. We apply our approach to the Red-cockaded Woodpecker conservation problem over the coast of North Carolina [Ahmadizadeh *et al.*, 2010]. The results show that our approach is highly robust to different EM initializations and can reliably learn parameters with varying numbers of cascades. Most notably, our approach accurately learns parameters even with more than 80% missing data.

In the ecology literature, Moilanen [1999] fit a metapopulation model that was similar to ours by maximum likelihood. He did not use the EM algorithm, but instead used Monte Carlo sampling of missing values to approximately compute the likelihood, which was then optimized numerically. Compared with our work, Moilanen uses a weak sampler, and one that only works when entire years are either missing or observed, so it is much less flexible with respect to the pattern of missing data. Ter Braak and Etienne [2003] used a much more efficient Gibbs sampling procedure to resample missing values, which is similar to our E step. One main difference between their work and ours is that they took a Bayesian approach, while we have developed an EM algorithm to maximize likelihood.

In addition, a key contribution that distinguishes our work from that of both Moilanen and ter Braak and Etienne is the fact that we have considerably increased the flexibility of the models used in those papers by moving from a rigid parameterization that depends on just a few environmental variables (distance between habitat nodes and habitat patch size) to a logistic regression parameterization that can model the effect of an arbitrary number of environmental variables on the transmission probabilities. To the best of our knowledge, ours is the first method in either the social network or ecology literature that addresses both the generality and missing data issues. Thus our work is a significant improvement over previous approaches.

2 The Metapopulation Modeling Problem

In metapopulation modeling, the goal is to describe the occupancy pattern of habitat patches for a certain species in a fragmented landscape over a period of time. Each habitat patch can be thought of as a node in a geospatial network. Edges connect two different habitat patches, intuitively representing the fact that species such as birds can migrate from one habitat patch to another. We say that a node is *active* or *infected* if the species in question is present within the node. Initially, a certain number of nodes are infected. The infection spreads from these initially active nodes in discrete time steps using the well-known *independent cascade* model [Kempe *et al.*, 2003; Sheldon *et al.*, 2010].

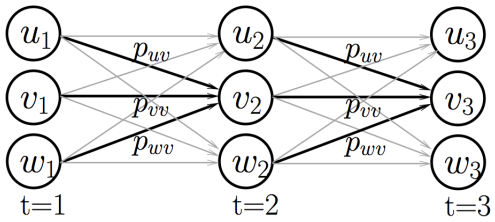

Figure 1: A layered graph for three nodes u, v and w

In the independent cascade model, once a node u becomes infected, it will attempt (only once) to infect each of its neighbors v independently using the transmission probability p_{uv}:

$$p_{uv} = P(\text{node } v \text{ gets infected by } u \mid \text{node } u \text{ is infected}) \quad (1)$$

For ease of exposition, we present the problem using the Susceptible–Infected (SI) model in which an infected node remains infected forever; each newly infected node is given only a single chance to infect its neighbors. As we will see in Section 3, it is easy to fold the species extinction and repopulation in a habitat patch over time within the SI model by using a layered time indexed graph [Kempe *et al.*, 2003; Sheldon *et al.*, 2010]. Unlike social networks, we assume that the transmission probability is parameterized using a logistic regression model:

$$p_{uv} = \frac{1}{1 + \exp\left(\langle \theta, \phi_{uv} \rangle\right)} \quad (2)$$

where ϕ_{uv} denote the features associated with the edge (u, v) and θ denotes the parameter vector that governs the underlying species spread. The features, for example, can include the geographical distance between the nodes u and v, and a measure of habitat suitability of node v. This is a very flexible approach that is capable of capturing models very similar to those widely used in metapopulation modeling [Hanski, 1994], as well as extending those models to depend on a much richer set of features. In our work, the goal is to estimate the unknown parameters θ using limited observations about patch occupancy, as described below.

Observation model: A cascade c over such a network starts with a set of initially active nodes at time t=1. As the cascade progresses in discrete time steps, we observe the infection time of nodes as they subsequently become infected; for nodes u that are never infected, we set the infection time $\tau_u^c = \infty$. Let $\mathcal{V}$ denote the set of all nodes. Instead of observing the infection time of all the nodes $u \in \mathcal{V}$, we assume that this observation is available only for a subset of nodes $\mathcal{V}^c \subseteq \mathcal{V}$. The nodes in the set $\mathcal{V} \backslash \mathcal{V}^c$ can be infected or uninfected, representing hidden data. Furthermore, for the observed nodes, we do not observe *how* they got infected. That is, we do not know which node activated them. This particular setting is relevant for offline networks used in metapopulation modeling, where collecting information about the infection status requires significant instrumentation. Thus, our goal is to learn diffusion parameters based on such incomplete observations.

