7-1-2013

Interactive Exploration of Multitask Dependency Networks

Diane Oyen

Diane Oyen

Candidate

Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

George Luger , Chairperson

Terran Lane

Eric Eaton

Vincent Clark

Lance Williams

Interactive Exploration of Multitask Dependency Networks

by

Diane Oyen

B.S., Electrical and Computer Engineering,
Carnegie Mellon University, 1998

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico
Albuquerque, New Mexico

May, 2013

Dedication

To
my cousin, Carla Modderno
for teaching me to dance in the rain.

And also to
my husband, Troy Lovata
for all of his support along the way.

Acknowledgments

Many people contributed ideas, support and encouragement throughout the research that culminated in this dissertation. First and foremost, I wish to acknowledge the support and contributions of all members of my dissertation committee. As my advisor throughout my time as a doctoral student, Terran Lane shaped much of the work and contributed countless ideas. George Luger encouraged me to pursue the PhD from the first time we met and provided support and guidance ever since. Eric Eaton has been the driving force behind the research in the direction of active transfer learning through many brainstorming sessions. I am indebted to Vince Clark for collecting data worth analyzing. Thanks to Lance Williams not only for stepping in at the last minute, but also for many interesting conversations.

I would also like to acknowledge my collaborator and mentor Alexandru Niculescu-Mizil at NEC Laboratories. This dissertation would not be complete without his suggestions for research directions, even if those directions outside of the typical machine learning domain. Thanks also for sharing the protein data and thanks to the biologists at SomaLogic for being so enthusiastic about our results.

There are a number of researchers that I worked with while a student, whose research goals have influenced my own goal of making machine learning practical for domain experts. Thanks to the Mind Research Network, and in particular Vince Calhoun, for providing challenging problems to be solved. Thanks to Frank Gilfeather for introducing me to a much broader view of national security research than I knew existed, especially through travel to Kazakhstan and Egypt. Thanks to Robin Ohls for giving me a chance to save babies with machine learning. While these experiences may not be directly reflected in this dissertation, they will shape my future career choices and research directions.

The machine learning research group of UNM have all helped with this research through brainstorming, criticism and commiseration. In particular, I would like to thank Vamsi Potluru, Sergey Plis, Sushmita Roy, John Burge, Eva Besada-Portas, Xiaoran Yan, Ben Yackley, Thanaphon Tangchoopong, and Blake Anderson.

Of course, no dissertation gets written without the encouragement and help from friends and family. Thanks to all who provided personal support, and even brought me food, when I was discouraged, frustrated or just plain busy with deadlines.

This dissertation was supported financially by various funding agencies throughout my time as a graduate student. I would like to acknowledge the financial support from NSF, NIMH, DARPA, DTRA, and ONR. Additionally, NEC Laboratories provided a paid internship that allowed me to pursue my research.

Interactive Exploration of Multitask Dependency Networks

by

Diane Oyen

B.S., Electrical and Computer Engineering,
Carnegie Mellon University, 1998
Ph.D., Computer Science, University of New Mexico, 2013

Abstract

Scientists increasingly depend on machine learning algorithms to discover patterns in complex data. Two examples addressed in this dissertation are identifying how information sharing among regions of the brain develops due to learning; and, learning dependency networks of blood proteins associated with cancer. Dependency networks, or graphical models, are learned from the observed data in order to make comparisons between the sub-populations of the dataset. Rarely is there sufficient data to infer robust individual networks for each sub-population. The multiple networks must be considered simultaneously; exploding the hypothesis space of the learning problem. Exploring this complex solution space requires input from the domain scientist to refine the objective function.

This dissertation introduces a framework to incorporate domain knowledge in transfer learning to facilitate the exploration of solutions. The framework is a generalization of existing algorithms for multiple network structure identification. Solutions produced with human input narrow down the variance of solutions to those that answer questions of interest to domain scientists. Patterns, such as identifying differences between networks, are learned with higher confidence using transfer learning than through the standard method of bootstrapping. Transfer learning may be the

ideal method for making comparisons among dependency networks, whether looking for similarities or differences. Domain knowledge input and visualization of solutions are combined in an interactive tool that enables domain scientists to explore the space of solutions efficiently.

Contents

List of Figures

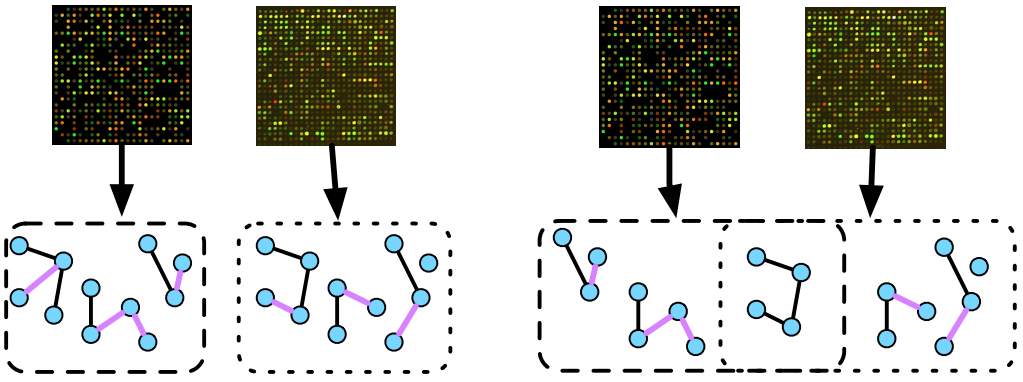
List of Tables

Chapter 1

Introduction

Scientists in many domains, such as biology and neuroscience, increasingly rely on machine learning algorithms to aid the discovery of patterns in complex data. Finding patterns that suggest potential hypotheses is a problem known as *knowledge discovery*. For example, neuroscientists look for pathways of information sharing in the brain, known as *functional brain networks* to understand how development or mental illness affects these pathways. The underlying functional network of the brain is difficult to extract purely from neuroimaging data, yet it still may be possible to learn highly likely differences and similarities in these pathways under various experimental contexts. In another example, biologists look for how concentration correlations among blood proteins are different in patients with cancer or heart disease. The goal is to get a better understanding of the biological processes underlying disease, and to generate hypotheses about protein markers that can be used for the early diagnosis of disease.

The data in these problems looks quite different, yet the same tools can be used for both. In both cases, the goal is to discover dependencies in multivariate data, particularly for making comparisons about similarities and differences among various related sets of data. Probabilistic graphical models provide a compact representation of the joint probability distribution of variables in multivariate data (Friedman, Hastie, and Tibshirani, 2008; Friedman, Nachman, and Peér, 1999; Heckerman, Geiger, and Chickering, 1995; Koller and Friedman, 2009; Meinshausen and Bühlmann, 2006); enabling scientists to visualize the conditional dependencies among the variables. Therefore,



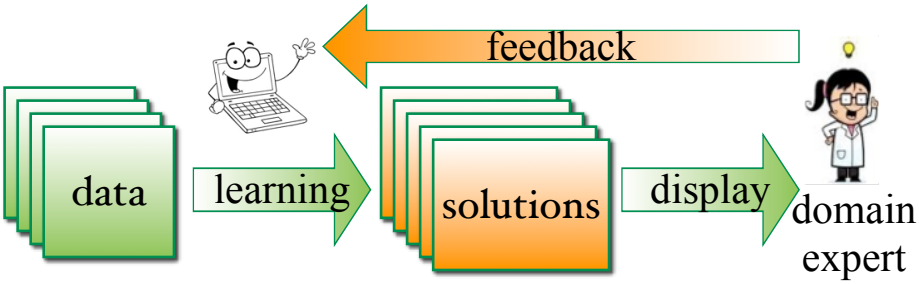
(a) Two networks learned independently (b) Two networks learned with transfer

Figure 1.1: Networks learned from data of two related tasks. The first example shows networks being learned independently of each other. The second example shows networks being learned simultaneously with transfer learning that encourages part of the network structure to be shared among the two tasks. The amount of overlap in shared structures is controlled by the degree of transfer bias.

the primary focus of this dissertation is on learning the structure of multiple graphical models from related sets of data in order to make comparisons among those models about likely dependencies. Figure 1.1a shows an example of learning two network models from two sets of data. Conditional dependencies are shown as an edge between nodes in each network. Black edges are common to both sets of data, while purple edges are unique to each task in this example.

Challenges in learning multiple networks are statistical, computational, and practical. The statistical challenges are primarily due to the fact that learning graphical models, or network structure, requires large amounts of data (the number of data samples must be at least square the number of variables) to guarantee the accuracy of the solution (Friedman and Yakhini, 1996). Scientific data often includes hundreds or thousands of variables, yet there is rarely sufficient data to guarantee the accuracy of the solution. The problem of learning multiple graphical models, as addressed in this dissertation, if approached naively, presents a computational challenge because the solution space grows exponentially with the number of networks being learned. Searching through these solution spaces may not be tractable. Furthermore, issues of practical use are important. Scientists realize that their dataset is but one snap-

(a) Traditional machine learning workflow

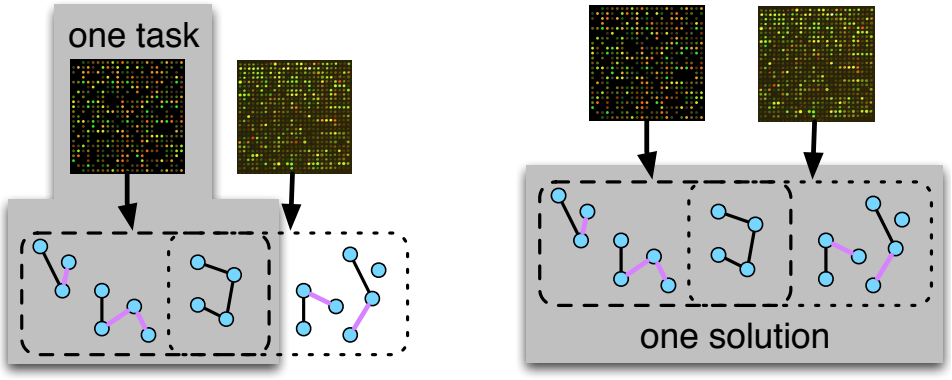


(b) Proposed interactive machine learning workflow

Figure 1.2: Machine learning workflows. The top figure shows how the traditional machine learning algorithms take data as input and output a single solution that is presented to the domain expert or end-user. The bottom figure shows the interactive machine learning approach given in this dissertation. The improvements are shown in orange: multiple solutions are given rather than just one; and the domain expert can give feedback to the learning algorithm to update the displayed solutions.

shot, possibly noisy, from a larger, unobserved population and therefore any single solution is just one of many possible solutions. Machine learning algorithms traditionally present the single most likely solution given the data and a learning objective. Figure 1.2a shows a typical machine learning workflow, in which data is used as the input to a machine learning algorithm. A single solution is found and displayed to the end-user, a domain expert looking for patterns in scientific data. However, a more complete picture of all (or many) likely solutions would be more informative to the end-user. Figure 1.2b demonstrates the proposed interactive workflow for improving the practicality of using machine learning algorithms. Multiple solutions would be learned by the algorithm and displayed to the end-user. The end-user or domain expert would give feedback to the learning algorithm to explore possible solutions of patterns in the data. Existing algorithms are of limited practical use until these issues are addressed.

Various methods have been attempted to address the statistical, computational



(a) A single *task* outlined in gray (b) A single *solution* outlined in gray

Figure 1.3: Terminology used in this dissertation. (a) A *task* is a subset of the data, from which a task-specific network model will be learned. (b) A single *solution* is a set of task-specific networks (one network per task).

and practical challenges of multiple network learning, but none fully address the issues unique to comparative analysis of learned networks. Statistical challenges have been addressed previously through the use of *transfer* and *multitask* learning; that is, sharing information among related sets of data to leverage the data across multiple sets of related data (or *tasks*) (Caruana, 1997; Danaher, Wang, and Witten, 2011; Niculescu-Mizil and Caruana, 2007; Thrun, 1996). See Figure 1.1b for an example of transfer learning of two networks and Figure 1.3a for a pictorial definition of a *task*. Transfer and multitask learning methods have all been shown to improve the accuracy of learned network models assuming that much of the network is the same across tasks. Figure 1.1b shows two tasks learned with a transfer bias that encourages some shared network structure. The amount of overlap in the learned structure can be controlled through the strength of the transfer bias. With more transfer bias, a larger portion of each learned network will be shared with the other task. Despite the success of existing transfer learning methods, they generally have rather naive assumptions about the degree of transfer between tasks. Therefore, this dissertation extends transfer and multitask learning algorithms to improve the learning of multiple network structures.

As a note on terminology, in this dissertation, we refer to transfer learning as a

mechanism for biasing the learned model of a target task to be similar to the structure of one or more source tasks (Thrun, 1996). We use the term multitask learning to define a special case of transfer learning in which all tasks are learned simultaneously as both target and source tasks where all structures are biased to be similar to each other (Caruana, 1997).

While transfer and multitask algorithms learn multiple networks, our goal is to perform comparative analysis among the learned task-specific dependencies. This is a somewhat different statistical question to answer than the typical machine learning goal of distribution matching. Existing approaches for comparative dependency analysis generally fall into three types. The first is the naive approach of learning networks for each related dataset independently and then making comparisons. To assess the validity of comparisons among the learned networks, these approaches often employ a bootstrap re-sampling or post-hoc network analysis on the learned models. When there is limited data available to learn network structures, approaches that learn the network independently suffer from statistical limitations. The second approach is to use discriminative models to look for differences in dependency structures between two related datasets. This approach has the advantage of encoding the question being asked as the learning objective for the algorithm. Yet, this approach answers a fundamentally different question than what is considered in this dissertation. Discriminative models learn dependencies that differentiate one class of data from the other; however, it is rarely the case that the differential dependency learned represents a direct dependency that actually exists in the data of that class. Furthermore, learned discriminative networks are difficult to interpret, as the discriminative networks do not represent joint probability distributions that could have generated the given dataset. The third approach learns the multiple networks simultaneously through transfer learning. Transfer learning assumes that some of the network structures are common to multiple tasks, and therefore biases solutions towards those that share common structure among tasks. When comparing learned networks, transfer learning reduces the variance of individual task-specific solutions so that comparisons are easier to make. This approach has been shown to perform well in terms of producing robust

models the match the generating distribution especially when the amount of training data is limited. Thus, this dissertation uses transfer learning as the underlying basis for learning comparative network structures while customizing the learning objective toward comparison rather than distribution matching.

Learning one network model can be computationally expensive and so learning multiple networks is even more challenging. Most transfer and multitask learning algorithms have dealt with this issue by performing heuristic search for a locally optimal single point solution (Niculescu-Mizil and Caruana, 2007; Roy, Werner-Washburne, and Lane, 2011) or by introducing a convenient regularizer that is convex to produce a globally optimal single point solution (Honorio and Samaras, 2010). However, it has been shown that when learning a single network given limited data, it is better to learn a posterior distribution over all possible solutions rather than find a single highly-likely point solution because many other nearly score-equivalent solutions could exist (Friedman and Koller, 2003; Koivisto, 2006). However, these algorithms have not been extended to include a transfer bias for learning multiple networks. One approximation algorithm has been proposed that incorporates a prior knowledge bias (Grzegorczyk and Husmeier, 2008), but extending the exact calculation of Koivisto and Sood (2004) would be preferable, yet it is not trivial to do this efficiently. Thus, computational challenges in learning multiple networks simultaneously have limited the usefulness of existing algorithms.

The practical challenges facing domain scientists using machine learning algorithms for comparative analysis of learned networks have been addressed primarily through the creation of special-purpose algorithms. These algorithms are developed with objective functions that are specific to particular domains. For example, in genomic research, changes in dependency structure are expected to change over the life-cycle of algorithms. This domain knowledge can be encoded in the learning objective of a transfer learning algorithm (Husmeier, Dondelinger, and Lèbre, 2010). The algorithm created for one domain may not be useful for another domain. Yet, the concept of encoding domain knowledge in the objective function is applicable to many domains. Therefore, generalizing algorithms to take this domain knowledge as

an input would enhance the practical use of transfer learning. This is a first step in the human-in-the-loop workflow proposed to create a practically useful algorithm. An algorithm that incorporates domain knowledge will bias solutions toward human desires. Yet, human desires are often difficult to encode mathematically a priori (Kapoor et al., 2012). Instead, it is preferable to allow the domain scientist to respond to the learned solution with feedback for the algorithm that refines the domain knowledge. Extending existing algorithms to include this feedback loop requires a method for translating user preferences (about the solution space) into prior knowledge (about the appropriate bias in the learning objective). This concept of user feedback about transfer bias in network structure learning has not been previously explored.

1.1 Goals

Until now, little research has gone into showing whether transfer learning actually improves the comparative analysis of learned dependency networks. This is the broad goal of this dissertation. Several practical limitations of existing transfer algorithms have been identified and addressed. In particular, existing multitask algorithms provide only a single optimal solution (Danaher, Wang, and Witten, 2011; Niculescu-Mizil and Caruana, 2007). One solution is defined to mean a task-specific network for each task (see Figure 1.3b). Other solutions may be nearly as likely, and so to interpret the patterns found in the data, the end-user will need a comprehensive picture of multiple solutions to make any meaningful comparison. Another serious limitation of multitask network structure algorithms is that they assume all pairs of tasks are equally related and provide no mechanism for incorporating domain knowledge about task-relatedness. We address this limitation by providing a framework for incorporating task-relatedness knowledge in any multitask network learning algorithm. Finally, this dissertation provides a visualization tool for exploring the space of solutions with a mechanism for the end-user to provide feedback to the learning algorithm (see Figure 1.2b).

The overall goal of this dissertation is to provide an interactive method for learning

multiple network structures with the purpose of comparing dependency structures of related scientific data sets. The goals of this dissertation are summarized in the following questions:

- Does domain knowledge about task-relatedness lead to better comparative solutions?
- Is it tractable to learn multiple solutions and display them to the end-user?
- Can feedback from the domain expert to the machine learning algorithm be automated into an interactive process?

1.2 Impact

By achieving the goals laid out in this dissertation, we can narrow the gap between the types of questions being asked by domain scientists and the pattern recognition tools of machine learning that are currently available. Knowledge discovery in scientific domains is a unique problem for machine learning. Other, more typical machine learning objectives find parsimonious solutions that fit the data well, usually assuming that the end-user is relatively unknowledgeable and cannot provide any further input. In scientific domains, the end users are scientists that care more about understanding the breadth of likely solutions than finding any one solution, even if it is a statistically optimal solution. To these knowledgeable end-users, any machine learning solution is just one possible explanation of the true generating process underlying the data set that has been collected by the domain scientist.

Often in machine learning, the least-informative or agnostic learning objective is the preferred objective. The assumption is that the data alone should drive the solution. However, in scientific domains, our end-users are themselves quite knowledgeable and they recognize the limitations of the particular set of data that they have collected. This is a fundamentally different way of thinking about machine learning. For these reasons, we want to include prior knowledge, informative priors and

human input as a bias in the learning objective. Without such input from the domain scientist, machine learning algorithms are of little practical use in scientific discovery.

Furthermore, once a solution has been found, scientists often are surprised by the results. They then would like to change the parameters of the learning algorithm to see such choices affect the solution. Automating this process is critical to making learning algorithms practical for use by practitioners. Integrating user-feedback into the solution visualization software enables scientists to explore solutions in real-time. Using the software tools detailed in this dissertation, our domain expert collaborators have been able to find patterns in dependency networks that give them insight into how functional brain networks are affected by mental illness and medication, and patterns of blood protein correlations associated with cancer.

1.3 Contributions

This dissertation makes several contributions to the field of machine learning, as itemized here.

- Develops the task-relatedness aware multitask (TRAM) general framework for incorporating domain knowledge about the relative relatedness of tasks in multitask network structure learning formulations.
- Demonstrates that a relaxation of the standard assumption of multitask learning that all pairs of tasks are equally related is able to learn networks that better generalize to validation data.
- Demonstrates that domain knowledge about the relative relatedness of tasks shapes the topology of the solution space to reflect human biases.
- Develops a multitask Bayesian network discovery algorithm for incorporating prior knowledge about network features in estimating full posterior distributions over multitask Bayesian networks.

- Demonstrates that transfer learning identifies higher confidence differences between networks than existing bootstrap methods.

- Develops an interactive algorithm for updating both network solutions and hyper-parameters using human feedback.

- Develops an interactive software tool for visualizing network solutions and providing feedback from the end-user.

- Learns comparative dependency networks from neuroimaging and blood proteins that are of interest to domain scientists.

1.4 Organization of this Dissertation

This dissertation is organized as follows: Chapter 2 provides some context and background information in the form of a survey of related literature. The related work includes machine learning algorithms for learning Bayesian networks and undirected graphical models. Particular attention is paid to algorithms that consider multiple networks at a time in the form of transfer and multitask learning algorithms, as well as interactive machine learning approaches. In addition, background is provided on neuroimaging and blood proteins.

Chapter 3 gives a general framework for incorporating human or other prior domain knowledge into a task-relatedness metric for multitask network structure learning. A specific application is given for learning multitask Bayesian networks from several related datasets. This chapter demonstrates that this generalized version of multitask network structure learning is better able to find solutions that generalize to validation data as well as fit the preferences of the domain expert.

Chapter 4 addresses the problem of incorporating a transfer bias on graph structures in algorithms that are optimized for estimating the posterior distribution of each individual edge in the network. These algorithms give the fullest picture possible to the domain expert about potential dependencies that exist in the related datasets.

The contribution of this chapter is two-fold. First, it addresses a limitation of the previous chapter by giving a distribution over solutions rather than a point solution. Second, it provides an efficient multitask Bayesian discovery algorithm that can be used in any scenario where prior knowledge about graph structures is available.

Chapter 5 looks at the problem of identifying differences between learned network structures and demonstrates that transfer learning is an effective tool for this goal. This particular issue has typically been solved by computationally expensive bootstrap procedures. Yet this chapter shows that transfer learning actually produces higher confidence differences than existing methods. Case studies are detailed from neuroimaging and cancer protein data.

Chapter 6 introduces an interactive approach to exploring the space of multiple network solutions for comparative analysis. The end-user can observe a solution and provide feedback about whether they would prefer to see more differences or more similarities among the task-specific networks. A method for incorporating this feedback and an approach to update the parameters and solution based on the feedback is presented.

Chapter 7 concludes the dissertation with a summary of major findings and implications.

Chapter 2

Background and Context

In machine learning, probabilistic graphical models describe a compact representation of a joint probability distribution. Learning such structures from data enables a domain scientist to visualize complex pathways of dependencies among variables, as described in Section 2.1. The comparison of such models has been studied previously, and so we discuss relevant applications (neuroimaging and protein correlation analysis) and open challenges in the comparison of graphical models in Section 2.2. Our approach to learning multiple graphical models is via transfer learning. Section 2.3 describes existing machine learning literature on transfer learning with a focus on existing algorithms within the framework of network structure learning. Finally, Section 2.4 describes existing methods for human-in-the-loop machine learning, also known as interactive and active learning algorithms.

