Interactive Exploration of Comparative Dependency Network Learning

Diane Oyen
Los Alamos National Lab
Los Alamos, New Mexico, USA
doyen@lanl.gov

Terran Lane
Google, Inc
Cambridge, MA, USA
terran@cs.unm.edu

ABSTRACT

Comparative dependency network learning is a growing field of research, especially in systems biology. Domain scientists would like to discover patterns of variable dependency that are conserved across conditions or discover pathways that are disrupted due to disease. In machine learning, multitask graphical structure learning algorithms have been developed to help solve this problem by learning network models from multiple related datasets. These algorithms typically have regularization hyper-parameters that have the effect of reducing the number of spurious edges learned and the number of spurious differences learned. We propose a mechanism to allow the end-user to control these regularization hyper-parameters in real-time to interactively explore the huge space of potential dependency network solutions. This is a critical element of a visualization system that enables domain scientists to discover interesting patterns in multivariate data. Yet, this is a computationally challenging endeavor as complex models must be learned in real-time and, additionally the number of differences learned in each network and the number of differences between them must be translated by the machine learning algorithm into the correct change in the setting of the hyper-parameters. This paper introduces a general framework for interactively exploring the similarities and differences among a set of dependency networks and demonstrates our work-in-progress on a specific implementation for multiple Bayesian networks.

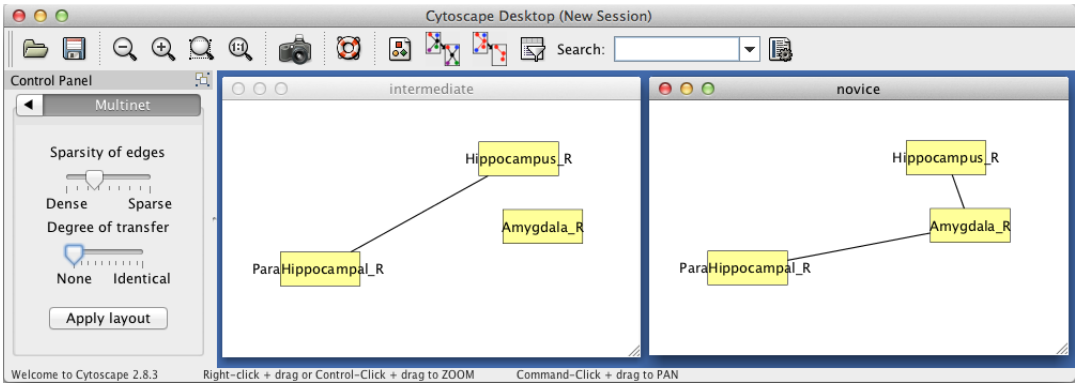
1. INTRODUCTION

Probabilistic graphical models encode patterns of dependency among variables in multivariate data [11]. Attention is turning to the problem of comparative network analysis; that is, identifying dependencies that are conserved or different among related sets of data. In machine learning, multitask graph learning algorithms have been developed to address this problem [17, 4]. Multiple graphs are learned simultaneously, producing models that are similar except where the data strongly supports differences, easing comparison (see example in Figure 1). The results of these learning algorithms are multiple graphical models, requiring visualization software to help the user understand the results.

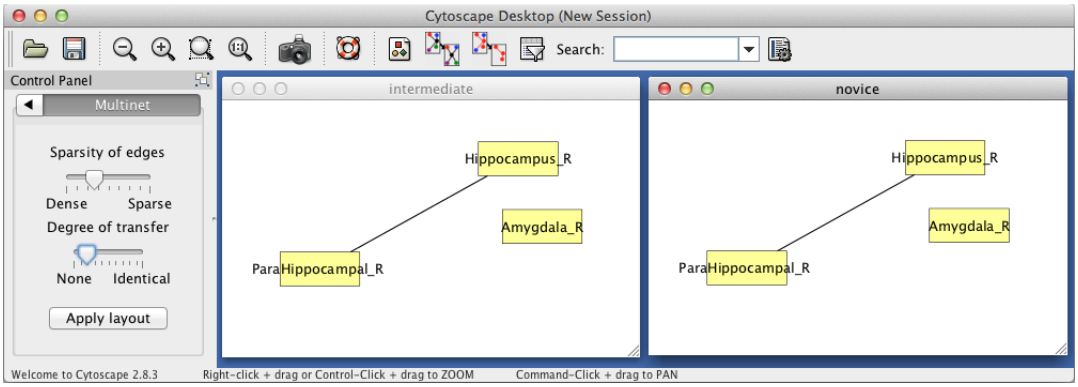
Multitask graph structure learning is a promising direction for knowledge discovery in many scientific domains [25, 16, 5, 19]. However, there remain issues of practical concern; namely, the exploration of the solution space for different settings of hyper-parameters. The solution space includes many graph structures that fit the data nearly equally well, but the learned solutions vary based on the choice of hyper-parameters given to the learning algorithm. Multitask graph structure learning algorithms typically have two hyper-parameters, one that affects the number of edges learned (sparsity) and the other that controls the strength of transfer bias (how similar the graphs will be to each other). Machine learning typically treats these hyper-parameters as nuisance parameters that must be tuned to learn an optimal model [14, 13, 24, 15, 17]. Yet, to the end user, all learned models — no matter the choice of hyper-parameters — are good fits to the data. There is a natural tradeoff between sensitivity and specificity in machine learning algorithms governed by the hyper-parameters. Domain scientists would like to be able to vary the level of confidence in learned models to see which are the highest confidence edges and/or differences to explore all potential dependency patterns in the data.

Naively, exploration of the solution space is achieved by performing a grid search over hyper-parameter settings and then presenting each of these learned graphs to the end user. Yet, this grid search is not an ideal approach. To illustrate the problem of grid search, we take a look at example results from a neuroimaging study. Using an existing multitask graph learning algorithm [4], we assign the sparsity hyper-parameter to 10 values evenly spaced in the range $[0.1, 1]$ and we assign the transfer hyper-parameter to 11 values evenly spaced in the range $[0, 1]$. All 110 combinations of sparsity and transfer settings were run through a multitask algorithm and the solutions were displayed to a domain expert who found which solution was most interesting. Taking a broader look at the results from all 110 settings of these parameters, Figure 2a summarizes the number of edges learned in the networks for all of the values in the grid. We can see from this that if the sparsity setting is too high, then no edges are learned in the graphs. Yet, if the sparsity parameter is too small, then all variables are dependent on each other and this gives little new information to the end-user. Figure 2b shows the number of edges that are different between the two learned networks. We see that for many settings of transfer and sparsity there are many solutions that are uninteresting because there are no differences. Another frustration with this grid search is that the number of edges or differences learned does not change linearly with evenly-spaced steps in parameter space. Tuning the hyper-parameters over a coarse grid like this could easily miss the optimal hyper-parameter setting.