3 Our Approach

In our approach, to fold the metapopulation model within the SI model, we use a time indexed graph. If $v \in \mathcal{V}$ denotes a node in the original geospatial network, then a node in the time indexed graph is denoted $v_t \in \mathcal{V}_T := \mathcal{V} \times \{1, \ldots, T\}$, where T denotes the total length of a cascade. There is an edge (u_t, v_{t+1}), if there exists an edge (u, v) in the original graph. Figure 1 shows an example for a three node complete graph and 3 time steps. Note that the probability p_{uv} for $u \neq v$ represents colonization of habitat patch v by individuals from patch u, while p_{vv} represents "self-colonization", i.e., it is the probability that a population in a given habitat patch *does not* go extinct in a given time step. Let $X^c_{u,t}$ be a binary random variable such that $X^c_{u,t} = 1$ denotes that node u_t is infected in cascade c, and 0 indicates otherwise.

Our strategy is to use the EM algorithm to fill in the values $X^c_{u,t}$ for unobserved nodes. We first provide a brief overview of the EM algorithm and a description of the E and M-steps. Let $X = (X^c_{u,t})$ be the complete vector of random variables, and let Y and Z be the subvectors of variables corresponding to the observed and unobserved nodes, respectively. The EM algorithm iteratively finds parameters $\theta^\star$ that maximize the following expected log-likelihood:

$$Q(\theta^\star, \theta) = \sum_Z P(Z|Y;\theta) \log P(Z, Y; \theta^\star) \tag{3}$$

where $\theta^\star$ denotes the parameters to optimize and θ denotes the previous iteration's parameters. In the E-step, we need to calculate $Q(\theta^\star, \theta)$, which is an expected value with respect to the distribution $P(Z|Y;\theta)$. However, often the posterior probability $P(Z|Y;\theta)$ is considered hard or intractable to calculate as in our case. In the Monte Carlo variant of EM (MCEM), we generate a certain number of samples of Z, say n, following the posterior probability $P(Z|Y;\theta)$ using techniques such as Gibbs sampling. The modified M-step maximizes the following approximate likelihood w.r.t. $\theta^\star$:

$$Q(\theta^\star, \theta) = \mathbb{E}_Z\big[\log P(Z, Y; \theta^\star)\big] \approx \frac{1}{n} \sum_{i=1:n} \log P(Z^i, Y; \theta^\star) \tag{4}$$

where Z^i denotes the ith sample generated from $P(Z|Y;\theta)$.

Complete Data Likelihood: For any fixed setting of the complete random vector X, let $\mathcal{I}^c(t)$ denote the set of nodes u_t that were infected ($X^c_{u,t} = 1$) at time step t, and let $\mathcal{U}^c(t)$ denote the set of nodes that were not infected at time step t. Also, let $X^c_{\cdot,t}$ denote the complete set of variables from time slice t, and let $X^c_{-u,t}$ denote all variables other than $X^c_{u,t}$ at time t. Because of the Markov structure of the cascade process, the conditional distribution of $X^c_{u,t}$ given all predecessors depends only on $X^c_{\cdot,t-1}$:

$$P\big(X^c_{u,t} = 1 \mid X^c_{\cdot,t-1}; \theta\big) = 1 - \prod_{v_{t-1} \in \mathcal{I}^c(t-1)} \frac{1}{1 + e^{-\langle \theta, \phi(v,u) \rangle}} \tag{5}$$

The above expression is analogous to saying that the probability of a node u_t being infected is one minus the probability that this node is never infected. Furthermore, each variable at time t is conditionally independent all other variables in the same time slice given $X^c_{\cdot,t-1}$. Using this conditional independence structure, we can write the joint distribution of a single complete sample, $X^c = (X^c_{u,t})$ for a particular cascade c as:

$$\begin{aligned} P(X^c; \theta) &= \prod_{t=2}^{T} \Bigg[\prod_{u_t \in \mathcal{I}^c(t)} P\big(X^c_{u,t} = 1 \mid X^c_{\cdot,t-1}; \theta\big) \\ &\qquad \prod_{u_t \in \mathcal{U}^c(t)} P\big(X^c_{u,t} = 0 \mid X^c_{\cdot,t-1}; \theta\big) \Bigg] \\ &= \prod_{t=2}^{T} \Bigg[\prod_{u_t \in \mathcal{I}^c(t)} \Bigg(1 - \prod_{v_{t-1} \in \mathcal{I}^c(t-1)} \frac{1}{1 + e^{-\langle \theta, \phi(v,u) \rangle}} \Bigg) \\ &\qquad \prod_{u_t \in \mathcal{U}^c(t)} \prod_{v_{t-1} \in \mathcal{I}^c(t-1)} \frac{1}{1 + e^{-\langle \theta, \phi(v,u) \rangle}} \Bigg] \end{aligned} \tag{6}$$

In the above expression, we are assuming the complete observability of seed nodes that are infected at time step 1.

