Scalable Structure Learning of Graphical Models

Walid Chaabene

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Bert Huang, Chair
Liqing Zhang
Serkan Gugercin

April 28$^{\text{th}}$, 2017
Blacksburg, Virginia

Keywords: ℓ_1-based Structure Learning, Linear Dynamical Systems, Markov Random Fields

Scalable Structure Learning of Graphical Models

Walid Chaabene

(ABSTRACT)

Hypothesis-free learning is increasingly popular given the large amounts of data becoming available. Structure learning, a hypothesis-free approach, of graphical models is a field of growing interest due to the power of such models and lack of domain knowledge when applied on complex real-world data. State-of-the-art techniques improve on scalability of structure learning, which is often characterized by a large problem space. Nonetheless, these techniques still suffer computational bottlenecks that are yet to be approached.

In this work, we focus on two popular models: dynamical linear systems and Markov random fields. For each case, we investigate major computational bottlenecks of baseline learning techniques. Next, we propose two frameworks that provide higher scalability using appropriate problem reformulation and efficient structure based heuristics.

We perform experiments on synthetic and real data to validate our theoretical analysis. Current results show that we obtain a quality similar to expensive baseline techniques but with higher scalability.

Scalable Structure Learning of Graphical Models

Walid Chaabene

(GENERAL AUDIENCE ABSTRACT)

Structure learning of graphical models is the process of understanding the interactions and influence between the variables of a given system. A few examples of such systems are road traffic systems, stock markets, and social networks. Learning the structure uncovers the invisible inter-variables relationships that govern their evolution. This process is key to qualitative analysis and forecasting. A classic approach to obtain the structure is through domain experts. For example, a financial expert could draw a graphical structure that encodes the relationships between different software companies based on his knowledge in the field. However, the absence of domain experts in the case of complex and heterogeneous systems has been a great motivation for the field of data driven, hypothesis free structure learning. Current techniques produce good results but unfortunately require a high computational cost and are often slow to execute.

In this work, we focus on two popular graphical models that require computationally expensive structure learning methods. We first propose theoretical analysis of the high computational cost of current techniques. Next, we propose a novel approach for each model. Our proposed methods perform structure learning faster than baseline methods and provide a higher scalability to systems of large number of variables and large datasets as shown in our theoretical analysis and experimental results.

Acknowledgments

I would like to thank Dr. Bert Huang, for his availability, guidance, and support throughout this work. It was a privilege to have him as an adviser.

I would also like to extend my gratitude to Dr. Serkan Gugercin and Dr. Liqing Zhang for kindly accepting to serve on my committee.

Contents

List of Figures

Abbreviations And Notations

- We use the following abbreviations:
  - LDS : Linear Dynamical Systems;
  - ODE : Ordinary Diffrential Equations;
  - MRFs : Markov Random Fields.
- We use the following notations:
  - $G(E,V)$: graph with a node set V and edge set E;
  - $||.||_p$: the L_p norm.
  - For a matrix $M \in \mathbb{R}^{m \times n}$, two integers $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$, and two sets of integers $L \subset \{1, \ldots, m\}$ and $J \subset \{1, \ldots, n\}$:

    $M_{i,j}$: the entry in the i^{th} row and j^{th} column of M;

    $M_{L,J}$: an $|L| \times |J|$ matrix consisting of the intersection between the rows and columns of M having indices in L and J, respectively;

    $M_{L,:}$: an $|L| \times n$ matrix consisting of the rows of M having indices in L;

    $M_{:,J}$: an $m \times |J|$ matrix consisting of the columns of M having indices in J; and
  - $m_{i \to j}(x_j)$: message from node i to node j about a state of (x_j) ;
  - $H[x]$: value associated with key x in hash-table H.

Introduction

Sparse graphical models are powerful tools that provide high computational and analysis convenience (Wainwright and Jordan, 2008). Learning such models has been a popular area of research. In classical problems, we are often given the graphical models' structure by a domain expert. The learning task is then to estimate the model's parameters. However, with the increase of available datasets and the complexity of systems that are being studied nowadays, structure is not always provided.

Techniques using sparsity-inducing norms, *e.g.,* ℓ_1-learning, were introduced to enable learning the structure of sparse graphical models. Some successful applications can be found in (Schmidt, 2010), (Mairal et al., 2010), and (Donoho and Elad, 2003). This family of methods has the common advantage of simultaneously selecting relevant variables and learning the model parameters by combining both variable-selection and parameter-learning steps. Hence, rather than pre-processing the data using an aggressive and premature feature-selection step, ℓ_1-learning does it in accordance with the learning objective.

The convenience of ℓ_1-learning approaches is that no knowledge of the underlying structure is required. In fact, the ℓ_1-penalty treats each variable independently and selects the most relevant in a data driven manner.

Advanced techniques such as group-ℓ_1 allow putting together parameters into groups and hence promote group sparsity. Parameters of the same group become homogeneous and are either selected or zeroed out together. In the case where information about the structural relationship between variable is available, group-ℓ_1 can be leveraged to inject structural information into the learning objective. This is particularly convenient for graphical models where nodes and edges are often encoded as groups of parameters.

Unfortunately, as it will be shown in future chapters, ℓ_1-learning can face scalability problems, which could make it overwhelmingly expensive for large

complex systems.

Proposed Work

In this work, we propose to investigate ways to bring more scalability to ℓ_1-learning methods for two types of models: linear dynamical systems (LDS) (Kalman, 1963) and Markov random fields (MRFs) (Chellappa and Jain, 1993).

Scalable LDS structure learning In Chapter 2, we show that reasoning about LDS as graphical models opens the door to leveraging important partial structure information. We propose a scalable ℓ_1-learning method based on structure heuristics: We exploit direct and indirect dependencies of few seed nodes to infer the entire structure without having to perform ℓ_1-learning on the entire set of nodes.

Scalable MRFs structure learning In Chapter 3, we propose a new approach based on grafting edges in an online manner. First, we construct a group-ℓ_1 objective to derive an edge-activation test. We then propose a framework to activate edges on the fly without having to look at the entire dataset but instead, reason about parts of it in an on-demand fashion.

Applying the proposed frameworks on synthetic and real datasets provides agreeing performance trends that show how we avoid the computational overload of batch data processing necessary for classic approaches while providing a high quality structure recovery in both cases.

Chapter 1

Structure Learning as Sparse Parameter Learning

Introduction

In this chapter, we address the paradigm of ℓ_1-learning of graphical models. First, we introduce the notion of graphical models before specifically presenting LDS and MRFs models. We then discuss the basics and different formulations of ℓ_1-based methods. Finally, we present a brief survey on state-of-the-art gradient-based learning methods that are currently used.

1.1 Graphical Models

A graphical model is defined over a graph structure $G = (V, E)$, where V is a set of nodes representing the model variables and E is a set of edges that describes different variable interactions. These interactions are often parametric as they depend on a set of p parameters $\theta = \{\theta_1, \ldots, \theta_p\}$.

1.1.1 Examples

As examples, we introduce LDS and MRF graphical models. To have a consistent notation for each model, we use A and w respectively as the model parameters for LDS and MRFs.

Linear dynamical systems

Many forecasting methods are based discrete time linear dynamical systems (Kalman, 1963; Ghahramani and Hinton, 1996; Rosenbrock, 1974). A general linear dynamical system is modeled, over N time steps, as

$$\begin{aligned} z^{(l+1)} &= A\ z^{(l)} + E\ u^{(l)} + v^{(l)} \\ x^{(l+1)} &= F\ z^{(l+1)} + w^{(l)} \\ l &= 1, \ldots, N-1\ , \end{aligned}$$

where A is a state matrix denoting the evolution of z, E describes the exterior actions on the system, F maps the states vector to the observation vector, and v and w are vectors representing (time-dependent) Gaussian noise.

Many popular approaches use this representation. For example, Kalman filters recursively predict the most likely values of z in an online fashion. Like other approaches, the classic Kalman filter framework requires prior knowledge of A, E and F. Although defining E and F can be fairly easy—as it is a design related task—defining A is not. In fact, A defines the intrinsic dynamics of a system. In some applications, knowing A is straightforward. For example, tracking the location of a moving robot in a controlled environment that obeys an established physics model (Jetto et al., 1999). In other cases, such as traffic networks, A can be completely unknown as it is related to the nature of the considered network and is often complex and nontrivial.

From a structural point of view, A can be viewed as a hidden influence network (Friedkin and Johnsen, 1999). We simplify the general linear dynamical system formulation to focus on learning A as an ordinary differential equations (ODE) system. To do so, we reduce E to the zero matrix (*i.e.*, when no external action is applied on the system) and F to the identity matrix. Hence, we can simply cast the model as

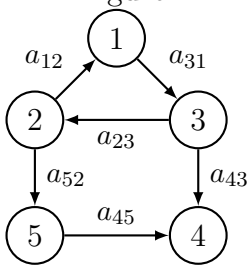
$$\begin{aligned} x^{(l+1)} &= A\ x^{(l)} + e^{(l)} \\ l &= 1, \ldots, N-1\ , \end{aligned}$$

where e is a residual noise.

This formulation means that we observe a noisy version of the true states since the F matrix is an identity matrix, and the noise, e remains spherical.

We can then define a directed graph $G(V,E)$, where V is the set of n nodes and E is the set of edges. We refer to each node by an index $i \in \{1, \ldots, n\}$ as in Figure 1.1.1.

Figure 1.1: Toy example of an ODE graph



$$A = \begin{pmatrix} 0 & a_{12} & 0 & 0 & 0 \\ 0 & 0 & a_{23} & 0 & 0 \\ a_{31} & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{43} & 0 & a_{45} \\ 0 & a_{52} & 0 & 0 & 0 \end{pmatrix}$$

Markov random fields

Markov random fields are popular probabilistic graphical models that have been widely used in different areas such as image analysis, computer vision, and natural language processing (Rue and Held, 2005; Li, 2009). These models capture probabilistic dependencies between variables and encode them in a joint probability factorization.

We consider the case of log-linear, pairwise Markov random fields (MRFs) (Wang et al., 2013). Based on a given MRF structure, the probability of a set of variables $x = \{x_1,..,x_n\}$ is

$$p_w(x) = \frac{1}{Z(w)} \prod_{c \in C} \phi_c(x; w) \tag{1.1}$$

where, $Z(w)$ and ϕ are respectively a normalizing partition function and clique potential function:

$$Z(w) = \sum_x \prod_{c \in C} \phi_c(x; w)$$

$$\phi_c(x; w) = \exp\left(\sum_{k \in c} w_k f_k(x) \right) .$$

Set C contains indexes representing cliques and $f_k(x)$ are feature functions often defined as indicator functions. In the case of pairwise Markov random fields, a clique can either refer to a node or an edge.

An indicator function is a binary function that indicates the state of a variable or a pair of variables given a data instance. The following is an example of unary and pairwise indicator functions:

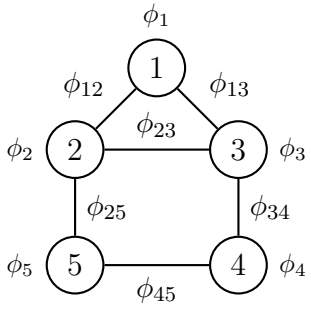
$$f_{k_{\{x_1=1\}}} = \begin{cases} 1 & \text{if } x_1 = 1 \\ 0 & \text{otherwise.} \end{cases} \qquad f_{k_{\{x_1=0\}}} = \begin{cases} 1 & \text{if } x_1 = 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$f_{k_{\{x_1=0,x_2=1\}}} = \begin{cases} 1 & \text{if } x_1 = 0 \text{ and } x_2 = 1 \\ 0 & \text{otherwise.} \end{cases}$$

A pairwise MRF is associated with an undirected graph $G(V, E)$, where V is the set of n nodes corresponding to variables and the set of edges E correspond to pairwise cliques. We refer to a variable x_i as a node i as in Figure 1.2.

$$p_w(x) = \frac{1}{Z(w)} \prod_{i \in V} \phi_i(x; w) \prod_{(i,j) \in E} \phi_{ij}(x; w) \tag{1.2}$$

Figure 1.2: Toy example of a binary pairwise MRF graph



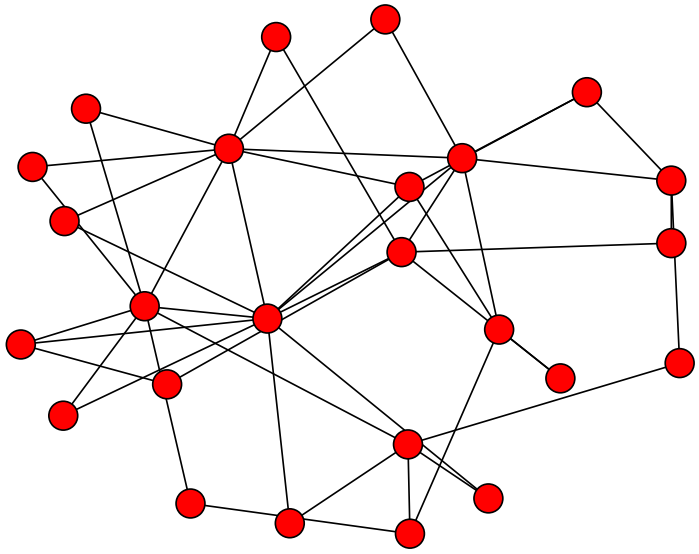
$$\phi_{12} = \begin{array}{c|c|c} & x_1 = 0 & x_1 = 1 \\ \hline x_2 = 0 & \phi_{12}^{(00)} & \phi_{12}^{(10)} \\ \hline x_2 = 1 & \phi_{12}^{(01)} & \phi_{12}^{(11)} \end{array}$$

1.1.2 Scale-free model

It is a fair assumption to treat graphs of models such as LDS or MRFs as random structures. However, it has been shown that natural networks tend to follow a scale-free structure (Albert and Barabási, 2002). This was a result of studying a diverse set of networks such as biological, social, financial and Web graphs. In a scale-free structure, there are few predominant nodes referred to as "hubs". These nodes have the highest number of incident edges. The remaining nodes tend to share their edges with hubs.