2.1 Structure Learning of Graphical Models

Structure learning of graphical models refers to the problem of identifying conditional dependencies among variables from a set of data. The underlying joint probability distribution that produced the data is assumed to be made up of only a handful of direct dependencies. Other dependencies may exist, but are indirect. This structure is represented as the edges in a graphical model. We consider two different forms of graphical models: discrete Bayesian networks and Gaussian graphical models, described in the

rest of this section.

2.1.1 Bayesian Networks

Bayesian networks are directed acyclic graphs that give a compact representation of the joint probability distribution among a set of variables. Directed edges encode conditional dependencies between variables. A node in the network is independent of all other non-descendant nodes given the value of its parent nodes. Given a set of multivariate data, we can search for the Bayesian network that best describes the joint probability distribution of the data (Heckerman, Geiger, and Chickering, 1995). If the conditional dependencies (the edges or structure of the network) are known and data fully observed, then it is straightforward to calculate the maximum-likelihood estimates of the parameters of the distribution. However, learning the conditional dependency structure itself is more difficult. Learning structure is further complicated by Markov equivalence; that is, a joint probability distribution can be described by several different structures. Structure learning is required by applications in which we want to discover the interactions among variables in the system, such as in functional brain networks.

Several methods have been proposed that search over the space of Bayesian networks to find a network that best fits the given data (Grzegorczyk and Husmeier, 2008; Heckerman, Geiger, and Chickering, 1995; Madigan, York, and Allard, 1995). When there are a large number of variables, it takes more data samples to characterize the joint distribution over variables (Buntine, 1991). Therefore, in the case of limited data it is more likely that spurious edges will be inferred or that true edges will not be detected.

Network structure discovery in the face of limited data is an extensively studied problem. With limited data, the posterior probability of even the optimal network may be quite small; however, Friedman and Koller (2003) show that the marginal posterior probabilities over subgraphs or structural features can be quite high given the same data. They propose the so-called order-MCMC algorithm for estimating

such posterior probabilities. Koivisto and Sood (2004) give a dynamic programming method for calculating exact posterior probabilities of network features conditioned on orders. Further improvements are made to make the approach more memory efficient (Parviainen and Koivisto, 2009) and to produce partial-order MCMC (Niinimaki, Parviainen, and Koivisto, 2011).

2.1.2 Graphical Lasso

Gaussian graphical models (GGMs) infer a network of conditional dependencies from multivariate data by approximating the inverse covariance matrix (Dempster, 1972; Lauritzen and Spiegelhalter, 1988). The lasso algorithm, which employs an $\ell 1$ regularizer to select a sparse set of non-zero parameters to represent the data, has been proposed to find the neighborhood of each node in the graph (Tibshirani, 1996). This neighborhood selection is consistent with learning the full joint distribution (Meinshausen and Bühlmann, 2006). Various efficient algorithms for *graphical lasso*, the application of lasso to the inverse covariance selection problem, have been developed (Banerjee, El Ghaoui, and d'Aspremont, 2008; Friedman, Hastie, and Tibshirani, 2008; Yuan and Lin, 2007). The learning problem is formally stated as follows. If X is a p-dimensional Gaussian random variable $X \sim \mathcal{N}(0, \Sigma)$, then $\Theta = \Sigma^{-1}$ is the precision matrix. Entries in the precision matrix are partial correlations, i.e. θ_{ij} is the correlation of variables X_i and X_j given all other variables X_m, $m \neq i, j$. A value of $\theta_{ij} = 0$ implies conditional independence of X_i and X_j. Therefore, the precision matrix can be interpreted as an undirected network where nodes are the variables in the precision matrix and edges connect variables with non-zero partial correlations.

Given the dense covariance matrix estimated from data, $\hat{\Sigma}$, the learning objective for a single network is:

$$\widehat{\Theta} = \arg\max_{\Theta \succ 0} \log\det\Theta - \mathrm{tr}(\hat{\Sigma}\Theta) - \lambda\|\Theta\|_1 \ .$$

The parameter λ, $0 \leq \lambda \leq 1$, controls the degree of sparsity. Varying this parameter affects the precision-recall tradeoff of identifying dependencies. Rather than selecting an individual value for λ, it is usually more informative to inspect the networks in-

ferred at various values to see how edges appear/disappear along the precision-recall curve. Many standard methods for automatically tuning λ, such as cross validation, produce networks that are denser than the true network when limited data is available (Wasserman and Roeder, 2009). Other methods, such as stability selection, produce sparser networks with high-precision edges, at the risk of missing some true dependencies (Liu, Roeder, and Wasserman, 2010). Stability selection requires significantly more computation due to the bootstrapping approach that requires the algorithm to be run many times on different samples of data.

Often, the Gaussian assumption is too strong for real data. Extreme values in just a few samples, like 1% of the data, can produce a large number of Gaussian correlations that do not exist when those samples are not present. Transelliptical models replace the Gaussian covariance matrix with a non-parametric correlation matrix that is less sensitive to extreme values (Liu, Han, and Zhang, 2012). Any graphical lasso algorithm can learn a transelliptical graphical model, simply by replacing the sample inverse covariance matrix Σ with a Kendall's tau correlation matrix. This small change makes the learning significantly more robust to outliers and non-Gaussianity, without any significant loss in accuracy, even when data is truly Gaussian.

2.2 Comparing Learned Graph Structures

Often, domain scientists ask questions about how dependencies change due to adaptation or disease. To answer such a question, data are collected for various sub-populations in a population of interest. Then dependencies that are the same or different among those sub-populations can be assessed. To motivate and clarify the purpose of comparing graph structures, we describe the applications that this dissertation considers. Then, we outline the open challenges to answering such comparisons.

2.2.1 Applications

We provide examples from two forms of biological data to motivate the comparison of learned dependency networks.

Functional brain networks from neuroimaging data

Functional Magnetic Resonance Imaging (fMRI) measures the level of activity in voxels of the brain over time (Sarty, 2007). Brains of different individuals are mapped onto standard atlases of regions of interest (Lancaster et al., 2000; Tzourio-Mazoyer et al., 2002). The resulting data are a multivariate timeseries where the variables represent the activity levels in regions of the brain.

Due to individual physiological differences, pooling data together from multiple subjects is problematic, yet there is often not enough data to learn a different model for each subject. Multitask learning has been used to bias models of different subjects toward each other while still allowing for individual differences (Honorio and Samaras, 2010; Varoquaux et al., 2010). The intuition is that edges that are common to many populations will be detected correctly and those edges that are specific to a population will only be learned when there is enough data to support them, thus reducing the effect of noise in the data.

Bayesian networks and Gaussian graphical models show how regions of the brain interact with each other as functional networks (Smith et al., 2011). These so-called "whole brain" network models can investigate interactions among all regions of the brain simultaneously to discover patterns of interaction which would otherwise be unknown to researchers, as opposed to other models of functional brain networks which test the validity of hypotheses about the interaction of a few areas of the brain (Friston et al., 1994; Friston, Harrison, and Penny, 2003).

Plasma protein correlation networks for disease biomarkers

The concentration of thousands of plasma proteins can be measured from a single drop of blood using Somamer technology (Gold et al., 2010). Our goal in studying plasma protein networks is to find protein pathways that are associated with a disease, such as cancer. If such distinguishing relationships exist, then biologists can develop blood tests for fast, non-invasive early diagnosis of cancer, as well as gain insight into the underlying biological process of disease. In this application, it is important to identify differences with high confidence. While many differences may potentially exist, it is very expensive to test for multiple protein concentrations, therefore we need to limit the number of proteins to those that are most influential.

2.2.2 Open Challenges

Scarcity of data makes learning individual networks difficult. Theoretical guarantees on the accuracy of learned models require at least as many samples as the square of the number of variables (Akaike, 1973; Buntine, 1991; Schwarz, 1978). We know that we are not learning the correct network for each population, yet we still believe that it is possible to make comparisons among various networks. The first challenge is identifying more robust networks from limited data. Much work has been devoted to this in recent years. We have already discussed the estimation of posterior feature probabilities rather than entire networks (Friedman and Koller, 2003; Koivisto, 2006). Additionally, transfer learning incorporates information from related sets of data, as discussed in more detail in the next section (Caruana, 1997; Thrun, 1996). Transfer learning, naively applied, provides just one type of bias that can reduce the variance of solutions learned. The primary aim of this dissertation is to show that various forms of bias can be incorporated into the learning objective. Bias can take the form of preferring solutions that include specified features, numbers of features, or similarity of features among tasks. Bias will reduce the variance of learned solutions. The trick is to include the correct kind of bias to produce solutions that answer questions of interest to the user. That is, it remains an open question whether the robustness of

comparative analysis can be improved in light of the limitations of the robustness of individual networks.

Machine learning solutions for comparative network analysis have been data-driven. However, including human knowledge in the learning process is one way to improve the quality of learned networks. Existing algorithms that incorporate human knowledge typically rely on knowledge about the dependencies themselves (Grzegorczyk and Husmeier, 2008; Tong and Koller, 2001; Werhli and Husmeier, 2007). Yet, in the applications of interest, prior information about individual dependencies is not available and cannot be obtained. However, there may be other human knowledge that can reduce the variance of solutions. In particular, in transfer learning algorithms, domain experts may be able to provide valuable information about the transfer relationships among tasks. Furthermore, by incorporating domain knowledge, the human guides the algorithm to narrow down solutions in a way that helps to identify the highest confidence similarities and differences among the networks being learned. Existing comparative network analysis algorithms do not have a mechanism for incorporating this knowledge.

A special interest of many domain scientists is to identify the differences between dependency networks of related tasks (Bergmann, Ihmels, and Barkai, 2004; Burge and Lane, 2005; Roy, Werner-Washburne, and Lane, 2011; Zhang et al., 2009). Transfer and multitask algorithms do not address this issue. Discriminative models do look for differences between networks that differentiate the data from one class compared with another class. Yet, discriminative models answer a fundamentally different question because it is rarely the case the the differential dependency learned represents a conditional dependency that actually exists in the data of that class. Furthermore discriminative networks are difficult to interpret because they do not represent joint probability distributions that could have generated the given dataset. They lose the context of which dependencies *do* exist. Reliably identifying conditional dependencies that exist in one network but not the other (a difference), remains an open challenge for comparative network analysis.

2.3 Transfer and Multitask Learning

In the real world when encountering a novel problem, people leverage previous experience to avoid starting from scratch to learn a new task. Taking inspiration from this aspect of human learning, a machine learning approach known as transfer learning is concerned with applying knowledge from source tasks to improve the learning of a target task (Thrun, 1996). Variants on transfer learning and related frameworks, such as learning to learn, lifelong learning, multitask learning, domain adaptation, and self-taught learning are differentiated by the definition of source data and the form of transfer to the target task (Pan and Yang, 2010).

Transfer learning is the foundation for all of the proposed methods in this dissertation. Some of the methods are improvements on *multitask learning*, in which all tasks are considered simultaneously. The basic assumption in multitask learning is that there are some similarities between the various sets of data. This assumption will be built upon for all the variations on multitask learning that we propose. When the tasks are considered in a sequential manner rather than simultaneously, we use the more general term *transfer learning*.

2.3.1 Existing methods

Multitask learning is a version of transfer learning in which all tasks are included as both source and target tasks (Baxter, 1997; Caruana, 1997). The goal is improved model generalization for each task as compared with learning a model for each task independently, using the assumption that tasks are not independent of each other. Multitask learning shares information between tasks through inductive bias. Multitask algorithms generally learn models simultaneously for all given tasks.

Multitask (and transfer) learning encourages structures common across multiple tasks to be learned easily while also allowing for specialization between tasks, where the data supports specialization. Multitask learning has been successfully applied to learn models individualized to specific users or subjects, such as text classifica-

tion which tailors spam filtering to individual users (Dredze, Kulesza, and Crammer, 2010), online advertisement targeting (Chen et al., 2010), and learning localization parameters of individual devices in a sensor network (Zheng et al., 2008). In the analysis of neuroimaging data, we need to allow for subject variability while leveraging information across the pool of subjects. Jbabdi, Woolrich, and Behrens (2009) successfully achieve this with a hierarchical Bayesian approach to clustering of brain voxels. More closely related to our work, Honorio and Samaras (2010) treat each subject as a task while learning Gaussian graphical models of interactions among brain regions.

2.3.2 Task-relatedness knowledge

Existing multitask and transfer learning models handle task-relatedness in one of three ways: 1) assume all tasks are equally similar; 2) estimate the relatedness of tasks from the training data; or 3) estimate the relatedness of tasks from some information other than the training data. All existing multitask structure learning algorithms for graphical models use the first method. However, in the supervised setting of multitask learning, it has been shown that greater knowledge of task-relatedness is useful (as described in the following paragraphs). Conversely, assuming tasks are similar when they are not can result in negative transfer.

Rather than assuming all tasks are equally similar, the task clustering algorithm (TC) (Thrun and O'Sullivan, 1996) explores clustering tasks by some metric of the training data representing task-relatedness. This approach breaks the problem into several independent multitask problems. More sophisticated models of task relatedness have been shown to improve classifier performance. Eaton, desJardins, and Lane (2008) learn a graph of task-relatedness from data which is then used to direct transfer among tasks. Yu, Tresp, and Yu (2007) identify noisy or outlier tasks to avoid negative transfer.

Bakker and Heskes (2003) introduced the idea of using "higher level task characteristics" to improve the clustering of related tasks in classification and regression

problems rather than using the data itself. Such an approach is used with classifiers for natural language processing using meta-information from WordNet (Epshteyn and DeJong, 2006; Miller, 1995). Another application builds multitask genomic sequence classifiers with task-relatedness based on phylogenetic trees (Widmer et al., 2010). We build on these findings to show that multitask structure learning algorithms can also benefit from incorporating intelligent information about task-relatedness.

With a task-relatedness metric and learned models for some source tasks, it is possible to predict models for a target task without any training data specific to that task. This problem is known as zero-shot learning. Empirical results in classification problems demonstrate the feasibility of zero-shot learning and they differentiate between two frameworks (Larochelle, Erhan, and Bengio, 2008). In the terminology of Larochelle, Erhan, and Bengio (2008), in the data-space-view, task-relatedness information is simply appended to the input data vector so that learning a model for a new task is equivalent to generalizing over the hidden values of the missing data of the input vector. The model-space-view on the other hand, biases the parameters of the model toward those for similar tasks. Our work fits into the model-space-view paradigm. An application to neuroimaging data indicates that the zero-shot approach is feasible even with noisy, high-dimensional data (Palatucci et al., 2009). This approach is similar to using meta-data "hints" in which information from an expert about metadata, but separate from the training data, can increase performance of classification (Abu-Mostafa, 1995).

2.3.3 Transfer in unsupervised learning

One of the most fundamental unsupervised machine learning problems is clustering, in which inherent groupings of data points are learned in a dataset without labels. Transfer learning has been employed to bring some supervision into clustering using human-clustered or human-corrected machine-generated models as source datasets to help cluster a target dataset (Bhattacharya et al., 2009; Gu and Zhou, 2009; Zhang and Zhang, 2010).

Several multitask structure learning algorithms for graphical models have been proposed recently. These models cover Bayesian networks (Luis, Sucar, and Morales, 2009; Niculescu-Mizil and Caruana, 2007), dynamic Bayesian networks (Husmeier, Dondelinger, and Lèbre, 2010), Markov random fields and Gaussian graphical models (Chiquet, Grandvalet, and Ambroise, 2011; Danaher, Wang, and Witten, 2011; Honorio and Samaras, 2010). Most of the algorithms iteratively use the learned structure of other tasks as a prior on the current structure. All of these models assume that tasks are equally related, in contrast to the proposed approach of this dissertation, which incorporates task-relatedness knowledge. Many of the Gaussian graphical model learning algorithms use a regularization framework that forces all edges to be the same across tasks, as opposed to allowing for flexibility between learned structures of different tasks.

Steele and Tucker (2009) transfer networks from previously published gene regulatory results to generate more robust network models which do not rely so heavily on a single dataset. Similarly, Neumann et al. (2010) use meta-analysis to discover patterns of activity in brain regions from previously published work.

2.4 Interactive Machine Learning

Various forms of human-machine dialog (or interaction or human-in-the-loop) have been studied in machine learning for decades. In recent years, the focus has been on machine-directed interaction, known as *active* learning. We provide a review of such methods with an emphasis on active learning for unsupervised problems. We then draw a distinction between active learning and the more general *interactive* learning which need not be directed by the machine. The goals of interactive learning differ greatly between various applications, and so we divide the discussion into supervised and unsupervised domains. Most of the interactive paradigms involve gathering labels for data instances from humans; however, the approach taken in this work is to provide interaction with the parameters of the algorithm. Therefore, special attention is given to interactive algorithms that address parameter search.

2.4.1 Active Learning

Active learning algorithms have access to an oracle that can answer a certain class of queries, at a cost, to give more information or data relevant to the problem. The goal of the active learner is to select the lowest cost of queries that learns the best model. Cohn, Ghahramani, and Jordan (1996) defines active learning as choosing a query based on the statistically optimal method for improving the performance of the learner. A successful active learner needs fewer labels/samples to learn good models than if the labels/samples were drawn randomly. See Settles (2009) for a survey of active learning paradigms.

Little work exists in active structure learning of Bayesian networks. Tong and Koller (2001), as well as Murphy (2001), actively speed up learning through interventions - forcing the value at a specified node or nodes and then drawing a sample from the rest of the variables. Both algorithms use interventions which are not feasible in our applications because we cannot force the activity level in a region of the brain or the concentration of an individual plasma protein.

2.4.2 Interactive Supervised Learning

The most research in interactive machine learning falls under the broad category of supervised learning in which the goal is to predict a classification or regression label for each point of data. In the interactive paradigm, typically, only a few of the data points are labelled initially. The machine learning algorithm makes predictions on the remaining points. The user can label more points to refine the model in regions where the machine learning algorithm makes incorrect predictions. The main difference between interactive learning and active learning is that active learning is driven by the machine while interactive learning is user-driven. The user select which points to label. Examples of such interactive applications include image segmentation (Fails and Olsen Jr, 2003), image classification (Fogarty et al., 2008), and regression (Eaton, Holness, and McFarlane, 2010). An interesting variation on this theme is a human-built classifier, in which the machine only provides feedback about the current model

while the human makes all decisions about which feature to split at each branch of a decision tree (Ware et al., 2001). This work shows just how well a good visualization system helps a human to make sense out of data even without any machine learning.

All of these supervised learning approaches expect the end-user to provide labels describing the data itself. The work in this dissertation focuses on unsupervised learning in which no such labels exist. Instead, we concentrate on providing multiple solutions to the end-user and using interaction to navigate through those solutions. The idea of considering multiple models in the supervised learning setting is covered by Amershi et al. (2010) with an image classification application. While the classic active learning approach only gives a user a method for increasing the amount of information available to the machine, their case study of interaction shows that end-users prefer to explore how their feedback impacts the solution and often "undo" that feedback to revert to a previous model. This attitude toward interaction is similar to the approach considered here. The interaction is provided primarily to allow the end-user the ability to consider many different solutions that fit the data, rather than treating the user as a one-way oracle that provides unchanging bits of information.

2.4.3 Interactive Unsupervised Learning

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. Interaction in unsupervised learning, such as clustering and topic modeling, have been studied. For unsupervised learning, the motivation for interaction is that the user knows a good solution when he sees it, but otherwise cannot specify a learning criterion in advance. Empirical work in topic models justifies this assumption and shows that data-driven learning criteria do not match human judgments of good topics (Chang et al., 2009). Several interactive approaches to unsupervised learning recast the problem as a semi-supervised learning problem by allowing the user to provide labels that help to guide the objective functions (Cohn, Caruana, and McCallum, 2003; desJardins, MacGlashan, and Ferraioli,

2007; Dubey, Bhattacharya, and Godbole, 2010). These models are quite different than the approach taken here, where no labeling of dependencies is expected from the user. Although, such an approach would be possible algorithmically, there is little reason to believe that a user could provide knowledge about the existence of individual dependencies because these are precisely the unknowns to be discovered. It should be noted that desJardins, MacGlashan, and Ferraioli (2007) provides a nice visualization interface to facilitate interaction, an important component of interactive data mining for knowledge discovery. Rather than labeling instances, another approach has the user give feedback about the features used for clustering (Bekkerman et al., 2007), but this approach is not applicable to graphical model learning in which the dependencies are the features.

More related to our work, is the approach of learning multiple solutions and then allowing a user to select among them by providing information about each of the solutions. Dasgupta and Ng (2009) provide such an algorithm for multiple clusterings. However this approach is not interactive. Instead, multiple clusterings are learned as a batch and then the user selects the preferred clustering without any further interaction with the algorithm. Dy and Brodley (2000) interactively select a subset of features for clustering by presenting several choices to the user at each iteration. They provide a visualization interface that allows a user to find a preferred subset of features.

2.4.4 Interactive Parameter Search

The previous examples of interactive learning involve getting information from the user about labels on the data or preferred features. In contrast to these, our problem is not about getting information from the user about the data itself. Rather, it is giving the user a mechanism for exploring the various models that could be learned from a static set of data. Most learning algorithms are dependent not only on the data but also input parameters, referred to as algorithm parameters, to determine the learned model. Generally, these parameters are hidden from the end-user and chosen through a data-driven model selection procedure. The early parts of this dissertation

project demonstrate that in some applications, these parameters can actually be used to tailor the solution models to focus on particular queries from the user. Instead of selecting a single model, the end-user varies the parameters to explore the space of models.

Model selection is an inherently difficult, if not impossible, problem throughout machine learning. In graphical model structure learning, model selection concerns the choice of the number of edges present in the model. Several methods have been proposed, generally involving some form of *information criterion* (Akaike, 1973; Schwarz, 1978). These criteria attempt to choose the smallest model that accurately fits the data. For a given choice of criterion, a model can be selected in a principled manner. Yet, choosing an appropriate criterion is still a subjective matter left to a human expert. Thus, for learning a single graphical model, no single choice of the number of edges will be ideal in all cases. Similarly, for learning multiple related graphs, the parameter controlling the degree of transfer among models must be selected (Danaher, Wang, and Witten, 2011; Niculescu-Mizil and Caruana, 2007). The ideal setting of this parameter is typically determined through fit to holdout data (Efron, 1982), but this is only the correct criterion if fit to holdout data is the user's priority for learning the model (Meinshausen and Bühlmann, 2006; Van Allen and Greiner, 2000).