After noticing that the most interesting results are located within a narrow range of hyper-parameter settings, we can re-run the multitask network learning algorithm for



(a) Independently learned graphs



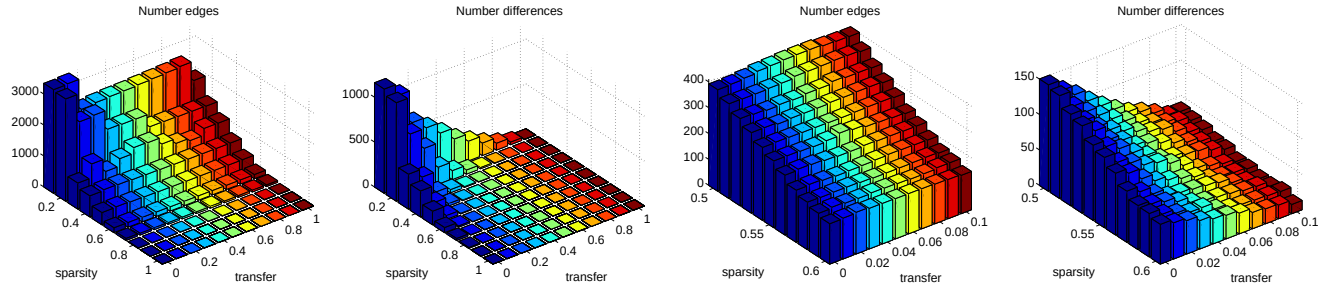
(b) Graphs learned with some transfer

Figure 1: Example of a sub-graph learned from neuroimaging data. The nodes in the graph are regions of the brain. Edges indicate a direct dependent relationship in functional activity as modeled by a multinomial distribution; i.e. an excitatory or inhibitory pathway between brain regions. When the graphs are learned independently (a), the connections are different. If this were the only result given to the domain scientist, we might conclude that the Amygdala has a regulating effect on the pathway between the Hippocampus and ParaHippocampus in one group of subjects, but not in the other group. However, with even a little bit of transfer bias encouraging them to be similar (b), the differences disappear, suggesting low confidence that this difference is real. This is a small sub-graph of a much larger graph. Higher confidence differences in the larger graph could still remain at this low value of the transfer hyper-parameter.

new values of hyper-parameters. As an example, we "zoom in" on the range $[0.5, 0.6]$ for the sparsity parameter and the range $[0, 0.1]$ for the transfer parameter. The algorithm is run with another 100 combinations of values for hyper-parameters evenly spaced in this new range. Figure 2c gives the number of individual edges learned in each networks while Figure 2d displays the number of differences between the two networks. Here we see that the numbers of edges and differences learned change smoothly in this local region of hyper-parameter space.

The above example demonstrates key limitations of existing exploratory approaches in comparative dependency networks, which we address with our interactive approach. First, instead of changing the hyper-parameters, we must think about how the results will change. In this case, the end user thinks about seeing more or fewer edges or differences (assuming that fewer edges are higher confidence edges, etc). Therefore, we need a computational model that translates the human desires (number of edges and differences) into the domain of the hyper-parameters of the machine learning algorithm. Second, the end user needs to be able to get fine-grained results in realtime to effectively explore the space of solutions that are of interest. Thus, an effective interactive exploration of dependencies must translate human desire into machine learning objectives and update in realtime in response to user feedback.

This paper introduces an interactive machine learning algorithm for multitask graphical models as part of ongoing research into creating an interactive data exploration visualization system (pictured in Figure 3). A single setting of the hyper-parameters does not give the full picture that domain scientists want to see. Therefore, we propose a graphical structure learning algorithm that allows the user to interactively adjust the number of edges and the number of differences learned between graphs. As the user makes selections about increasing/decreasing the number of edges or the number of differences between graphs, we estimate the necessary change in the hyper-parameter values and re-learn the networks, displaying the results and allowing further interaction. This approach gives the user an exploration of the solution space directly, rather than having to guess pairs of values for hyper-parameters. Essentially, we are giving the user the ability to explore fine-grained steps in the solution space, and making the appropriate steps in the hyper-



(a) Number of learned edges (b) Number of differences (c) Number of learned edges (d) Number of differences

Figure 2: Neuroimaging study: Summary statistics about learned network models for a course (a,b) and fine (c,d) grid of values of sparsity and transfer hyper-parameters.

parameters to achieve that result, rather than using a typical grid search in hyper-parameter space.

2. RELATED WORK

To interactively explore graphical models, we need to provide a means to adjust parameters of interest to the user and display the resulting graphs. Display of the graphs is handled through the Cytoscape software that is popular in bioinformatics [23]. The plugin interface allows us to customize the display for comparison of multiple graphs (see Figure 3). We incorporate sliders that allow a user to modify the sparsity and degree of transfer among networks. Previously, in our implementation, these sliders simply looked up the pre-computed graphs learned from a list of hyper-parameter values [20]. The user did not have any control over the granularity of the slider, and furthermore, changing a hyper-parameter value may not always have the desired effect (for example, on sparse graphs, even a small amount of transfer will cause the graphs to be identical). Therefore, we propose to provide more intuitive controls to the user, allowing them to change the number of edges or the number of similarities directly.

The idea of interactive parameter search is inspired by work in supervised learning models that show that with human interaction, the optimal parameter settings are found faster [1] and gives the user control over the objective function [9]. As in these papers, to achieve this interactive exploration in multitask graph structure learning, we must be able to estimate the values of hyper-parameters that will produce the desired change in the solution space. We achieve this by calculating the gradient of the solution with respect to the hyper-parameters and then taking a step in the direction of the gradient to produce a new solution that meets the requirements of the user.

In graph structure learning literature, much research has gone into optimizing the selection of the sparsity parameter [14, 13] without a clear resolution to the problem. Traditionally, the hyper-parameters are tuned through trial and error after examining the learned graphs [4] or through a computationally expensive grid search that optimizes with respect to holdout data [24, 15, 17, 19]. Graph structure learning is an unsupervised learning domain and so there may not be an optimal parameter setting. Even using the oracle value of hyper-parameters does not guarantee optimal performance [15], instead that paper recommends using known non-interactions to gauge the optimal level of sparsity. Selecting the ideal setting of transfer parameters has received less attention, with cross-validation being the preferred method [17, 19] and subjective human-selection being another choice [4]. Cross-validation selects the best model to match the empirical distribution; yet, distribution matching is not always the primary goal for using transfer learning, and therefore cross-validation will not give optimal results. Giving the user the ability to explore the solution space is even more important in unsupervised learning. The user may have desires about learned models that are not expressible until the learned models are seen [3]. Furthermore, allowing a user to give feedback about the solutions is more intuitive than asking the user to adjust hyper-parameters in the hopes that the adjustments will have the desired effect.

3. FRAMEWORK OF USER INTERACTION

We formalize the general problem of learning multiple graphical model structures and describe a method for including user input in response to learned models.

3.1 Problem Formulation

A graphical model is a joint probability distribution of a random vector $X = [x_1, x_2, \ldots, x_p]$ that can be represented compactly as factors of local structure, $P(X) = \prod_{i=1}^{p} f(x_i, ne(x_i))$, where the set of neighbors of each node, $ne(x_i)$, is some subset of variables. The elements of vector $X = [x_1, x_2, \ldots, x_p]$ are random variables represented in the graphical model as vertices (or nodes) as the set V. If $x_p \in ne(x_q)$, then there is said to be a direct dependency between these two variables which is represented with an edge e_{pq}. The set of all edges is called E. In many cases, the graph structure itself $G = \{V, E\}$ is of particular interest.