3.1 E-Step

Our strategy is to use Gibbs sampling to generate complete samples $X^1, \ldots, X^n$ of the random vector X by filling in the values of the hidden variables Z with samples $Z^i \sim P(Z \mid Y; \theta)$. The Gibbs sampling works by initializing the hidden variables to arbitrary values and then stochastically changing their values one by one in some order. The new value for a variable $X^c_{u,t}$ is sampled from the conditional distribution of that variable given all the other variables. Once the values of all the variables are recomputed, one iteration of the Gibbs algorithm is finished. Usually, we collect samples only after a certain number of iterations, which guarantees the distribution that samples are drawn from is independent from the initial values of variables. We now derive the conditional distribution used by Gibbs sampling for our case and show that it is computationally tractable even for larger networks.

The conditional probability used in Gibbs sampling must consider not only predecessors but also variables from subsequent time steps. It can be computed by considering the Markov blanket of the variable $X^c_{u,t}$, as shown below:

$$\begin{aligned} P\big(X^c_{u,t} \,\big|\, X^c_{\cdot,t-1}, X^c_{-u,t}, X^c_{\cdot,t+1}; \theta\big) \propto \\ P\big(X^c_{u,t} \,\big|\, X^c_{\cdot,t-1}; \theta\big) P\big(X^c_{\cdot,t+1} \,\big|\, X^c_{\cdot,t}; \theta\big) \end{aligned}$$

The first factor in the RHS of the above equation is given in Eq. (5). The second factor is given in the expression below, in which the set $\mathcal{I}^c(t)$ is updated to reflect the particular setting of $X^c_{u,t}$ to either 0 or 1:

$$\begin{aligned} P\big(X^c_{\cdot,t+1} \,\big|\, X^c_{\cdot,t}; \theta\big) = \prod_{w_{t+1} \in \mathcal{U}^c(t+1)} \prod_{v_t \in \mathcal{I}^c(t)} \frac{1}{1 + e^{-\langle \theta, \phi(v,w) \rangle}} \\ \prod_{w_{t+1} \in \mathcal{I}^c(t+1)} \Bigg(1 - \prod_{v_t \in \mathcal{I}^c(t)} \frac{1}{1 + e^{-\langle \theta, \phi(v,w) \rangle}} \Bigg) \end{aligned} \tag{7}$$

Notice that the complexity of computing the above expression increases linearly with respect to the number of edges in the time indexed graph.

3.2 M-Step

Taking the log of the complete joint distribution in Eq. (6), we get:

$$P(X^c;\theta)=\sum_{t=2}^{T}\sum_{u_t\in\mathcal{I}^c(t)}\log\left(1-\prod_{v_{t-1}\in\mathcal{I}^c(t-1)}\frac{1}{1+e^{-\langle\theta,\phi(v,u)\rangle}}\right)$$
$$-\sum_{t=2}^{T}\sum_{u_t\in\mathcal{U}^c(t)}\sum_{v_{t-1}\in\mathcal{I}^c(t-1)}\log(1+e^{-\langle\theta,\phi(v,u)\rangle}) \quad (8)$$

In order to maximize the log-likelihood in Eq. (4), we sum over different samples for each cascade c. If there are m partially observed cascades with n complete samples obtained using Gibbs sampling per cascade, then we require mn total summations of the type in Eq. (8) for the log-likelihood of Eq. (4). We omit the two outermost summations for brevity.

Nonlinear programming based optimization

Once we represent the log-likelihood as in Eq. (8), we can maximize it with respect to the parameters θ by using an off-the-shelf nonlinear programming (NLP) solver. Notice that this optimization problem is not convex in θ. Therefore, an NLP solver may increase the log-likelihood with respect to the previous iteration of EM, but may not maximize it. This is a common setting for EM, also known as the *generalized EM* (GEM) algorithm. Most of the properties of the EM algorithm, such as convergence to a stationary point, are preserved by the GEM algorithm [Neal and Hinton, 1998].

A disadvantage of the above technique is that while NLP solvers can work well for moderately sized optimization problems, their performance degrades significantly when the parameter space is large or there are additional constraints on the parameters θ. In contrast, convex optimization solvers are much more scalable and well understood. Even though the optimization problem of Eq. (8) is not convex in θ, we show that it can be reformulated as a difference-of-convex functions (DC) program. The main advantage of DC programming is that we can solve a DC program iteratively using a sequence of *convex* programs. Therefore, this reformulation allows us to use highly efficient convex optimization solvers for this non-convex problem. Another attractive property of iteratively solving a DC program is that the objective improves monotonically with each iteration, thus fitting well within the GEM framework.