Interactive parameter search has been studied in supervised learning domains. Talbot et al. (2009) optimizes the ensemble weights of a collection of classifiers via interaction. Other work allows an end-user to adjust the cost-function weights and show that human interaction finds optimal parameter settings faster (Amershi et al., 2011) and gives the user control over the objective function (Kapoor et al., 2012). Graph structure learning is an unsupervised learning domain and so there may not be an optimal parameter setting. 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 on the solution. This dissertation borrows ideas from interactive parameter search in these classification problems to address the limits of model selection in learning multiple graphical models.

Chapter 3

Prior Domain Knowledge about Task-Relatedness

As discussed in the Introduction, transfer learning is a promising technique for learning multiple graphical models that are related to each other. Transfer and multitask learning algorithms for network structure identification generally assume that all pairs of tasks are equally related. This assumption simplifies the formulation of the objective function, but it is not always accurate. This chapter provides a framework applicable to any multitask network structure learning algorithm for incorporating prior knowledge about the relationships among pairs of tasks. Empirical results on synthetic and real data indicate that human knowledge about task-relatedness improves the quality of learned models and directs the solution towards those of interest to the end-user.

3.1 Motivation

Scientists use network structure learning algorithms to discover patterns of interaction in multivariate data, such as functional brain networks from neuroimaging data. For these datasets, multitask learning algorithms learn robust models even when the number of data samples collected in a specific task are limited, but there are several tasks that are believed to be similar (Baxter, 2000; Caruana, 1997). For example, in group neuroimaging studies, we learn functional brain networks for several pop-

ulations of subjects and treat the data from each population as a task. We expect the population-specific networks to have a lot in common but not to be identical. Furthermore, we may be able to describe relationships among the populations. For example in a study of functional brain networks there are populations of healthy control subjects and schizophrenia patients on various types of medication (see Figure 3.1). In another example, we could learn protein interaction networks for various species where species that are near each other in the evolutionary tree should have the most similar networks (Figure 3.1).

Our goal is to learn the best set of networks that are interesting to the domain expert. The concept of incorporating human preferences into unsupervised learning objectives is an active research trajectory in clustering (Dasgupta and Ng, 2009) and topic modeling (Chang et al., 2009). In these unsupervised learning problems data-driven measures of goodness of the learned model are insufficient and can only be addressed by incorporating human objectives. We bring this concept to network structure learning. The search space of Bayesian network structures is characterized by large basins of approximately equivalent solutions (Friedman and Koller, 2003). Multitask algorithms provide a first step toward incorporating a human bias by selecting a set of networks that are near each other in the search space, rather than a score-equivalent set of networks chosen from independent regions of the space (Niculescu-Mizil and Caruana, 2007).

Existing multitask methods for unsupervised problems typically assume that all pairs of tasks are equally related. This assumption makes these algorithms too rigid to handle datasets where different pairs of tasks have widely varying degrees of task-relatedness. Furthermore, they provide no mechanism for incorporating human objectives for which learned networks should be most similar to each other. A few specialized multitask network structure learning applications have recently incorporated specific domain knowledge about task-relatedness (Dondelinger, Lèbre, and Husmeier, 2010; Husmeier, Dondelinger, and Lèbre, 2010; Liu et al., 2010).

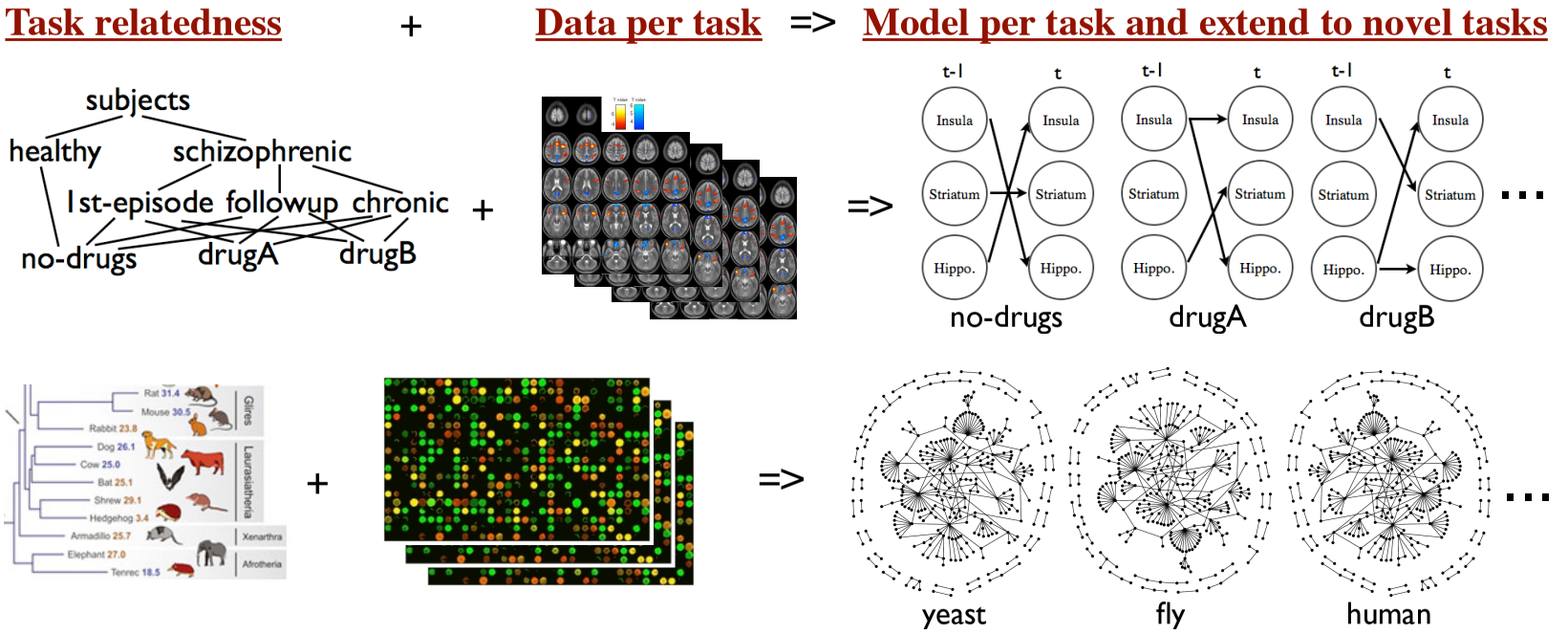


Figure 3.1: Example applications for learning multiple networks. Knowledge about task-relatedness is provided by a domain expert, such as the relationship among schizophrenia subjects (top) or among species in an evolutionary tree (bottom). Data is available for each task in the form of a data matrix comprising several samples of a multivariate random vector. The learning problem then is to identify a network model of dependencies for each task.

The first contribution of this dissertation is to introduce a framework for multitask structure learning that relaxes the assumption that tasks are equally related. In many applications we have prior beliefs about the relatedness of tasks based on metadata or domain expert knowledge. Using this framework, we develop the first multitask Bayesian network structure learning algorithm to incorporate task-relatedness as a parameter. With this algorithm we explore various factors in the problem space: the number of tasks, the true similarity between tasks, and the topology of task-relatedness. We compare the performance of the algorithm with naive algorithms (those without task-relatedness knowledge). This new algorithm generalizes to validation data and fits ground truth better than naive algorithms.

Finally, functional brain networks are learned from neuroimaging data with this Bayesian network algorithm. For a given pool of subjects, there are a number of "natural" task divisions (e.g, pooling by age or by medication). We explore different divisions of subjects into tasks, with corresponding task-relatedness metrics and discuss the interesting patterns found by the algorithm. Domain knowledge about task-relatedness improves both the robustness of learned networks and addresses human objectives.

3.2 Related Work

Multitask learning algorithms generally represent the similarity among tasks in one of three ways: all tasks are assumed to be equally similar (Caruana, 1997; Niculescu-Mizil and Caruana, 2007); the similarity among tasks is estimated from the same data that is used to train the model (Eaton, desJardins, and Lane, 2008; Thrun and O'Sullivan, 1996); or the similarity between tasks is provided by another source such as task-specific domain information or an expert (Bakker and Heskes, 2003). This third option, which we apply to network structure learning, has been used successfully for zero-shot classification problems when no training data are available for certain tasks (Larochelle, Erhan, and Bengio, 2008; Palatucci et al., 2009).

Multitask learning has been applied to learn Gaussian graphical models (Hon-

orio and Samaras, 2010) and Bayesian networks (Luis, Sucar, and Morales, 2009; Niculescu-Mizil and Caruana, 2007). Unlike our framework, these models assume that all tasks are equally related. There is recent work in specialized application-specific algorithms that share information only between tasks that are believed to be most similar (Dondelinger, Lèbre, and Husmeier, 2010; Husmeier, Dondelinger, and Lèbre, 2010; Liu et al., 2010). These applications demonstrate the benefit of domain knowledge.

3.3 Preliminaries: Multitask Structure Learning

Probabilistic graphical models compactly describe joint probability distributions by encoding independencies in multivariate data. Multitask learning in graphical models enforces a bias toward learning similar independency patterns among tasks. We introduce a general framework for multitask learning of graphical models with domain knowledge by optimizing the likelihood of the data given the learned model with a Bayesian prior over the model with a task-relatedness metric. We show that two existing applications are examples of our general model, and then use the framework to develop a novel algorithm.

A Bayesian network $B = \{G, \theta\}$ describes the joint probability distribution over n random variables $\mathbf{X} = [X_1, X_2, \ldots, X_n]$, where G is a directed acyclic graph and the conditional probability distributions are parameterized by θ (Heckerman, Geiger, and Chickering, 1995). An edge (X_i, X_j) in G means that the child X_j is conditionally independent of all non-descendants given its parent X_i. A Markov random field (MRF) encodes similar conditional independencies with an undirected graphical model (Kindermann and Snell, 1980). The *structure* of the network, G, is of particular interest in many domains as it is easy to interpret and gives valuable information about the interaction of variables.

A set of tasks with data sets D_k and networks G_k for $k \in \{1, \ldots, K\}$ can be

learned by optimizing:

$$P(G_{1:K}|D_{1:K}) \propto P(D_{1:K}|G_{1:K})P(G_{1:K}) \ , \tag{3.1}$$

where $P(G_{1:K})$ is a joint prior distribution over network structures. Assumptions about the similarity of network structures are encoded in this joint distribution. The simplest naive assumption is to assume that the K tasks are independent of each other. We refer to this model as single-task learning (STL) because it is equivalent to learning each model individually. By breaking the prior into independent single-task learning problems, STL assumes that all tasks are independent of each other, simplifying Equation 4.4 to:

$$P_{STL}(G_{1:K}|D_{1:K}) \propto \prod_{k=1}^{K} P(D_k|G_k)P(G_k) \ .$$

Multitask learning (MTL) does not assume that tasks are independent; however, it does generally assume that $P(D_k|G_k)$ is independent of all other G_i so Equation 4.4 simplifies to:

$$P_{MTL}(G_{1:K}|D_{1:K}) \propto P(G_{1:K}) \prod_{k=1}^{K} P(D_k|G_k) \ .$$

In multitask learning, the joint structure prior, $P(G_{1:K})$, is used to encode a bias toward similar structures. We can decompose this joint distribution into a product of conditionals, so that $P(G_{1:K}) = P(G_1) \prod_{i=2}^{K} P(G_i|G_{1:i-1})$. Many multitask algorithms (including those outlined in this chapter), make a further simplifying assumption that $P(G_k|G_{1:k-1}) = \prod_{i=1}^{k-1} P(G_k|G_i)$. That is, the joint prior over structures can be described by pairwise sharing of information among tasks:

$$P(G_{1:K}) \triangleq \prod_{i=2}^{K} P(G_i) \prod_{j=1}^{i-1} P(G_i|G_j) \ . \tag{3.2}$$

3.4 Task-Relatedness Aware Multitask Learning

We introduce our general framework for incorporating prior knowledge about task-relatedness in multitask structure learning. The goal is to include a weighting scheme for the amount of information sharing between different pairs of tasks. First, we define

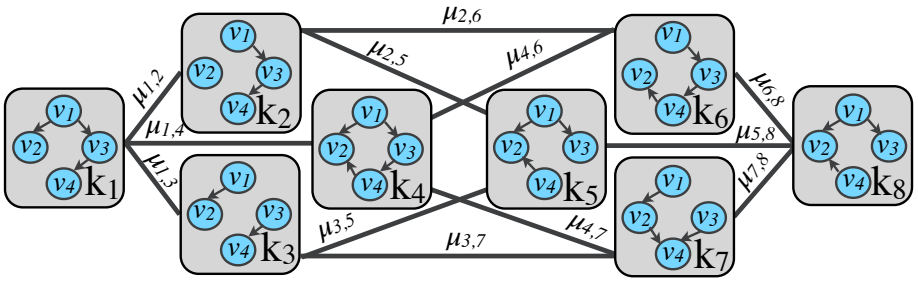


Figure 3.2: Task-relatedness graph: square vertices are tasks, weighted edges are task-relatedness. Small circles within each task are variables of task-specific networks.

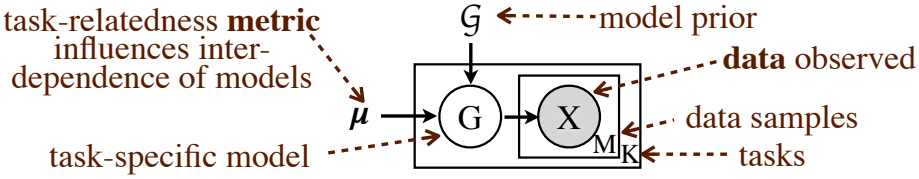


Figure 3.3: Generative model of multitask network structure learning use plate notation.

a symmetric matrix, $\boldsymbol{\mu}$, of size $K \times K$ where each element, $\mu_{ij} \geq 0$, describes prior information about the degree of relatedness between tasks i and j. These values come from a task-relatedness metric that describes the degree of relatedness of pairs of tasks. Figure 3.2 shows an example of what such a metric could look like. We find it more convenient to work with the inverse of the metric so that $\mu_{ij} = 0$ means that the tasks are independent, and large values of μ_{ij} mean a high degree of relatedness. Then, using the general description of the joint prior over network structure, in Equation 3.2, we use $\boldsymbol{\mu}$ to weight the transfer bias among pairs of tasks. With this additional input about the relatedness of tasks, the new **T**ask-**R**elatedness **A**ware **M**ultitask (TRAM) generative model is shown using plate notation in Figure 3.3. The TRAM learning objective to maximize is:

$$
\begin{aligned}
P_{TRAM}(G_{1:K}|D_{1:K}, \boldsymbol{\mu}) &\propto P(G_{1:K}|\boldsymbol{\mu}) \prod_{i=1}^{K} P(D_i|G_i) \\
P(G_{1:K}|\boldsymbol{\mu}) &\triangleq \frac{1}{Z_{\boldsymbol{\mu}}} \prod_{i=2}^{K} P(G_i) \prod_{j=1}^{i-1} P(G_i|G_j)^{\mu_{ij}} \quad .
\end{aligned}
\tag{3.3}
$$

The key benefits of the TRAM framework are that it:

- Introduces a task-relatedness metric that allows explicit control over information sharing among tasks.
- Subsumes MTL (all elements of $\boldsymbol{\mu}$ are 1) and STL (all elements of $\boldsymbol{\mu}$ are 0).
- Includes existing application-specific models as discussed in Section 3.4.2.
- Provides a convenient mechanism for incorporating task-relatedness in any multitask network structure learner, such as our Bayesian network learner discussed in the next section.

This framework is general enough to cover any network structure learning algorithm that enforces bias between pairs of tasks. Extensions would be required to cover higher-order relationships among tasks, such as describing task-relatedness as a Markov random field with appropriate higher-order potential functions to penalize differences among related tasks.

3.4.1 Multitask Learning of Bayesian Networks

Our novel task-relatedness aware multitask Bayesian network structure learning algorithm illustrates the use of the framework. To apply the objective function in Equation 3.3 to multitask Bayesian networks we define $P(D_i|G_i)$ as the Bayesian likelihood score $P(D|G) = \int P(D|G,\theta)P(\theta|G)d\theta$. The prior over structures encodes the bias toward similar structures by penalizing differences in edges among tasks:

$$P(G_i|G_j) \triangleq \frac{1}{Z_{ij}}(1-\alpha)^{\Delta(G_i,G_j)} \ ,$$

where Δ is a graph distance metric. The parameter $\alpha \in [0,1]$ controls the relative strength of fit to data versus bias toward similar models. When $\alpha = 0$, the objective function is equivalent to learning the tasks independently. When $\alpha = 1$ the only solutions that produce a non-zero probability are those in which $\Delta(G_i, G_j) = 0$, in other words all structures must be identical. The parameters are always inferred independently for each task.

Any graph distance metric can be used for Δ depending on the desired definition of structure similarity. If all D_i come from analogous random variables, the distance metric can be a simple graph edit distance. In our experiments, we use edit distance (the number of edge additions, deletions or reversals necessary to change G_i into G_j).

Optimization of the multitask Bayesian network structure learning objective proceeds by searching over the space of directed acyclic graphs (DAG) for a high-scoring set of DAGs. We follow a commonly used search heuristic, greedy search, which starts from an initial structure and then iteratively makes the best change (the addition, deletion or reversal of a single edge) to the network structure until no further improvements can be made. The best change is the one that gives the greatest improvement in score. We are optimizing several network structures simultaneously, therefore one edge in each task can be changed at each iteration. Incorporating the task-relatedness metric does not incur any computational cost above standard multitask learning.

3.4.2 Special Cases

Now we show how the framework subsumes two application-specific examples from the literature. The dynamic Bayesian network structure learning algorithms with inter-time segment sharing in Husmeier, Dondelinger, and Lèbre (2010) and Dondelinger, Lèbre, and Husmeier (2010) can be written using the TRAM framework as follows. Each time segment is a task i for which they learn a graph G_i that is biased toward having a similar structure to the graph of the previous time segment G_{i-1}. The structure prior is $P(G_{1:K}) = P(G_1)\prod_{i=2}^{K}(1/Z_i)\exp(-\beta\Delta(G_i, G_{i-1}))$, where β is a hyper-parameter and $\Delta(G_i, G_j)$ is the Hamming distance between edge sets. To fit into our framework, we write the prior according to Equation 3.3 with the task-relatedness metric defined as $\mu_{ij} = 1$ for $i = \{2 \dots K\}$ and $j = i - 1$, and $\mu_{ij} = 0$ otherwise.

3.5 Experiments on Synthetic Data

We empirically evaluate our TRAM Bayesian network learning algorithm on synthetic and real-world data. For comparison, we also learn each network independently with single-task learning (STL) and learn a single network structure for all contexts (AVG), so named because this assumes that there is some "average" network that is representative of all tasks. Note that AVG learns the same structure for all tasks, but the parameters are independent of the other tasks. We also compare against a standard multitask learning algorithm (MTL) that assumes all tasks are equally related (all $\mu_{ij} = 1$) (Niculescu-Mizil and Caruana, 2007). For these experiments, we use greedy structure search, starting from an empty network and use a Bayesian score. For TRAM and MTL, we tune the strength parameter, α, with a small holdout set (10% of the training data). All reported results are averaged over 10-fold cross validation.

3.5.1 NetSim Data

We use benchmark data from Smith et al. (2011) which is generated from known networks to simulate rich realistic functional magnetic resonance imaging (fMRI) data. The purpose of this benchmark data is to provide a means to compare how well various algorithms identify the underlying dependencies among variables. They generate data using a hemodynamic response model (Friston, Harrison, and Penny, 2003). We quantize the given continuous data into binary and fit multinomial functions when learning the networks. We use the benchmark data for 50 synthetic subjects with 200 training samples per subject from 50-node networks. The given network structures for all subjects are identical (the functional relationships between nodes are subject-specific) but we are interested in models where the structure differs. Therefore, we modify the structures by re-labeling various numbers of nodes for some tasks and then combining those re-labelings for other tasks. For example, in the first column of Figure 3.4 the adjacency matrix for a given network is used as the first task. To create a related task, we swap the node labels between a few pairs of nodes thus changing some edges of the adjacency matrix as seen in the next task. Re-labeling

produces isomorphic networks, so that we can use the data provided and maintain the general properties of each network while giving different-looking network structures to the structure learning algorithms. TRAM is not given the true measure of similarity between tasks, instead we set $\mu_{ij} = 1$ for each pair of tasks i, j with an edge in the task-relatedness topology, and 0 otherwise. Figure 3.4 shows the ground truth networks derived from the benchmark data.

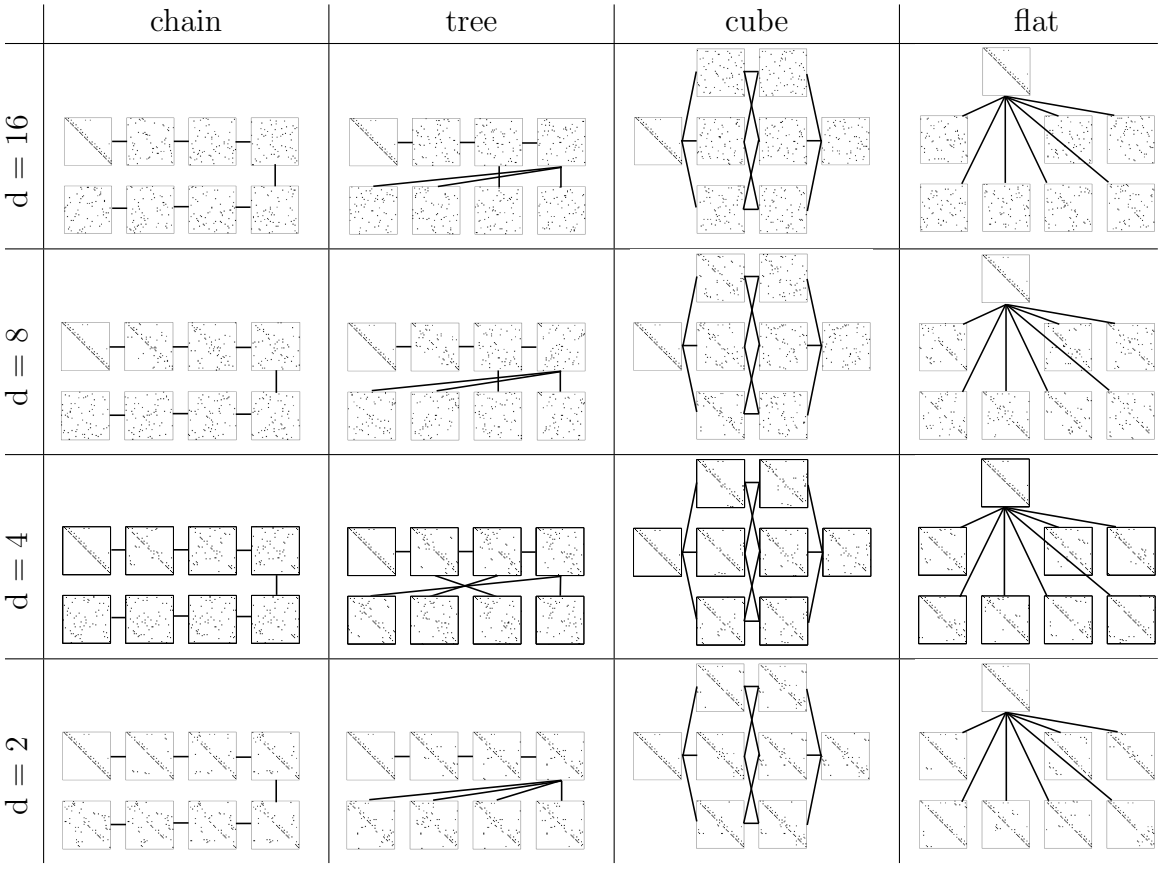


Figure 3.4: Ground truth of NetSim data networks (for one example fold). Each square node in the task-relatedness graph represents a task. The square itself is an image of the adjacency matrix of the ground truth network where dots in the images represent directed edges in the ground truth network. The lines between nodes in the task-relatedness topology indicate that the two tasks are similar with $\mu_{ij} = 1$. Note that when fewer than 8 tasks are used to learn models, we take the first 2, 4, or 6 tasks from the set of 8 tasks.