A method to grow a scale-free graph based on the preferential attachment model was proposed in (Albert and Barabási, 2002). In this model, a graph

Figure 1.3: An example of a simulated scale-free graph with 25 nodes and 2 added edges per node using Python's Networkx library.



is grown incrementally by adding nodes to the graph. A new node has more preference to share its k edges with an existing node that has a high number of connections. In fact, the probability p_i that the t^{th} added node becomes connected to some node i is

$$p_i = \frac{k_i}{\sum_{j \in V^{(t)}} k_j} \ ,$$

where k_j is the number of neighbors of node j and $V^{(t)}$ is the set of nodes in the graph at iteration t.

This mechanism eventually leads to creating a graph dominated by hubs as shown in Figure 1.3.

1.2 Sparsity-Inducing Norms

Many current approaches to structure learning are based on constructing a loss function $F(\theta)$ and minimizing it given a set of data points $X = \{x^{(m)}\}_{m=1\ldots N}$.

$$\min_{\theta} F(X,\theta) \tag{1.3}$$

To avoid over-fitting the learned parameters to dataset X, a regularization term R is often added to F as follows:

$$\mathbb{F}(X,\theta) = F(X,\theta) + R(\theta) \ . \tag{1.4}$$

To promote sparsity of the learned weights, and hence of the learned graphical model, an ℓ_1-norm is often used as a regularizer. There is a difference between promoting sparsity in the solution and avoiding over-fitting; although the former implies the latter.

$$\mathbb{F}_{\mho}(\theta) = F(X,\theta) + \lambda||\theta||_1 \tag{1.5}$$

This regularization encourages irrelevant parameters to have zero values at the solution. As λ increases, more parameters tend to zero.

The classical ℓ_1-formulation ignores any available structure information and treats parameters as generated using the same distribution. If parameter group memberships are available, as in the case of MRFs, it becomes more convenient to use the group-ℓ_1 as will be shown in Chapter 3. In doing so, the learned parameters of each group of parameters tend to be more homogeneous.

$$\mathbb{F}_{\eth}(\theta) = F(\theta, X) + \sum_{g \in G} \lambda_g ||\theta_g||_2 \tag{1.6}$$

The quantity λ_g is the regularization parameter of group g, and θ_g refers to the subset of parameters corresponding to group g in the set of groups G. In principle, at least, a group g can either refer to a variable or an edge but other formulation based on domain knowledge can motivate the construction of additional types of groups.

It is interesting to notice that full-ℓ_1 is a special case of group-ℓ_1, where every coordinate of θ constitutes a group.

1.3 Gradient Methods

In this section, we discuss the aspects of solving Equation (1.4) with a full-ℓ_1 or group-ℓ_1 regularization.

1.3.1 Gradient updates

Gradient-based methods consist of performing parameter updates using gradient descent (or ascent) following the best possible direction. For objectives using group-ℓ_1 regularization, the gradient is however not defined at zero:

$$\nabla_\theta \mathbb{F}_{\eth}(\theta, X) = \nabla_\theta F(\theta, X) + \sum_{g \in G} \lambda_g \frac{\theta_g}{||\theta_g||_2} \ . \tag{1.7}$$

In the case of full-ℓ_1 norm, objectives suffer non-smoothness at zero. This introduces a challenge for gradient-based methods. To work around this issue, we often use sub-gradient methods. In our case, we approximate the gradient of (3.5) by a sub-gradient of the form:

$$\nabla_\theta \mathbb{F}_{\mho}(\theta, X) = \nabla_\theta F(\theta, X) + \text{sign}(\theta) \tag{1.8}$$

where the function $\text{sign}(\theta)$ is defined coordinate-wise as

$$\text{sign}(\theta_i) = \begin{cases} 1 & \text{if } \theta_i \geq 0 \\ -1 & \text{otherwise} \ . \end{cases}$$

In the general case, a parameter-wise update is performed as follows:

$$\theta_i^{(t+1)} = \theta_i^{(t)} + \eta^{(t)} d_i^{(t)}$$

where $d_i^{(t)}$ is the descent direction for coordinate θ_i and $\eta(t)$ is the descent rate at iteration t.

Quasi-Newton methods are state-of-the-art methods for solving unconstrained large-scale optimization problems (Dennis and Moré, 1977). These approaches estimate the best possible direction based on the secant method by computing the inverse of the objective Hessian using a second degree Taylor expansion. The best direction is then derived by setting the future step gradient to zero, which is the goal of the optimization. BFGS, L-BFGS and OWL-QN (Liu and Nocedal, 1989; Zhu et al., 1997; Andrew and Gao, 2007) are all based on the quasi-Newton method but each is the derivation of

the other. The methods within this family differ in the way they handle the Hessian matrix computation and storage.

The basic quasi-Newton method computes the Hessian by solving a linear system. BFGS incrementally estimates the Hessian and avoids solving a linear system as in quasi-Newton. L-BFGS is limited memory BFGS, which avoids storing the entire inverse of the Hessian matrix, which can be expensive in large-scale problems. L-BRGS instead stores a small set of vectors that represent the Hessian implicitly. OWL-QN is an orthant-wise descent method specially developed for ℓ_1-regularized methods. This method avoids the issues produced by the non smoothness of the the ℓ_1-norm by adaptively modifying the descent direction and promoting sparsity of its solutions.

1.3.2 Stochastic and mini-batch gradient descent

Vanilla batch learning consists of using the entire dataset X to compute $F^{(t)}(\theta, X)$ at each iteration of gradient descent. However, the size of current datasets is increasingly large, which introduces computational challenges to learning algorithms. Furthermore, batch learning does not allow online learning, which requires that data be available as one, or a group of, data instances at each gradient-descent iteration.

To work around these limitations, mini-batch and stochastic gradient methods were introduced.

Stochastic gradient (Bottou, 2010) optimization, is amenable to online learning as it performs a parameter update using one data point $x^{(m)}$ at each descent iteration by computing $F(\theta^{(t)}, x^{(m)})$. Therefore, it is fast and also avoids redundant computation on the same data points at each descent iteration. However, stochastic gradient descent produces large fluctuations when minimizing the objective as it is sensitive to data variance. Nonetheless, it could be useful and practical to use stochastic-gradient in some cases as shown in Chapter 2, where the task is not to provide the exact solution to an optimization problem, but to select relevant parameters.

In-between, both extremes of batch and stochastic gradient is mini-batch learning (Cotter et al., 2011). This method randomly selects a subset of $|I|$ data points $\{x^{(i)}\}_{i \in I}$ and computes the objective function as $F(\theta^{(t)}, \{x^{(i)}\}_{i \in I})$. This method is much faster than batch gradient descent and is less sensitive to data variance than stochastic gradient descent.

1.3.3 Active-set methods

Structure learning of graphical models is often challenged by the dimension of its parameter space. Active-set methods (Kelley, 1999) were introduced to classical machine learning problems to significantly reduce the number of variables during optimization as this cost becomes prohibitive in large real-world datasets that include large numbers of variables (Kim and Park, 2008; Schmidt et al., 2007; Perkins et al., 2003).

Active-set methods split the parameter space into two sets — A search space F and an active space S — such that $F \cap S = \emptyset$. Only constraints related to parameters in the set S are considered for optimization. At each iteration, one inactive parameter associated with the maximum violation of the optimality condition is moved to set S. This two-step method can be described as follows:
Step 1: Optimizing over the active set S.
Step 2: Expanding the active set by activating one parameter associated with the maximum violation of the solution's optimality condition.

Active set methods provide the convenience of a warm start from an iteration to another for active parameters, yielding a faster convergence.

The optimality condition is specific to the task in hand but related to the ℓ_1-regularization that is used. In Chapter 3 for example, we present the optimality condition for the grafting algorithm. We then extend the active-set method into a version of group activation that constitutes the basis of our proposed framework for MRFs structure learning.

Chapter 2

Scalable Structure Learning of Linear Dynamical Systems

Introduction

It is often assumed that the structure of linear dynamical systems is provided. However, in many settings, the structure of these systems is unknown, such as in traffic networks (Samaranayake et al., 2011). Most available data about real-life dynamical systems is limited to time evolving observations. Traditional techniques for high-dimensional systems such as subspace identification (Van Overschee and De Moor, 2012; De Cock) have been developed to learn the underlying dynamics of linear systems such as singular value (Golub and Reinsch, 1970) and QR decompositions (Katayama, 2006). However, they come at a cubic-time cost (Brand, 2006). In large-scale datasets, learning such systems presents a significant challenge in scalability. Furthermore these techniques do not consider the fact that real systems tend to be sparse (Aykanat et al., 1988), and hence do not exploit this fact to establish faster and more scalable solutions.

In this chapter, we propose a general framework for scalable learning of linear dynamical systems. The framework approaches the learning problem by identifying key structural patterns in partial structure and inferring the rest of the structure using effective heuristics.

In graph terminology, the method uses data to detect which nodes could be two-hop parents of another. The algorithm then treats this two-hop information as a collaborative filtering pattern, inferring from potentially

noisy estimates of two-hop parents the likely direct parents for much of the network. This filtering is done without having to directly reason about all nodes or relationships. We evaluate this approach, comparing it against variants that do not exploit this data-driven sparsity, on synthetic and real data. Our results demonstrate that it is an effective strategy for learning the dynamics of a system with significant savings in computational cost.

2.1 Background

In this section, we introduce the context, baseline techniques, and building block algorithms used to learn linear dynamical systems.

We consider time evolving observations $X = [x^{(1)} \dots x^{(N)}]$. Observation $x^{(t)}$ is a snapshot of noisy observations of the system states vector z. Variable $x \in \mathbb{R}^n$ is captured over N equidistant time steps. The learning goal is to uncover the underlying linear dynamical system that governs the evolution of $x(t)$.

A fundamental basis of our work is to treat the system as a graph where each component of $x(t)$ refers to a node. We hence define a graph (V,E), where V is the set of n nodes and E is the set of edges. We refer to each node by an index $i \in \{1,\dots,n\}$. Each state can be seen as a time evolving node attribute. We define observations of state evolution matrix as $X = [x^{(1)} \dots x^{(N)}] \in \mathbb{R}^{n \times N}$.

Our aim is to learn A in a data-driven way from our observation of X. We start by discussing a well-established technique that is used to solve this inverse problem. We refer to this baseline as *dense learning*, since it does not promote sparsity in the learned system. We then introduce different sparsity-based approaches that we utilize later as building blocks of our proposed framework.

2.1.1 Dense learning

As in (Schmidt et al., 2007), the first step is to define $X' = [x^{(1)},\dots,x^{(N-1)}]$ and $X'' = [x^{(2)},\dots,x^{(N)}] = [A\,x(1),\dots,A\,x^{(N-1)}]$ for a given attribute x. Hence,

$$X'' = AX' + E \tag{2.1}$$

where $E = [e^{(1)},\dots,e^{(N-1)}]$, using the same notation as in Chapter 1.

Algorithm 1 Dense Learning

1: **Input:** $X = [x^{(1)}, \ldots, x^{(N)}]$
2: Construct $X' = [x^{(1)}, \ldots, x^{(N-1)}]$, $X'' = [x^{(2)}, \ldots, x^{(N)}]$
3: $[U, \Sigma, V^*] = \text{SVD}(X')$
4: $A = X''V\Sigma^{-1}U^*$
5: **Return:** A

The learning task is reduced to solving the following optimization problem:

$$\min_A ||X'' - AX'||_{\text{F}} \tag{2.2}$$

where $||.||_{\text{F}}$ refers to the Frobenius norm (Golub and Reinsch, 1970).

The above problem falls under the category of inverse problems (Cotter et al., 2005). A well-established and efficient approach is the singular-value decomposition based approach. We first compute the singular value decomposition (SVD) of X':

$$X' = U\Sigma V^* \tag{2.3}$$

where, Σ is an $r \times r$ diagonal matrix determined by the ordered r singular values $\{\sigma_1, \ldots, \sigma_r\}$ of X; $U \in C^{n \times r}$ s.t. $U^*U = I_n; UU^* = I_r$; $V \in C^{l \times r}$ s.t. $V^*V = I_l$; and r is the rank of X.

Let $X'^{\dagger} = V\Sigma^{-1}U^*$ be the right pseudo inverse of X', *i.e.* $X'X'^{\dagger} = I_n$. We then recover the state space matrix by solving

$$A \approx X''V\Sigma^{-1}U^* \ . \tag{2.4}$$

Algorithm 1 summarizes the above steps. Note that computing the SVD of X has an asymptotic time complexity of $O\left(\min\{nN^2, Nn^2\}\right)$, which is therefore the dominating cost of Algorithm 1.

The resulting A is fully connected, as there is no sparsity promoting component in this first approach. This presents a high risk of over-fitting and a high computational cost to inference using this learned A.

2.1.2 Sparsity enforcing learning

Enforcing a certain structure on the system translates into selecting a reduced number of parents for each state i, *i.e.*, reducing the number of states that

influence node i in any time step, and learning the corresponding weights for every incoming influence. For this reason, we break the learning task into smaller problems, where we learn the influence of parents of each state separately. This decomposition scales better with systems with a high number of states, providing significant computational savings over the dense learning procedure. We refer to this procedure as *sparsity enforcing learning*.

An adjacency matrix M encodes the the structure of the system such that $\forall n,k \in \{1,\ldots,n\}$

$$M_{k,i} = \begin{cases} 1 & \text{if state k influences i in one time step} \\ 0 & \text{otherwise .} \end{cases}$$

Let N_i be the neighborhood (set of parents) of i, *i.e.*, $M_{k,i} = 1\ \forall k \in N_i$. We select the sub-matrix $X'_{N_i,:}$ consisting of the rows referring to the parent nodes of i and the i^{th} row of X'', $X''_{i,:}$.