DC programming based optimization

We now briefly describe DC programming and the concave-convex procedure (CCCP) used to solve DC programs. Consider the optimization problem: $\min\{g(x) : x \in \Omega\}$, where $g(x) = h(x) - f(x)$ is an arbitrary function with h, f being real-valued, differentiable *convex* functions and Ω being a constraint set. The CCCP method provides an iterative procedure that generates a sequence of points x^l by solving the following *convex* program:

$$x^{l+1} = \arg\min\{h(x) - x^T\nabla f(x^l) : x \in \Omega\} \quad (9)$$

Each iteration of CCCP monotonically decreases the objective function $g(x)$ for any Ω [Sriperumbudur and Lanckriet, 2009]. CCCP was originally proposed for Ω that is described by linear equality constraints, but Sriperumbudur and Lanckriet [2009] showed the same idea extends to any constraint set including non-convex constraints that may themselves be represented as a DC function. Furthermore, it is guaranteed to converge to a stationary point where the Karush-Kuhn-Tucker (KKT) conditions are satisfied [Sriperumbudur and Lanckriet, 2009]. Note that the objective $g(x)$ may be non-convex, which makes CCCP a general approach for non-linear optimization. For ease of exposition, we show how the objective in Eq. (8) fits into a DC programming framework. The analogous result applies to the log-likelihood function of Eq. (4) which has the same structure as Eq. (8). For a node $u_t \in \mathcal{I}^c(t)$, let us use the following substitution:

$$\gamma(u_t) = 1 - \prod_{v_{t-1}\in\mathcal{I}^c(t-1)}\frac{1}{1+e^{-\langle\theta,\phi(v,u)\rangle}} \quad (10)$$

We can write the problem of maximizing (8) for a single complete cascade as following:

$$\min_{\theta,\{\gamma(u_t)\}} -\sum_{t=2}^{T}\sum_{u_t\in\mathcal{I}^c(t)}\log\gamma(u_t)$$
$$+\sum_{t=2}^{T}\sum_{u_t\in\mathcal{U}^c(t)}\sum_{v_{t-1}\in\mathcal{I}^c(t-1)}\log(1+e^{-\langle\theta,\phi(v,u)\rangle})$$
$$\text{Subject to: } \gamma(u_t) \le 1 - \prod_{v_{t-1}\in\mathcal{I}^c(t-1)}\frac{1}{1+e^{-\langle\theta,\phi(v,u)\rangle}}$$
$$\forall u_t \in \mathcal{I}^c(t), \forall t = 2:T \quad (11)$$
$$0 \le \gamma(u_t) \le 1 \quad \forall u_t \in \mathcal{I}^c(t), \forall t = 2:T \quad (12)$$

In the above optimization problem, the objective function is convex. Furthermore, the inequality constraint will become equality as the objective function is a decreasing function of γ and the product term, thus satisfying equality constraint (10). Notice that the constraint function (11) is not convex. Our strategy is to represent the constraint as a difference of two convex functions. We manipulate this constraint as following:

$$\prod_{v_{t-1}\in\mathcal{I}^c(t-1)}\frac{1}{1+e^{-\langle\theta,\phi(v,u)\rangle}} \le 1-\gamma(u_t) \quad (13)$$
$$-\log(1-\gamma(u_t)) - \sum_{v_{t-1}\in\mathcal{I}^c(t-1)}\log(1+e^{-\langle\theta,\phi(v,u)\rangle}) \le 0 \quad (14)$$

The last inequality resulted by taking the log of both sides of the previous inequality. The constraint (14) is a DC constraint as the first term is convex in $\gamma(u_t)$ and the second term is also convex due to the log-sum-exp function being convex. Thus, we now have a DC program with convex objective function and DC constraints. Each iteration of CCCP entails solving the following convex program:

$$\min_{\theta,\{\gamma(u_t)\}} -\sum_{t=2}^{T}\sum_{u_t\in\mathcal{I}^c(t)}\log\gamma(u_t)$$
$$+\sum_{t=2}^{T}\sum_{u_t\in\mathcal{U}^c(t)}\sum_{v_{t-1}\in\mathcal{I}^c(t-1)}\log(1+e^{-\langle\theta,\phi(v,u)\rangle}) \quad (15)$$