In the problem of multitask graph structure learning, we have several sets of data, D_k for $k \in \{1, 2, \ldots, K\}$, from which we learn several graphs $\mathcal{G} = \{G_1, \ldots, G_K\}$. The multitask structure learning algorithm relies on two hyper-parameters, which we call $\Lambda = [\lambda_1, \lambda_2]$, where generally $0 < \lambda_1 \leq 1$ controls the sparsity and $0 \leq \lambda_2 \leq 1$ controls the strength of transfer. We treat the graph structures $\mathcal{G}$ and Λ as unknowns to be learned. For a fixed Λ, the graphs can be learned from the data with existing algorithms. The user will interactively learn Λ by giving feedback to the learning algorithm about the number of edges and edge similarities that they would like to see in the learned graphs.

In this paper, we represent the set of edges in all of the K graphs with $\mathbf{G}$, an $m \times K$ binary matrix, where m is the total

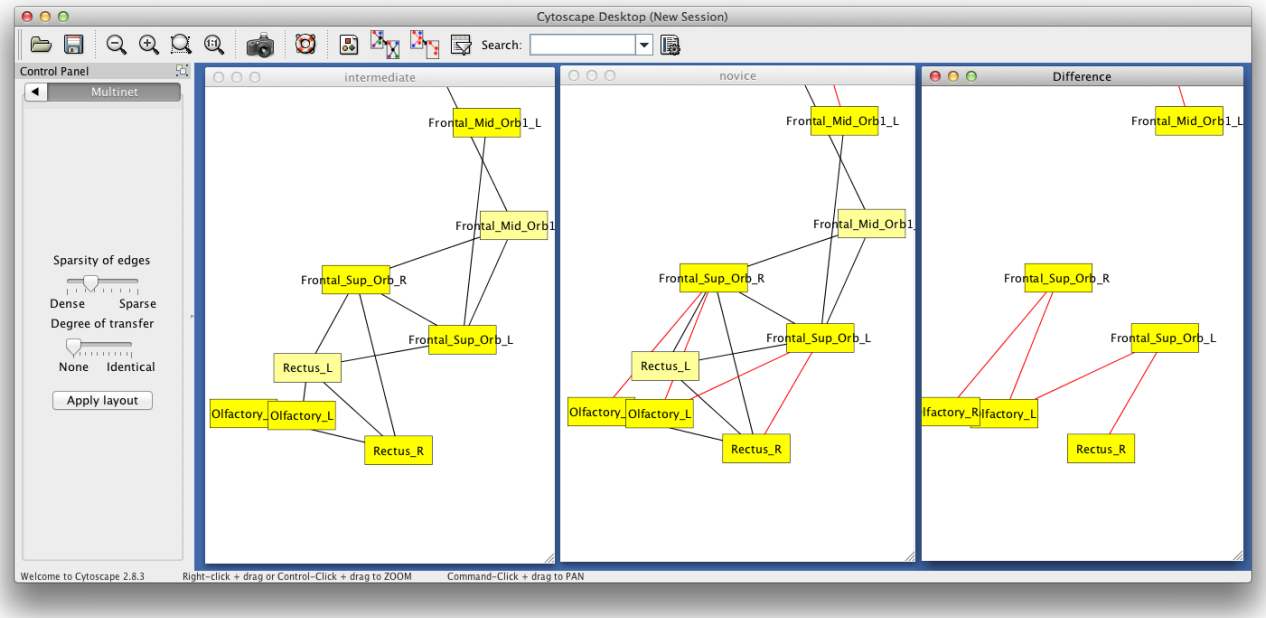


Figure 3: Interactive multi-graph visualization. Our system consists of the following components: a visual display of multiple learned graphs, user controls to increase/decrease the number of edges in each graph, user controls to increase/decrease the degree of similarity among pairs of graphs, efficient update of learned graphs in response to user controls.

number of potential edges (for directed models $m = p(p-1)$ and for undirected models $m = p(p-1)/2$). Each entry G_{ik} represents the presence ($G_{ik} = 1$) or absence ($G_{ik} = 0$) of the edge i in task k. In a slight abuse of notation, we use G_{ik} to refer to edge i in task k, while G_k refers to all potential edges in task k. The structure of the learned graphs depends on the training data and the hyper-parameters $\Lambda = [\lambda_1, \lambda_2]$. While looking at a given solution, a human end-user may desire to see a solution with more (or fewer) edges in some G_k or with more (or fewer) edge differences between some G_i and G_j for tasks i and j. These desires are encoded in a binary matrix $\mathbf{S}$ that correspond to the edge matrix $\mathbf{G}$ (explained in further detail later).

3.2 Sketch of Interactive Approach

Our interactive approach alternates between learning graph structure, $\mathcal{G} = f(\mathcal{D}, \Lambda)$, and learning hyper-parameters, $\Lambda = g(\mathcal{D}, \mathcal{G}, \mathcal{S})$, based on feedback $\mathcal{S}$ from a human who is looking at a visualization of the learned graphs $\mathcal{G}$. To initialize the interaction, we learn a set of graphs from given datasets. These graphs can be learned with transfer bias ($\lambda_2 = 0$) initially with an arbitrary value for the sparsity (e.g. $\lambda_1 = 0.5$). These graphs will be displayed in Cytoscape, along with information in the Control Panel about the number of edges learned in each graph and the number of differences in edges among the tasks. The user can then adjust the desired number of edges learned (up or down) or adjust the number of differences among pairs of tasks. Based on the user input, $\mathbf{S}$, we compute the necessary Λ to achieve the requested change (details in the next section). Using the computed Λ, we relearn the graphs, $\mathcal{G}$, and update the visualization, allowing the user to further interact until satisfied with the solution.

3.3 Representation of User Feedback

When a user clicks to change the number of edges in a graph or the number of differences among graphs, the user is not directly changing the hyper-parameters. We must translate the feedback into an appropriate change in the hyper-parameters to produce the desired outcome. To represent user preferences, we use a binary matrix, $\mathbf{S}$, the same size as $\mathbf{G}$ ($m \times K$). Each entry, S_{ik}, indicates the user's desire to see the presence or absence of edge i in task k. Using this input, we can move the learned structures $\mathbf{G}$ in the direction of the user preferences $\mathbf{S}$ by finding an appropriate adjustment to Λ.

An example illustrates how the user representation works. Consider the case where a user wishes to see fewer differences between tasks a and b. Let the currently existing set of non-zero edges in graph a be A and the set of non-zero edges in graph b be B. Then the user feedback defines a set U of edges, any one of which could change to satisfy the user. If the user wishes to see fewer edges that exist in a but not b then the set difference $A \setminus B$ must get smaller. Therefore $U = A \setminus B$. We set $S_{ea} = 0 \ \forall e \in U$ and $S_{eb} = 1 \ \forall e \in U$. $\mathbf{S}$ encodes the user-preferences to see one of the specific edges to be added or removed. The remaining entries in $\mathbf{S}$ are set to the current values of $\mathbf{G}$, i.e. $S_{ek} = G_{ek} \ \forall e \notin U$ and $\forall k$.