The learning problem is then solved via (Lawson and Hanson, 1995)

$$\min_{\hat{A}_i} ||(X''_{i,:} - \hat{A}_i X'_{N_i,:})^\top||_2. \tag{2.5}$$

As in Section 2.1.1, we compute the SVD of $X'_{N_i,:}$,

$$X'_{N_i,:} = U_i \Sigma_i V_i^*. \tag{2.6}$$

The solution for this new optimization problem is $\hat{A}_i \approx X''_{i,:} V_i \Sigma_i^{-1} U_i^*$. We summarize these steps in Algorithm 2. Assuming that there exists a $K \in \mathbb{N}$ such that $\forall i \in V$, $|N_i| \leq K$, the algorithm has a time complexity of $O(nNK^2)$. The steps of Algorithm 2 can be implemented in parallel, making it well-suited to large networks. Another important outcome that this approach presents is reducing the risk of over-fitting. Since *sparsity enforcing learning* requires the sparsity structure as an input, we discuss in the next section methods for learning it, leading to our main framework.

2.1.3 Sparsity inducing learning

The task of learning the optimal structure of the system consists of detecting the k most influential states for every state $i \in V$, where k is predefined as the target cardinality of all neighborhoods across the graph. One well-established

Algorithm 2 Sparsity Enforcing Learning

Input: Adjacency matrix M and X
Initialize $A = 0_{n \times n}$
Construct $X' = [x^{(1)}, \ldots, x^{(N-1)}]$, $X'' = [x^{(2)}, \ldots, x^{(N)}]$
for $i \in \{1, \ldots, n\}$ **do**
    $N_i =$ Indices of nonzero entries in $M(:, i)$
    Select $X'_{N_i,:}$ and $X''_{i,:}$
    $[U_i, \Sigma_i, V_i] = \text{SVD}(X'_{N_i,:})$
    $\hat{A}_i = X''_{i,:} V_i \Sigma_i^{-1} U_i^*$
    $A_{i,N_i} = \hat{A}_i$
end for
Return: A

approach to enforce sparsity is using ℓ_1-regularization. The learning task becomes that of minimizing F_i with respect to $\widetilde{A}_i$ such that,

$$F_i = ||(X''_{i,:} - \widetilde{A}_i X')^\top||_2 + \lambda ||\widetilde{A}_i^\top||_1, \ \forall i \in 1, \ldots, n,$$

where λ is a parameter that controls the sparsity induced in the solution.

The main drawbacks in using such a method is the loss of an exact, closed-form solution, and the non-smoothness of the ℓ_1-norm. However, sub-gradient based descent methods were designed to solve this exact kind of optimization problem as discussed in Chapter 1. In our case, the sub-gradient is given by the following equation:

$$\nabla_{\widetilde{A}_i} F_i = -e_i X' + \lambda \, \text{sign}(\widetilde{A}_i), \tag{2.7}$$

where, $e_i = X''_{i,:} - \widetilde{A}_i X'$.

Each gradient iteration updates the vector $\widetilde{A}_i$ with $\widetilde{A}_i = \widetilde{A}_i - \gamma \nabla_{\widetilde{A}_i} F_i$, where γ is the learning rate or step size.

However, one important consideration is that we are not directly interested in learning the exact solution $\widetilde{A}_i$. In fact, our learning task as stated above is to detect the k most influential nodes for every node $i \in V$, *i.e.*, select the k indices in $|\widetilde{A}_i|$ that have the highest inputs. Getting the right order for the inputs of $|\widetilde{A}_i|$ should require much fewer iterations than learning the exact $\widetilde{A}_i$. This method can be seen as a feature selection task using ℓ_1 optimization, as has also been suggested in previous work Ng (2004); Guyon et al. (2008).

Algorithm 3 Sparsity Inducing Learning

Input: X, γ, k, λ
Initialize $M = 0_{n \times n}$
$X' = [x^{(1)}, \ldots, x^{(N-1)}]$, $X'' = [x^{(2)}, \ldots, x^{(N)}]$
for $i \in \{1, \ldots, n\}$ **do**
  Generate a random $\widetilde{A}_i$
  repeat
    Pick a random time step l and select $X''_{i,l}, \quad X'_{:,l}$
    $e_i = X''_{i,l} - \widetilde{A}_i X'_{:,l}$
    $\nabla_{\widetilde{A}_i} F = -e_i X'_{:,l} + \lambda \operatorname{sign}(\widetilde{A}_i)$
    $\widetilde{A}_i = \widetilde{A}_i - \gamma \nabla_{\widetilde{A}_i} F$
  until convergence
  $N_i =$ indices of the k highest inputs in$|\widetilde{A}_i|$
  $M_{N_i,i} = 1$
end for
Return: M

To obtain the fastest possible approximation of the sparsity structure, we use the stochastic gradient descent method. The steps of regularized sparsity induced learning are summarized in Algorithm 3.

Once we learn the optimal (estimated) structure of the influence network captured by M, we can apply Algorithm 2 to learn the weights of the edges. However, Algorithm 3 comes at a cost of $O(tn^2)$, where t is the maximum number of iteration needed for convergence of the SGD ($t < N$). Applying *sparsity inducing learning*, thus has a total time complexity of $O(tn^2 + nNk^2)$.

2.2 Scalable Structure Learning

Time complexity can be a major drawback while applying the sparsity-inducing approach, especially when dealing with extremely large numbers of states. Though learning a sparse structure allows the coefficient learning phase to be inexpensive, the structure-learning step requires considering all possible entries in the $n \times n$ matrix. In this section, we present *scalable structure learning*, which exploits multi-hop neighborhood structure to infer the sparsity structure without having to consider all possible influence pairs.

Given the node set V, we select a random subset S of nodes that we refer

to as seed nodes. We define L as $V - S$. For each node i in S, we learn N_i, the one-hop parents of i.

Learning the set of immediate parents N_i is straightforward using the inner loop of Algorithm 3. However, we can also leverage the same approach to reason about two-hop parents. We compute a score b_{ki} for each node k in V that describes the likelihood of having a two-hop walk from k to i. We can then directly learn about two-hop parents using a similar approach to the one presented in Algorithm 3, by learning from the observations X. The main difference is that we would like to learn a matrix B that governs the system

$$B\ X^{(l)} = X^{(l+2)},\ \ \forall l = 1, \ldots, N-2. \tag{2.8}$$

Note that B can be viewed as A^2, but since we are directly learning it from data, it is an approximation. The motivation behind this approach is the fact that the likelihood of k linking to any node in N_i can be indirectly measured by the likelihood of k having a two-hop walk to i since any two-hop parents of i must link to at least one node in N_i. Taking advantage of the partial structure information, the likelihood of having a link between k and i is measured by the amount of the re-occurrences of such pattern throughout the analysis of all seed nodes.

Figure 2.1 is an example where z and x are seeds who share $s \in L$ as a common one-hop parent and β as a common two-hop parent. Based only on this information and following the above assumption, we estimate a higher likelihood for the existence of an edge between β and s than an edge between β and any other node in the one-hop parents of z and x.

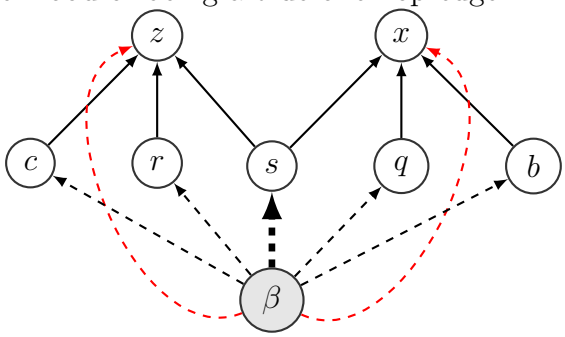
Table 2.1: Time complexity of different methods

Algorithm	Structure Learning	Coefficient Learning	Total Cost
Dense	0	$O\left(\min\{nN^2, Nn^2\}\right)$	$O\left(\min\{nN^2, Nn^2\}\right)$
Sparsity Inducing	$O(tn^2)$	$O(nNk^2)$	$O(tn^2 + nNk^2)$
Scalable	$O((t+k)\|S\|n)$	$O(nNk^2)$	$O((t+k)\|S\|n + nNk^2)$

Computing the one-hop parents of each node $i \in S$ and the scores of two-hop walks leading to them allows us to create an edge score matrix $E \in \mathbb{R}^{n \times n}$ such that $\forall j \in L$ and $u \in V$:

$$E_{j,u} = \sum_{i \in S} b_{i,u} \delta(j,i), \tag{2.9}$$

Figure 2.1: Example of relationship between two-hop influence estimates and one-hop structure. Bold black arrows are true edges, red dashed arrows represent two-hop links, and the thickness of black dashed arrows describes the estimated likelihood of being a true one-hop edge.



where $b_{i,:} = \frac{|B_{i,:}|}{||B_{i,:}||_1}$ and $\delta(j,i) = \begin{cases} 1 & \text{if } j \in N_i \\ 0 & \text{otherwise} \end{cases}$

We then select the k top scoring entries in $E_{j,:}$ as the parents of j, *i.e.*, the non-zero entries in A. Algorithm 4 summarizes the steps of this approach. Scalable structure learning has a time complexity of $O(t|S|n)$. Hence, applying scalable sparsity influence learning comes with a time complexity of $O(t|S|n + nNk^2)$, where again, t is the user-defined parameter for the number of stochastic gradient steps, $|S|$ is the user-defined size of the seed set, N is the number of observed time steps, and k is the user-defined target number of parents. With these parameters held constant, the total cost is linear in the number of nodes.

Time complexity analysis Table 2.1 summarizes time complexities of the different approaches. Even though *dense learning* has no structure recovery step, it still has a cubic time complexity, which does not scale with very large networks. On the other hand, breaking the task of structure recovery in the *sparsity inducing learning* method yields a lower overall cost since we have $t < N$ and $k \ll n$ in most cases. However, this still has a quadratic growth with respect to n. With *scalable structure learning*, choosing a random seed set avoids this issue and reduces the overall complexity to a linear cost with respect to n and N, while still obtaining reliable estimates of the sparsity structure. Although *scalable structure learning* has a time complexity term

Algorithm 4 Scalable Structure Learning

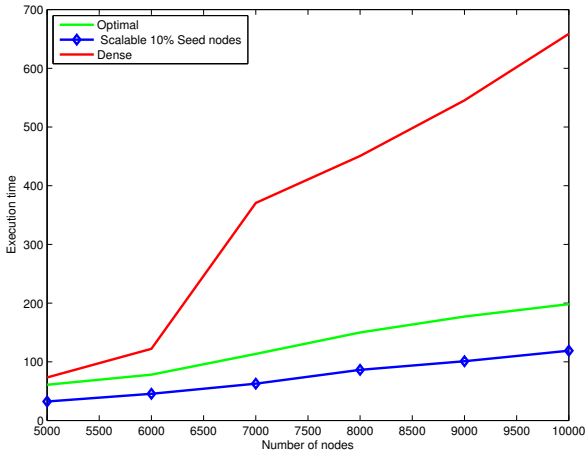
Input: X, γ_1, γ_2, k, λ
$X' = [x^{(1)}, \ldots, x^{(N-2)}]$, $X'' = [x^{(2)}, \ldots, x^{(N-1)}]$, $X''' = [x^{(3)}, \ldots, x^{(N)}]$.
Select a random subset S of V
for $i \in S$ **do**
  Initialize $\widetilde{A}_i$ and B_i randomly, $L = 0_{n \times n}$, $M = 0_{n \times n}$
  repeat
    Pick a random l and select $X''_{i,l}$, $X'''_{i,l}$ and $X'_{:,l}$
    $e_i^{(1)} = X''_{i,l} - \widetilde{A}_i X'_{:,l}$; $e_i^{(2)} = X'''_{i,l} - B^{(i)} X'_{:,l}$
    $\nabla_{\widetilde{A}_i} F_1 = -e_i^{(1)} X'_{:,l} + \lambda \operatorname{sign}(\widetilde{A}_i)$
    $\nabla_{B_i} F_2 = -e_i^{(2)} X'_{:,l} + \lambda \operatorname{sign}(B_i)$
    $\widetilde{A}_i = \widetilde{A}_i - \gamma_1 \nabla_{\widetilde{A}_i} F_1$; $B_i = B_i - \gamma_2 \nabla_{B_i} F_2$
  until convergence
  $N_i =$ indices of the k highest inputs in$|\widetilde{A}_i|$
  $M_{N_i,i} = 1$
  $L_{N_i,:} = L_{N_i,:} + \frac{|B_{i,:}|}{||B_{i,:}||_1}$
end for
for $j \in V - S$ **do**
  $N_j =$ indices of the k highest inputs in $L_{j,:}$
  $M_{N_j,j} = 1$
end for
Return: M

of nNk^2 corresponding to the cost of coefficient learning, this procedure can be done in parallel for each node, yielding a cost of Nk^2 per node.

2.3 Experiments

In this section, we present experiments performed on both synthetic and real data. In the synthetic setting, we randomly generate a ground-truth system state matrix A as well as observations of state values over equidistant time snapshots. For real data experiments, we analyze New York City taxicab data, where we model city blocks as system states, and we learn the underlying dynamical system that can be used as a predictive model for taxicab activity.

Figure 2.2: Execution time (s) of different methods vs increasing LDS size. The synthetic structure is generated randomly with a density of 0.1. Training datasets of 12,500 instances are generated from each synthetic system.



2.3.1 Synthetic experiments

In our synthetic experiments, we generate data such that we can measure the scalability and ability of methods to recover the true sparsity structure and coefficients of the dynamic system matrix.

Scalability evaluation

We first investigate the scalability of different methods to an increasing size of LDS systems. Figure 2.2 shows that *scalable structure learning* with 10% of nodes selected as seed nodes has the best scalability where the execution time evolves slower than *sparsity inducing learning*. The figure also shows the poor scalability of *dense learning* as the execution time increases drastically with an increasing number of nodes.