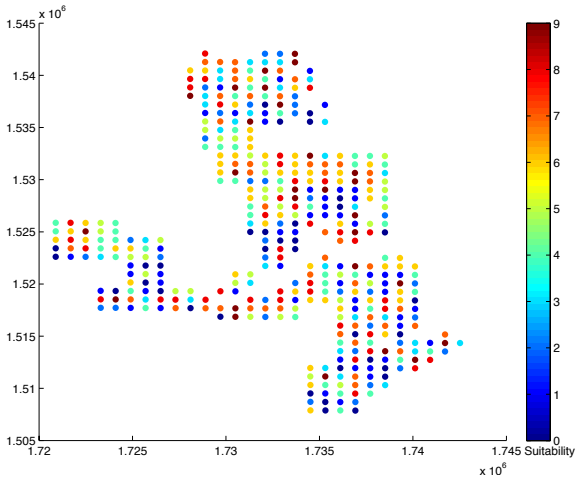

Figure 2: Territories plotted based on their GIS coordinates, with color showing habitat suitability score, according to color bar.

The constraint set for the above program is described as follows:

$$\text{Subject to: } -\log(1-\gamma(u_t)) - \Big\{ f_{u_t}(\theta^l) + (\theta-\theta^l)\cdot\nabla f_{u_t}(\theta^l) \Big\} \leq 0 \ \forall u_t \in \mathcal{I}^c(t), \forall t = 2:T \quad (16)$$

$$0 \leq \gamma(u_t) \leq 1 \ \forall u_t \in \mathcal{I}^c(t), \forall t = 2:T \quad (17)$$

where θ^l denotes the previous CCCP iteration's parameters, $\nabla f_{u_t}(\cdot)$ is the gradient of the function:

$$f_{u_t}(\theta) = \sum_{v_{t-1} \in \mathcal{I}^c(t-1)} \log(1 + e^{-\langle \theta, \phi(v,u) \rangle}) \quad (18)$$

The convex program (15) can be solved using highly efficient off-the-shelf solvers such as KNITRO [Byrd *et al.*, 2006]. The optimization program of (15) includes additional variables $\gamma(\cdot)$. For efficiency purpose, we can in fact eliminate these variables using constraint (16). The resulting optimization program has θ as the only variables to optimize and is much more compact. Details are omitted for brevity.

4 Experiments

We used a publicly available conservation planning benchmark which represents a geographical region on the coast of North Carolina [Ahmadizadeh *et al.*, 2010]. The conservation target species is the Red-cockaded Woodpecker. The graph consists of 411 territories or habitat patches. We assume that the species can move between any two nodes. We simulate the cascades for 10 time steps, so the resulting time indexed graph has 4110 nodes and 842550 edges.

For each territory in the network, the GIS coordinates and the habitat suitability scores are provided. Figure 2 shows all the territories and their habitat suitability scores in color. With this information, the features $\phi(v,u)$ are defined as $\langle 1, dist(v,u), suit(u) \rangle$, where $dist(v,u)$ is the distance between territories v and u, and $suit(u)$ is the suitability score of territory u. Therefore, there are three parameters in $\theta = \langle \theta_1, \theta_2, \theta_3 \rangle$, one per feature. To generate the observations, we choose a value for θ and simulate the diffusion process for 10 time steps. The signs of the individual parameters in θ are chosen to realistically model the problem at hand, so the colonization probability between two patches increases as the suitability of the target patch increases, and decreases as the distance between patches increases. Furthermore, we make sure that the resulting probabilities are distributed between 0 and 1 so that there is sufficient stochasticity in the edge transmission probabilities to make the problem challenging. The M-step optimization was done using nonlinear programming, unless stated otherwise.

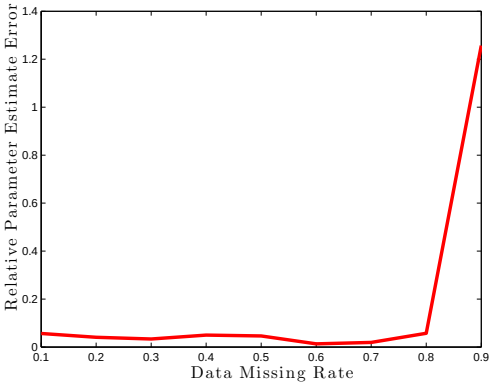

Figure 3: Relative estimated parameter errors for 411 territories with different amounts of missing data, using 10 cascades and 10 samples per cascade.

Each simulation phase generates one complete cascade using the true parameters θ. Depending on how much missing data is desired, we randomly choose the appropriate number of observations from each complete cascade and make them available for learning. We evaluate our algorithm by trying to learn the correct parameters under different settings, using the observed data and starting from a random initialization of parameters θ. With a limit of 20 iterations, EM converged in almost every case. The relative parameter estimate error shown in different plots is computed as follows:

$$\frac{1}{3}\sum_{i=1}^{3}\left|\frac{\theta_i^{estimate} - \theta_i^{true}}{\theta_i^{true}}\right|.$$

We describe below different sets of experiments, designed to assess the characteristics of our algorithm.