Formally, the rules for representing user feedback depends on the action taken by the user. The rules are defined as:

- **Fewer edges in task i:**
  Assign $S_{ei} = 0 \quad \forall\, e = \{1, 2, \ldots, m\}$.
  Assign $S_{ek} = G_{ek} \quad \forall\, e = \{1, 2, \ldots, m\}$ and $\forall\, k \neq i$.
- **More edges in task i:**
  Assign $S_{ei} = 1 \quad \forall\, e = \{1, 2, \ldots, m\}$.
  Assign $S_{ek} = G_{ek} \quad \forall\, e = \{1, 2, \ldots, m\}$ and $\forall\, k \neq i$.

- **Fewer edges in task i that are not in task j:**
  Define set $U = \{e = \{1, 2, \ldots, m\} \mid G_{ei} = 1 \wedge G_{ej} = 0\}$.
  Assign $S_{ek} = 0 \quad \forall\, e \in U$ and $k = i$.
  Assign $S_{ek} = G_{ek}$ otherwise.

- **More edges in task i that are not in task j:**
  Define set $U = \{e = \{1, 2, \ldots, m\} \mid G_{ei} = 0 \wedge G_{ej} = 0\}$.
  Assign $S_{ek} = 1 \quad \forall\, e \in U$ and $k = i$.
  Assign $S_{ek} = G_{ek}$ otherwise.

3.4 Local Move Toward User Desires

The goal is to obtain a setting for $\Lambda = [\lambda_1, \lambda_2]$ that creates graphs that are nearly the same as the current solution, but one edge closer to the user's desires **S**. Therefore, we define an objective function that measures the squared error between **S** and **G**:

$$g(\Lambda) = \sum_{k=1}^{K} \sum_{e \in E} (S_{ek} - G_{ek}(\Lambda))^2 \quad . \tag{1}$$

The user's feedback asks us to take just one step in the direction of this objective (only one edge is added or deleted at a time). We are not fully optimizing the objective. The gradient is given in Eq 2:

$$\begin{aligned} \nabla_\Lambda g &= -2 \sum_{k=1}^{K} \sum_{e \in E} (S_{ek} - G_{ek}(\Lambda)) \cdot \nabla_\Lambda G_{ek}(\Lambda) \\ &= -2 \cdot \mathbf{J}_\Lambda(\vec{G}) \cdot (\vec{S} - \vec{G}) \quad , \end{aligned} \tag{2}$$

where $\vec{S}$ and $\vec{G}$ are vectors formed by stacking the columns of the **S** and **G** matrices respectively. $\mathbf{J}_\Lambda(\vec{G})$ is the $2 \times |\vec{G}|$ Jacobian matrix, with each entry in the first row the partial derivative of G_{ek} with respect to λ_1 while the second row is with respect to λ_2. Our objective is to find the minimum step size η that gives the incremental change requested. Once η is found, the new value of the hyper-parameters is:

$$\Lambda^{\text{new}} = \Lambda - \eta \cdot \nabla_\Lambda g \quad . \tag{3}$$

The new hyper-parameter values are fed back into the learning algorithm, the visualized results are updated and the cycle continues if the user gives more feedback.

3.5 Computational Challenges

The above objective requires two computationally expensive steps. The first is the calculation of the Jacobian (the gradient $\nabla_\Lambda G_{ek}(\Lambda)$). The computational complexity of this depends on the specific model of multitask graph structure learning used. For the Bayesian discovery of multitask Bayesian networks format given in the next section, the partial derivative with respect to λ_1 (sparsity) is trivial, but the partial derivative with respect to λ_2 (the transfer strength) is computationally equivalent to calculating the multi-task family scores. Which is to say that it is exponential and could take minutes (depending on complexity-reducing approximations). However, we note that the gradient depends only on the current model and not user feedback. Therefore, the gradient can be calculated in the background while the user is looking at the previously learned graphs and making a choice about feedback to give.

The other computationally expensive procedure is the inference of $G(\Lambda)$ for each task. For the Bayesian discovery of multitask Bayesian networks approach given here, to update G the graphs must be re-learned (exponential time, or approximated with MCMC).

4. EXPLORATION OF MULTITASK BAYESIAN NETWORKS

Here we apply the above framework for interactive graph exploration to the specific problem of Bayesian discovery of multiple Bayesian networks, particularly those with transfer bias from related data. Then we discuss how to cache intermediate calculations to make updating the transfer bias faster on subsequent calculations. Finally, we show how discrete graphs are obtained from the expectations on edges.

4.1 Preliminaries

A Bayesian network is a directed acyclic graph that represents a joint probability distribution as $P(X) = \prod_{i=1}^{p} P(x_i|\pi_i)$, where π_i is the parent set of child i. That is, the value of x_i depends directly on the values of all $x_j \in \pi_i$. Bayesian structure discovery produces a posterior estimate of the expectation of each edge in a Bayesian network [7, 10]. For multitask Bayesian networks, there is a posterior estimate of the expectation of each edge in each task, which we organize into a matrix **W**, denoted $0 \leq w_{ek} \leq 1$. An edge is described by an indicator function $f_i(\pi_i)$ such that the edge $x_v \to x_i$ exists (and $f_i(\pi_i) = 1$) iff $x_v \in \pi_i$, otherwise $f_i(\pi_i) = 0$. The probability of the edge w_{ek} is therefore the expectation of f in task k for that edge. The expectation is calculated over all orderings, $\prec$, of the nodes in the Bayesian network, as in Equation 4. For a given ordering, the parents of a node i must precede i in the order.

$$w_{ek} = \sum_{\prec} P(\prec) \sum_{G \subseteq \prec} P(G|\prec) P(\mathcal{D}|G) f_e(\pi_e) \ , \tag{4}$$

where $G \subseteq \prec$ means that the graph structure G is consistent with the order $\prec$; and π_e is the parent set of node x_e in graph G.

Relatively efficient methods for exactly calculating each w_e for single-task learning exist [10]. The method breaks down into three steps:

1. Calculate the family scores from data. These are called the β functions, $\beta_i(\pi_i) = P(\pi_i)P(x_i|\pi_i)f_i(\pi_i)$. It is assumed that the computational complexity of each of these is some function $C(n)$ that depends on the number of samples n. The maximum number of parents allowed for any node is typically fixed to a small natural number, r. Therefore, there are $O(p^{r+1})$ of these functions to calculate for a total computational complexity of $O(p^{r+1}C(n))$.

2. Calculate the local contribution of each subset $U \subseteq V - \{i\}$ of potential parents of i. These are called the α functions, $\alpha_i(U) = \sum_{\pi_i \subseteq U} P(\pi_i)P(x_i|\pi_i)f_i(\pi_i)$. There are an exponential number of subsets U, therefore there are an exponential number of α functions. Using a truncated fast Möbius transform [2], all of the α functions can be computed in $O(p2^p)$ time, assuming that the β functions are pre-computed and that there is a limit, r, on the maximum size of the parent sets.

3. Sum over the subset lattice of the various U_i to obtain the sum over orders $\prec$. Although the number of orders is $p!$ there is no need to enumerate each order explicitly. The potential parents of each node i depend only on the set of parents U_i that precede it, not on the ordering of the parents within U_i. Using dynamic programming, this sum takes time $O(p2^p)$ [10].

The total computational complexity for a single task is $O(p2^p + p^{r+1}C(n))$. This is the exact calculation of the posterior. For large networks, roughly $p > 30$, the exponential term is intractable. In these cases, we can use MCMC to approximate the sum over orders [18]. To limit the computation of the polynomial term, we can choose a sufficiently small r or further reduce the number of potential families using candidate parent sets [8].