Data and metrics for performance evaluation

We generate 100 system state matrices A of different sparsity patterns. For each experiment we generate a sparse matrix with 1,000 uniformly distributed nonzero entries. These serve as ground-truth examples. We then generate snapshots of node attributes. To do so, we start by generating two matrices

X'_{train} and X'_{test}. We then generate X''_{train} and X''_{test} such that $X''_{\text{train}} = AX'_{\text{train}}$ and $X''_{\text{test}} = AX'_{\text{test}}$.

We recover the dynamical system by learning $\bar{A}$ using the training data via different methods. We set the number of desired parent nodes to 15 when recovering the system structure. We measure performance using the following metrics:

1. Coefficient recovery:

$$\text{CoeffRecError} = \frac{||\bar{A} - A||_{\text{F}}}{||A||_{\text{F}}} \tag{2.10}$$

2. Predictive performance:

$$\text{PredError} = \frac{||\bar{A}X'_{test} - X''_{\text{test}}||_{\text{F}}}{||X''_{\text{test}}||_{\text{F}}} \tag{2.11}$$

3. Structure recovery (NetStructRec):

   Average true positive rate of sparsity structure recovery over all states.

To measure the behavior of the algorithms in varying conditions, we measure performance at different noise levels applied on the training data.

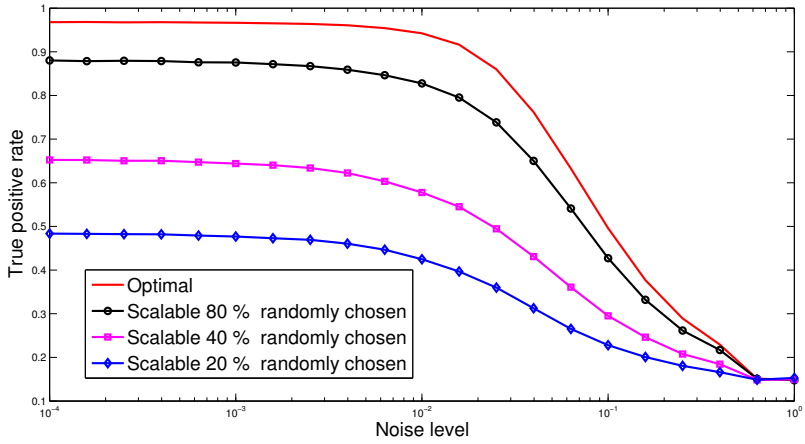
Performance results

Figure 2.3 plots the evolution of different methods' performance on recovering the true structure of the dynamical system, (StructRec). Unsurprisingly, *sparsity inducing learning* performs the best at recovering the true structure of the influence network. We also see a positive correlation between the growth of the set of seed nodes and the performance amelioration of *scalable structure learning*.

Similar behavior can be observed in recovery of coefficients (CoeffRecError), shown in Figure 2.4. As noise levels increase, CoeffRecError grows for each method, however the higher the number of seed points, the more robust the module is to noise.

Although *dense learning* is shown to have roughly the same predictive performance in a noiseless setting, Figure 2.5 shows that using optimal structure methods in both its complete and scalable versions are less sensitive to noise and yield a lower predictive error in noisy settings, such as most real

Figure 2.3: StructRec vs. noise. The optimal sparsity inducing learning obtains the best accuracy at the cost of more computation, while the scalable approaches trade off accuracy for speed.



world applications. On the other hand, the trend between the cardinality of the set seed nodes and PredError is also clear. However, we can see that the scalable method reaches the performance of the optimal sparsity inducing approach when noise levels increase. This trend should be taken under consideration when dealing with noisy data, where the scalable version is likely to give same results with much less computational cost.

2.3.2 Dynamic system of traffic in NYC

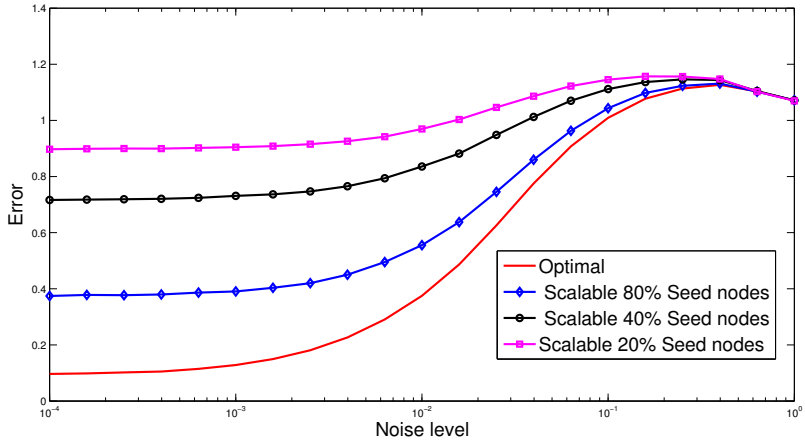
In this task, we propose to uncover traffic influence relations between different blocks of New York City using taxi data. This kind of application can help city planners design better road maps and emergency plans. It can also help traffic managers forecast traffic jams based on the current state of traffic. Finally, this type of analysis can help transportation service providers better manage their cab fleets to optimize their service and costs.

Data

The data is publicly provided by the City Taxi and Limousine Committee[1]. It contains records of time-stamped pick-ups and drop-offs tagged with GPS

[1]http://www.nyc.gov/html/tlc

Figure 2.4: CoeffRecError vs. noise. The high-cost *sparsity inducing learning* method achieves the least error, and using small fractions of the nodes as seeds for scalable learning enables trade-offs between cost and accuracy.



coordinates. We focus on yellow taxi data gathered from January to May, 2015.

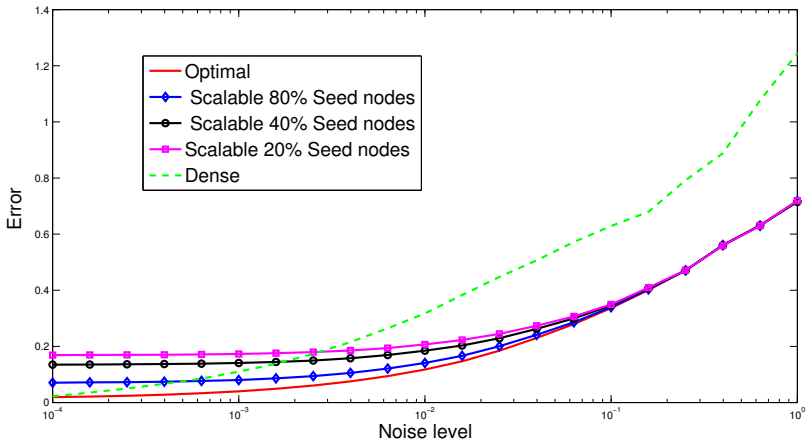
First we construct a grid over New York City. Each block in the grid is considered to be a node. We then define the dynamic attribute as the total taxi activity in a certain city block, which is the max-normalized sum of pickups and drop-offs per time-step in each block. We choose a time step of one hour, and we construct the dynamic node attribute matrix X.

We build an initial grid over New York city using 200 longitude and latitude equidistant boundaries (100 each). The resulting grid contains 9,801 blocks. In order to improve the reliability of the data, we only select active blocks to be nodes in the network. The selection condition is done using a threshold of 100 drop-offs and pick-ups during the considered period. This pruning reduces the number of nodes to 4,057 nodes. We then split our data into training and testing sets. We use the first set to learn the influence network using different methods, and we use the second set to compare their performances.

Results

Quantitative analysis Although our method allows the user to set the cardinality of neighborhoods per state k as an input, we are interested in

Figure 2.5: PredError vs. noise. Dense learning method tends to over-fit. Inducing or enforcing sparsity reduces the model complexity.



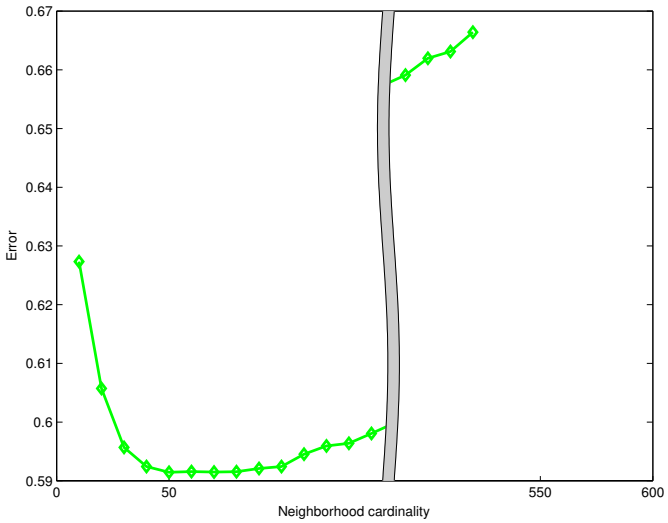
finding the optimal k in an empirical manner. The optimal number of parents per node is an intrinsic attribute of the considered network. It tells us about the volume of influence and the amount of hidden interactions between nodes.

Figure 2.6 plots PredError with the neighborhood cardinality using *sparsity inducing learning*. Using 50 parents per state yields the best performance. It is worth noting that choosing only 50 out of 4057 nodes to be parents of a given state presents and important space and time savings, especially when applying the model for predictive purposes.

We are also interested in seeing how the percentage of seed nodes in *scalable structure learning* affects the performance of our model. For this purpose, we observe trend of PredError with respect to the percentage of seeds, and we compare it with the performance of *sparsity inducing learning* and *dense learning* in Figure 2.7.

As expected, *dense learning* has a high risk of over-fitting which can be seen in the poor predictive performance in Figure 2.7. On the other hand, the scalable method performs well and reaches a performance very close to that of the optimal structure learning (using *sparsity inducing learning*) with as few seed nodes as 10% of the full graph, reaching the same performance at 20%. This trend can be explained by the robustness of the scalable method to noise also seen in our synthetic experiments. In fact, *sparsity inducing learning* performs well in noiseless settings with a predefine true sparse influence network. This noiselessness should not be expected in real

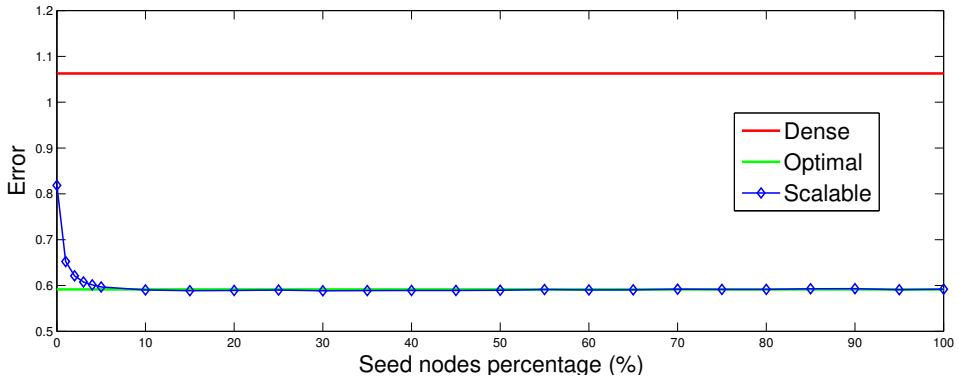
Figure 2.6: PredError vs. neighborhood cardinality. Using a neighborhood size of approximately 50 with a 15% of seed nodes leads to the best predictive performance.



world systems that are complex and have many confounding factors. Hence using *sparsity inducing learning* can be seen as an abuse of resources when one can obtain similar results with *scalable structure learning* using a much less computational cost.

Qualitative analysis In this section, we evaluate how well *scalable structure learning* captures real dynamic patterns of taxicab activity. We should expect the cab activity to alternate between high and low volumes throughout the hours of the day and between week days. Figure 2.8 shows the dynamics of cab activity over one week in four of the busiest neighborhoods of New York City. The plots show that, while dense influence learning fails to capture the traffic patterns, our approach does it with a satisfactory accuracy. Following our analysis above, these results were obtained using neighborhood cardinality $k = 50$ and a percentage of seed nodes of only 20%, making the learning and prediction very inexpensive.

Figure 2.7: PredError vs. seed set cardinality. Dense method is likely overfitting. Scalable learning is able to achieve a closely similar error to optimal using only 10% of the nodes as seeds and a neighborhood cardinality of 50.

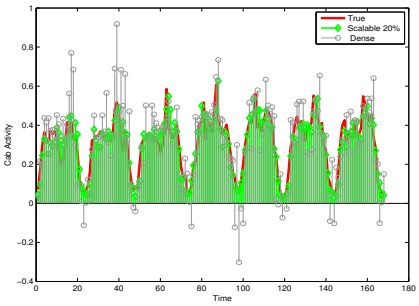


Conclusion

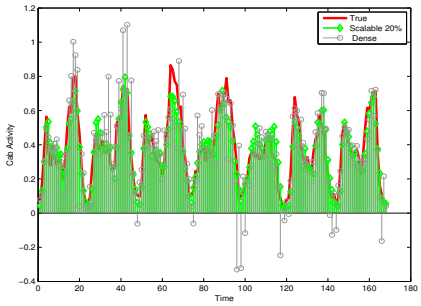
In this chapter, we proposed *scalable structure learning*, a general framework for learning dynamical systems. The framework includes various parameters that allow significant savings in computational cost and provides high sparsity scalability. The learning procedure first estimates the structure of the dynamical system by reasoning about a set of seed nodes. It does this by learning the local structure of the seed nodes using observed data, estimating their immediate and two-hop parents. Using the learned structure information of the seed subset, we infer the parents structures for the second subset using low-cost and effective heuristics. In addition, our algorithm can be implemented in parallel, enabling it to scale to large network settings.

Experiments with synthetic data showed that our method has high tolerance for noise and is able to effectively recover and estimate the coefficients of linear dynamical networks. In the real-data experiments, the algorithm gave convincing results, by effectively modeling traffic activity between blocks of NYC. We showed that the scalable-learning method can get as accurate a result as the optimal-structure learning with the sparsity inducing approach using a seed subset containing only 20% of the nodes. The algorithm provides a high-accuracy predictor of traffic patterns, even when the dense baseline method failed.