How much missing data can the algorithm tolerate?
We first test our algorithm on the complete 411-node graph. For this set of experiments, we vary the percentage of missing data and test its effects on the accuracy of the MCEM algorithm. Figure 3 shows the relative estimated parameter errors for the 411 node network with respect to varying missing data percentage. It is encouraging to see that with up to 80% missing data, implying 329 unobservable nodes out of 411 per time step, our method is still able to learn the true parameters accurately. It is also interesting to see that the estimation becomes arbitrary with 90% missing data and MCEM

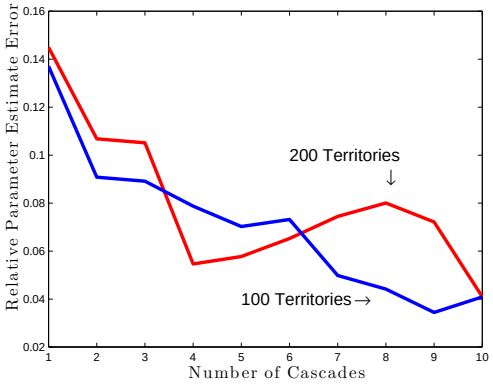

Figure 4: Relative estimated parameter errors for different number of cascades, using 10 samples per cascade with 70% missing data.

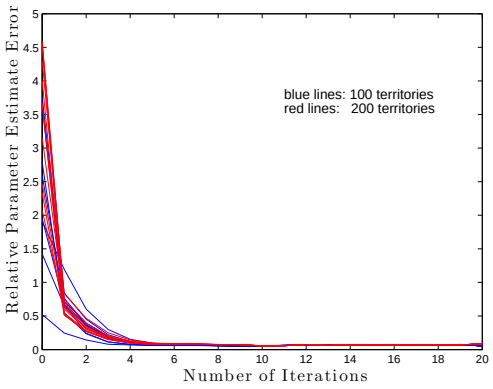

Figure 5: Relative estimated parameter errors for different initial parameters, using 10 cascades, 10 samples per cascade with 50% missing data. Each line represents a different initialization.

fails to converge to a good setting within the fixed number of iterations.

How much data is needed to yield a good estimate?

This set of experiments is designed to test the number of (partially) observed cascades needed to accurately estimate the parameters. An underlying observation is that a single cascade may not provide enough information, therefore, multiple cascades are needed. Netrapalli and Sanghavi [2012] provide information-theoretic bounds on the number of cascades needed to learn a social network accurately. While their results are not directly applicable to our setting, we still empirically test the number of cascades required for reliable learning. To make the problem challenging, we randomly extract 100- and 200-node networks from the original network and simulate the diffusion on these smaller sized networks. The intuition is that the effect of the number of cascades will be more pronounced on a smaller network, where fewer observations are available for learning than the complete 411 node network. We used a fixed setting of 70% missing data in these experiments. Figure 4 plots the relative estimation error with

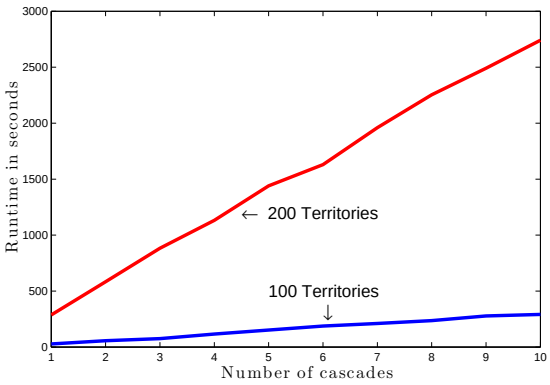

Figure 6: Runtime for different number of cascades, using 10 samples per cascade with 50% missing data.

respect to the number of cascades. Each data point is an average of 10 trials in order to decrease the noise. The results show that learning is generally more accurate with more cascades. As the number of cascades increases, the accuracy also increases, albeit slowly after the first few cascades and with some fluctuation.

How robust is MCEM to initializations?

A critical factor for the MCEM algorithm is the initial parameter setting. In this set of experiments, we test the robustness of MCEM in this respect. Figure 5 plots the relative estimation errors for each MCEM iteration for different random initial parameters (a total of 18 settings) for 100 territories and 200 territories. In both the cases, the relative error uniformly converges to 0.05 after around 6 iterations. The results show that our technique is able to robustly recover the actual parameter starting with different arbitrary initializations.