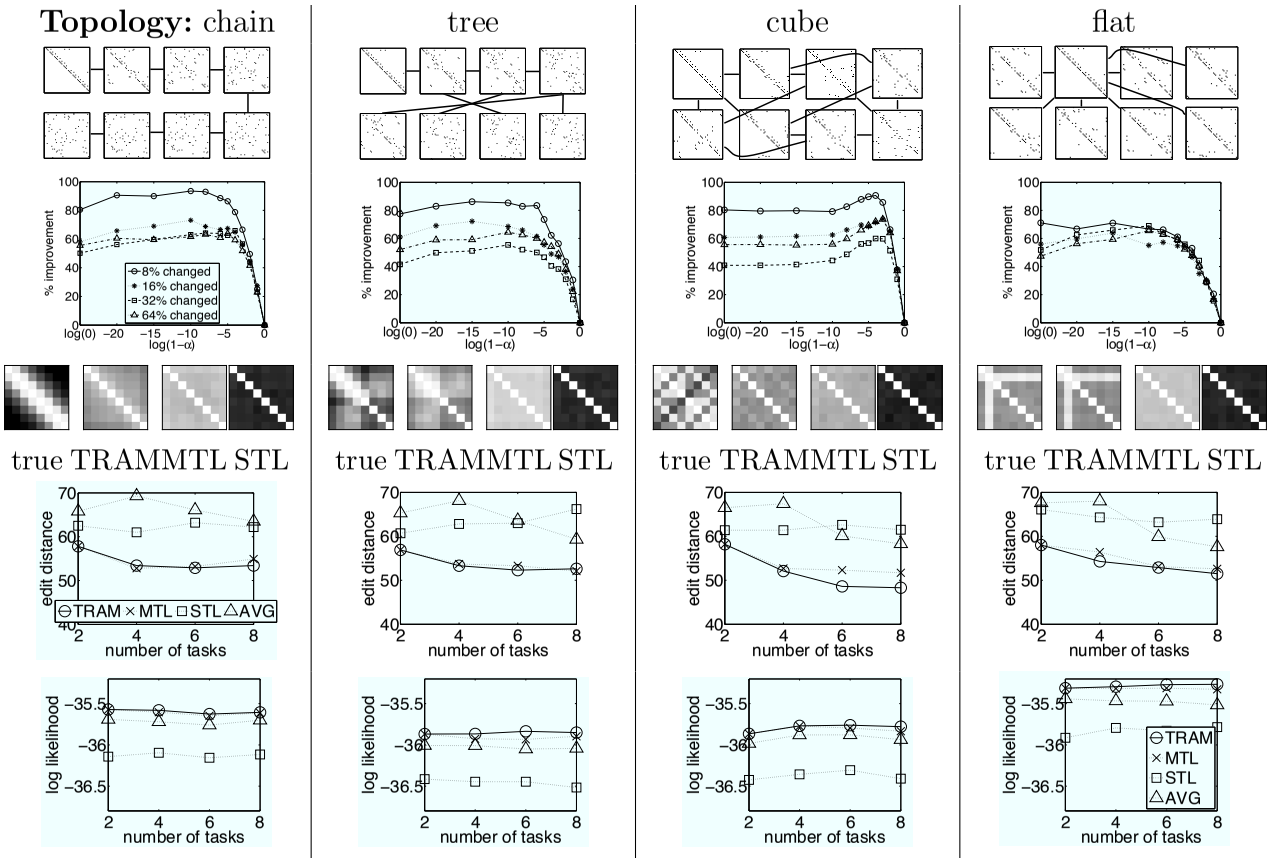


Figure 3.5: NetSim data results. **Top row:** Our generated task-relatedness topologies. Each square node in the topology graph represents a task. The square itself is an image of the adjacency matrix of the ground truth network where dots in the images represent directed edges in the ground truth network. The lines between nodes in the task-relatedness topology indicate that the two tasks are similar with $\mu_{ij} = 1$. **Second row:** TRAM's percent improvement in likelihood on holdout data over STL, across values of α for various levels of true task similarity (% changed). **Third row:** Similarity between tasks for the *true* networks and learned networks from TRAM, MTL and STL for 8 tasks as measured by graph edit distance. White squares mean < 30, black squares mean > 100. **Fourth row:** Edit distance (down is good) of learned networks to ground truth for the four algorithms for 2, 4, 6, and 8 tasks. **Bottom row:** Score of learned networks (up is good) for the four algorithms for 2, 4, 6, and 8 tasks.

3.5.2 Overall Results on NetSim Data

We vary the number of differences between tasks, the number of tasks, and the topology of task-relatedness. Figure 3.5 shows a summary of the types of comparisons made between algorithms; more detailed results follow after this general discussion. The second row of Figure 3.5 shows the average percent improvement in likelihood of holdout data for TRAM over STL. On the x-axis, we vary the strength parameter α on a log scale. When $\alpha = 0$, the tasks are learned independently (the right end of the plot). As we move across the plot to the left, the bias toward learning similar models increases until $\alpha = 1$ at the left end of the plot, where all structures learned are identical to each other. Each line in the plot corresponds to a generative process of node re-labeling that changes the node label for the given percentage of nodes in the network, where high percentages mean there are more differences in the true networks between tasks. As expected, the plots show that when the true networks are most similar (8% changed), the performance gained by TRAM over STL is greatest. As the number of true differences between networks increases, biasing the models toward each other is still a large improvement over STL, but if the strength parameter gets too high then performance degrades.

The bottom three rows of plots in Figure 3.5 compare the performance of all algorithms on the datasets with 32% of nodes relabeled (other results show similar trends and are omitted for space). The row of grayscale images show the similarity among task-specific networks as measured by graph edit distance. For example, in the true networks for the *chain* topology, we see that the first task is increasingly dissimilar to the other tasks as we look across the the top row. STL learns networks that are highly dissimilar to each other while MTL and TRAM learn networks that are more similar, reflecting the bias in these algorithms. TRAM is the only algorithm that reflects the patterns of task similarity given by the true networks.

Perhaps more importantly, the bottom two rows of Figure 3.5 indicate that TRAM learns models that are as close to ground truth as MTL and always better than STL and AVG. We did not perform experiments on training set size because existing

literature has well documented that multitask algorithms are most beneficial on small datasets (Niculescu-Mizil and Caruana, 2007). The NetSim data provides 200 samples per task which we find is insufficient for good single-task learning, making the data a good candidate for multitask learning. We ran experiments with smaller amounts of training data and found those results to be consistent with existing literature.

3.5.3 Detailed NetSim Results

Sensitivity curves show how the performance of the learning algorithm varies as the parameter α is varied for a given set of data. The plots in Figures 3.6 through 3.9 show the average percent improvement in log-likelihood on holdout data (or edit distance to ground truth) for TRAM (or MTL) over STL. On the x-axis, we vary the strength parameter α on a log scale. When $\alpha = 0$, the tasks are learned independently (the right end of the plot). As we move across the plot to the left, the bias toward learning similar models increases until $\alpha = 1$ at the left end of the plot, where all structures learned are identical to each other. Each line in the plot corresponds to a particular setting of d, where high values for d mean there are more differences in the true networks between tasks. All NetSim data is for networks with 50 variables and 200 samples per task.

The learned networks should reflect the amount of similarity that was assumed through the given task-relatedness metric. By looking at results in Figure 3.10, MTL clearly overly-constrains the learned networks to be similar even when they should not be. MTL produces networks that are always more similar to each other than they actually are, while STL often produces networks that are more different than they should be. TRAM produces relatively similar networks, even when there should be more differences, yet it allows for more variation in the number of differences than either MTL or STL provide.

The TRAM, MTL and AVG algorithms are directly compared to each other in terms of edit-distance to ground truth and fit to validation data. Figure 3.11 shows the edit distance results. Figure 3.12 shows the fit to validation data.

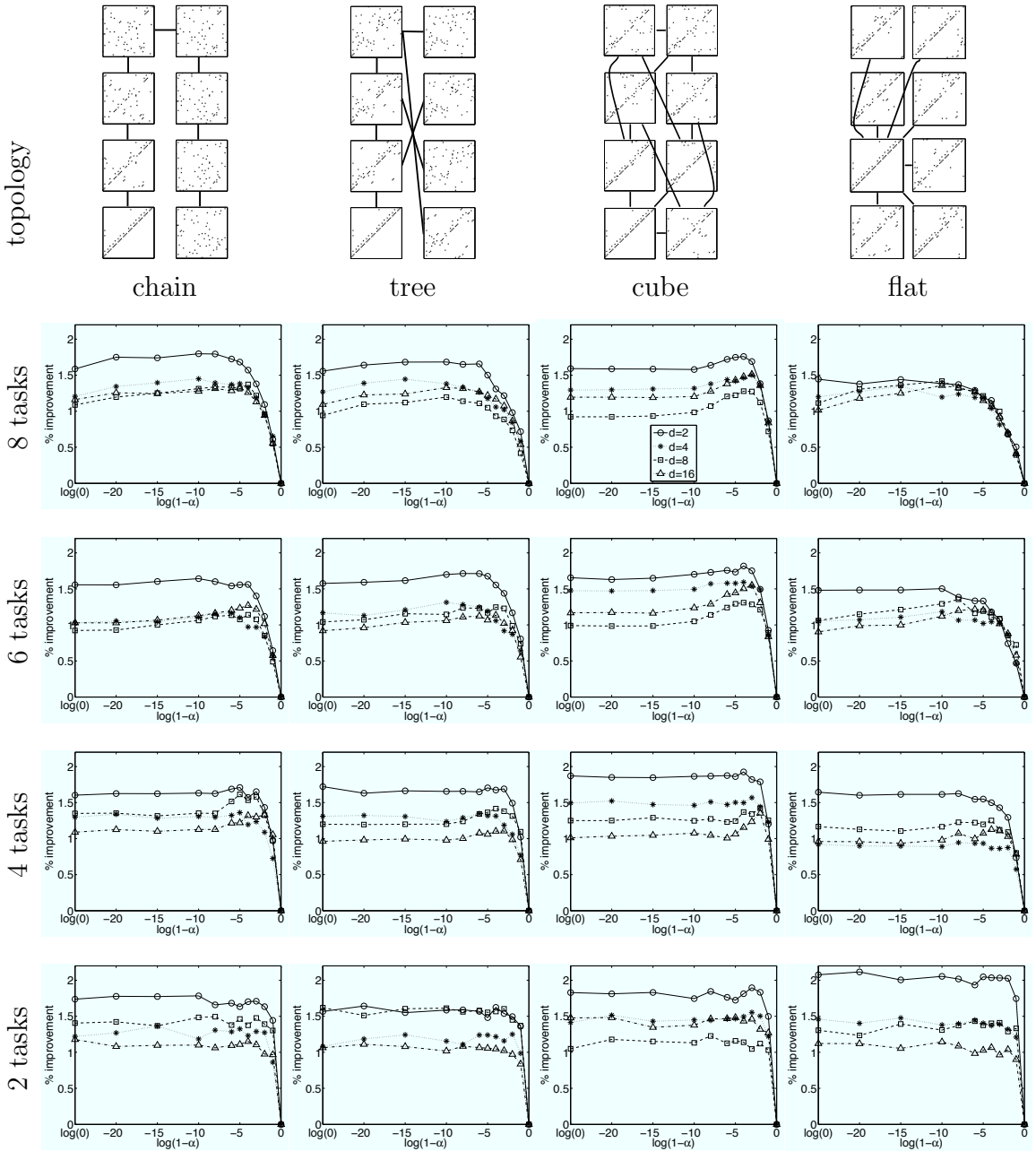


Figure 3.6: **TRAM sensitivity curves as fit to holdout data:** TRAM's percent improvement in log-likelihood on holdout data over STL, for various values of α. The top row (topology) shows our generated task-relatedness topologies. Each square node in the topology graph represents a task. The square itself is an image of the adjacency matrix of the ground truth network where dots in the images represent directed edges in the ground truth network. The lines between nodes in the task-relatedness topology indicate that the two tasks are similar with $\mu_{ij} = 1$.

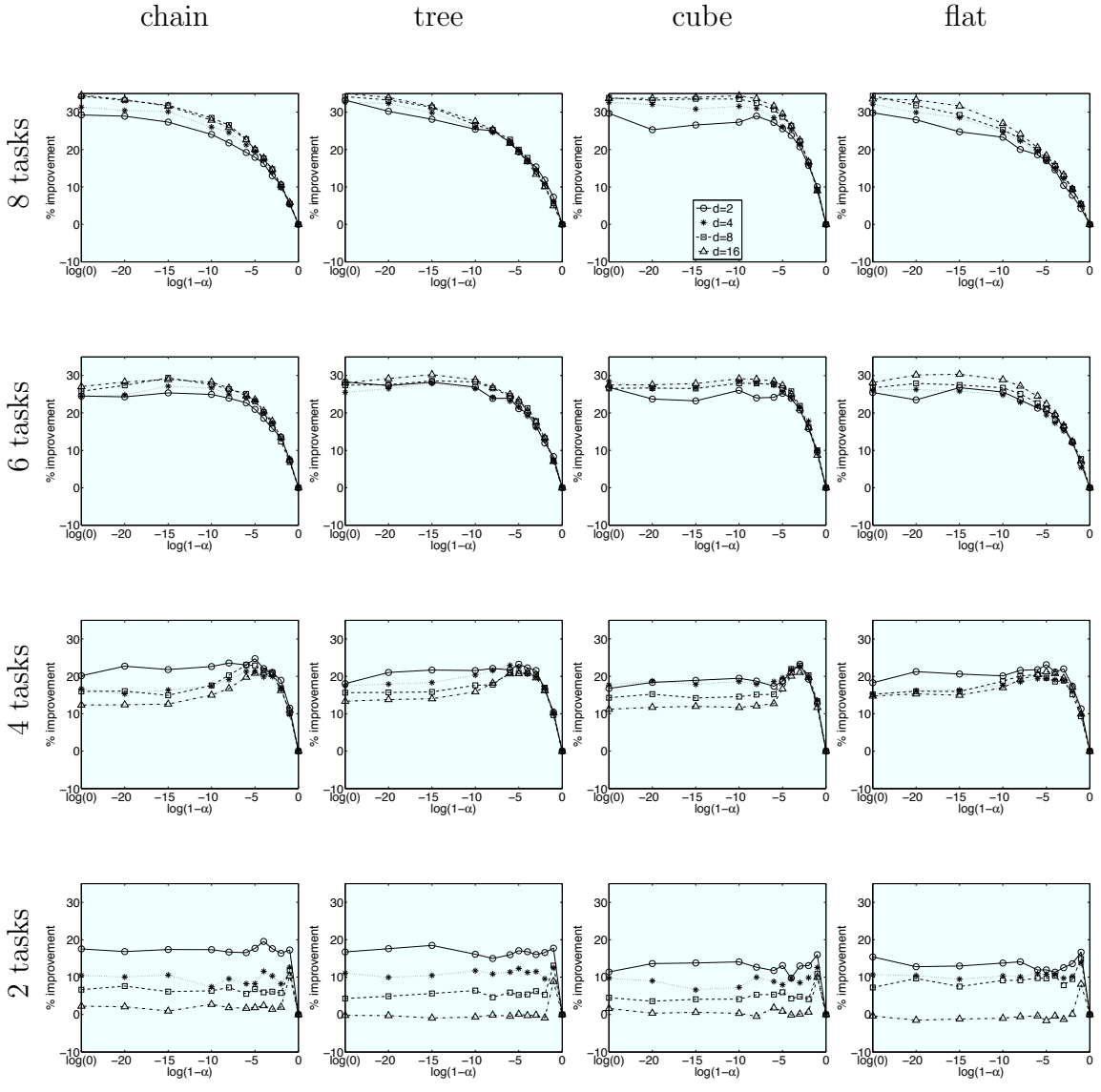


Figure 3.7: **TRAM sensitivity curves as edit distance:** TRAM's percent improvement (up is good) in edit-distance to ground truth over STL, for various values of α. On the x-axis, we vary the strength parameter α on a log scale. When $\alpha = 0$, the tasks are learned independently (the right end of the plot). As we move across the plot to the left, the bias toward learning similar models increases until $\alpha = 1$ at the left end of the plot, where all structures learned are identical to each other. Each line in the plot corresponds to a particular setting of d, where high values for d mean there are more differences in the true networks between tasks.

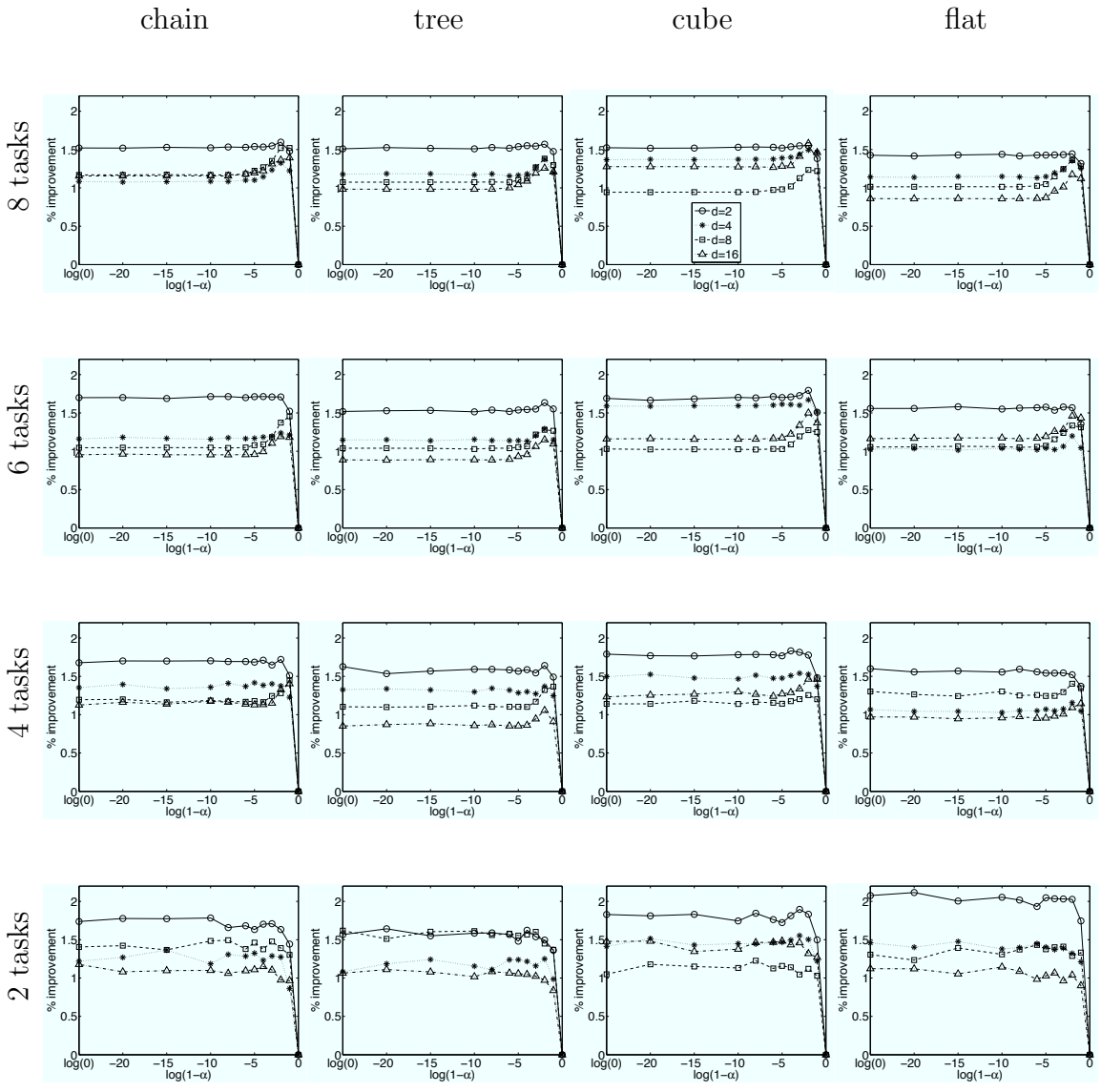


Figure 3.8: **MTL sensitivity curves as fit to holdout data:** MTL's percent improvement (up is good) in log-likelihood on a large test set over STL, for various values of α. On the x-axis, we vary the strength parameter α on a log scale. When $\alpha = 0$, the tasks are learned independently (the right end of the plot). As we move across the plot to the left, the bias toward learning similar models increases until $\alpha = 1$ at the left end of the plot, where all structures learned are identical to each other. Each line in the plot corresponds to a particular setting of d, where high values for d mean there are more differences in the true networks between tasks. Notice that these sensitivity curves look much more similar across the topologies of task-relatedness than do the same curves for TRAM.