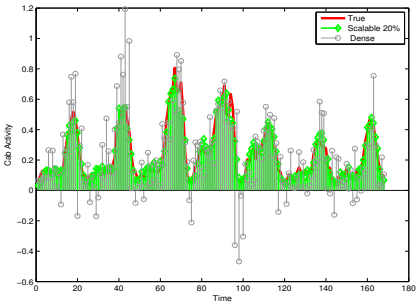
Figure 2.8: Predictions of cab activity in different NYC areas. Our learned dynamical system is able to predict the dynamic rise and fall of activity in these blocks, while dense learning makes predictions that are clearly incorrect.



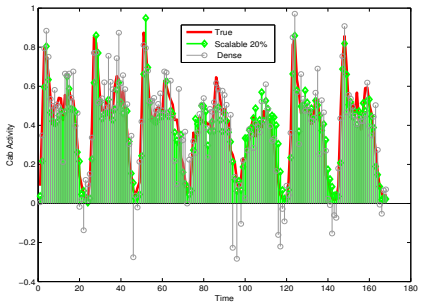
(a) Times Square



(b) Wall Street



(c) East Village



(d) Upper East Side

Chapter 3

Scalable Edge Grafting in Pairwise Markov Random Fields

Introduction

One well-established approach to learning methods of Markov random fields is based on minimizing the ℓ_1-regularized negative log likelihood (Andrew and Gao, 2007). At the end of the optimization task, parameters of irrelevant edges are reduced to zero. The structure learning problem hence becomes that of finding the structure, and parameters, that optimize the ℓ_1-regularized negative log likelihood.

The main challenge when using these particular methods is that, for large enough MRFs, the feature space becomes extremely large and could result in an overwhelmingly high computational cost. Active set methods were introduced to promote more scalability. One popular approach is *grafting* (Perkins et al., 2003; Lee et al., 2006).

However, *grafting* incurs a significant computational cost: As a mandatory pre-learning step, *grafting* requires the sufficient statistics of states of all possible variable pairs to enable a greedy activation tests on the entire search space. Furthermore, *grafting* does not reason about the structure of the MRFs as it activates single parameters.

This chapter introduces *online edge grafting*, an active set method that activates edges, in an online fashion, as groups of parameters. We derive an

edge activation test using a structured group-ℓ_1 learning objective formulation.

The edge search space is stored in a min-heap priority queue (van Emde Boas et al., 1976) where edges are assigned different search priorities. We perform online edge activation using a control sample of candidate edges which we refer to as a limited-memory reservoir. Search priorities are updated based on the search history and structural information derived from the partially constructed MRFs' structure learned so far. The framework provides high scalability since it allows on-demand computation of sufficient statistics tables for edges that are likely to be activated. This avoids a heavy computational overload in the pre-learning phase. It is also possible to control a trade-off between speed of convergence and quality of learned MRFs. These attributes lead to a significant computational speedup for structure learning of pairwise MRFs.

Our experiments show that *online edge grafting* scales better to large datasets than the edge activation version of *grafting*. Synthetic experiments show that *online edge grafting* performs well at recovering the true structure, while evaluating our method on real data shows that we obtain high quality learned MRFs on large and diverse datasets. We also show how to control a trade-off between speed of convergence and quality of learned MRFs.

3.1 Parameter Learning Through ℓ_1-Regularized Likelihood

Popular approaches of MRFs structure learning are based on optimizing a likelihood-based objective function. Given a set of data $X = \{x^{(m)}\}_{m=1\ldots N}$, the likelihood is expressed as follows:

$$l(w) = \frac{1}{N} \prod_{m=1}^{N} p_w(x^{(m)}) \tag{3.1}$$

A high likelihood $l(w)$ reflects the model's ability to represent and predict data.

Negative Log Likelihood

The likelihood function is converted to the logarithmic space which provides further convenience for exponential based models and avoids problems of

numerical precision loss. Therefore, we formulate the learning objective as a negative log likelihood $L(w)$:

$$\min_{w} L(w) \tag{3.2}$$

where,

$$\begin{aligned} L(w) &= -\frac{1}{N}\sum_{m=1}^{N} \log p_w(x^{(m)}) \\ &= -\frac{1}{N}\sum_{m=1}^{N} \left(w^\top f(x^{(m)})\right) + \log Z(w) \end{aligned} \tag{3.3}$$

and w and f correspond to the vectors of w_k and f_k respectively.

Minimizing $L(w)$ is often done using gradient based methods.

The gradient of $\log Z(W)$ with respect to the k^{th} feature is the model's expectation of the k^{th} feature function (Koller and Friedman, 2009):

$$\frac{\partial \log Z}{\partial w_k} = E_w[f_k(x)]$$

where, $E_w[f_k(x)] = \sum_x p_w(x) f_k(x)$.

Therefore, the gradient of L with respect to the k^{th} feature is

$$\begin{aligned} \frac{\partial L}{\partial w_k} &= -\frac{1}{N}\sum_{m=1}^{N} f_k(x^{(m)}) + E_w[f_k(x)] \\ &= E_w[f_k(x)] - E_D[f_k(x)] \end{aligned} \tag{3.4}$$

In other words, the feature-wise gradient $\delta_k L$ is the error between the data expectation E_D and the model expectation E_w of $f_k(x)$. The goal of leaning can be seen as minimizing that error.

ℓ_1-Regularization

To avoid over-fitting and to promote sparsity of the learned weights and hence of the learned Markov network, classic methods add an ℓ_1 regularization to the objective function:

$$\mathbb{L}(w) = L(w) + \lambda ||w||_1 \tag{3.5}$$

Learning challenges

Despite the clarity of its expression, the computation of the gradient comes with major complications that result in its high cost:

- Data expectation requires the computation of sufficient statistics for every unary and pairwise feature which requires non-negligible computations, especially when it must be done on extremely large datasets.
- Classic ℓ_1 formulation ignores the structure of MRFs and treats parameters as if generated using the same distribution. This flaw results in dismissing the importance of local consistencies and the intrinsic identity of each variable and each edge.
- Inference, which is generally #P-complete (Koller and Friedman, 2009), has to be made with every gradient update.

The third limitation has been the main focus of current techniques. In the next section, we present current inference approximation methods. Next, we present active set methods that further address the computational overload of inference which remains high even with approximation methods.

3.1.1 Inference through belief propagation

Inference is expensive due to the intractability of the partition faction in the denominator term of the probability factorization. For tree like graphs, inference can be accurately performed using fairly inexpensive message passing (Pearl, 2014). These techniques thrive in sparse graphs and converge to optimal marginals. Unfortunately, for cyclic graphs, these algorithms do not guarantee convergence to exact marginals. Approximation algorithms such as Gibbs sampling, variational methods and loopy belief propagation were introduced (Wainwright and Jordan, 2008; Ihler et al., 2005; Murphy et al., 1999). Loopy belief propagation is a widely popular approach due to its simplicity and efficiency.

Loopy Belief Propagation This technique can be thought of as propagating information between variables along the Markov network edges. For

each edge (x_i,x_j), a message $m_{i\to j}(x_j)$ is initialized as $m_{i\to j}(x_j) = 1$. Then, all messages are updated as follows:

$$m_{i\to j}(x_j) = \sum_{x_i} \phi_{ij}(x;w)\phi_i(x;w)\phi_j(x;w) \prod_{x_k \in N_{x_i}\setminus x_j} m_{x_k\to x_i}(x_i)$$

Message updates converge when no relevant change occurs in the message update. We define N_i as the set of variables that share an edge with variable x_i in G. At the end of message passing, the pairwise probabilities are computed as

$$\begin{aligned} p_w(x_i,x_j) =& \frac{1}{Z_{ij}}\phi_{ij}(x;w)\phi_i(x;w)\phi_j(x;w) \\ & \prod_{k\in N_i\setminus x_j} m_{x_k\to x_i}(x_i) \prod_{x_k\in N_j\setminus x_i} m_{x_k\to x_j}(x_j) \end{aligned} \tag{3.6}$$

where

$$Z_{ij} = \sum_{x_i,x_j} \phi_{ij}(x;w)\phi_i(x;w)\phi_j(x;w) \prod_{x_k\in N_i\setminus x_j} m_{k\to x_i}(x_i) \prod_{x_k\in N_j\setminus x_i} m_{x_k\to x_j}(x_j)$$

We can also derive the model's pairwise probability for nodes that are not connected as

$$\begin{aligned} \hat{p}_w(x_i,x_j) =& \frac{1}{\hat{Z}_{ij}} \sum_{x_i,x_j} \phi_i(x;w)\phi_j(x;w) \\ & \prod_{x_k\in N_i\setminus x_j} m_{k\to x_i}(x_i) \prod_{x_k\in N_j\setminus x_i} m_{x_k\to x_j}(x_j) \end{aligned} \tag{3.7}$$

where

$$\hat{Z}_{ij} = \sum_{x_i,x_j} \phi_i(x;w)\phi_j(x;w) \prod_{k_i\in N_i} m_{k_i\to i}(x_i) \prod_{k_j\in N_j} m_{k_j\to j}(x_j)$$

3.2 Current Active-Set Methods

To further address the expense of inference, active-set methods were introduced. These methods avoid performing inference over the entire feature

space in each optimization step and instead perform it on an incrementally grown active set. *Grafting* (Perkins et al., 2003; Lee et al., 2006) and *grafting light* (Zhu et al., 2010), which we present below, are state-of-the-art active set methods for pairwise MRFs structure learning.

3.2.1 Grafting

This method is an application of the active-set approach, alternating between two steps:
Step 1: Optimizing over the active set S using a sub-gradient method.
Step 2: Expanding the active set by activating one feature using a gradient test 3.9.

Step 2 is based on the KKT optimality condition (Zhang et al., 2014) of problem (3.5)

$$\begin{cases} \delta_k L = 0 \text{ if } w_k \neq 0 \\ |\delta_k L| \leq \lambda \text{ if } w_k = 0 \end{cases} \tag{3.8}$$

The selection step is performed by selecting the weight that has the maximum violation of the optimality condition. Hence, we select the j^{th} feature using the following selection criteria C_1 that is derived from the optimality condition 3.8:

$$C_1 : j = \arg\max_k |\delta_k L| \wedge |\delta_k L| > \lambda \tag{3.9}$$

This method converges when Step 2 does not activate any new feature or when a predefined maximum number of features $f_{\max}$ is reached.

Algorithm 5 Grafting

1: Compute sufficient statistics of $f \ \forall f \in F$ # cost: $O(n^2 N s_{\max}^2)$
2: **repeat**
3: Select the top violating feature f^* using C_1. # cost: $O(n^2 s_{\max}^2)$
4: Activate f^*
5: Optimize the ℓ_1-regularized L over the active set.
6: **until** convergence

Grafting is summarized in Algorithm 5. Line 3 is a major bottelneck as the algorithm needs to compute $O(n^2 N s_{\max}^2)$ feature sufficient statistics, for

a system of n variable and $s_{\max}$ maximum number of states. In addition, *grafting* performs inference every time a feature is activated along with an activation operation with an $O(n^2 s_{\max}^2)$ complexity.

These computational bottlenecks translate to a poor scalability for large systems and datasets.

3.2.2 Grafting-light

A modified version called *grafting light* was introduced in (Zhu et al., 2010) as a faster extension of *grafting*. This method aims to reduce the computational cost of optimization by running one orthant-wise gradient step each time a new feature is selected. Although this method produces a faster optimization step, it does not solve the time complexity of the activation test and is more likely to produce a high number of irrelevant active features.

Furthermore, it has been shown by the authors that the method suffers limitations. First, the method is very likely to introduce irrelevant features which can diverge the solution to a bad local minima. In addition, the introduction of irrelevant features can introduce an important slow down of the method and might introduce a stronger risk of over-fitting.

3.2.3 Limitations

Although grafting and grafting-light are fairly suitable to learning MRFs, they do not consider structural information and reason about fully connected graphs in the activation step. Features are treated as one homogeneous group while they belong to distinct variables and edges. One possible outcome of *grafting*, or *grafting light*, is to activate a subset of features of an edge and keep the remaining ones at zero. This defeats the purpose of using an MRFs model which is based on the hypothesis that variables structurally influence each other.

Furthermore, current methods are batch learning methods that require processing the entire dataset to extract sufficient statistics tables which is a costly procedure and a bottleneck for large scale datasets. Grafting methods require computing the sufficient statistics for each feature in the search space before even activating the first feature. This requires $O(n^2 N s_{\max}^2)$ for a dataset with n variables, N instances and a maximum number of states per variable $s_{\max}$.

We address these issues in the proposed method.

3.3 Online Edge Grafting

We propose a method that grafts edges as opposed to features, in an online fashion using a variation of reservoir sampling (Olken and Rotem, 1995). This method is particularly useful for large-scale systems as it does not require computing all possible sufficient statistics tables in advance but instead does it on demand within the learning loop. This enables the fast learning of MRFs with a quality that improves with each added edge.

Furthermore, we reorganize the search space by assigning search priorities to edges based on search history and structure information derived from the partially constructed MRFs graph. The goal is to prioritize testing edges that are more likely to be relevant.

3.3.1 Grafting edges

Our approach injects the structural information into the objective function. In contrast with current grafting methods, *online edge grafting* activates edges as opposed to features. Hence, we redefine the search and active sets F and S respectively as the set of inactive and active edges.

To include structural information in the likelihood function, we use the group-ℓ_1 formulation. In doing so, the learned parameters of each node and each edge tend to be homogeneous.

We define the group-ℓ_1 negative log likelihood as follows:

$$\mathbb{L}(w) = L(w) + \sum_{g \in G} \lambda d_g ||w_g||_2 + \lambda_2 ||w||_2^2 \ . \tag{3.10}$$

A group g can either refer to a node or an edge. Consequently, w_g refers to the sub-vector containing all weights related to features of group g. We define d_g as the number of states per clique.