How does MCEM scale with respect to runtime?

In this set of experiments, we test the execution time of the MCEM algorithm. Figure 6 plots the runtime with respect to different number of cascades for 100- and 200-node networks. For both of these network sizes, the total runtime increases linearly with the number of cascades. The larger 200-node network requires more runtime as the number of edges and the sampling complexity increase quadratically with the number of nodes. Moreover, the size of the optimization problem also increases, leading to higher runtime in that case.

Table 1 shows the runtime of MCEM for the largest 441-node network with different amounts of missing data. We show the time required for the E and M steps separately. Interestingly, while the time of the E step increases linearly with the amount of missing data, the time for the M step remains nearly constant. This is expected, because the number of Gibbs iterations is proportional to the number of unobserved nodes, while the size of the optimization problem in the M step is proportional the total number of variables. Nevertheless, it is very useful to note that when the percentage of missing data is more than 40%, the E-step dominates the runtime. Therefore, for larger networks, developing efficient techniques for performing the E-step is going to be the key for scalability.

Missing Data Rate	Runtime (sec)		
	E-step	M-step	Total
0.1	105	388	9877
0.2	206	378	11700
0.3	314	384	13983
0.4	417	378	15915
0.5	529	363	17854
0.6	631	360	19831
0.7	734	382	22333
0.8	823	352	23534
0.9	809	220	20610

Table 1: Runtime for 411 territories with different amounts of missing data, using 10 cascades and 10 samples per cascade. The runtime in the E-step and M-step columns is average runtime per iteration.

Active Nodes	Estimation Error
$0-15\%$	15.4%
$16-30\%$	3.9%
$31-45\%$	4.8%
$46-60\%$	3.6%
$61-75\%$	4.3%
$76-90\%$	6.2%

Table 2: Relative parameter estimation errors for different types of diffusion strength, measured by percentage of active nodes after 10 time steps.

How does the strength of diffusion affect MCEM?

We now discuss the relative error achieved by MCEM for different types of diffusion models. We measure the strength of a particular cascade by measuring the number of active nodes after 10 time steps starting with 10% seeds. We generated a number of diffusion parameters covering a range of strengths from weak to aggressive diffusion. We used the largest 411 node network in these experiments. For each cascade, 80% of data was randomly chosen to be missing.

Table 2 shows a broad categorization of different cascades from weak diffusion ($0-15\%$ active nodes) to aggressive diffusion ($76-90\%$ active nodes). The table shows the average parameter estimation error calculated based on 5 random cases per entry. We can see clearly that MCEM performs reasonably well for all the settings. For the weak diffusion setting ($0-15\%$), the percentage error is the highest ($\approx 15\%$). This is expected as for weak diffusion, there are too few activations to provide enough data to learn the parameters reliably. For the rest of the settings, the error is well below 10%.

How does DC programming scale?

Figure 7 shows the parameter estimation error for the 411-node complete network with respect to the number of iterations of CCCP. For each M-step, we use 40 iterations of CCCP. Each blue dot in the figure denotes a point when a new E-step starts. As expected, CCCP monotonically improves the parameters and provides an error of only about 6% after 5 EM iterations. However, we did not see improvement in the total runtime while using CCCP over the nonlinear programming (NLP) based M-step. We believe the reason is that in the NLP, there are only 3 variables and importantly, no constraints. The objective function contains many terms involving these 3 variables. Therefore, the NLP solvers are able to efficiently handle this moderately sized program. We still believe that for problems which have richer parameter space and additional constraints on the parameters, using convex optimization solvers within CCCP is going to be advantageous relative to NLP.

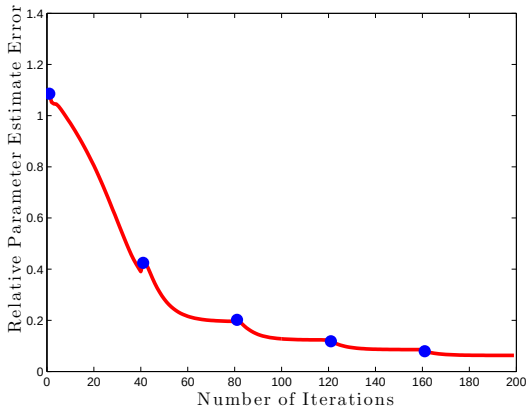

Figure 7: Relative estimated parameter errors using CCCP, with 10 cascades, 10 samples per cascade, 50% missing data. The blue points indicate when an E-step is finished.