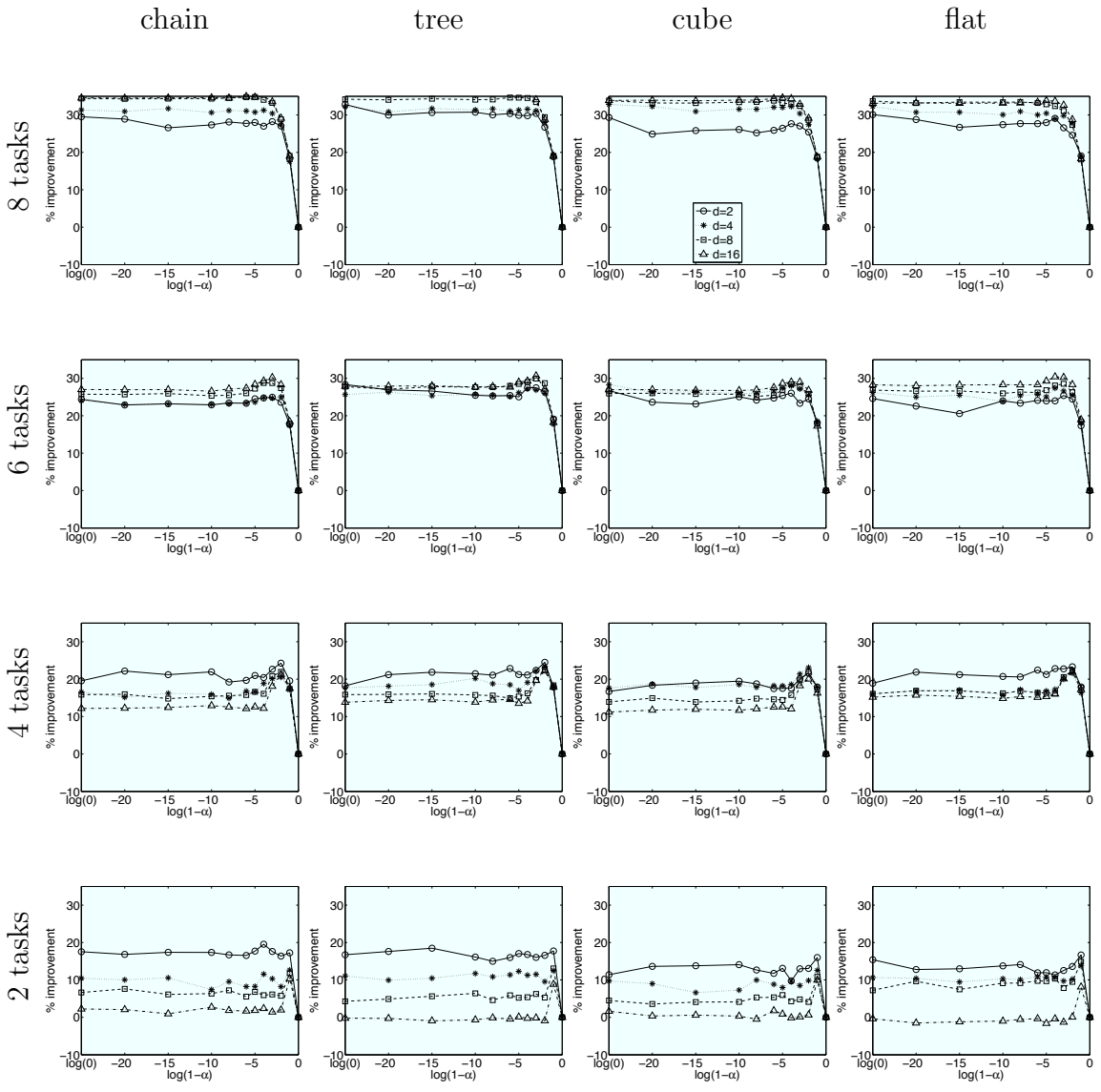


Figure 3.9: **MTL sensitivy curves as edit distance:** MTL's percent improvement (up is good) in edit-distance to ground truth over STL, for various values of α. On the x-axis, we vary the strength parameter α on a log scale. When $\alpha = 0$, the tasks are learned independently (the right end of the plot). As we move across the plot to the left, the bias toward learning similar models increases until $\alpha = 1$ at the left end of the plot, where all structures learned are identical to each other. Each line in the plot corresponds to a particular setting of d, where high values for d mean there are more differences in the true networks between tasks. Notice that these sensitivity curves look much more similar across the topologies of task-relatedness than do the same curves for TRAM.

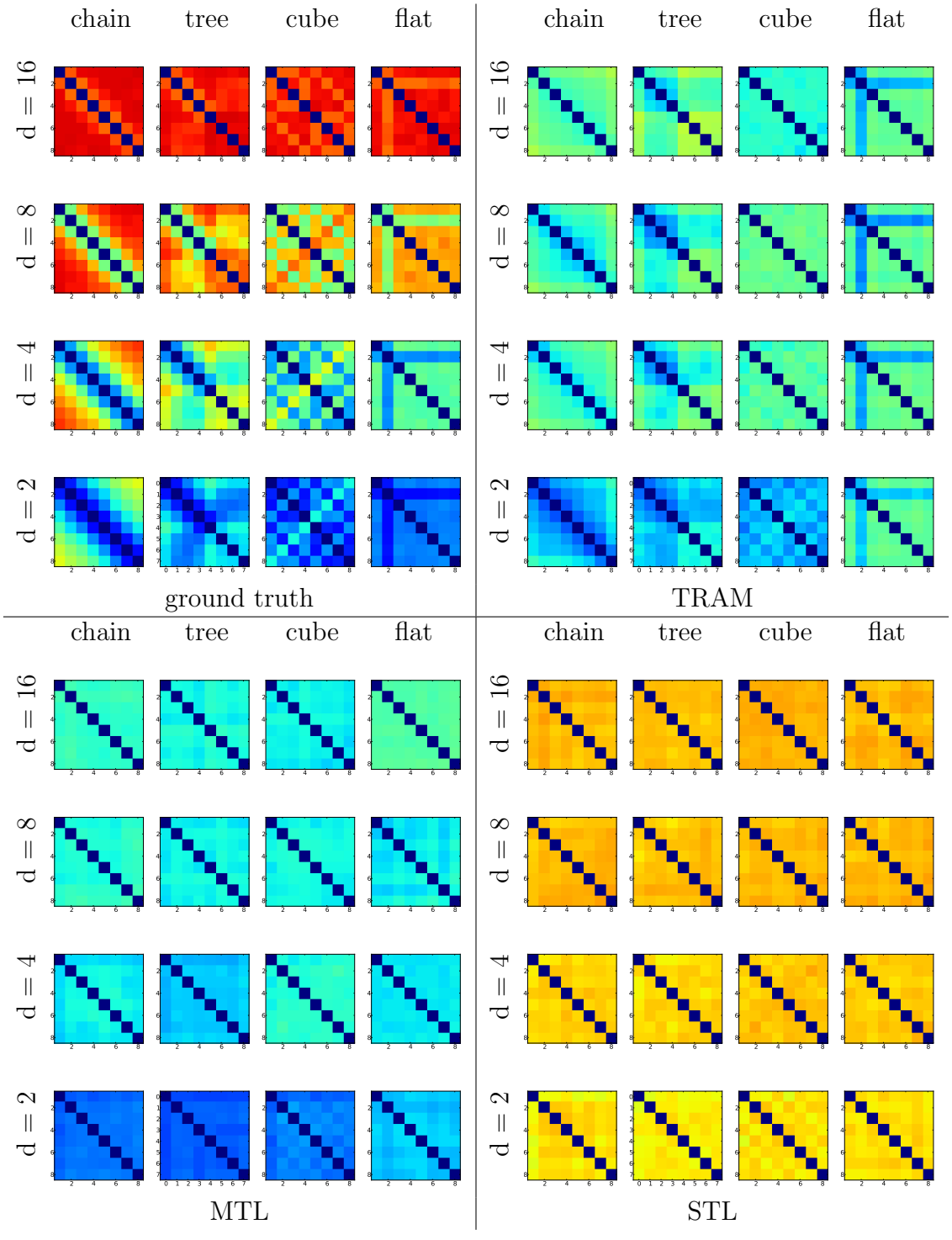


Figure 3.10: **Network similarity.** Top left: Ground truth of NetSim data task-relatedness. A dark blue square at position (i,j) indicates that there are no differences in the true network structures between task i and task j. Red squares indicate 130 edges are different between the two tasks (all heatmaps use the same color scale). The other quadrant of the table show similarity among learned networks for TRAM, STL and MTL.

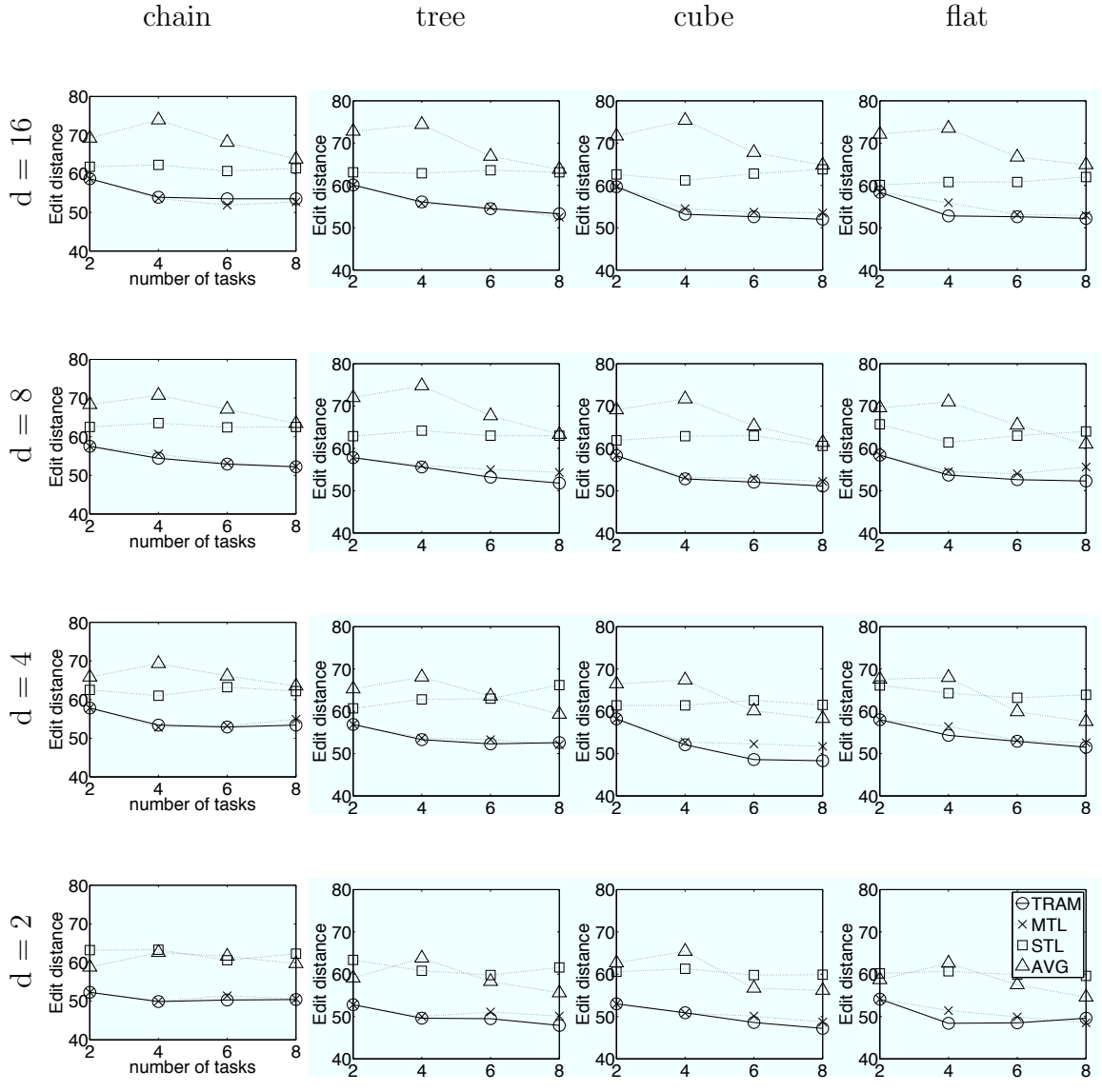


Figure 3.11: Edit distance to ground truth for TRAM, MTL, STL, and AVG averaged over 10 folds. Down is good.

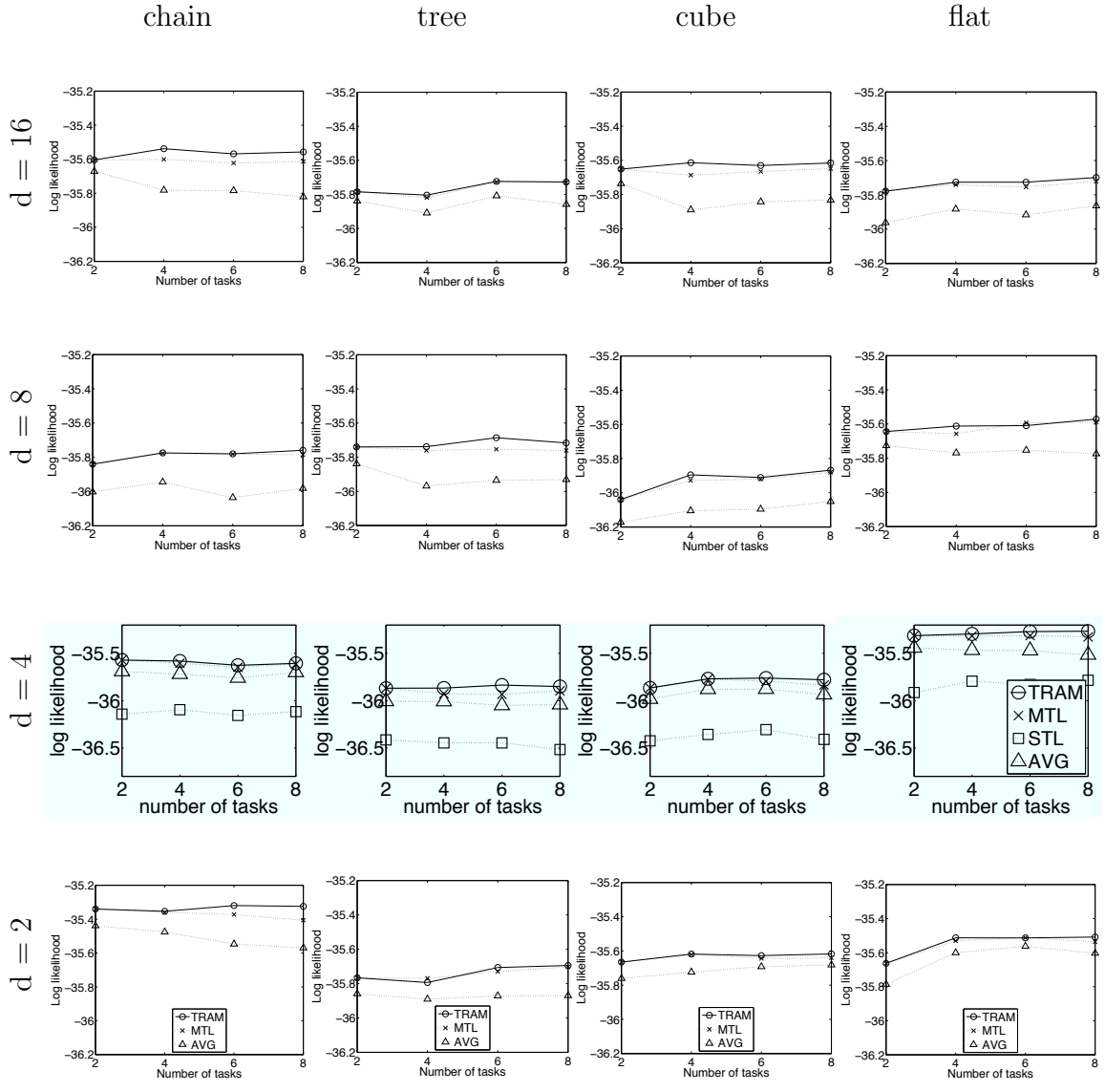


Figure 3.12: Log likelihood on a large test set for TRAM, MTL and AVG, averaged over 10 folds. Up is good.

3.6 Networks Learned from fMRI

Functional MRI measures the activation level of voxels in the brain. Typically, hundreds of such images are collected over the period of time that a subject is in the scanner. We use data from a large schizophrenia study, where 384 volumes are sampled per subject. Voxels are mapped into 150 regions of interest (ROIs) based on the Talaraich atlas (Lancaster et al., 2000). The fMRI data for each ROI in each subject is independently detrended and discretized into four levels of activity. Thus, our data is 150 variables by 384 samples per subject. We use the same algorithms as with NetSim.

3.6.1 Age-Groups as Tasks

A fundamental question for multitask learning in practice is — how do we define a task? While we cannot fully answer the question in this chapter, we do explore how the number of tasks for a fixed dataset affect the performance of the learned models. We experiment with dividing our dataset into various numbers of tasks by grouping subjects into tasks by age. We take 86 subjects from our dataset (the control subjects in the schizophrenia study) and group them based on the age of the subject. Figure 3.13 shows how we create four different learning problems by dividing the dataset into 2, 4, 8, and 16 tasks. The training data is the same across these problems, but the number of tasks is different. We define the task-relatedness values $\mu_{ij} = e^{-(\bar{a}_i - \bar{a}_j)^2/(2\sigma^2)}$ where $\bar{a}_i$ is the average age of subjects in task i and σ^2 is the variance of ages of all subjects. As an example, $\mu_{1j} = [1, .89, .67, .37, .18, .09, .03, .005]$ for the youngest group (task 1) versus tasks j in order of increasing age for 8 tasks. In the discussion of results, this task-relatedness metric is used for the TRAM algorithm. For comparison, we also try binary-valued μ with $\mu_{ij} = 1$ for pairs of tasks i, j that are adjacent to each other in age-order and 0 otherwise. We refer to this variation on the task-relatedness metric as TRAMbin in the following discussion of results.

Results from the age data are shown in Figures 3.15 and 3.14. Plots 3.15(b) and 3.15(c) show TRAMbin and TRAM's sensitivity to the strength parameter α as mea-

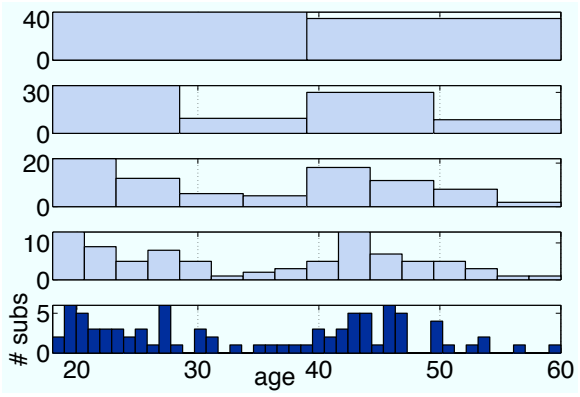


Figure 3.13: **Age groups.** Bins of subjects grouped by age for 2, 4, 6, and 16 tasks. Each box is a task; the width shows the age range and the height shows the number of subjects in the task. The bottom row is a histogram of all subjects.

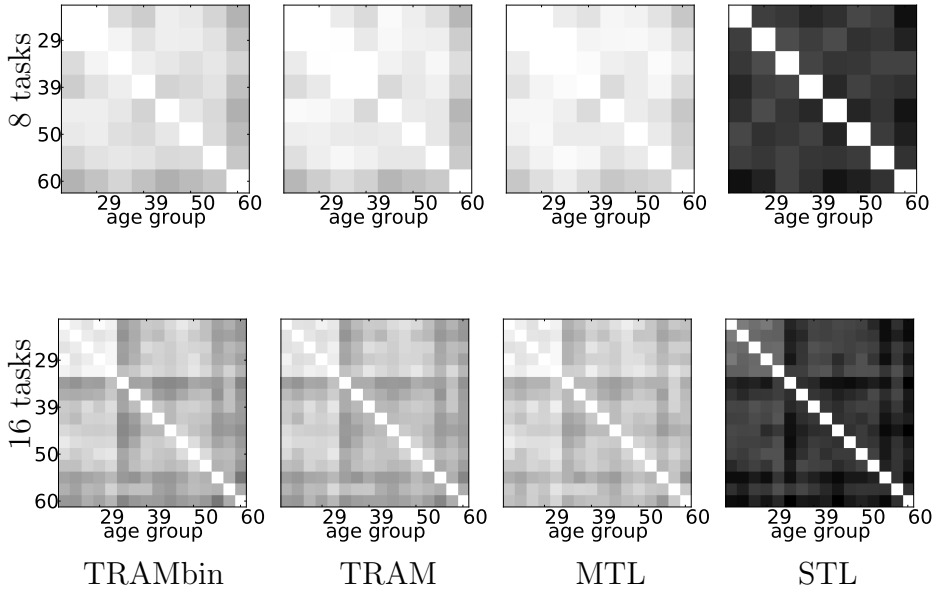
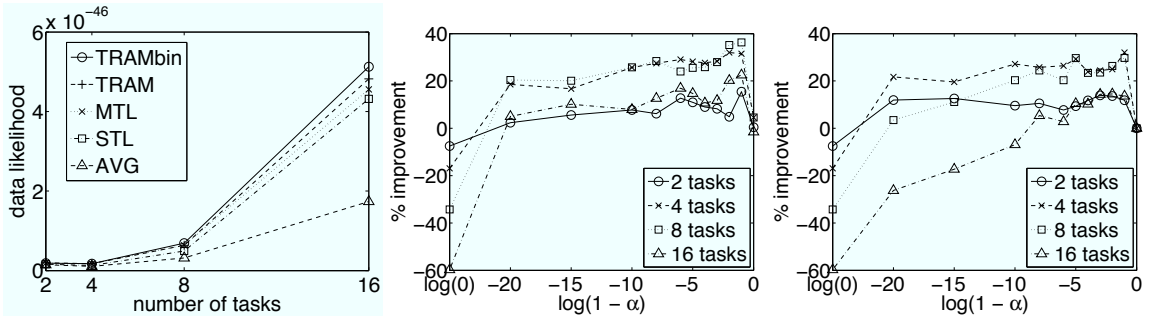


Figure 3.14: **Age data task similarity.** Edit distance between learned task-specific networks. White is <100, black is >300.

sured by the improvement in likelihood on test data versus STL. We see that both are an improvement over STL but TRAMbin appears less sensitive to α. AVG (the left edge of the plot at log(0)) actually causes negative transfer that is increasingly bad as the number of tasks grows. Therefore, some biasing of models helps, but too much degrades performance.



(a) All algorithms (b) TRAMbin (c) TRAM

Figure 3.15: **Age data.** (a) Likelihood of holdout data. All differences between algorithms are significant at p=.05 except for between TRAM, TRAMbin and MTL at 2 and 4 tasks. (b) TRAMbin's increase in performance over STL. (c) TRAM's increase in performance over STL.

Figure 3.15(a) shows the overall comparison between algorithms of data likelihood on test data. Here, the strength parameters of TRAMbin, TRAM and MTL have been tuned on 10% of the training data. We see that splitting the dataset into more tasks improves the performance of all algorithms, even though the underlying dataset is the same. TRAMbin is always the highest performing algorithm, with TRAM and MTL close behind. The lines appear quite close, but the differences are significant everywhere except between TRAMbin, TRAM and MTL for 2 and 4 tasks according to paired t-tests at p=0.05. The improvement in performance of AVG is somewhat surprising because tasks share the same structure. However, recall that the parameters of different tasks are independent, therefore splitting data into tasks also allows AVG to fit parameters to specific tasks.