As in elastic net methods (Zou and Hastie, 2005), we add the ℓ_2-norm to avoid the short comings of the group-ℓ_1 regularization such as parameter unbalance and aggressive group selection.

We derive a similar KKT optimality condition to group regression (Liu and Zhang) as follows:

$$\begin{cases} \frac{||\delta_g L||_2}{d_g} + \lambda_2 ||w_g||_2^2 = 0 \text{ if } ||w_g||_2 \neq 0 \\ \frac{||\delta_g L||_2}{d_g} \leq \lambda \text{ if } ||w_g||_2 = 0 \end{cases} \tag{3.11}$$

where $\delta_g L$ is the sub vector constructed using the entries of the gradient vector L corresponding to group g.

Once we define the optimality violation criteria C_2 we utilize it as an edge activation test for a given edge $e \in F$.

$$C_2 : s_e > \lambda \tag{3.12}$$

here, s_e is the activation score for edge e and is defined as

$$s_e = \frac{||\delta_e L||_2}{d_e}. \tag{3.13}$$

Conveniently, we can derive an activation score for an inactive edge e using (3.7) as

$$s_e = \frac{||\hat{p}_w(e) - p_D(e)||_2}{d_e} \ . \tag{3.14}$$

We summarize the initial *edge grafting* method in Algorithm 6.

Algorithm 6 Edge Grafting

```
Define EdgeNum as the maximum allowed number of edges to graft
Initialize E = ∅ and F = set of all possible edges
Compute sufficient statistics of e ∀e ∈ F                # cost: O(n^2 N s_max^2)
AddedEges = 0
Continue = True
while EdgeNum > AddedEges and Continue do
   Compute score s_e ∀e ∈ F using (3.14)                 # cost: O(n^2 s_max^2)
   e* = arg max_{s∈F} s_e.                               # cost: O(n^2)
   if s_{e*} > λ then
      E = E ∪ {e*}; F = F\{e*}
      AddedEges = AddedEges +1
      Perform optimization over new active set
   else
      Continue = False
   end if
end while
```

3.3.2 Online edge grafting using reservoir sampling

Edge grafting results in fewer activation, and optimization, iterations since it activates groups of parameters as opposed to *grafting* methods. However,

it still suffers major bottlenecks, mainly in lines 3, 7 and 8 in Algorithm 6, where there is a quadratic complexity in n.

To address this issue, we derive *online edge grafting*. This new formulation activates edges without paying the initial $O(n^2 N s_{\max}^2)$ necessary for the computation of all edge sufficient statistics tables. Instead, this approach starts activating edges by computing a small fraction of the edge sufficient statistics tables. A naive approach would be to activate the first encountered edge that satisfies (C_2). However, this can introduce a large number of spurious edges as shown in the experiments section where we refer to it as *first hit* method. We propose an adaptive approach that uses a control sample of edges satisfying C_2, by collecting them online with a limited memory reservoir R. Figure 3.2 presents a high level description of the online edge activation mechanism, which we discuss in detail below.

Reservoir management

We define $A \in F$ as the unknown set of all edges that satisfy C_2 at an activation iteration. We construct a control sample $R \subseteq A$ from which we activate edges using a confidence interval.

In the first activation iteration, the search set F is stored in a min-heap priority queue (van Emde Boas et al., 1976) with equal priorities. Then, we parse through the priority queue by extracting one edge at a time. This is similar to receiving a stream of edges to test. If a tested edge satisfies C_2, it gets added to R. Otherwise, it gets ignored. In the next paragraph, we discuss how we manage failing edges as part of the search space re-organization mechanism.

When a maximum number of edge tests $t_{\max}$ is reached, we start activating edges. To guarantee a high quality reservoir, edge activation only starts after we fill R to capacity in the first edge activation iteration. To decide what edges must be activated, we first estimate the mean of candidate edge scores:

$$\mu = \frac{1}{|R|} \sum_{e \in R} s_e \ . \tag{3.15}$$

We can then define a confidence interval for choosing edges that are likely to be relevant as:

$$I_\alpha = \left[\mu + \alpha(\max_{e \in R} s_e - \mu), \max_{e \in R} s_e\right] \tag{3.16}$$

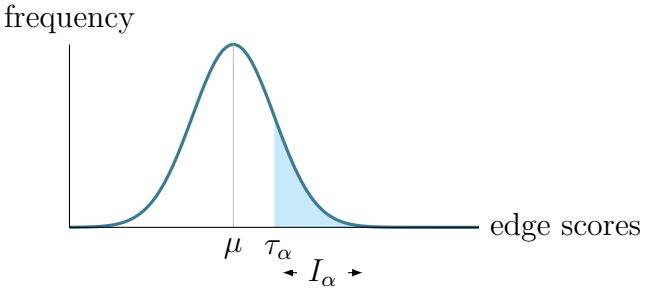
where $\alpha \in [0,1]$.

Hence, we define an activation threshold τ_α as shown in equation (3.17).

$$\tau_\alpha = (1-\alpha)\mu + \alpha \max_{e \in R} s_e \tag{3.17}$$

Figure 3.1 is a clarifying example of a confidence interval.

Figure 3.1: Example of a score confidence interval for edge selection. Area in light blue represent the α-parametrized confidence interval.



Note that when $\alpha = 1$, we only select the maximum scoring edge. Reducing α increases the number of edges to be activated at a certain step but can also result in adding spurious edges. In our experiments, we show the impact of different values of α on the quality of the learned MRFs and convergence speed.

After deriving τ, edges are activated in a decreasing order with respect to their scores. To avoid redundant edges and to promote scale-free structure, we only activate edges that are not adjacent as shown in Algorithm 8 in line 27.

After each optimization step, edge gradients change. Therefore, scores of reservoir edges are updated, and edges that no longer satisfy C_2 are dropped from the reservoir.

If R reaches its capacity before $t_{\max}$ is reached, we replace the minimum scoring edge in R, $R_{\min}$, by the newly tested edge whenever it has a higher score. This operation increases the quality of R.

Search space reorganization

Search space reorganization is performed by assigning and updating search priority of edges in the priority queue *pq*. The benefit of such an approach is

to increase the chance of testing relevant edges and hence increase the quality of the reservoir. We leverage search history information as well as structural information.

Search history At a given activation iteration, an edge with a small priority is considered to have a low chance of satisfying C_2 in the future and is hence placed further towards the tail of the priority queue. This gives way to edges with higher priorities to get tested first. Therefore, we define an edge violation offset v_e as

$$v_e = 1 - \frac{s_e}{\lambda} \ . \tag{3.18}$$

Ignored edges and edges dropped from R are not immediately returned to the priority queue but instead placed, along with their violation offsets, in a separate container L. We refer to such edges as "frozen" edges. When the search priority queue is emptied, we refill it by re-injecting frozen edges from L with their respective violation offset as their new priorities as shown in Figure 3.2.

Partial structure information As the active set grows, so does the underlying MRFs graph G. The resulting partial structure contains rich information about the dependencies between variables which enables leveraging additional information to the learning process.

We rely on the hypothesis that graphs of real networks have a scale-free structure as discussed in Chapter 1. We hence promote such structure in the learned MRF graph. The strategy is based on detecting hub and non-hub nodes in the partially constructed graphs. As the graph grows, the contrast between hubs and non-hubs increases as hub nodes acquire more incident edges than non-hub nodes.

To promote a scale-free structure, we prioritize testing edges that are incident to hub nodes. We start by measuring node centrality on the partially constructed MRF graph G to detect hub and non-hub nodes in the search space. A degree-based node centrality c_i for a node i is the fraction of its neighbor nodes N_i with respect to the entire set of nodes $|V|$ and is computed as:

$$c_i = \frac{|N_i|}{|V| - 1}$$

We then, use a centrality threshold $\hat{c}$ to detect the set of hubs H:

$$H = \{i \in V \text{ such that } c_i > \hat{c}\} \tag{3.19}$$

Finally, we prioritize all edges that are incident to nodes in the set H.

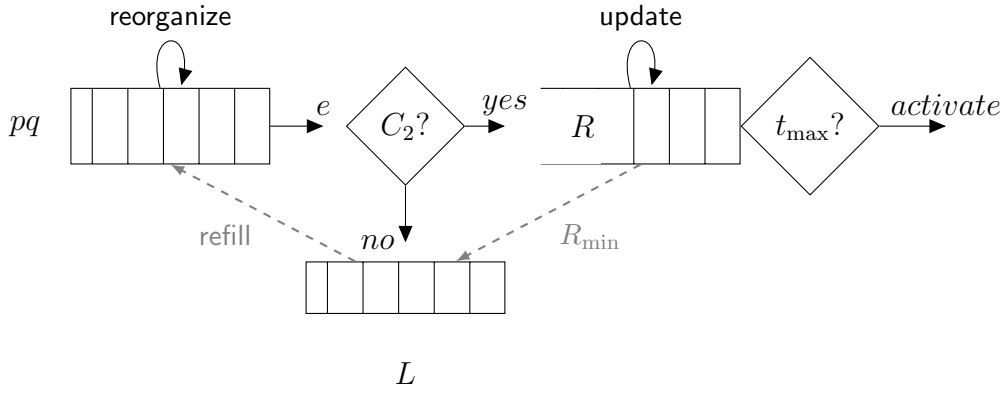
We summarize these steps in Algorithm 7. The total cost of updating the priority queue is $O(|H|n\log(n))$, where $\log(n)$ is the cost of updating the priority of an edge in the min-heap structure.

Algorithm 7 Reorganize PQ

```
Compute centrality measures over partially constructed MRFs graph G
Construct set H using (3.19)                          # cost: O(n)
for h ∈ H do
    for n ∈ V do
        pq[(h,n)] = pq[(h,n)] − 1          # total loop cost: O(|H|n log(n))
    end for
end for
```

Reorganizing the priority queue pushes edges that have a higher chance of being relevant to the front of the priority queue. This means that we obtain higher-quality reservoir and hence activate higher-quality edges at each activation iteration.

Figure 3.2: High-level operational scheme of the edge activation mechanism. From left to right, the first structure is the priority queue (pq) initiated with the set of all possible edges, The next diamond shaped box presents the activation test C_2 after which an edge is either added to the reservoir (R) or to the frozen edges container L. The $t_{\max}$ box describes reaching the maximum number of edge tests after which edges are activated. The gray dashed line on the right ($R_{\min}$) indicates the injection of the minimum scoring edge in R into L, when R is full. The gray dashed line on the left ('refill') indicates refilling the priority queue when it is emptied with the frozen edges.



We present the activation mechanism in detail in Algorithm 8.

Algorithm 8 Activation Test

Reorganize pq using Algorithm 7
$E^* = \emptyset$
$t = 0$
repeat
  if pq is empty **then**
    if R is empty **then**
      Break
    end if
    Refill pq using L
  end if
  Extract edge e with highest priority from pq
  if Sufficient statistics of e not already computed **then**
    Compute sufficient stats of e # cost: $O(Ns^2_{\max})$
  end if
  Compute score s_e. # cost: $O(s^2_{\max})$
  if $s_e > \lambda$ and R not full **then**
    Add e to R # add e if capacity not reached
  else if $s_e > \lambda$ and (R is full and $R_{\min} < s_e$) **then**
    Replace $R_{\min}$ by e. # $R_{\min}$: minimum scroing edge in R
    Place $R_{\min}$ in L.
  else
    Place e in L
  end if
until $t = t_{\max}$
Compute τ
for $e \in R$ s.t $s_e \geq \tau$ **do**
  if e not adjacent to edges in E^* **then**
    $E^* \cup \{e\}$
  end if
end for

3.3.3 Algorithm summary and complexity analysis

We finally have all the pieces to construct the main framework as described in Algorithm 9.

Algorithm 9 Online Edge Grafting

1: Define EdgeNum as the maximum allowed number of edges to graft
2: Initialize $E = \emptyset$, $F =$ set all possible edges
3: Fill pq with all elements of F and assign equal priorities
4: AddedEges = 0
5: Fill R to capacity
6: **while** EdgeNum > AddedEges **do**
7: Get set E^* of edges to activate using Algorithm 8
8: **if** E^* is empty **then**
9: Break
10: **end if**
11: **for** e^* in E^* **do**
12: $E = E \cup \{e^*\}$; $F = F \backslash \{e^*\}$
13: AddedEges = AddedEges +1
14: **end for**
15: Perform an optimization over active parameters.
16: **end while**

If it takes the algorithm r edge tests to fill the reservoir in one pass, then Algorithm 9 performs $O(rNs^2_{\max})$ operations to compute the sufficient statistics necessary to fill the reservoir and activate the first edges. To activate the j^{th} edge, the algorithm needs to perform at most $O\big((r + jt_{\max})Ns^2_{\max}\big)$ where $t_{\max}$ is the allowed number of edge tests between two activation steps. The computational complexity decreases with $\alpha < 1$ since more than one edge is activated per activation iterations.

In the usual case, we assume that it takes the algorithm $r = O(|R|)$ tests to fill the reservoir R. Furthermore, to construct a relevant reservoir we set $|R| = O(n)$. We also set $t_{\max} \ll n$.

This means that to activate the j^{th} edge the algorithm needs to compute at most $O\big((n + jt_{\max})Ns^2_{\max}\big)$ sufficent statistics tables. In the case of *edge grafting*, the algorithm needs to compute $O(n^2Ns^2_{\max})$ to start grafting the first edge. Since $(n + jt_{\max}) \ll n^2$, our method provides a drastic speed up, as we get rid of the quadratic term in n and replace it with a linear term.

It is worth noting that the pq initialization step (line 3 in Algorithm 9) has a complexity of $O(n^2 \log(n))$ for *online edge grafting*. Better initialization of the search space can avoid this cost. We do not approach this issue in the current version of the algorithm as this cost only occurs once at the initialization stage.

Different algorithms complexities are summarized in Table 3.1.