To learn multiple Bayesian networks simultaneously, we replace the single-task prior $P(\pi_i)$ with a transfer bias $P(\pi_i^{(k)}, \pi_i^{(j)})$ that shares information among tasks k and j [19]. The transfer bias penalizes the number of parents in $\pi_i^{(k)}$ that are not also in $\pi_i^{(j)}$ for all pairs of tasks (k, j). This transfer bias encourages similar graph structures to be learned for each task, and has been shown to produce more robust networks [17, 19]. In terms of the three-step method above [10], this means replacing the β functions with [19]:

$$\begin{aligned}\beta_{ki}(\pi_i, \lambda_2) =& f_i(\pi_i^{(k)})P(x_i^{(k)}|\pi_i^{(k)})P(\pi_i^{(k)}, \pi_i^{(j)}) \\ =& f_i(\pi_i^{(k)})P(x_i^{(k)}|\pi_i^{(k)}) \times \frac{1}{(K-1)(4-\lambda_2)^{|U_i|}} \times \\ & \left[\sum_{j\neq k}\sum_{\pi_i^{(j)}\subseteq U_i} P(x_i^{(j)}|\pi_i^{(j)})(1-\lambda_2)^{\Delta(\pi_i^{(k)},\pi_i^{(j)})}\right] ,\end{aligned} \tag{5}$$

where $\Delta(\pi_i^{(k)}, \pi_i^{(j)}) = |\pi_i^{(k)} \setminus \pi_i^{(j)}|$. We assume each β function takes time $C(n_k)$ to compute, where n_k is the number of samples in task k. Under transfer learning, there is a now a sum over parent sets for each task, therefore the computational complexity is $O(Kp^r)$ for each β function. There are Kp^{r+1} of these functions to calculate. This gives a total computational complexity for all multitask β functions of $O(K^2p^{2r+1})$. Note that for visualization and interactive purposes, the number of tasks is typically $K = 2$ for ease of end-user interpretation of the results.

Once the multitask β functions are calculated, the rest of the posterior estimate can be calculated using existing algorithms, such as exact expectation over orders [10, 21] or MCMC approximations [18].

4.2 Efficient Computation of Transfer Bias

We store intermediate calculations to speed up any future calculations with different values for λ_2. We achieve this by noting that the function Δ can only produce a finite number of integer values in the range $[0, r]$. By grouping the parents sets, we can re-arrange terms to group together the parent sets $\pi_i^{(j)}$ that will produce the same value in the Δ function.

$$\begin{aligned}&\sum_{\pi_i^{(j)}\subseteq U_i} P(x_i^{(j)}|\pi_i^{(j)})(1-\lambda_2)^{\Delta(\pi_i^{(k)},\pi_i^{(j)})} = \\ &\quad = \sum_{\delta=0}^{r}\sum_{\pi_i^{(j)}|\Delta(\pi_i^{(k)},\pi_i^{(j)})=\delta} P(x_i^{(j)}|\pi_i^{(j)})(1-\lambda_2)^{\delta} \\ &\quad = \sum_{\delta=0}^{r}(1-\lambda_2)^{\delta}\sum_{\pi_i^{(j)}|\Delta(\pi_i^{(k)},\pi_i^{(j)})=\delta} P(x_i^{(j)}|\pi_i^{(j)})\end{aligned} \tag{6}$$

By separating the sum over individual scores, we can store the sums and re-use them later if λ_2 changes. We define the γ functions as these sums:

$$\begin{aligned}\gamma_{ki\delta}(\pi_i, \delta) =& \sum_{j\neq k}\sum_{\pi_i^{(j)}|\Delta(\pi_i^{(k)},\pi_i^{(j)})=\delta} P(x_i^{(j)}|\pi_i^{(j)}) \ , \\ & \text{for all } \pi_i \subseteq V - \{i\},\ \delta \in \mathbb{Z},\ 0 \leq \delta \leq r \ .\end{aligned} \tag{7}$$

With a maximum parent set size r, the maximum value that δ can take is r. Therefore, the number of γ functions to be calculated are: Krp^{r+1}, one for every family in every task for every value of δ. The calculation of all of these γ functions is $O(K^2rp^{2r+1}C(n))$.

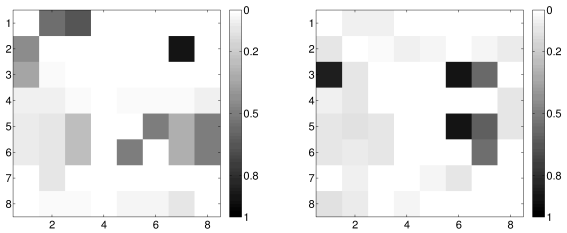
We rewrite the β functions using the pre-computed γ functions. Notice that the computational complexity of the β function is now linear in r. This means that the functions can be computed quickly for various values of λ_2.

$$\beta_{ki}(\pi_i, \lambda_2) = \frac{f_i(\pi_i^{(k)})P(x_i^{(k)}|\pi_i^{(k)})}{(K-1)(4-\lambda_2)^{|U_i|}} \cdot \sum_{\delta=0}^{r}(1-\lambda_2)^{\delta}\gamma_{ki\delta}(\pi_i, \delta) \tag{8}$$

These γ functions are also used in the calculation of the Jacobian.

4.3 Thresholding for graphs

The feature probabilities, w_{ek}, learned from Equation 4 can be organized into square matrices W_k for each task k representing the directed edges of a network. Figure 4 shows an example of these learned feature posterior probabilities.



(a) W_1 (b) W_2

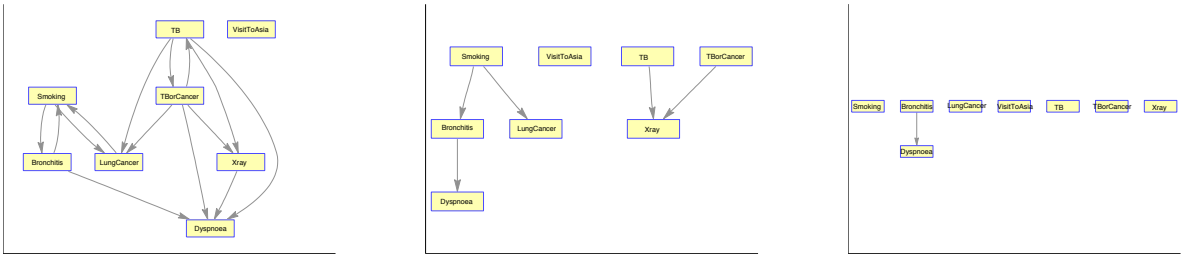
Figure 4: Estimated posterior likelihoods for two tasks with $\lambda_2 = 0$. There are 8 variables, and 8×7 possible directed edges, which are organized as a weighted adjacency matrix.