Interestingly, TRAMbin and TRAM find quite a few network edges that are different for the oldest age group than for the others (see Figure 3.14). All algorithms find more differences from the oldest group to the others, but for TRAM and TRAMbin many of these edges exist with high robustness in the oldest group, but none of the others. Other edges exist with high robustness in the younger groups but never in the oldest group.

Table 3.1: **Patient type.** Mean log-likelihood of holdout data. Bold indicates the best score, * indicates that the score is significant by t-test at p=.05.

task	# subs	TRAM	MTL	STL	AVG
1 chronic control	86	**-108.59***	-108.67	-108.75	-108.84
2 followup control	15	-112.78	-112.80	-113.22	**-111.92***
3 1st episode control	32	-110.39	-110.33	-110.63	**-110.14***
4 1st episode	14	-114.67	-114.62	-114.94	**-113.78***
5 followup	15	-110.64	-110.66	-111.17	**-109.94**
6 chronic	74	**-111.02**	-111.09	-111.37	-111.13
aggregate		-111.35	-111.36	-111.68	**-110.96***

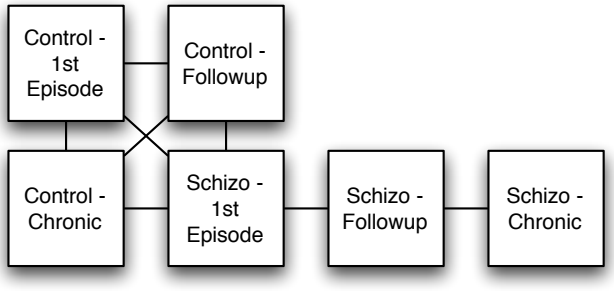


Figure 3.16: **Patient type.** Metric of task-relatedness. Edges between tasks represent $\mu = 1$.

3.6.2 Stages of Schizophrenia as Tasks

We divide the subjects from the schizophrenia study into six tasks based on the degree of the disease of the patients, a clinical variable that can take on one of three values: first episode baseline, first episode followup, or chronic in increasing order of length of time with the disease. Each group of patients has a corresponding group of control subjects (see Table 3.1). Our task-relatedness metric assumes that all of the tasks with control subjects are similar to each other, and the other tasks are related in increasing order of length of time with the disease, as shown in Figure 3.16. A 10-fold cross validation of held-out subjects in each task is performed. In this data set, we hold out the full set of samples from individual subjects. This evaluation tells us how well the learned model for each task fits a population of subjects represented by the corresponding task.

Results in Table 3.1 indicate that sharing information among tasks improves the

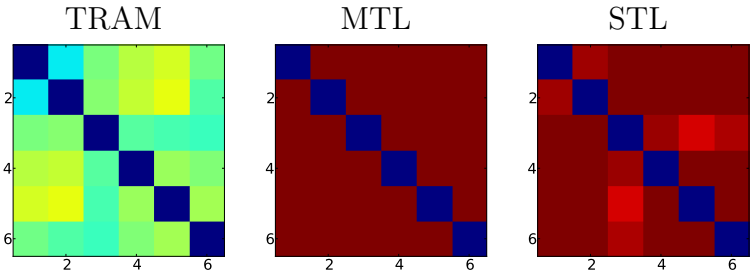


Figure 3.17: **Patient type.** Learned similarity among networks for TRAM, MTL and STL. A dark blue square at position (i,j) indicates that there are no differences in the network structures between task i and task j. Dark red squares indicate 300 edges are different between the two tasks.

fit to holdout data, to the point that learning identical structure for all tasks (AVG) performs the best on 3 of 6 tasks. However, learning identical structures would give no insight to the end-user about changes in the brain functional network of patients with schizophrenia. TRAM performs second-best in terms of fitting the data and still allows for some differences between tasks. Figure 3.17 shows that TRAM learns far fewer differences between pairs of tasks than either MTL or STL. With fewer differences, it is easier to make comparisons between networks.

3.6.3 Tasks Defined by Medication Type

Often we want to look at populations of subjects with a certain type of mental illness, but we expect that different drug treatments will have an effect on brain activity. To address this, we divide the subjects from the schizophrenia study into seven tasks. One of the tasks is the group of control subjects. The other tasks are schizophrenic patients divided into six groups representing the medication they are taking (Figure 3.18).

Figure 3.19 shows improvement in data likelihood versus STL. The improvement is greater for TRAM than MTL. AVG always performs worst of all of the algorithms. As expected, the improvement of TRAM over STL is greatest when there is the least data (Figure 3.19(b)). All algorithms learn networks that show high variation between the control group and all other tasks (see Figure 3.20). The networks learned by TRAM in particular show that networks for subjects on Clozapine type drugs are

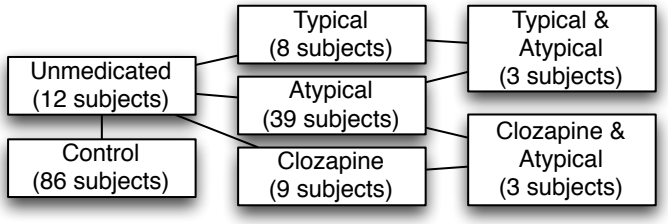
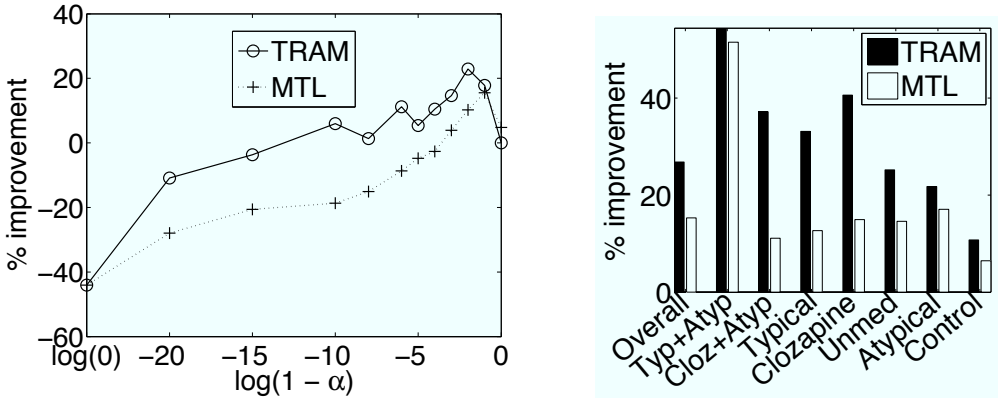


Figure 3.18: **Task relatedness for Drug data.** Each square is a task, edges between tasks represent $\mu = 1$.



(a) Sensitivity curve (b) Per-task improvement

Figure 3.19: **Drug dataset results.** (a) Increase in performance over STL across values of the strength parameter for TRAM and MTL. (b) Improvement over STL for tuned TRAM and MTL. Note tasks are ordered by increasing number of subjects.

most similar to those on drug combinations including Clozapine than they are to any other group. On the other hand, for subjects on Typical type drugs TRAM learns brain networks that are highly similar to all other groups.

3.7 Discussion

Task-relatedness knowledge can improve both the robustness of learned networks and address human objectives. A natural question is how to define the task-relatedness metric $\boldsymbol{\mu}$. Previous application specific algorithms employed binary task-relatedness weights. Our experiments support the intuition that binary weights that give the topology of tasks that are directly related is preferable to fine-tuning real-valued

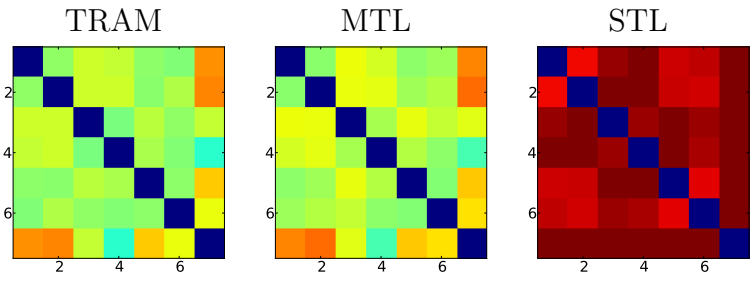


Figure 3.20: **Drug dataset results.** Learned similarity among networks for TRAM, MTL and STL. A dark blue square at position (i,j) indicates that there are no differences in the network structures between task i and task j. Dark red squares indicate 300 edges are different between the two tasks.

weights. A similar question is how fragile algorithms become when domain knowledge is poor. In practice, we found that using a misleading $\boldsymbol{\mu}$ causes TRAM to produce results equivalent to MTL, which is not surprising because MTL is a case of TRAM with a fixed $\boldsymbol{\mu}$. Theoretical definitions of good task-relatedness knowledge would be interesting future work.

Another important direction of research is to estimate task-relatedness from data. However, the data-driven approach answers a different question than addressed in this chapter. TRAM is explicitly incorporating a human-specified objective that cannot be ascertained directly from data. This concept of incorporating human preferences into learning objectives is an active research trajectory in unsupervised learning (Chang et al., 2009; Dasgupta and Ng, 2009). In these problems data-driven measures of goodness of the learned model are insufficient and can only be addressed by incorporating human objectives. This chapter introduced the concept of human-specified objectives to multitask network structure learning.

Learning a large number of Bayesian network tasks efficiently is also important, though not explicitly addressed in this chapter. Currently, no multitask Bayesian network learning algorithm adequately addresses this. Task-relatedness knowledge may be useful to break up the problem into manageable-sized chunks. It would also be interesting to investigate transferring bias among only the parts of the Bayesian network model that we are most interested in and allow a more efficient independent

search for other parts of the model.

3.8 Conclusion

Naive assumptions about task-relatedness limit the effectiveness of multitask learning algorithms. Relaxing these assumptions in the objective function through a task-relatedness metric is a necessary step in improving the performance of multitask network structure learning algorithms. This chapter introduces a general framework, TRAM, for incorporating domain knowledge into multitask network structure learning objectives. This framework allows a natural and flexible way to represent domain knowledge. Specifically, we develop a novel multitask Bayesian network structure learning algorithm with TRAM. Empirical evaluation shows that leveraging domain knowledge produces models that are both robust and reflect a domain expert's objective.

Chapter 4

Bayesian Discovery of Multiple Bayesian Networks

Transfer learning improves the robustness of learned networks by leveraging data from related tasks, as the previous chapter shows. Existing transfer learning algorithms for Bayesian network structure learning give a single maximum a posteriori estimate of network models. Yet, many other models may be equally likely, and so a more informative result is provided by Bayesian structure discovery. Bayesian structure discovery algorithms estimate posterior probabilities of structural features, such as edges. This chapter presents transfer learning for Bayesian structure discovery; providing a more comprehensive view of the many possible solutions to the problem of identifying the shared and unique structural features among related tasks. Efficient computation requires that the transfer learning objective factors into local calculations, which we prove theoretically. Empirical results on benchmark data demonstrate that compared to single-task learning, transfer learning is better able to positively identify true edges.

4.1 Motivation

The multitask Bayesian network learning algorithms presented in the previous chapter have two major limitations: 1) they use *heuristic* search over the space of sets

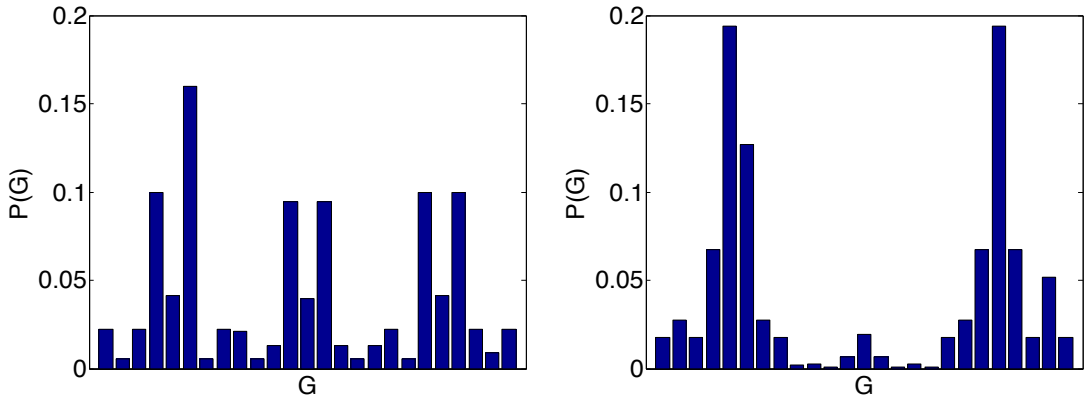


Figure 4.1: Example posterior distributions estimated from sample data generated by two different 3-node Bayesian networks.

of graphs; and 2) produce a *single* point maximum a posteriori model. Yet, there may be many other solutions of similar likelihood and therefore a point solution will not give a full picture of likely relationships among variables. For example Figure 4.1 shows the posterior likelihood of networks from sample data generated by two different Bayesian networks. In both cases, there is not just one likely network that dominates the posterior likelihood, and so different solutions made up of various combinations of conditional dependencies could be nearly equally likely. In terms of learning multiple biological networks, the danger in learning only a single representative network for each task, is that it gives no hint to the domain expert that other dependencies may indeed be highly likely. Therefore, we would like to learn a posterior distribution over solutions and extract meaningful summary statistics, such as the expectation of a particular edge (dependency). Such algorithms exist for learning individual Bayesian networks and are referred to as *network discovery* (Friedman and Koller, 2003; Koivisto, 2006). The focus of this chapter is on extending these algorithms for transfer learning of multiple networks.

Extending Bayesian network discovery algorithms for transfer learning is not a trival problem. Algorithms for estimating the posterior probability distribution for a single network generally fall into two categories: those that search over *structure* space and those that search over *order* space. Structure-space algorithms make small local

changes to the learned graph structure (typically, the addition, removal or reversal of a single edge). For these small changes, updating the likelihood of the graph, and therefore the posterior distribution, is fast but covering the full space of structures can be slow and can get stuck in local maxima (Grzegorczyk and Husmeier, 2008; Madigan, York, and Allard, 1995). Extending those structure-search algorithms to multiple tasks would explode the model search space exponentially, exacerbating convergence issues. On the other hand, order-search algorithms exploit the tractability of calculating posteriors given a fixed ordering of the variables (described in more detail later). Node order dictates which nodes are allowed to be parents to any given node, and therefore changes in node order are more global than structure-space changes. There are relatively efficient algorithms for calculating exact posteriors of structural features (Koivisto and Sood, 2004; Parviainen and Koivisto, 2009) or approximate posteriors (Friedman and Koller, 2003; Niinimaki, Parviainen, and Koivisto, 2011) that have been shown to be faster than structure-search. However, there is no *structural* prior provided in these order-space formulations; instead there is a prior over orders. We do not want to impose transfer at the level of *orders*, but rather at the level of structures (a particular edge appearing in one task will be preferred in other tasks).

Our main challenge, therefore, is to incorporate a structural bias term into the order-search formulation. With such a bias term, we can impose a transfer bias to learn more robust networks, while leveraging the most efficient Bayesian discovery algorithms that currently exist. Our major contribution is efficiently incorporating a structural bias into the order-conditioned network discovery formulation. We prove that our transfer formulation factors into local calculations. Thus, we provide the first transfer algorithm that can calculate *exact* posteriors for multiple networks of moderate size. We are also the first to show how transfer can incorporated into state-of-the-art order-search approximation algorithms for larger networks.

Our contribution is a proof that structure bias can be efficiently incorporated into order-conditioned Bayesian structure discovery: a necessary requirement for using the efficient algorithms of network discovery (Friedman and Koller, 2003; Koivisto and Sood, 2004). This is a finding that can impact many structure discovery problems.

We give a specific formulation of *multitask* Bayesian network discovery that uses the structure bias to transfer information among tasks. We further show that we can take a Bayesian approach to average over all possible settings of the transfer parameter rather than needing to select this parameter. Empirical results on networks, of size 8 variables and 37 variables, indicate that our transfer approach learns posterior probabilities that are closer to the optimal values than single-task learning algorithms.

4.2 Related Work

Network structure discovery in the face of limited data is an extensively studied problem. With limited data, the posterior probability of even the optimal network may be quite small; however, Friedman and Koller (2003) show that the marginal posterior probabilities over subgraphs or structural features can be quite high given the same data. They propose the order-MCMC algorithm for estimating such posterior probabilities. Koivisto and Sood (2004) give a dynamic programming method for calculating exact posterior probabilities of network features conditioned on orders. Further improvements are made to make the approach more memory efficient (Parviainen and Koivisto, 2009) and to produce partial-order MCMC (Niinimaki, Parviainen, and Koivisto, 2011). We show how to extend these single-task learning approaches to the transfer learning problem.

Starting from existing multitask algorithms (Niculescu-Mizil and Caruana, 2007; Oyen and Lane, 2012), we derive an inductive bias toward similar structures among related tasks. However, the solutions to the multitask problems in these papers and the previous chapter are found through heuristic search, producing a point estimate rather than a posterior distribution, which we desire. They also require a parameter that determines the strength of transfer bias.

4.3 Preliminaries

First, we introduce background information about Bayesian structure discovery for learning a single task and then describe maximum a posteriori (MAP) multitask Bayesian network objectives. We combine ideas from both of these approaches to produce Bayesian structure discovery of multitask Bayesian networks.

Bayesian networks compactly describe joint probability distributions by encoding conditional independencies in multivariate data. A Bayesian network $B = \{G, \theta\}$ describes a joint probability distribution over n random variables $\mathbf{X} = [X_1, X_2, \ldots, X_n]$ where G is a directed acyclic graph (DAG) and the conditional probability distributions are parameterized by θ (Heckerman, Geiger, and Chickering, 1995). An edge (X_i, X_j) in G means that the child X_j is conditionally independent of all non-descendants given its parent X_i. The *structure* of the network, G, is of particular interest in many domains as it is easy to interpret and gives valuable information about the interaction of variables.

4.3.1 Structural Feature Discovery

Given a limited amount of data, the posterior probability of any network may be quite small. However, summary statistics regarding structural features of networks may have high posterior even with limited data (Friedman and Koller, 2003). Structural features (such as an edge) can be described by an indicator function f such that for $f(G) = 1$ the feature exists in graph G, otherwise $f(G) = 0$. The posterior probability of the feature is equivalent to the expectation of its indicator,

$$P(f|D) = \sum_G P(G|D) f(G) \ . \tag{4.1}$$

However, this sum can be intractable, as the number of DAGs is super-exponential in the number of variables.

An important insight to making this sum tractable is that we could fix the *order* of the variables. An order, $\prec$, is a permutation on the indices of the variables

$X_{\prec(1)}, X_{\prec(2)}, \ldots, X_{\prec(n)}$ such that parents must precede children in the order, i.e. X_j cannot be a parent of X_i if $\prec(j) \geq \prec(i)$. Given an order, learning optimal parents for each child factors into local calculations, and summing over DAGs consistent with the order is tractable (Buntine, 1991; Cooper and Herskovits, 1992). Friedman and Koller (2003) condition on a node order, and then obtain the unconditional posterior by summing over orders:

$$P(f|D) = \frac{1}{P(D)} \sum_{\prec} P(\prec) \sum_{G \subseteq \prec} P(D|G) P(G|\prec) f(G) \ . \tag{4.2}$$

Note that these two formulations for $P(f|D)$ in Equations 4.1 and 4.2 are not the same, as most DAGs, G, will be consistent with multiple orders, $\prec$. Typically, this formulation produces an acceptable bias in favor of simpler structures.

Koivisto and Sood (2004) give an efficient method for calculating the sum in Equation 4.2. The approach is rather involved, so we summarize only the key points here. They make several reasonable assumptions, then break the calculation into three steps. First, we describe the modularity assumptions. Modularity refers to the property of network calculations to depend only on local information, rather than global information about the whole network. The modularity assumptions are:

1. **Parameter modularity**: Modularity of the Bayesian network parameters must hold, $P(\theta|G) = \prod_{i=1}^{n} P(\theta_{i,\pi_i}|\pi_i)$ and $P(X = x|G) = \prod_{i=1}^{n} P(x_i|x_{\pi_i}, \theta_{i,\pi_i})$.

2. **Structure prior modularity**: The network model prior must be modular so that $P(G,\prec) = c \prod_{i=1}^{n} P(U_i) P(\pi(X_i))$, where U_i is the set of variables preceding X_i in the order $\prec$ (potential parents of X_i) and c is a normalization constant.

3. **Feature modularity**: The features must be modular, $f(G) = \prod_{i=1}^{n} f_i(\pi(X_i))$ where $\pi(X_i)$ is the parent set of variable i.

The most common feature to look for is a directed edge $u \to v$ s.t. $f = 1$ if $X_u \in \pi_v$, which is clearly modular. If these modularity assumptions hold, then the likelihood over order space factors into local calculations as shown in Equation 4.3 (Koivisto

and Sood, 2004).

$$\begin{aligned} P(f, D|\prec) &= \prod_{i=1}^{n} \sum_{\pi_i \subseteq U_i} P(\pi_i|U_i) P(x_i|\pi_i) f_i(\pi_i) \\ P(f|D) &= \frac{1}{P(D)} \sum_{\prec} P(\prec) P(f, D|\prec) \end{aligned} \tag{4.3}$$

where $\pi_i = \pi(X_i)$ is the parent set of variable i. The unconditional posterior for the features is obtained by summing over orders, using the following steps:

1. Calculate family scores: $\beta_i(\pi_i) = P(\pi_i)P(x_i|\pi_i)f_i(\pi_i)$ for each node i and potential set of parents π_i. The computational complexity of each of these is some function $C(m)$ of the number of samples m. The maximum number of parents allowed for any node is typically fixed to a small natural number, r. Therefore, there are $O(N^{r+1})$ of these functions to calculate for a total complexity of $O(N^{r+1}C(m))$.

2. Calculate local contribution of each subset $U \subseteq V - \{i\}$ of potential parents of i: $\alpha_i(U) = \sum_{\pi_i \subseteq U} P(\pi_i)P(x_i|\pi_i)f_i(\pi_i)$. Using a truncated fast Möbius transform and pre-computed β's, all of the α functions are computed in $O(n2^n)$ time.

3. Sum over the subset lattice of the various U_i to obtain the sum over orders $\prec$. Using dynamic programming, this sum takes time $O(n2^n)$.

The total computational complexity for a single task is $O(n2^n + n^{r+1}C(m))$. This is the exact calculation of the posterior. For large networks, roughly $n > 30$, the exponential term is intractable. In these cases, MCMC simulations give an approximation to the posterior probability, so that $P(f|D) \approx \frac{1}{T}\sum_{t=1}^{T} P(\prec_t)P(f|D,\prec_t)$ for $\prec_t$ sampled from order space (Friedman and Koller, 2003) or partial orders (Niinimaki, Parviainen, and Koivisto, 2011).