Table 3.1: Time complexity of different methods

Algorithm	Suff. stats. at j^{th} edge	Activation step
Feature grafting	$O\big(n^2 N s_{\max}^2\big)$	$O\big(n^2 s_{\max}^2\big)$
Edge grafting	$O\big(n^2 N s_{\max}^2\big)$	$O\big(n^2 s_{\max}^2\big)$
Online edge grafting	$O\big((n + j t_{\max}) N s_{\max}^2\big)$	$O\big(t_{\max} s_{\max}^2\big)$

3.4 Experiments

We perform experiments with both synthetic and real data on a Linux machine having 16 GB of RAM and a 3 GHz processor. To evaluate the scalability of our method, we use datasets of varying variable dimensions and dataset sizes. In the synthetic setting, we simulate MRFs with scale-free structures, and we generate data using a Gibbs sampler. In the real setting, we use three diverse datasets to see how well our framework performs in practice. We define *edge grafting* as the baseline. Experiments using *grafting* (classic feature based method) proved to be extremely slow and hence would not fit well with edge grafting based methods. We derive a naive approach called *first hit* which consists of activating the first encountered edge that satisfies C_2.

Before getting into the details of the performed experiments we first define the metrics that we used for evaluation.

3.4.1 Metrics

As a structure recovery metric, we use recall, defined as

$$\text{recall} = \frac{\text{number of recovered true edges}}{\text{total number of true edges}} \ .$$

In our experiments, we monitor recall over time. The slope of recall against time is an indicator of precision as a high slope indicates that a method adds relevant edges and a small slope indicates the activation of several false-positive edges.

We are also interested in evaluating how well the learned MRFs represent the true data distribution. Since computing the likelihood is intractable, we measure the pseudo-likelihood (Gourieroux et al., 1984), replacing the likelihood by a more tractable objective. Experiments show that the learning objective and the testing negative log pseudo-likelihood exhibit similar trends.

The pseudo-likelihood formulation is based on the following approximation:

$$p(x) = \prod_i p(x_i | x \setminus \{x_i\}) = \prod_i p(x_i | N_i) \tag{3.20}$$

where N_i is the set of variables in the Markov blanket (direct neighbors in the Markov network) of x_i. Since each variable has a limited number of neighbors, this yields a drastic gain in running time complexity compared with the true likelihood formulation. In fact, taking the above approximation into consideration we get:

$$p(x) = \prod_i \frac{\exp\left(\phi_i(x_i) + \sum_{j \in N_i} \phi_{ij}(x_j, x_i)\right)}{\sum_{\hat{x}_i} \exp\left(\phi_i(\hat{x}_i) + \sum_{j \in N_i} \phi_{ij}(x_j, \hat{x}_i)\right)} \tag{3.21}$$

For M different data points $x^{(m)}$, the negative log pseudo-likelihood (*nlpl*) is hence given by:

$$\begin{aligned} \text{NPLL} =& \frac{1}{M} \sum_m \left(\log \prod_i p(x_i^{(m)} | N_i) \right) \\ =& \frac{1}{M} \sum_m \sum_i \log p(x_i^{(m)} | N_i) \\ =& \frac{1}{M} \sum_m \sum_i \left[\phi_i(x_i) + \sum_{j \in N_i} \phi_{ij}(x_j, x_i) \right. \\ & \left. - \log \sum_{\hat{x}_i} \exp\left(\phi_i(\hat{x}_i) + \sum_{j \in N_i} \phi(x_j, \hat{x}_i) \right) \right] \end{aligned} \tag{3.22}$$

3.4.2 Synthetic data

We generate structures with varying sizes: 50 nodes (30,875 parameters), 100 nodes (124,250 parameters), 150 nodes (280,125 parameters), and 200

nodes (498,500 parameters). For each case, we construct random scale-free-strcutured MRFs with variables having 5 states each. We use the Barabasi-Albert preferential attachment model (Albert and Barabási, 2002). Following the model described in Section 1.1.2, we grow a graph by attaching new nodes each with 2 edges that are preferentially attached to existing nodes with high node centrality. We then use Gibbs sampler (Bishop, 2006) to generate a set of 20,000 data points, which we randomly split as train and held-out test data. We use grid search to detect optimal learning parameters. The task is to recover the correct structure and parameters and to ensure that we obtain a learned MRFs that has a quality close to the one learned using *edge grafting*. In our experiments, we fix $|R| = n$, and $t_{\max} \ll n$. We would like to see the evolution of the quality of learned MRFs in time and also investigate the impact of α on the learning curve.

We first investigate the behavior of different methods without early stopping, *i.e.*, learning only stops when no violating edge remains in the search space. We are interested in validating that different methods lead to the same solution. This is indeed the case in Figure 3.3, as all methods lead to the same minimum value of the objective function but with a faster convergence rate for *online edge grafting*.

Figure 3.4 shows an interesting trend produced by experiments, where we fix the number of maximum edges as shown below each graph. For increasing values of α we get better recall for different MRFs sizes. However, this comes at the cost of a higher convergence time. In fact, for smaller values of α, *online edge grafting* tends to activate more edges but with lower quality, which introduces a higher number of false-positive edges. We also observe that, for increasing number of variables, the learning gap between *online edge grafting* and classical *edge grafting* (EG) increases, which demonstrates a better scalability for the online algorithm.

Figure 3.5 exhibits similar trends. Lower values of α give faster convergence time. However, the difference in the qualitative quality is less contrasted. In fact, there is not an important quality gap in the evolution of the learning objective for different values of α. This can be explained by the fact that the model can fit a different structure that is approximate of the true one.

Experiments on testing data presented in Figure 3.6 show that there is a positive correlation between the learning objective and the test negative log pseudo-likelihood, which confirms that the learned models do not over-fit the data even with small values of α. This result suggests that higher values of α may be used to recover a good structure. However, if the sole goal of learning

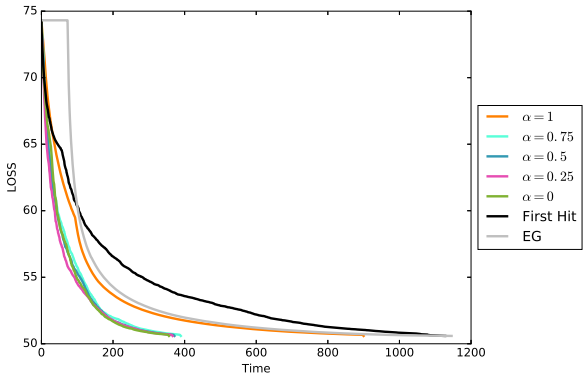
the MRF structure is to produce a good generative model, then lower values of α can be used to speed-up learning with a small loss of quality.

As expected, *first hit* reaches the maximum number of added edges faster. However, this results in a poor qualitative and quantitative behavior as seen in Figure 3.4 and Figure 3.5 since it adds bad quality edges. This is also verified in Figure 3.3 where this method has the worst convergence rate.

These experiments also confirm our theoretical hypothesis on the computational bottleneck that classical *edge grafting* suffers when computing all the possible sufficient statistics as a pre-learning step.

To evaluate the effectiveness of structure heuristics, we generated Figure 3.7. The graphs show that we obtain lower quality structure recovery when we do not apply the proposed heuristics which validates their role in prioritizing relevant edges. This produces higher quality edge stream generated from the priority queue, resulting in higher quality reservoir and hence higher quality of activated edges.

Figure 3.3: Time (s) evolution of objective functions during full convergence of different methods (50 nodes).

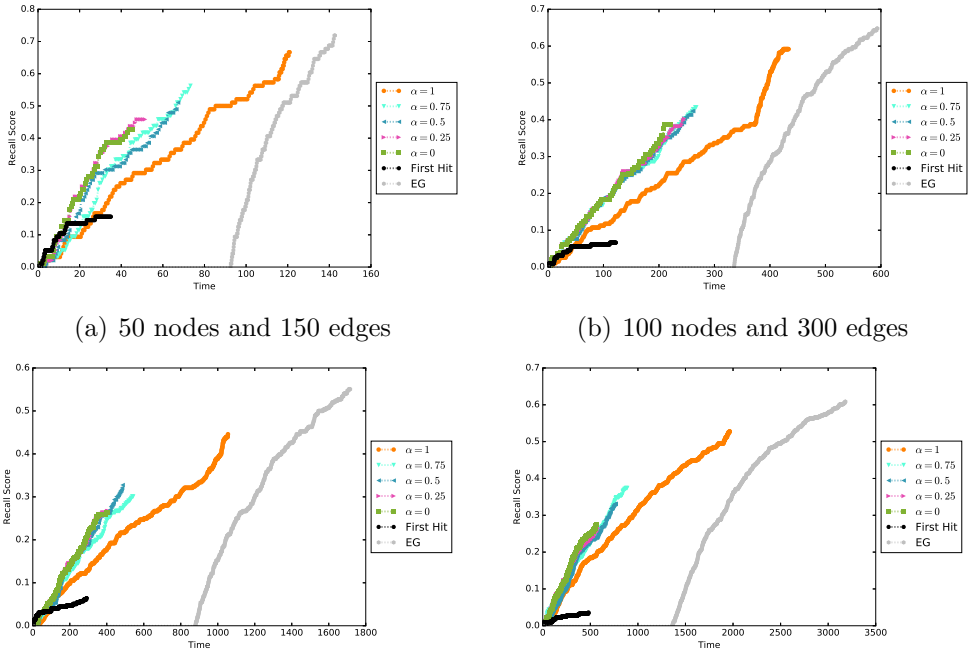


3.4.3 Real data

We use three datasets: Jester,[1] Yummly recipes,[2] and USDA plants database[3] (Hämäläinen and Nykänen, 2008). The Jester dataset is a joke ratings dataset

[1]http://goldberg.berkeley.edu/jester-data/
[2]https://www.kaggle.com/c/whats-cooking
[3]https://archive.ics.uci.edu/ml/datasets/Plants

Figure 3.4: Recall vs time (s) for varying MRFs sizes with an O(n) number of edges.



(a) 50 nodes and 150 edges

(b) 100 nodes and 300 edges

(c) 150 nodes and 450 edges

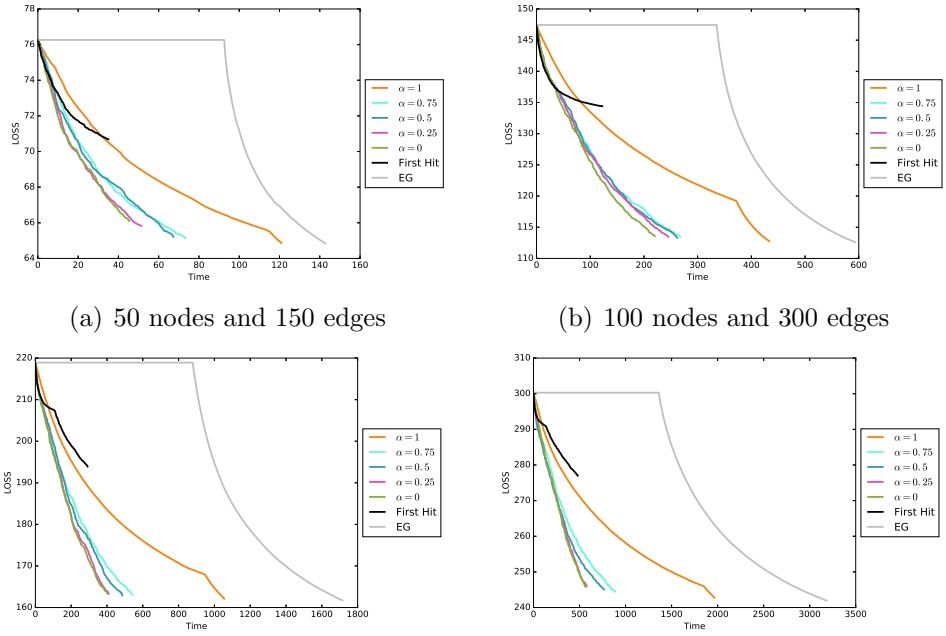
(d) 200 nodes and 600 edges

containing 100 jokes and 73,421 user ratings. We use jokes as variables and user ratings as instances. We map the ratings interval from a continuous interval in $[0, 20]$ to a discrete interval from 1 to 5. This results in 124,250 parameters. We sample 10,000 recipes from Yummly dataset containing recipes with a total of 153 ingredients. Each ingredient is treated as a binary variable and recipes are considered as data instances. This results in 36,450 parameters. The plants dataset comprises 34,781 plants with 68 different locations in the world. We treat plants as instances and locations as binary variables. This produces 9,248 parameters.

Our aim with these experiments is to assess how the proposed framework performs on different and diverse real datasets. As in the synthetic setting, we monitor the learned MRFs over time and evaluate them using their learning objectives and their negative log pseudo-likelihood.

Figures 3.8, 3.9, and 3.10 show a positive correlation between the evolution of the learning objective and the testing negative log pseudo-likelihood. We

Figure 3.5: Loss vs time (s) for varying MRFs sizes with an O(n) number of edges.



(a) 50 nodes and 150 edges

(b) 100 nodes and 300 edges

(c) 150 nodes and 450 edges

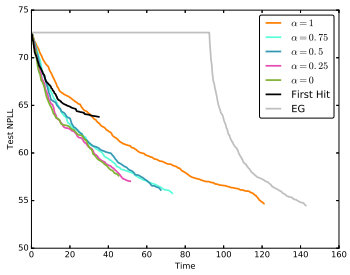
(d) 200 nodes and and 600 edges

also see similar results to the ones obtained with synthetic experiments where *online edge grafting* converges faster than *edge grafting*, and smaller values of α result in a faster convergence. In datasets where there are low dependencies between variables, such as in the case of USDA plants dataset, the difference in the quality for different values of α becomes more contrasted. Hence, it is recommended to set a high value of α. *First hit* displays a behavior similar to that observed in the synthetic setting, as it converges quicly but produces low quality MRFs compared to the rest of the methods. We also observe a higher scalability of *online edge grafting* as the gab in convergence time between the latter and *edge grafting* gets larger with larger datasets.