In order to display graphs to the user (see Figure 5), we threshold the w_{ek} values, showing only the edges with likelihoods greater than some cut-off value $0 \leq \lambda_1 \leq 1$. Clearly, λ_1 will control the density of edges in the displayed graphs. In this work, we employ a soft-threshold sigmoid function to define the learned graph:

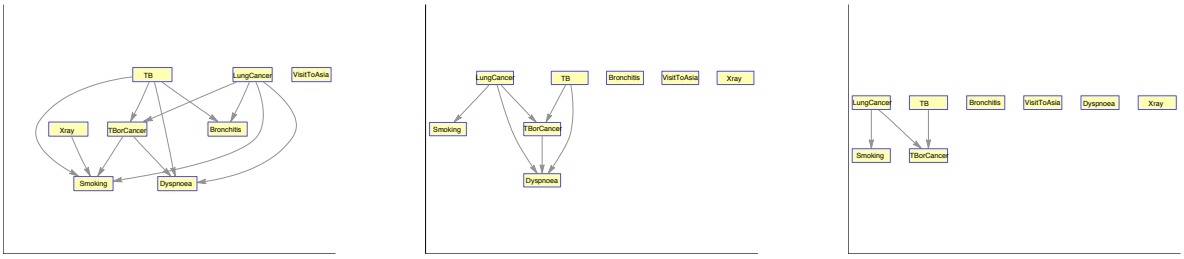
$$G_{ek} = \frac{1}{1 + \exp[-\beta(w_{ek} - \lambda_1)]} \ . \tag{9}$$

For sufficiently large values of $\beta > 1$ this is equivalent to a hard threshold at λ_1.

Mathematically, this thresholding of edge expectations is a loss of information. For comparative network analysis, it may seem desirable to keep all edges weighted by their expectation. However, from the end user perspective, nearly all graph visualization systems allow thresholding out the weakest edges to get a clearer picture of the network, and so we treat thresholding as a necessary step for visualization. When comparing the similarities and differences among a set of graphs, it is helpful to be able to control for the number of

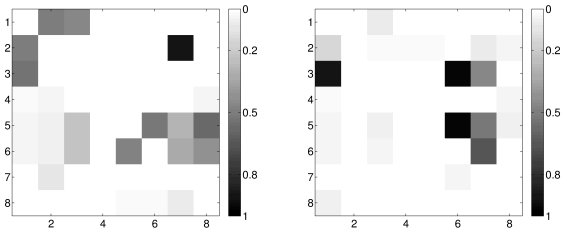


(a) G_1 for $\lambda_1 = 0.1$ (b) G_1 for $\lambda_1 = 0.5$ (c) G_1 for $\lambda_1 = 0.8$



(d) G_2 for $\lambda_1 = 0.1$ (e) G_2 for $\lambda_1 = 0.5$ (f) G_2 for $\lambda_1 = 0.8$

Figure 5: By thresholding at λ_1, we obtain graphs G_k from the weighted adjacency matrices W_k.



(a) W_1 (b) W_2

Figure 6: Estimated posterior likelihoods for two tasks with $\lambda_2 \neq 0$. There are 8 variables, and therefore 8×7 possible directed edges, which we have organized into a weighted adjacency matrix.

differences between the graphs. By encouraging the graphs to be similar, we can reduce the number of spurious differences learned, and display only differences that are most likely to be real (see Figures 6 and 7). The λ_2 parameter controls the amount of similarity bias which encourages the w_{ek} and w_{ej} values of tasks k and j to be close, as in Figure 6. This means that as the w values move closer together, some will cross the threshold, as in Figure 7, therefore, both parameters will have a noticeable effect on both the number of edges learned and the number of differences.

5. NUMERICAL ESTIMATION OF HYPER-PARAMETERS

The previous section showed how to learn multiple Bayesian networks given data and hyper-parameter settings. This section outlines our method for incorporating user preferences into the learning algorithm. Once feedback has been received from the user, the hyper-parameters $\Lambda(G, S)$ need to be updated. This is computationally expensive, and so we lay out a numerical estimation of Λ.

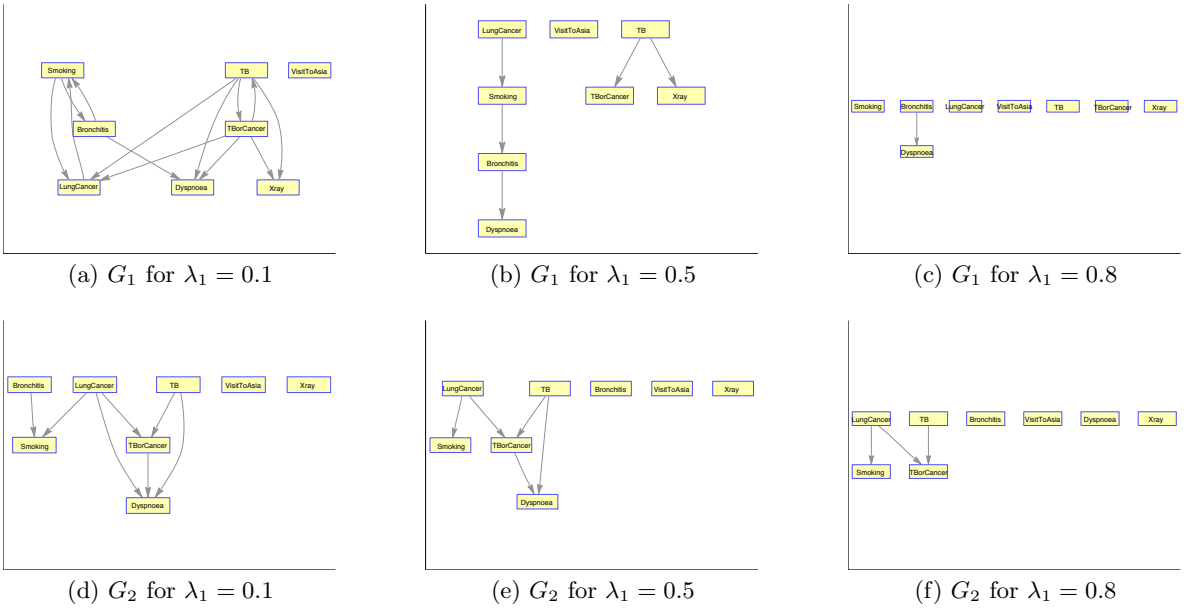
5.1 Estimation of Λ for Multitask Bayesian Networks

To re-learn graphs after getting feedback from the user, we take one step toward minimizing the distance between the learned graphs an the given user preferences, as in Equation 1. First, we need to calculate the Jacobian $\nabla_\Lambda G_{ek}(\Lambda)$ in Equation 2. For multitask Bayesian networks, the partial derivative with respect to λ_1 is fairly straightforward.

$$\begin{aligned}\frac{\partial}{\partial \lambda_1} G_{ek} &= \frac{-\beta e^{-\beta(w_{ek}-\lambda_1)}}{1+e^{-\beta(w_{ek}-\lambda_1)}} \\ &= -\beta e^{-\beta(w_{ek}-\lambda_1)} G_{ek}\end{aligned} \tag{10}$$

The partial derivative with respect to λ_2, on the other hand, is more complicated to calculate because the family scores within the sums depend on λ_2. Therefore, the partial derivative of each of these family scores must be computed, and the sums re-calculated.