5 Conclusion

We have addressed the problem of learning the underlying parameters that govern the diffusion process over a network. We examined the diffusion process in the context of offline networks used for modeling the spread of wildlife or the metapopulation modeling. The key ingredient to learning the underlying parameters is the availability of information about when different nodes become active. Unlike previous works, our technique can tolerate missing observations for some nodes in the network. We highlighted how this setting is particularly relevant for metapopulation modeling. We developed the EM algorithm to address learning with such missing data. To address the intractability of the E and M steps, we used a number of techniques such as Gibbs sampling to compute the expectations, nonlinear programming and DC programming for optimization in the M-step.

Our results on a real-world Red-cockaded Woodpecker conservation problem show that our technique is highly robust and accurately learns parameters in various settings, even with more than 80% missing data. Moreover, the approach provides consistently good results for a wide range of diffusion models with different strengths of influence.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback and suggestions. This work was supported in part by National Science Foundation Grant No. IIS-1116917 and Grant No. IIS-1117954.

References

[Ahmadizadeh *et al.*, 2010] Kiyan Ahmadizadeh, Bistra Dilkina, Carla P. Gomes, and Ashish Sabharwal. An empirical study of optimization for maximizing diffusion in networks. In *Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming*, pages 514–521, 2010.

[Anderson and May, 2002] Roy M. Anderson and Robert M. May, editors. *Infectious Diseases of Humans: Dynamics and Control*. Oxford University Press, 2002.

[Byrd *et al.*, 2006] Richard H. Byrd, Jorge Nocedal, and Richard A. Waltz. KNITRO: An integrated package for nonlinear optimization. In *Large Scale Nonlinear Optimization*, pages 35–59, 2006.

[Dempster *et al.*, 1977] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

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

[Golovin *et al.*, 2011] Daniel Golovin, Andreas Krause, Beth Gardner, Sarah J. Converse, and Steve Morey. Dynamic resource allocation in conservation planning. In *Proceedings of the 25th Conference on Artificial Intelligence*, pages 1331–1336, 2011.

[Gomez-Rodriguez *et al.*, 2012] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. *ACM Transactions on Knowledge Discovery From Data*, 5(4):21, 2012.

[Hanski, 1994] I. Hanski. A practical model of metapopulation dynamics. *Journal of Animal Ecology*, pages 151–162, 1994.

[Hanski, 1999] I. Hanski, editor. *Metapopulation Ecology*. Oxford University Press, 1999.

[Kempe *et al.*, 2003] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146, 2003.

[Kumar *et al.*, 2012] Akshat Kumar, Xiaojian Wu, and Shlomo Zilberstein. Lagrangian relaxation techniques for scalable spatial conservation planning. In *AAAI Conference on Artificial Intelligence*, pages 309–315, 2012.

[Leskovec *et al.*, 2007] Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. The dynamics of viral marketing. *ACM Transactions on the Web*, 1(1), May 2007.

[Moilanen, 1999] A. Moilanen. Patch occupancy models of metapopulation dynamics: Efficient parameter estimation using implicit statistical inference. *Ecology*, 80(3):1031–1043, 1999.

[Myers and Leskovec, 2010] Seth A. Myers and Jure Leskovec. On the convexity of latent social network inference. In *Advances in Neural Information Processing Systems*, pages 1741–1749, 2010.

[Neal and Hinton, 1998] Radford Neal and Geoffrey E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368, 1998.

[Netrapalli and Sanghavi, 2012] Praneeth Netrapalli and Sujay Sanghavi. Learning the graph of epidemic cascades. In *ACM Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 211–222, 2012.

[Sheldon *et al.*, 2010] Daniel Sheldon, Bistra Dilkina, Adam Elmachtoub, Ryan Finseth, Ashish Sabharwal, Jon Conrad, Carla Gomes, David Shmoys, William Allen, Ole Amundsen, and William Vaughan. Maximizing the spread of cascades using network design. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence*, pages 517–526, 2010.

[Sriperumbudur and Lanckriet, 2009] Bharath Sriperumbudur and Gert Lanckriet. On the convergence of the concave-convex procedure. In *Advances in Neural Information Processing Systems*, pages 1759–1767, 2009.

[ter Braak and Etienne, 2003] Cajo J.F. ter Braak and Rampal S. Etienne. Improved Bayesian analysis of metapopulation data with an application to a tree frog metapopulation. *Ecology*, 84(1):231–241, 2003.

[Wang *et al.*, 2012] Liaoruo Wang, Stefano Ermon, and John E. Hopcroft. Feature-enhanced probabilistic models for diffusion network inference. In *European conference on Machine Learning and Knowledge Discovery in Databases*, ECML PKDD'12, pages 499–514, 2012.

[Wei and Tanner, 1990] Greg Wei and Martin Tanner. A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association*, 85:699–704, 1990.