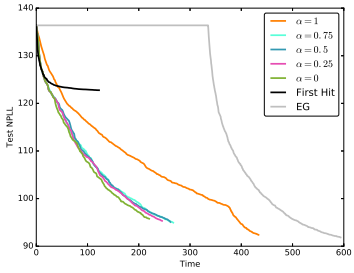
Conclusion

In this chapter, we presented *online edge grafting*, a method based on a group lasso formulation and online reservoir sampling. This method is derived from

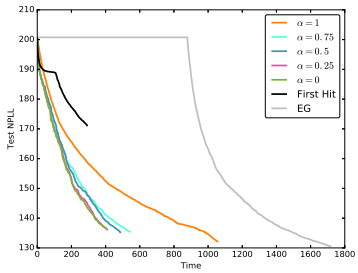
Figure 3.6: Negative log pseudo-likelihood vs time (s) for varying MRFs sizes with an O(n) number of edges.



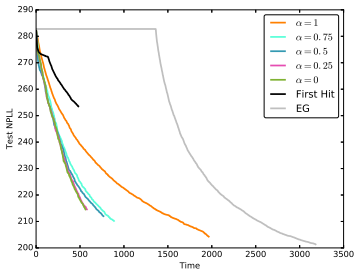
(a) 50 nodes and 150 edges



(b) 100 nodes and 300 edges



(c) 150 nodes and 450 edges

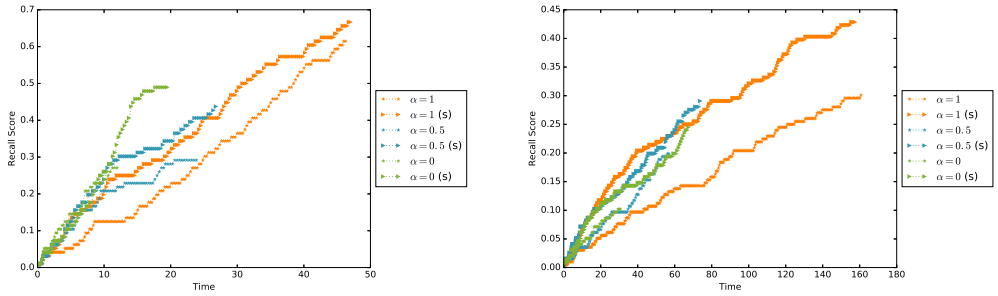


(d) 200 nodes and 600 edges

an initially developed method, *edge grafting*, that is based on activating edges as opposed to features as in the case of the classical *feature grafting* method. Theoretical analysis shows that the method provides higher scalability than batch edge grafting as it avoids its major bottlenecks. Experiments on both synthetic and real data confirm our theoretical analysis. They demonstrate a clear advantage of applying *online edge grafting* since it provides higher scalability and faster convergence on multiple datasets while producing similar performance for both structure recovery and predictive ability.

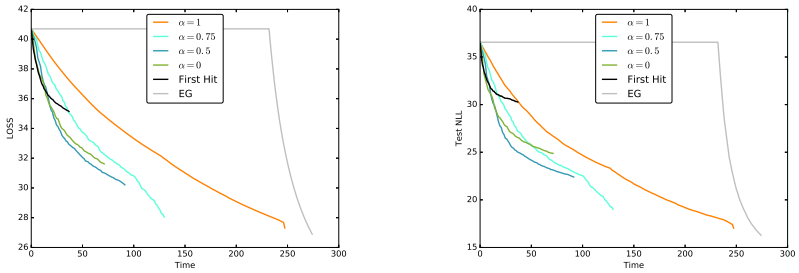
Further work can investigate better structure heuristics for a more efficient search space reorganization. This should enable an even faster convergence and higher quality MRFs.

Figure 3.7: Role of structure heuristics in improving the quality of the learned MRF over training time (s). The proposed heuristics (labeled as 's') produce higher recall for different MRFs sizes (50 and 100 nodes)



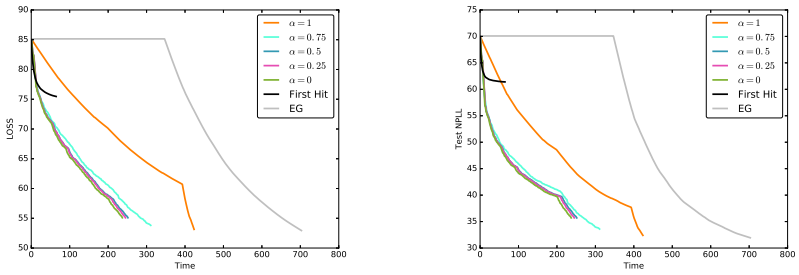
(a) 50 nodes (b) 100 nodes

Figure 3.8: Learning objective and negative log pseudo-likelihood vs time (s) with 200 edges (USDA plants).



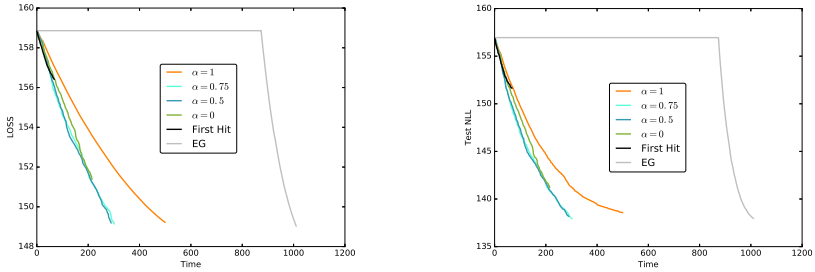
(a) Learning objective (b) NPLL

Figure 3.9: Learning objective and negative log pseudo-likelihood vs time (s) with 400 edges (Yummly).



(a) Learning objective (b) NPLL

Figure 3.10: Learning objective and negative log pseudo-likelihood vs time (s) with 250 edges (Jester).



(a) Learning objective (b) NPLL

Conclusion

In this work, we investigated the extent of scalability for current ℓ_1-learning methods in the case of linear dynamical systems and Markov random fields. We identified computational bottlenecks and then proposed new approaches that took advantage of problem reformulation and efficient structure heuristics.

In Chapter 2, we proposed *scalable structure learning*, a general framework for learning dynamical systems. The framework includes various parameters that allow significant savings in computational cost and provides high sparsity scalability. The learning procedure first estimates the structure of the dynamical system by reasoning about a set of seed nodes. It does this by learning the local structure of the seed nodes using observed data, estimating their immediate and two-hop parents. Using the learned structure information of the seed subset, we infer the parents structures for the second subset using low-cost and effective heuristics. In addition, our algorithm can be implemented in parallel, enabling it to scale to large network settings. Experiments with synthetic data showed that our method has high tolerance for noise and is able to effectively recover and estimate the coefficients of linear dynamical networks. In the real data experiments, the algorithm gave convincing results, by effectively modeling traffic activity between blocks of NYC using a seed subset containing only 20% of the nodes. Investigating ways to strategically choose seed nodes could bring benefits on the quality of the inferred structure and result in requiring fewer seeds.

In Chapter 3, we proposed *online edge grafting*, a method that is based on group lasso formulation and online reservoir sampling of edges. Theoretical analysis shows that the method provides higher scalability than batch *edge grafting* as it avoids its major bottlenecks while producing similar performance for both structure recovery and predictive ability. Experiments on both synthetic and real data confirm our theoretical analysis as we see a clear advantage of applying *online edge grafting* since it provides higher scalability

and faster convergence on multiple datasets. Further work can investigate better structure heuristics for a more efficient search space reorganization. This should enable an even faster convergence and higher quality of learned MRFs.

Both applications share the same high-level strategy: we reformulated each problem with emphasis on the structure of each system. In the case of LDS, we treated the state space matrix as an adjacency matrix which allowed us to reason about variable neighborhoods and create a rich source of information. In the case of MRFs, we developed a structured group-ℓ_1 regularized objective that allowed us to derive an online active-set method that reasons about edges. Furthermore, we exploited rich structure information derived from the partially constructed graph and developed efficient heuristics that infer the rest of the graph in the case of LDS and speed up the learning process in the case of MRFs. Finally, both frameworks avoid costly batch ℓ_1-learning on the entire problem space as ℓ_1-learning is only conducted on a smaller subspace, *i.e.*, seed nodes in the case of LDS and high priority edges in the case of MRFs. This strategy enabled us to build faster learning frameworks that scale better to large datasets while producing high quality models.

Bibliography

Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

Galen Andrew and Jianfeng Gao. Scalable training of L1-regularized log-linear models. In *Proceedings of the 24th International Conference on Machine Learning*, pages 33–40. ACM, 2007.

Cevdet Aykanat, Fusun Ozguner, Fikret Ercal, and Ponnuswamy Sadayappan. Iterative algorithms for solution of large sparse systems of linear equations on hypercubes. *IEEE Transactions on computers*, 37(12):1554–1568, 1988.

Christopher M. Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, 415(1):20–30, 2006.

Rama Chellappa and Anil Jain. Markov random fields. theory and application. *Boston: Academic Press, 1993, edited by Chellappa, Rama; Jain, Anil*, 1, 1993.

Andrew Cotter, Ohad Shamir, Nati Srebro, and Karthik Sridharan. Better mini-batch algorithms via accelerated gradient methods. In *Advances in Neural Information Processing Systems*, pages 1647–1655, 2011.

Shane F. Cotter, Bhaskar D. Rao, Kjersti Engan, and Kenneth Kreutz-Delgado. Sparse solutions to linear inverse problems with multiple measurement vectors. *IEEE Transactions on Signal Processing*, 53(7):2477–2488, 2005.

Katrien De Cock. Subspace identification methods.

John E. Dennis, Jr. and Jorge J. Moré. Quasi-newton methods, motivation and theory. *SIAM Review*, 19(1):46–89, 1977.

David L. Donoho and Michael Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via L1 minimization. *Proceedings of the National Academy of Sciences*, 100(5):2197–2202, 2003.

Noah E Friedkin and Eugene C Johnsen. Social influence networks and opinion change. *Advances in group processes*, 16(1):1–29, 1999.

Zoubin Ghahramani and Geoffrey E Hinton. Parameter estimation for linear dynamical systems. Technical report, Technical Report CRG-TR-96-2, University of Totronto, Dept. of Computer Science, 1996.

Gene H. Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420, 1970.

Christian Gourieroux, Alain Monfort, and Alain Trognon. Pseudo maximum likelihood methods: Theory. *Econometrica: Journal of the Econometric Society*, pages 681–700, 1984.

Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A. Zadeh. *Feature extraction: foundations and applications*, volume 207. Springer, 2008.

Wilhelmiina Hämäläinen and Matti Nykänen. Efficient discovery of statistically significant association rules. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 203–212. IEEE, 2008.

Alexander T. Ihler, W. Fisher John III, and Alan S. Willsky. Loopy belief propagation: Convergence and effects of message errors. *Journal of Machine Learning Research*, 6(May):905–936, 2005.

Leopoldo Jetto, Sauro Longhi, and Giuseppe Venturini. Development and experimental validation of an adaptive extended kalman filter for the localization of mobile robots. *IEEE Transactions on Robotics and Automation*, 15(2):219–229, 1999.

Rudolf Emil Kalman. Mathematical description of linear dynamical systems. *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, 1(2):152–192, 1963.

Tohru Katayama. *Subspace methods for system identification*. Springer Science & Business Media, 2006.

Carl T. Kelley. *Iterative Methods for Optimization*. SIAM, 1999.

Hyunsoo Kim and Haesun Park. Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. *SIAM Journal on Matrix Analysis and Applications*, 30(2):713–730, 2008.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.

Charles L. Lawson and Richard J. Hanson. *Solving least squares problems*, volume 15. SIAM, 1995.

Su-In Lee, Varun Ganapathi, and Daphne Koller. Efficient structure learning of markov networks using L1-regularization. In *Advances in neural Information processing systems*, pages 817–824, 2006.

Stan Z. Li. *Markov random field modeling in image analysis*. Springer Science & Business Media, 2009.

Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

Han Liu and Jian Zhang. Estimation consistency of the group lasso and its applications.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(Jan):19–60, 2010.

Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

Andrew Y. Ng. Feature selection, L1 vs. L2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

Frank Olken and Doron Rotem. Random sampling from databases: a survey. *Statistics and Computing*, 5(1):25–42, 1995.

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.

Simon Perkins, Kevin Lacker, and James Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *The Journal of Machine Learning Research*, 3:1333–1356, 2003.

Howard Harry Rosenbrock. Structural properties of linear dynamical systems. *International Journal of Control*, 20(2):191–202, 1974.

Havard Rue and Leonhard Held. *Gaussian Markov random fields: Theory and Applications*. CRC Press, 2005.

Samitha Samaranayake, Sébastien Blandin, and Alexandre Bayen. Learning the dependency structure of highway networks for traffic forecast. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 5983–5988. IEEE, 2011.

Mark Schmidt. *Graphical model structure learning using L1-regularization*. PhD thesis, University of British Columbia, 2010.

Mark Schmidt, Glenn Fung, and Rómer Rosales. Fast optimization methods for L1 regularization: A comparative study and two new approaches. In *Machine Learning: ECML 2007*, pages 286–297. Springer, 2007.

Peter van Emde Boas, Robert Kaas, and Erik Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10(1): 99–127, 1976.

Peter Van Overschee and B.L. De Moor. *Subspace identification for linear systems: Theory Implementation Applications*. Springer Science & Business Media, 2012.

Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

Chaohui Wang, Nikos Komodakis, and Nikos Paragios. Markov random field modeling, inference & learning in computer vision & image understanding: A survey. *Computer Vision and Image Understanding*, 117(11):1610–1627, 2013.

Jianke Zhang, Sanyang Liu, Lifeng Li, and Quanxi Feng. The KKT optimality conditions in a class of generalized convex optimization problems with an interval-valued objective function. *Optimization Letters*, 8(2):607–631, 2014.

Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23 (4):550–560, 1997.

Jun Zhu, Ni Lao, and Eric P. Xing. Grafting-light: fast, incremental feature selection and structure learning of Markov random fields. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 303–312. ACM, 2010.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.