$$\begin{aligned}&\frac{\partial}{\partial \lambda_2} G_{ek} = \\ &= -\beta e^{-\beta(w_{ek}(\lambda_2)-\lambda_1)} G_{ek} \sum_{\prec} \sum_{\pi_e^{(k)} \subseteq U_e} f_e(G_k) P(x_e^{(k)} | \pi_e^{(k)}) \times \\ &\left[\sum_{\pi_e^{(j)} \subseteq U_e} P(x_e^{(j)} | \pi_e^{(j)}) \frac{-\Delta(1-\lambda_2)^{\Delta-1} + |U_i|(1-\lambda_2)^{\Delta}(4-\lambda_2)^{-1}}{(4-\lambda_2)^2} \right] \\ &= -\beta e^{-\beta(w_{ek}(\lambda_2)-\lambda_1)} G_{ek} \sum_{\prec} \sum_{\pi_e^{(k)} \subseteq U_e} f_e(G_k) P(x_e^{(k)} | \pi_e^{(k)}) \times \\ &\left[\sum_{\pi_e^{(j)} \subseteq U_e} P(x_e^{(j)} | \pi_e^{(j)}) \frac{(1-\lambda_2)^{\Delta_{ikj}}}{(4-\lambda_2)^2} \cdot \left(\frac{|U_i|}{4-\lambda_2} - \frac{\Delta_{ikj}}{1-\lambda_2} \right) \right]\end{aligned} \tag{11}$$



(a) G_1 for $\lambda_1 = 0.1$ (b) G_1 for $\lambda_1 = 0.5$ (c) G_1 for $\lambda_1 = 0.8$

(d) G_2 for $\lambda_1 = 0.1$ (e) G_2 for $\lambda_1 = 0.5$ (f) G_2 for $\lambda_1 = 0.8$

Figure 7: By thresholding at λ_1, we obtain graphs G_k from the weighted adjacency matrices W_k with $\lambda_2 > 0$.

This can be re-written using the pre-computed γ functions.

$$\frac{\partial}{\partial \lambda_2} G_{ek} = \\ = -\beta e^{-\beta(w_{ek}(\lambda_2)-\lambda_1)} G_{ek} \sum_{\prec} \sum_{\pi_e^{(k)} \subseteq U_e} f_e(G_k) P(x_e^{(k)} | \pi_e^{(k)}) \times \\ \left[\sum_{\delta=0}^{r} \frac{(1-\lambda_2)^\delta}{(4-\lambda_2)^2} \cdot \left(\frac{|U_i|}{4-\lambda_2} - \frac{\delta}{1-\lambda_2} \right) \gamma_{ke\delta}(\pi_e^{(k)}, \delta) \right] \tag{12}$$

The minimum step size is η such that $\Lambda^{\text{new}} = \Lambda - \eta \nabla_\Lambda g$ gets $G(\Lambda^{\text{new}})$ one edge closer to S.

$$\sum_{e,k} |S_{ek} - G_{ek}(\Lambda^{\text{new}})| - \sum_{e,k} |S_{ek} - G_{ek}(\Lambda)| = -1 \tag{13}$$

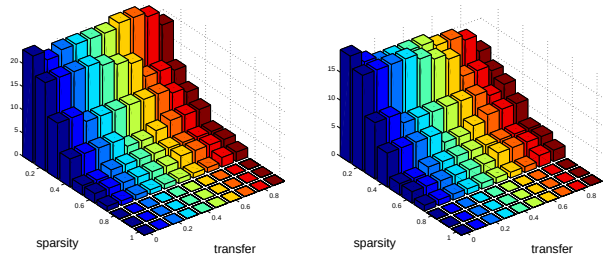
We solve for η using binary search until the above criteria is met.

6. DISCUSSION

There is strong motivation for creating interactive human-in-the-loop algorithms for exploring comparative dependency networks. Here we discuss our initial findings on benchmark networks, share case studies on real data and then suggest directions for future work.

6.1 Demonstration on Benchmark Networks

We use the benchmark *Asia* network to explore the practicality of this interactive approach. The *Asia* network contains 8 discrete variables [12]. In order to produce multiple networks with some edges different, we randomly delete each edge with probability $p = 0.1$. If an edge is deleted, the conditional probability table for the child is modified by summing over the removed parent. This produces a set of networks that are similar to the original *Asia* network but with a few edges different.



(a) Number of learned edges (b) Number of differences

Figure 8: Modified *asia* networks: summary statistics about learned network models for various values of sparsity and transfer hyper-parameters.

Using two tasks, and starting with an initial value for $\Lambda = [0.5, 0]$, we learn networks $\mathbf{G}$. Then simulated feedback responses, $\mathbf{S}$, are given by randomly choosing a user action at each stage. For each of these feedback matrices, we track the movement in Λ to investigate the effect of $\mathbf{S}$ on Λ. For comparison, we also perform a grid search by running the multitask network learning algorithm for combinations of settings of λ_1 and λ_2 evenly spaced in the valid range for each parameter. Figure 8 shows results of a grid search for one set of data, with 100 samples drawn from modified *Asia* networks. As expected, neither the number of edges nor the number of differences learned vary linearly with the input hyper-parameters. Whereas, by design, the interactive algorithm takes steps evenly in terms of the number of edges or differences learned.

One question is whether the gradient direction is typically aligned with just one hyper-parameter, or if it is usu-

ally more "diagonal". If it is typically aligned with just one hyper-parameter, then we could adjust each parameter independently rather than calculating the gradient. We observe that it is typically diagonal (takes a step in both λ_1 and λ_2 directions), however there are some cases where the gradient direction is nearly zero for either λ_1 or λ_2.

It is difficult to ascertain the interestingness of a solution for these benchmark networks. We have shown that grid search covers objectively uninteresting solutions; in the form of redundant solutions, overly dense solutions and empty solutions. These benchmark networks are not from a real domain (or it is an overly simplistic domain), therefore there is not a practical way to judge the subjective interestingness of the solutions to an uninteresting benchmark problem. To analyze the usefulness of the interactive algorithm from the end-user perspective, we therefore rely on case studies from real data.

6.2 Case Studies

Results on both neuroimaging and protein studies were presented to domain scientists using our interactive comparative network visualization. In both cases, a machine learning expert initially loaded the result networks into the visualization system and then manned the controls for adjusting the number of edges and the number of differences. After a few minutes of looking through network solutions with various numbers of edges and differences, the domain experts typically made requests, such as to see "the highest confidence edges shared by both tasks." The domain experts were able to take over the controls themselves and expressed appreciation for being able to visualize so many solutions quickly.

Anecdotally, we find that different domain experts are interested in different levels of confidence in edges and differences. For the neuroimaging study, the domain expert was most interested in extremely high confidence differences, selecting difference networks with only three dependencies in each. On the other hand, the biologists looking at protein data were interested in difference networks with 100 dependencies. These two anecdotes support the idea that different users could have different inexpressible objective functions in mind. However, we need to have different domain experts analyze the same data to see if the various interests are due to the users or if it is inherent in the data.

Often in machine learning, the goal is to find the single best solution to a problem. However, while looking through the various solutions produced by different hyper-parameter settings, the domain experts did not ask how to select the single best solution. They fully understand the concept of exploring the precision-recall tradeoff. Yet, they did ask whether there is any way to get a confidence interval for the dependencies and differences. Instead of adjusting the number of edges/differences, they would find it preferable to be able to quantify the confidence of edges/differences.