4.3.2 Multitask Bayesian Networks

As demonstrated in the previous chapter, multitask Bayesian network learning leverages knowledge among a set of related tasks by applying a bias toward learning similar

networks among the tasks. The underlying assumption is that much of the network is shared among tasks, yet a few differences may exist. We have already shown that by leveraging information among tasks, we can learn more robust networks than would be possible from a single small sample (Niculescu-Mizil and Caruana, 2007). We will apply a similar bias mechanism for network discovery. First we describe the objective function of existing MAP estimate algorithms, which has been shown to be effective at leveraging information. In this chapter, local calculations on a per-variable basis will be calculated. To distinguish indices over variables from indices over tasks, we will always index variables with a subscript and index tasks with a superscript. A set of tasks with datasets $D^{(k)}$ and networks $G^{(k)}$ for $k \in \{1, \ldots, K\}$ can be learned by optimizing:

$$\begin{aligned} &P(\mathcal{G}|\mathcal{D}) = P(G^{(1)}, \ldots, G^{(K)}|D^{(1)}, \ldots, D^{(K)}) = \\ &P(D^{(1)}, \ldots, D^{(K)}|G^{(1)}, \ldots, G^{(K)})\frac{P(\mathcal{G})}{P(\mathcal{D})} \quad . \end{aligned} \tag{4.4}$$

In existing multitask network learning formulations, the joint structure prior, $P(\mathcal{G})$, is used to encode a bias toward similar structures by penalizing differences in network structure among tasks (Niculescu-Mizil and Caruana, 2007; Oyen and Lane, 2012). We can assume that $P(D^{(k)}|G^{(k)})$ is independent of all other $G^{(i)}$ so Equation 4.4 simplifies and the joint prior over structures can be described by pairwise sharing of information among tasks as in Equation 4.5.

$$\begin{aligned} P_{MTL}(\mathcal{G}|\mathcal{D}, \lambda) &= \frac{P(\mathcal{G}|\lambda)}{P(\mathcal{D})} \prod_{k=1}^{K} P(D^{(k)}|G^{(k)}) \\ P(\mathcal{G}|\lambda) &= \frac{1}{Z} \prod_{k=1}^{K} P(G^{(k)}) \prod_{i=1}^{k-1} \lambda(1-\lambda)^{\Delta(G^{(k)}, G^{(i)})} \end{aligned} \tag{4.5}$$

where Z is a normalization constant and Δ is any graph distance metric, such as edit distance, that measures the number of structural differences between graphs $G^{(k)}$ and $G^{(i)}$. Note that in this chapter, we are using the standard MTL transfer formulation rather than incorporating domain knowledge via the TRAM formulation. The TRAM framework is general enough that we could combine the two. Instead, in this chapter, as is explained in Section 4.4.4, we learn the optimal degree of transfer strength directly from the data using Bayesian model averaging.

4.4 Multitask Feature Discovery

In this section, we present our novel Bayesian approach to structure discovery in multitask Bayesian networks. The challenge is finding a way to bias structures to be similar among tasks, like Equation 4.5, while maintaining the efficiency of calculating feature posteriors that factor into local calculations, like Equation 4.3. First we formulate the problem, describe structural bias terms that are order-modular, provide a Bayesian approach for handling the strength of the bias, and finally describe practical implementation issues.

4.4.1 Problem Formulation

Instead of learning the feature posteriors from a single task-specific dataset, we have K tasks from which we will leverage data. We define the indicator $f^{(k)} = f(G^{(k)})$. Our goal is to learn a feature for each task $P(f^{(k)}|D^{(1)}, \ldots, D^{(K)})\ \forall k \in \{1, \ldots, K\}$. All formulations are written for a single feature (e.g. a directed edge), but calculating them simultaneously (e.g. all edges in a network) takes the same time. To simplify the development of the objective, we will consider, without loss of generality, the case where $K = 2$.

$$\begin{aligned} P(f^{(1)}|D^{(1)}, D^{(2)}) &= \sum_{G^{(1)} \in \{G\}} P(G^{(1)}|D^{(1)}, D^{(2)}) f(G^{(1)}) \\ &= \sum_{G^{(1)} \in \{G\}} f(G^{(1)}) \sum_{G^{(2)} \in \{G\}} P(G^{(1)}, G^{(2)}|D^{(1)}, D^{(2)}) \\ &= \frac{1}{P(D^{(1)}, D^{(2)})} \sum_{G^{(1)} \in \{G\}} f(G^{(1)}) P(D^{(1)}|G^{(1)}) \times \\ &\quad \times \left[\sum_{G^{(2)} \in \{G\}} P(D^{(2)}|G^{(2)}) P(G^{(1)}, G^{(2)}) \right] \end{aligned}$$

By conditioning on an order, we get the following:

$$\begin{aligned} P(f^{(1)}, D^{(1)}, D^{(2)}|\prec) = &\sum_{G^{(1)} \subseteq \prec} f(G^{(1)}) P(D^{(1)}|G^{(1)}) \times \\ &\times \left[\sum_{G^{(2)} \subseteq \prec} P(D^{(2)}|G^{(2)}) P(G^{(1)}, G^{(2)}|\prec) \right] . \end{aligned} \tag{4.6}$$

This formulation imposes a transfer bias at the level of structure, $P(G^{(1)}, G^{(2)}| \prec)$, which is more intuitive than at the level of orders. To calculate this sum efficiently, it is necessary to factor it into a product over local sums. We prove that this is indeed possible, for appropriately chosen structure priors. In addition to the modularity assumptions already stated, we impose an additional modularity assumption, which we call Assumption 4) **Transfer prior modularity**: $P(G^{(1)}, G^{(2)}| \prec) = \prod_{i=1}^{n} P(\pi_i^{(1)}, \pi_i^{(2)}|U_i)$. Examples of priors that obey this assumption are graph distance measures that count the number of edge additions and deletions, so this is a reasonable requirement.

Theorem 1. *If $G^{(1)}, \ldots, G^{(K)}$ obey the four assumptions of modularity, then*

$$P(f^{(1)}, \mathcal{D}| \prec) = \prod_{i \in V} \sum_{\pi_i^{(1)} \subseteq U_i} f_i(\pi_i^{(1)}) P(x_i^{(1)}|\pi_i^{(1)}) \left[\sum_{\pi_i^{(2)} \subseteq U_i} P(x_i^{(2)}|\pi_i^{(2)}) P(\pi_i^{(1)}, \pi_i^{(2)}|U_i) \right] .$$

Proof Sketch: Apply the chain rule and marginalize over graph structure to get Equation 4.6. Use the modularity properties on each term in the product, and notice that the result factors into the desired form.

$$\begin{aligned} P(f^{(1)}, \mathcal{D}| \prec) &= \sum_{G^{(1)} \subseteq \prec} f(G^{(1)}) P(D^{(1)}|G^{(1)}) \left[\sum_{G^{(2)} \subseteq \prec} P(D^{(2)}|G^{(2)}) P(G^{(1)}, G^{(2)}| \prec) \right] \\ &= \sum_{\pi_1^{(1)} \subseteq U_1} \cdots \sum_{\pi_n^{(1)} \subseteq U_n} \prod_{i=1}^{n} \left[f_i(\pi_i^{(1)}) P(x_i^{(1)}|\pi_i^{(1)}) \right] \times \\ &\quad \times \left[\sum_{\pi_1^{(2)} \subseteq U_1} \cdots \sum_{\pi_n^{(2)} \subseteq U_n} \prod_{i=1}^{n} P(x_i^{(2)}|\pi_i^{(2)}) P(\pi_i^{(1)}, \pi_i^{(2)}|U_i) \right] \end{aligned}$$

See Appendix A.1 for a detailed proof.

4.4.2 Computational Complexity

The power of Theorem 1 is the computational savings that we gain. Using the factored posterior, we only need to change Step 1 of the order-space algorithm outlined in

Section 4.3.1, the calculation of the family scores. The transfer-biased family scores are calculated as:

$$\begin{aligned}\beta_{ki}(\pi_i) &= f_i(\pi_i^{(k)})P(x_i^{(k)}|\pi_i^{(k)})P(\pi_i^{(k)},\pi_i^{(j)}) \\ &= f_i(\pi_i^{(k)})P(x_i^{(k)}|\pi_i^{(k)}) \times \left[\sum_{j\neq k}\sum_{\pi_i^{(j)}\subseteq U_i} P(x_i^{(j)}|\pi_i^{(j)})P(\pi_i^{(k)},\pi_i^{(j)})\right] \quad .\end{aligned} \tag{4.7}$$

There are now Kn^{r+1} of these families to calculate and each one has a sum over $O(Kn^r)$ terms. The computational complexity increases from the single-task time of $O(n^{r+1}C(m))$ to $O(K^2n^{2r+1}C(m))$. Steps 2 and 3 remain unchanged with an exponential complexity that can be reduced through MCMC approximation.

Even the polynomial term becomes unmanageable for networks with more than 30 or so nodes and must be approximated. We note that in many cases the family scores $P(x_i^{(j)}|\pi_i^{(j)})$ are exponentially larger for some $\pi_i^{(j)} \subseteq U_i$ than others. Therefore, we can use a simple approximation by summing over only the most likely parent sets. While calculating the family scores, we create a set of the highest-scoring families, called set $\mathcal{H}_i^{(k)}$ for each node in each task. To populate this set, we simply include the h parent sets that give the highest $P(x_i^{(k)}|\pi_i^{(k)})$, for some positive constant h. Then we use the approximate structural prior:

$$P_i(\pi_i^{(k)}) \approx \sum_{\pi_i^{(j)}\in\mathcal{H}_i^{(j)}} P(x_i^{(j)}|\pi_i^{(j)})P(\pi_i^{(k)},\pi_i^{(j)}|U_i) \quad .$$

4.4.3 Transfer via Structure Bias

Now that we know we can incorporate transfer bias, we need to select a modular bias term that transfers knowledge among tasks. The MAP multitask algorithms use a penalty on the number of differences between tasks using a graph distance function. In that case, the number of edges that must be added, deleted, or reversed to edit one graph into the other is penalized. Due to our modularity constraint, our transfer bias must be defined as a function on pairs of parent sets $(\pi_i^{(k)},\pi_i^{(j)})$, rather than graphs. The main consequence of this limitation is that features that concern the parents of more than one child at a time (such as edge reversals) cannot be considered. We choose

to penalize the number of edge additions which breaks down into local calculations: the number of parents present in $\pi_i^{(k)}$ that are not present in $\pi_i^{(j)}$. In other words, the size of the set difference $\Delta_{ikj} = |\pi_i^{(k)} \setminus \pi_i^{(j)}|$ will be biased toward small values. To encourage the number of differences to be small, we apply a penalty in the form of a geometric distribution,

$$P(\pi_i^{(k)}, \pi_i^{(j)} | U_i, \lambda) = \frac{1}{Z}(1-\lambda)^{\Delta_{ikj}} \quad . \tag{4.8}$$

Calculation of the normalization constant, Z, requires summing over an exponential number of possible combinations of parent sets $(\pi_i^{(k)}, \pi_i^{(j)})$. However, we simplify the sum into an easily calculated closed form. We employ a shortcut by noting that there are a finite number of values that Δ_{ikj} can take and we find a closed form for calculating the number of parent-set combinations that produce each value of Δ_{ikj}.

$$\begin{aligned} Z &= \sum_{\pi_i^{(k)} \subseteq U_i} \sum_{\pi_i^{(j)} \subseteq U_i} (1-\lambda)^{\Delta_{ikj}} \\ &= (4-\lambda)^{|U_i|} \end{aligned} \tag{4.9}$$

Here we give a sketch of the derivation of Equation 4.9 (see Appendix A.2 for details). First, we simplify the inner sum by fixing parent set $\pi_i^{(1)}$ and counting how many parent sets $\pi_i^{(2)}$ will give $\Delta_{ikj} = 0$ ($\pi_i^{(2)}$ can contain any parents from the set $\{U_i \setminus \pi_i^{(1)}\}$ but *none* from $\pi_i^{(1)}$); then how many $\pi_i^{(2)}$ will give $\Delta_{ikj} = 1$ ($\pi_i^{(2)}$ can contain any parents from the set $\{U_i \setminus \pi_i^{(1)}\}$ and *exactly one* from $\pi_i^{(1)}$); etc, up to the maximum of $\Delta_{ikj} = |\pi_i^{(1)}|$. This sum turns out to be a binomial expansion, and so we can write it in closed form. Next, we perform a similar expansion of the outer sum over parent sets $\pi_i^{(1)}$ that have size $|\pi_i^{(1)}| = 0$, and $|\pi_i^{(1)}| = 1$, etc up to the maximum $|\pi_i^{(1)}| = |U_i|$. This sum also turns out to be a binomial expansion and therefore can be simplified into a closed form.

Plugging Equation 4.9 into Equation 4.8 gives the structure prior:

$$P(\pi_i^{(k)}, \pi_i^{(j)} | U_i, \lambda) = \frac{(1-\lambda)^{\Delta_{ikj}}}{(4-\lambda)^{|U_i|}} \quad . \tag{4.10}$$

The parameter λ, $0 \leq \lambda \leq 1$, controls the strength of transfer bias. When $\lambda = 0$, the prior becomes uniform and therefore there is no transfer. When $\lambda = 1$, the prior is

non-zero only when no edge additions occur, and therefore the only parents that are allowed are those that are likely in the other tasks.

4.4.4 Bayesian Model Averaging

We have just introduced an additional parameter, λ that must be given as an input or estimatedom. Existing MAP algorithms cannot avoid dealing with this, and they typically estimate λ by optimizing over a held out validation set. This is computationally expensive and reduces the amount of available data for training. Rather than selecting a fixed value for λ, we perform Bayesian model averaging over all possible values of λ. This Bayesian approach is compelling as the true amount of similarity among tasks is unknown, and the "true" value of λ is only incidental to our objective of learning the structure likelihoods. Furthermore, it saves us the computation of running the algorithm for several values of λ, and we do not need to hold-out data for tuning.

We set an uninformative uniform prior, $p(\lambda|U_i) = 1$ for $0 \leq \lambda \leq 1$, and marginalize over λ.

$$\begin{aligned} P(\pi_i^{(k)}, \pi_i^{(j)}|U_i) &= \int_0^1 P(\pi_i^{(k)}, \pi_i^{(j)}|U_i, \lambda) p(\lambda|U_i) d\lambda \\ &= \int_0^1 \frac{(1-\lambda)^{\Delta_{ikj}}}{(4-\lambda)^{|U_i|}} d\lambda \\ &= \frac{{}_2\mathrm{F}_1(|U_i|, 1; \Delta_{ikj}+2; 1/4)}{4^{|U_i|}(\Delta_{ikj}+1)} \end{aligned} \tag{4.11}$$

where ${}_2\mathrm{F}_1$ is the ordinary hypergeometric function:

$${}_2\mathrm{F}_1(|U_i|, 1; \Delta_{ikj}+2; 1/4) = \sum_{n=0}^{\infty} \frac{\Gamma(1+n)}{\Gamma(1)} \cdot \frac{\Gamma(|U_i|+n)}{\Gamma(|U_i|)} \cdot \frac{\Gamma(\Delta_{ikj}+2)}{\Gamma(\Delta_{ikj}+2+n)} \cdot \frac{1}{4^n n!} \ .$$

The last step in Equation 4.11 is obtained by applying an identity given by Euler in 1748 (Bailey, 1935). If β is the beta function and ${}_2\mathrm{F}_1$ is the ordinary hypergeometric function, then

$$\int_0^1 x^{b-1}(1-x)^{c-b-1}(1-zx)^{-a} dx = \beta(b, c-b) {}_2\mathrm{F}_1(a, b; c; z) \ ,$$

for $\Re(c) > \Re(b) > 0$. We let $x = \lambda$, $a = |U_i|$, $b = 1$, $c = \Delta_{ikj} + 2$, and $z = 1/4$. Then the condition, $\Delta_{ikj} + 2 > 1 > 0$, holds for any $\Delta_{ikj} \geq 0$ which is the valid range for Δ_{ikj}. Plugging these values into the identity gives the solution to the integral as:

$$\beta(1, \Delta_{ikj} + 1)_2\mathrm{F}_1(|U_i|, 1; \Delta_{ikj} + 2; 1/4) \ ,$$

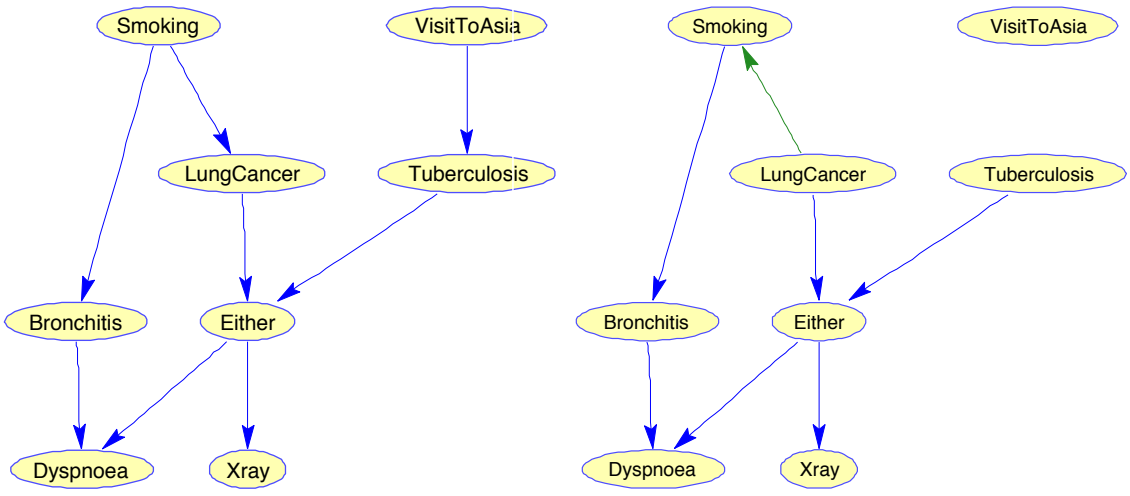
which simplies to the solution given in Equation 4.11. For complete details of these steps, see Appendix A.3.

We are only interested in calculating ${}_2\mathrm{F}_1$ for combinations of integer-values of Δ_{ikj} and $|U_i|$ for $0 \leq \Delta_{ikj} \leq |U_i| < n$. For these values, ${}_2\mathrm{F}_1$ is convergent and efficient solvers exist. Thus, we can plug the result of Equation 4.11 into the equation of Theorem 1.

Bayesian model averaging is made possible by conditioning on orders. Existing multitask network learning algorithms that search in DAG space (Niculescu-Mizil and Caruana, 2007; Oyen and Lane, 2012) would be required to calculate a normalization constant like that in Equation 4.9 but with sums over all possible DAGs, and no closed form has been found for such a sum.

4.5 Experiments

Multitask learning should be able to identify true edges and non-edges with less data than is possible with traditional single-task learning. We compare our MTL structure discovery algorithm against two baselines. The first baseline is single-task learning (STL), where each network is learned independently of the other tasks. The other baseline (POOL) takes the opposite extreme by pooling data from all tasks together and treating it as a single task. POOL uses the strongest leveraging of data among tasks possible and so it should perform best if the separate tasks are actually the same distribution. For all approaches, we use the BeanDisco implementation for exact and approximate network discovery (Niinimaki, Parviainen, and Koivisto, 2011). For MTL, the scoring function of BeanDisco is modified as described above. For POOL, the data are merged before applying the algorithm. The size of parent sets is also

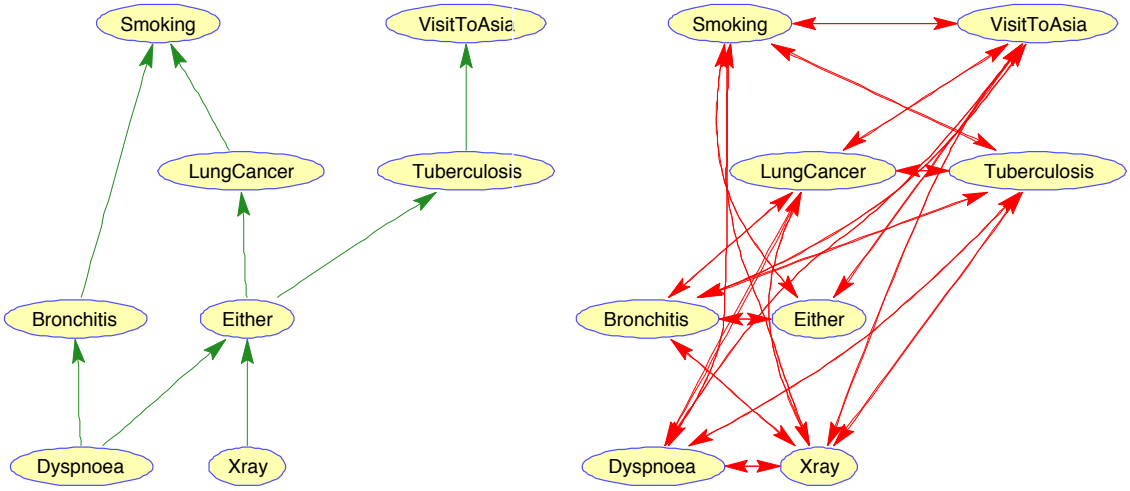


(a) Real network used to generate data.

(b) Likely edges learned from 5000 samples (thresholded at $P(f) > 0.5$)

Figure 4.2: *Asia* network. True edges are colored blue.

limited, as described for each dataset.



(a) Reverse of true edges are colored green

(b) Non-edges are colored red

Figure 4.3: Color coding of network edges. True edges are colored blue as in previous figure.

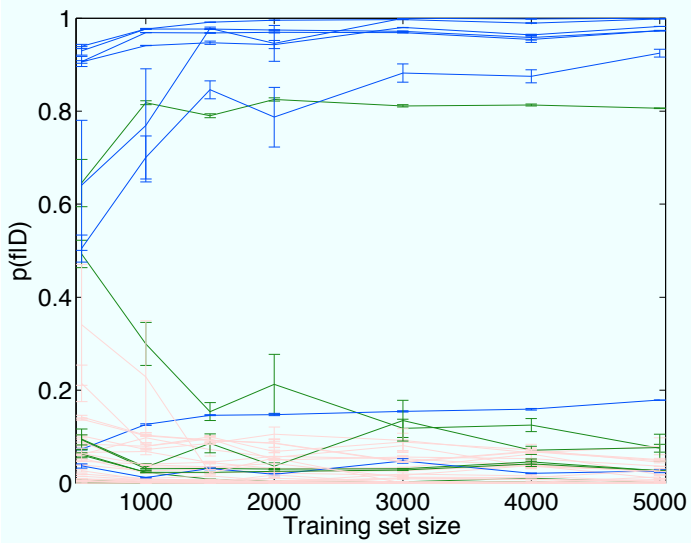
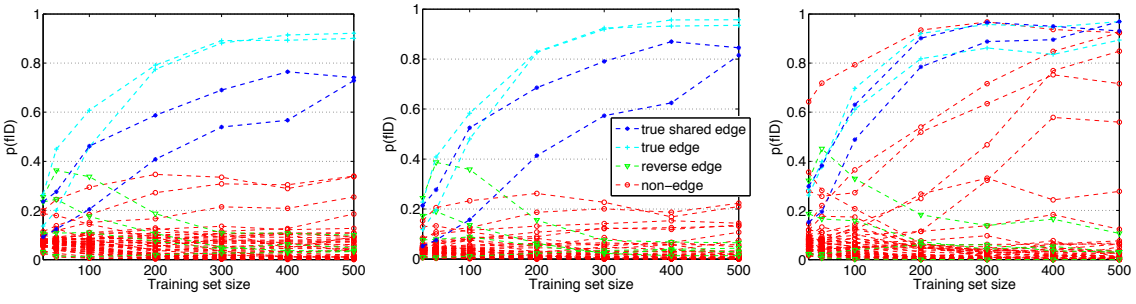


Figure 4.4: Posterior probability estimate for each edge in the *asia* network from various large sample sets (means calculated from 20 sample sets). Blue curves are true edges, green are reverse of a true edge, pink are non-edges.