6.3 Future Work

This paper provides a framework for creating interactive multitask graph structure learning algorithms. These algorithms remain computationally challenging. The Bayesian posterior distributions on multiple Bayesian networks, in particular, do not scale well to large networks. The scalability problem is endemic to the problem of Bayesian network learning. Performing updates in real-time for large networks will be computationally difficult. We could alleviate this problem through the use of approximate or heuristic network structure learning. Doing so requires extensive evaluation on the tradeoffs between speed and accuracy, and so we leave this for future work.

Other graph learning algorithms, such as graphical lasso [15, 4], scale to large networks much better than Bayesian networks. Therefore, we would like to apply the proposed interactive method to multitask graphical lasso. However, the graphical lasso objective with respect to λ_1 and λ_2 is discontinuous; therefore, the gradient (Equation 2) is undefined at precisely the points that we care about. Currently, we are investigating numerical approximations to the regularization path or heuristics for finding the discontinuous "hinge" points quickly. Such algorithms that calculate the regularization path for individual networks have been developed [6, 22]. However, there is not any such algorithm for multitask network learning.

Typical grid search methods are inefficient and information criteria based tuning guidelines often are not ideal. Interactive guidance provides fine-grained control over exploration of the solution space in those areas that are of highest interest to the user. We could take a hybrid approach, first computing results over a coarse grid, then giving the user the ability to take small local steps or to jump to another area of the hyper-parameter space using the pre-computed results.

Other forms of feedback could be incorporated rather than just increasing or decreasing the number edges and differences. For example, one request from domain scientists is being able to query a specific edge, and see what the whole network looks like at the threshold point where that edge appears. A similar query could be imagined for edge differences. These type of queries should be straightforward to implement algorithmically. The challenge is in creating a user interface to gather this type of feedback. Working closely with domain scientists, we could find other queries that would make exploring solutions easier for the user.

The interactive approach presented here assumes that a human will guide the objective function via feedback about the hyper-parameters. However, the idea of beginning at an initial point in the solution space and exploring solutions by modifying hyper-parameters could be accomplished without a human. A virtual user that begins with no transfer and repeatedly requests fewer differences, is essentially an automated process for exploring the regularization path along the "differences" axis. The result of such a solution path is a ranking of the strength of the differences found. Therefore, the updates to the algorithm presented in this paper could be used as steps in an automated iterative algorithm, instead of an interactive human-in-the-loop algorithm. This is an interesting direction to explore to see whether the human users or the automated approaches are more effective at finding interesting solutions.

7. CONCLUSIONS

The concept of interactive network comparison is compelling. The hypothesis space is large and the learned models are complex. Presenting only a single solution (even if it fits the data well) is unsatisfactory. Yet, it is not easy to display all possible solutions at once and summary statistics about the potential patterns only tell a part of the solution. Graphical models are frequently used in knowledge discov-

ery because they help to quickly visualize complex patterns of dependency. In an increasingly interactive world, it is frustrating to the end user to see static results of a learning algorithm and not able to explore alternative solutions on the fly. Therefore, human-in-the-loop interaction is necessary for comparative dependency network learning. The first challenge in making a machine learning algorithm interactive, is to translate user feedback into changes in hyper-parameters that control the learning algorithm. The second challenge is to efficiently update results to be seen in real-time. We introduce a framework for interactive multitask graph structure learning with a specific implementation of multitask Bayesian networks and show that the results are preferable to the standard grid search over hyper-parameter space. In practice, all machine learning applications involve some form of interaction between looking at results and adjusting the algorithm to investigate alternative results. Automating this interactive process allows domain scientists and other end users to work more efficiently while discovering patterns in their data.

8. REFERENCES

[1] S. Amershi, J. Fogarty, A. Kapoor, and D. Tan. Effective end-user interaction with machine learning. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[2] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, pages 67–74. ACM, 2007.

[3] J. Chang, J. Boyd-Graber, S. Gerrish, C. Wang, and D. Blei. Reading tea leaves: How humans interpret topic models. In *Neural Information Processing Systems*, 2009.

[4] P. Danaher, P. Wang, and D. Witten. The joint graphical lasso for inverse covariance estimation across multiple classes. *arXiv stat.ME*, 1111(00324v1), November 2011.

[5] A. de la Fuente. From 'differential expression' to 'differential networking' – identification of dysfunctional regulatory networks in diseases. *Trends in Genetics*, 26(7):326 – 333, 2010.

[6] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407–499, 2004.

[7] N. Friedman and D. Koller. Being Bayesian about network structure: A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50(1):95–125, 2003.

[8] N. Friedman, I. Nachman, and D. Peér. Learning Bayesian network structure from massive datasets: the sparse candidate algorithm. In *Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 206–215, 1999.

[9] A. Kapoor, B. Lee, D. Tan, and E. Horvitz. Performance and preferences: Interactive refinement of machine learning procedures. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[10] M. Koivisto and K. Sood. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, 5:549–573, 2004.

[11] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. Adaptive Computation and Machine Learning. MIT Press, 2009.

[12] S. Lauritzen and D. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–224, 1988.

[13] H. Liu, K. Roeder, and L. Wasserman. Stability approach to regularization selection (stars) for high dimensional graphical models. In *Neural Information Processing Systems*, 2010.

[14] S. Maslov and K. Sneppen. Specificity and stability in topology of protein networks. *Science*, 296(5569):910, 2002.

[15] N. Meinshausen and P. Bühlmann. High-dimensional graphs and variable selection with the lasso. *The Annals of Statistics*, 34(3):1436–1462, June 2006.

[16] K. Mohan, M. Chung, S. Han, D. Witten, S.-I. Lee, and M. Fazel. Structured learning of Gaussian graphical models. In *Advances in Neural Information Processing Systems 25*, pages 629–637. 2012.

[17] A. Niculescu-Mizil and R. Caruana. Inductive transfer for Bayesian network structure learning. In *Eleventh International Conference on Artificial Intelligence and Statistics*, 2007.

[18] T. Niinimaki, P. Parviainen, and M. Koivisto. Partial order MCMC for structure discovery in Bayesian networks. In *Twenty-Seventh Annual Conference on Uncertainty in Artificial Intelligence*, pages 557–564, 2011.

[19] D. Oyen and T. Lane. Bayesian discovery of multiple Bayesian networks via transfer learning. In *IEEE International Conference on Data Mining*, 2013.

[20] D. Oyen, A. Niculescu-Mizil, R. Ostroff, and A. Stewart. Controlling the precision-recall tradeoff in differential dependency network analysis. In *The Seventh Workshop on Machine Learning in Systems Biology*, 2013.

[21] P. Parviainen and M. Koivisto. Exact structure discovery in Bayesian networks with less space. In *Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 436–443, 2009.

[22] M. Schmidt, A. Niculescu-Mizil, and K. Murphy. Learning graphical model structure using l1-regularization paths. In *National Conference On Artificial Intelligence*, volume 22, page 1278. AAAI Press, 2007.

[23] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003.

[24] T. Van Allen and R. Greiner. Model selection criteria for learning belief nets: An empirical comparison. In *Seventeenth International Conference on Machine Learning*, pages 1047–1054, 2000.

[25] A. V. Werhli, M. Grzegorczyk, and D. Husmeier. Comparative evaluation of reverse engineering gene regulatory networks with relevance networks, graphical Gaussian models and Bayesian networks. *Bioinformatics*, 22(20):2523–2531, 2006.