4.5.1 Benchmark Data

Synthetic data is generated from benchmark Bayesian networks, *asia* which has 8 variables (Lauritzen and Spiegelhalter, 1988) and *alarm* which has 37 variables (Beinlich et al., 1989). Even when the generative model is known, it is not obvious how to measure the performance of a network discovery algorithm. Figure 4.2a shows the *asia* network. To give a clear picture of our objective, see the learning curve of posterior



(a) STL (b) MTL (c) POOL

Figure 4.5: Example posterior probability estimate for each edge in a modified *asia* network from various small sample sets (means calculated from 20 sample sets). Up is good for blue and cyan curves. Down is good for red and green curves.

features in Figure 4.4. The color-coding of these curves is explained in Figure 4.2a (blue for true edges), Figure 4.3a (green for edges that are reverse of true), and Figure 4.3b (red for non-existent edges). On the small *asia* network we can calculate the true posterior likelihoods of structural features given the data. Sample noise affects the true posterior likelihood of structural features but the posteriors appear to stabilize for large training sets. Even so, one edge has been consistently identified in the reverse direction of the true edge and another true edge represents such a subtle dependency that it is not discernible from this amount of data (see Figure 4.2b for a picture of the learned network). Therefore, we use the posterior estimates from large training sets as our ground truth $P^*(f|D)$. In the case of *asia*, $P^*(f|D) = \hat{P}_{STL}(f|D_{5000})$.

We need a set of related networks and so we modify some structures of the given benchmark network to create similar but different networks. We delete each edge with some probability p_{del} and vary p_{del} from 0.1 to 0.5 to create sets of networks with more or less features in common for various experiments. If an edge is deleted, the conditional probability table for the child of the deleted edge is updated by marginalizing over the deleted parent. In our experiments, the full generative model is repeated 10 times to produce 10 different sets of K networks each for a given p_{del}.

4.5.2 Benchmark Results

The goal of transfer learning is to accelerate the learning curve at smaller training set sizes by leveraging data among similar tasks. If we look closely at the results for one particular modified *asia* network, we can see the effect of multitask learning. Figure 4.5a shows the estimated posteriors from STL at smaller sample sizes. Even at these small sample sizes, the posteriors of the true edges tend to be higher than those of non-edges. However, compared to the estimates from large samples, these posteriors exhibit high variance and many are quite far from the large-sample posterior value (in the figure, error bars are omitted for readability). The question is whether multitask learning can produce a steeper learning curve.

Figure 4.5b shows the learning curve achieved by MTL on the same network, us-

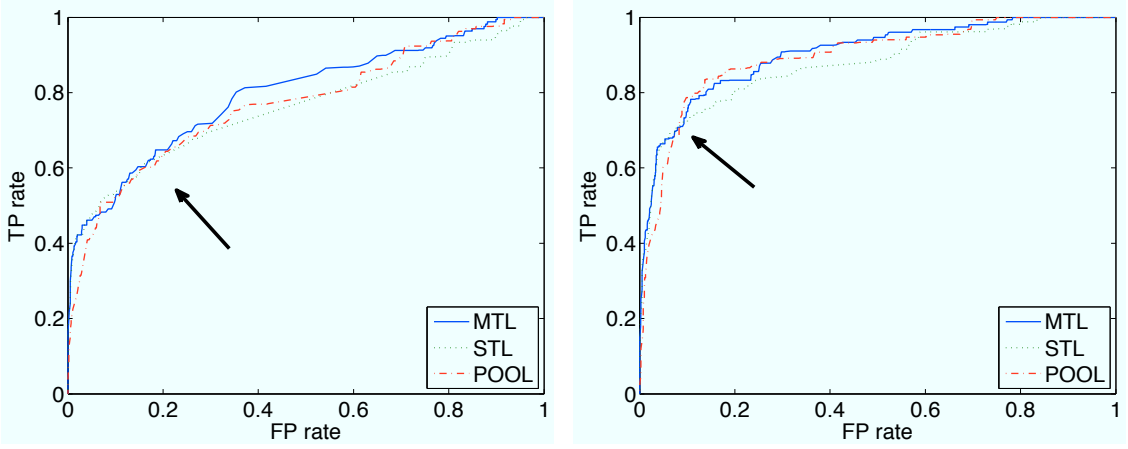
Table 4.1: Performance increase for *asia* in terms of AUC given by MTL vs STL and MTL vs POOL. The "pair-t" columns indicate which algorithm had the largest AUC according to a paired-t test.

Training	MTL vs STL		MTL vs POOL	
samples	% incr	pair-t	% incr	pair-t
5	3.06	-	10.72	MTL
10	9.02	MTL	4.10	-
20	4.90	MTL	0.34	-
30	4.98	MTL	0.85	-
40	7.60	MTL	3.66	-
50	3.00	MTL	3.08	MTL
100	1.97	MTL	2.96	MTL
200	0.53	-	2.82	MTL
400	0.14	-	4.14	MTL
500	-0.03	-	3.72	MTL

ing data leveraged from one other task, where some but not all edges are in common between the two tasks. This learning curve shows a wider gap between the estimated values of true edges and non-edges. In that sense, the MTL learning curve is better than STL because it is better at separating the true edges from the non-edges at smaller training set sizes. In particular, this gain is achieved through the lower estimates of non-edges. In other words, non-edges are more quickly identified as such through transfer learning than without. On the other hand, the raw estimates of true edges tend to have such high variance (both with STL and with MTL) that it is not possible to say that one algorithm is doing better than the other in terms of converging on the actual $P^*(f|D)$.

Figure 4.5c shows the learning curve obtained by POOL on the same modified *asia* network. POOL combined the data from two modified networks with some edges in common. The algorithm effectively has twice as much data to work with as STL, therefore the learning curves are steeper. Yet there are quite a few non-edges with high posterior values.

To quantify these results, we measure how well the estimates for true edges separate from the estimates for non-edges. There is potentially a directed edge between each ordered pair of nodes. We call this set of potential edges E. We quantify the



(a) 10 samples per task (b) 50 samples per task

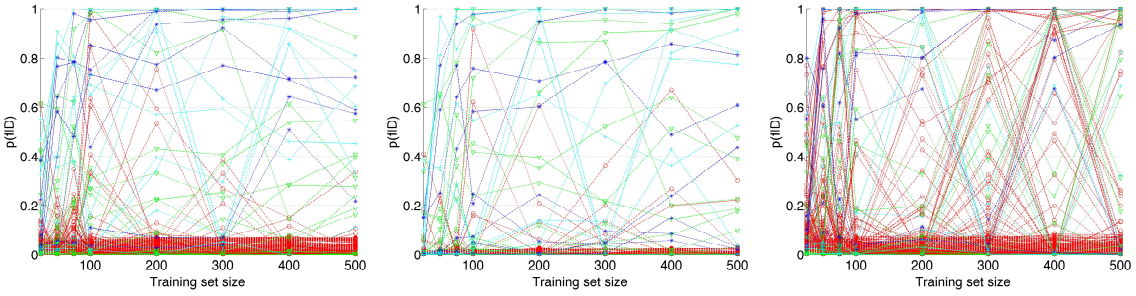
Figure 4.6: ROC curves for *asia*. Each point is the (FP rate, TP rate) aggregated over 2 tasks and 10 trials of the generative model, for a particular value of τ. The arrow indicates where STL and POOL curves cross.

ground truth using the $P^*(f|D)$ values obtained from large samples and identify the set of "true" edges $E^* = \{f \in E \mid P^*(f|D) > 0.5\}$.

To differentiate learned edges from learned non-edges, we assign a threshold τ and call any feature an edge if its posterior is greater than the threshold, $\hat{E} = \{f \in E | \hat{P}(f|D) > \tau\}$. By varying τ, $0 \leq \tau \leq 1$, we can investigate the tradeoff between the rates of true-positives (TP) and false-positives (FP) in $\hat{E}$ by constructing an ROC curve (Figure 4.6). The TP rate is $|\hat{E} \cap E^*|/|\hat{E}|$. The FP rate is $|\hat{E} \setminus E^*|/(|E| - |\hat{E}|)$.

Figure 4.6 shows that various algorithms have different strengths along the ROC curve. The ROC shows that the patterns indicated in Figure 4.5 are borne out more generally; that is, STL is slowest to positively identify true edges, while POOL has difficulty eliminating false positives. MTL achieves the greatest overall separation of true edges and non-edges. Initially, at the left end of the ROC curve, with low false positive rates both MTL and STL perform best, finding more true positives than POOL. However, the performance of STL falls off as the false positive rate increases: STL is missing some true positives that both MTL and POOL are able to identify. MTL gives us the best of both worlds, giving the best overall performance.

Area under the curve (AUC) summarizes the overall performance along the ROC



(a) STL (b) MTL (c) POOL

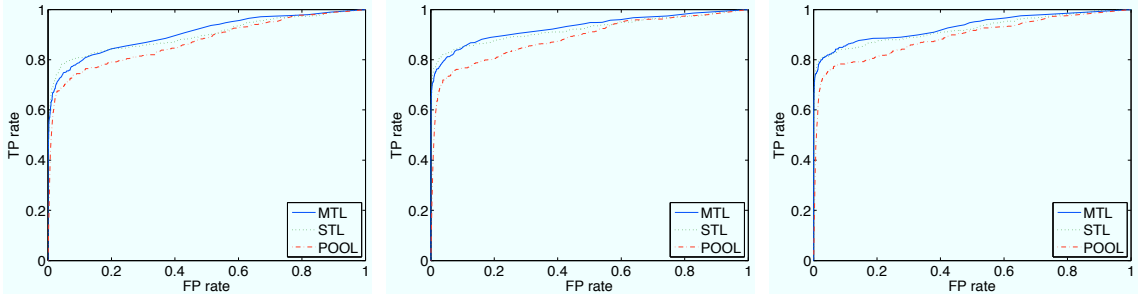
Figure 4.7: **Learning curves:** Example posterior probability estimate for each edge in a modified *alarm* network from various small sample sets (means calculated from 20 sample sets). Blue curves are true edges shared by both tasks, cyan curves are true edges unique to this task, green are reverse of a true edge, red are non-edges. Up is good for blue and cyan curves. Down is good for red and green curves. Note that likelihoods of non-edges are closer to zero in MTL than other algorithms, and that POOL has the highest number of falsely identified edges.

curve. We report AUC for various amounts of training data in Table 4.1 across 30 trials of the generative model. With these small training sets, the difficulty of the problem presented by each trial can vary quite a bit, therefore, the performance of each algorithm is compared directly on each trial by looking at how much greater the AUC is for MTL than the other algorithm. This increase in AUC score per trial is then averaged over all trials to give the numbers in Table 4.1. Furthermore, a paired-t test is performed to determine whether this increase in performance is significant at the 95% confidence level. The winner of the paired-t test is given in Table 4.1.

Similar experiments are performed on the larger *alarm* network. This network is too large for exact posterior computation, therefore we use MCMC approximation. We set MCMC hyper-parameters as recommended by Niinimaki, Parviainen, and Koivisto (2011); specifically, bucket size = 10, burn-in samples = 1000, sub-sample interval = 10 and total samples = 100. We tried other values (notably more samples, larger sub-sample interval, and longer burn-in) and found that they give nearly the same results. We also use the transfer approximation described in Section 4.4.2, with $h = 1000$. For the ground truth *alarm* network, we use 10,000 training samples to estimate the true posterior of each feature, $P^*(f|D) = \hat{P}_{STL}(f|D_{10,000})$.

Table 4.2: Performance increase on *alarm* for AUC given by MTL versus STL and MTL versus POOL. The "pair-t" columns indicate which algorithm had the largest AUC according to a paired-t test.

Training	MTL vs STL		MTL vs POOL	
samples	% incr	pair-t	% incr	pair-t
25	-0.12	-	0.04	-
50	0.29	MTL	1.76	MTL
75	0.43	MTL	3.63	MTL
100	0.67	MTL	4.49	MTL
200	0.05	-	5.19	MTL
300	-0.07	-	8.68	MTL
400	-0.03	-	8.85	MTL
500	-0.46	STL	10.45	MTL



(a) 50 samples per task (b) 75 training samples per task (c) 100 samples per task

Figure 4.8: Example ROC curves for *alarm* data. Each point represents the (FP rate, TP rate) aggregated over 2 tasks and 10 trials of the generative model, for a particular value of τ.

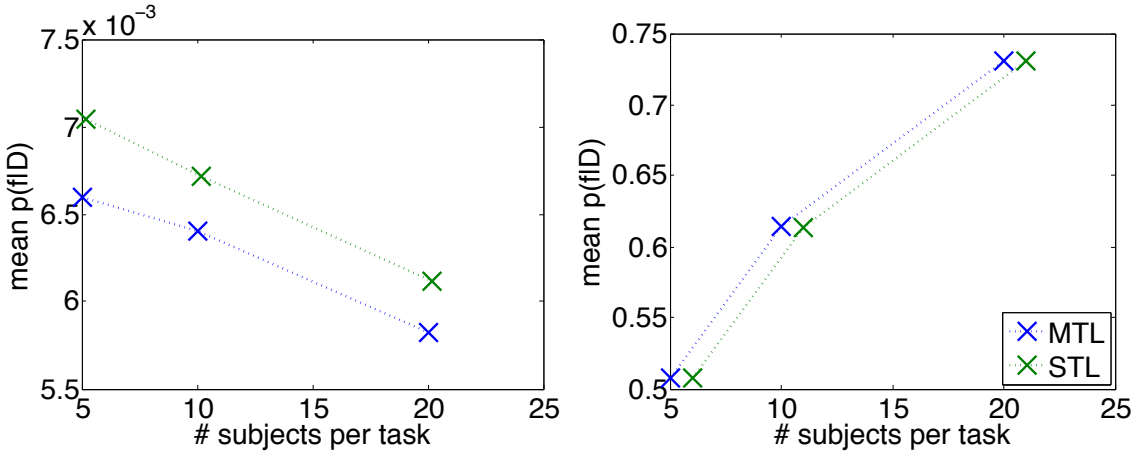
The alarm network contains 37 variables, therefore there are 1,332 ordered pairs of nodes or potential edges in the set E. Of these, only 46 are true edges. We see again that on this dataset, MTL estimates lower posteriors for the non-edges than STL or POOL (see Figure 4.7). The ROC curves in Figure 4.8 show that POOL routinely identifies many false positives. The curves for MTL and STL are closer, but again MTL is better at reducing the number of false positives. This makes differentiating the true edges from the false edges easier at small training set sizes, see Table 4.2 for AUC results. MTL dominates STL for small training sets. MTL dominates POOL at all training set sizes.

4.6 Application to Neuroimaging

Our goal is to find functional brain networks associated with schizophrenia. We start with functional magnetic resonance images (fMRI) data that measure the activity levels in regions of interest (ROI) in the brain. The functional brain network is modeled as a Bayesian network of information sharing (modeled by a multinomial of discretized activity level) among ROIs. Data has been collected from 86 healthy control subjects (*controls*) and 74 schizophrenia patients (*patients*). For each subject, there are 384 full-brain scans which are the samples in our training data. Brain images are parcellated using the Talaraich atlas giving 150 ROIs. Therefore, for each subject, we have a 150×384 data matrix. We concatenate the data from several subjects to create the training data for each task. We apply both our MTL algorithm and the standard STL Bayesian structure discovery algorithm. As our goal is to identify different structures between tasks, we do not use the POOL method that learns identical structures for both tasks.

To examine the consistency of results across various subsets of subjects for each task, we create cross-folds of the subjects. That is, for a 20-fold cross-fold, the subjects for a task are partitioned into 20 roughly equally-sized bins. For each fold, 19 of these 20 bins of subjects are included in the training data. The results show how well learned models over various subsets of subjects are representative of the larger *control* and *patient* populations.

The size of the parent sets is limited to $r = 2$. With this setting, the time to calculate family scores with STL is approximately 3 hours. For MTL family score calculation, we use the approximation method described in Section 4.4.2 with h =10,000. MCMC approximation is used to estimate the posterior likelihood of edges (Niinimaki, Parviainen, and Koivisto, 2011), with hyper-parameters bucket size = 10, burn-in samples = 5000, sub-sample interval = 10 and total samples = 1000.



(a) Non-edges (b) True edges

Figure 4.9: Estimated posterior of features from small subsets of subjects. Points are perturbed horizontally for visibility.

4.6.1 Small Samples

MTL estimates significantly lower posterior likelihoods on non-edges compared to STL. Evaluating results on real data is complicated by the fact that we do not have ground truth of known networks. Therefore, we look at the trend of estimates on smaller subset of the subjects and compare the results against the STL estimate from the full set of data. Figure 4.9a shows that for edges that are *not* determined to have real dependencies in the full dataset (i.e. non-edges), the posterior estimate is significantly lower for MTL than STL. Significance was determined via a paired-t test at the 95% confidence level over various subsets of subjects selected. Figure 4.9b shows that there is no difference between MTL and STL in terms of the posterior estimate of true edges (according to paired-t test at 95% confidence). Therefore, MTL is able to eliminate the non-edges with less data than STL.

These results on small samples show the estimation of non-edges is improved by using transfer. However, on this real dataset, there is high variability on the rest of the edges that may represent real dependencies. Neither MTL or STL is significantly better at reducing this variability. The implication is that it is not any easier to seperate non-edges from real edges using MTL than STL. This is a disappointing

result, particularly in light of the promising results given by the benchmark data. One likely explanation for this discrepency between benchmark data and real data is that there is a model mismatch problem. That is, the real data are not generated by a joint multinomial distribution, therefore inferring multinomial parameters produces a poor fit. This concern has been raised in discussions with domain experts. The fMRI data begin real-valued and there has been concern about loss of information due to discretization of the data. This issue is addressed in the next chapter, in which we consider a continuous joint probability distribution rather than a discrete distribution.

4.7 Discussion

Our structure bias in the order-modular framework for Bayesian network structure discovery can be applied to many other problems currently being researched. In this paper, we demonstrate the application of structural bias to the problem of multitask learning. We find promising results from our approach and expect that further improvements can be made by tailoring the bias term to the application. Additionally, more sophisticated methods for approximating the transfer bias on large networks could be explored.

Implementation of a structural bias in Bayesian structure discovery is critical for solving other problems as well. Grzegorczyk and Husmeier (2008) propose incorporating prior knowledge about biological networks in the form of a structural feature bias. Rather than using the exact calculation of posteriors that are possible when conditioning on orders, they propose a new MCMC sampling method for approximation. Their motivation is that it is inconvenient to define priors in the space of orders rather than structure. Our Theorem 1 shows that it is indeed possible to define structural priors at the structure level to use the efficient algorithms that rely on conditioning on orders.

This structural bias term could also be used to transfer knowledge about the direction of Bayesian network edges from interventional experiments (Cooper and Yoo, 1999). Active learning of Bayesian network structure has been shown to significantly

speed the learning of edges, particularly for getting directionality (Tong and Koller, 2001). Multitask active learning algorithms would be useful for transferring knowledge from an experiment where interventions are possible to a similar domain where such interventions may be more expensive or impossible. Recent work proposes principled methods for the transfer of causal relationships between domains (Pearl and Bareinboim, 2011). The order-modular structural bias term presented here provides a critical algorithmic mechanism to implement such transfer of knowledge.

4.8 Conclusions

This chapter introduces a Bayesian discovery algorithm for estimating multitask Bayesian network features. This algorithm is able to successfully leverage data from related tasks to improve the estimate of network structure features given limited amounts of data. The primary contribution is determining that structural priors that are order-modular can be used to impose inductive bias among tasks. By using local structural priors, we achieve three goals simultaneously: 1) an intuitive inductive bias at the level of structures rather than orders; 2) take advantage of the most efficient structure discovery algorithms; and 3) closed form Bayesian model averaging over the transfer strength parameter. Empirical evidence suggests that multitask learning of Bayesian networks reduces the number of spurious dependencies learned, especially at small sample set sizes. Overall, the end result is that we are better able to learn which network features exist in each task under the standard multitask assumption of similarity among tasks.

Chapter 5

Learning Differences between Network Structures via Transfer

Domain scientists often look for differences among related dependency networks, such as for regulatory networks of different species or for diseased versus healthy populations. The previous chapters indicate that transfer learning improves the overall fit to data of learned models, but there remains a question about the reliability of the differences learned between classes. The prevalent naive method for finding these differential dependencies is to learn individual networks independently and then compare them. This chapter shows why this approach is prone to high false discovery rates and that it provides no mechanism for controlling the quality of the identified differences. We then show that transfer learning identifies high-precision (high-confidence) differences and provides the domain expert with a natural "knob" for controlling the quality of the differential network analysis results.

5.1 Motivation

This chapter focusses on the problem of identifying differences in dependency networks among various classes of data. For example, we want to understand how regions of the brain share information before and after a person acquires a particular skill. The goal in this neuroimaging study is to identify the regions of the brain that are most

influential after a skill has been learned so that direct current stimulation can be applied to those few regions to accelerate a person's learning process (Clark et al., 2012). In another example, biologists want to analyze how the dependency structure of plasma proteins change between patients that have cancer and patients that do not have cancer, with the goal of better understanding the cancer biology and identifying improved cancer diagnostics.

Traditional methods for identifying these network structure differences learn a network for each task independently and then compare them. However, we show that this naive approach tends to produce a large number of spurious differences which significantly limit their usefulness. Large numbers of spurious differences hamper the analysis and prevent drawing any reliable conclusions. Furthermore, spurious differences are usually expensive to eliminate through followup tests. We also find that there is a need for an intuitive mechanism to control the quality of the learned differences; in other words, to trade off learning a small number of spurious differences (precision) with identifying a large number of differences (recall).

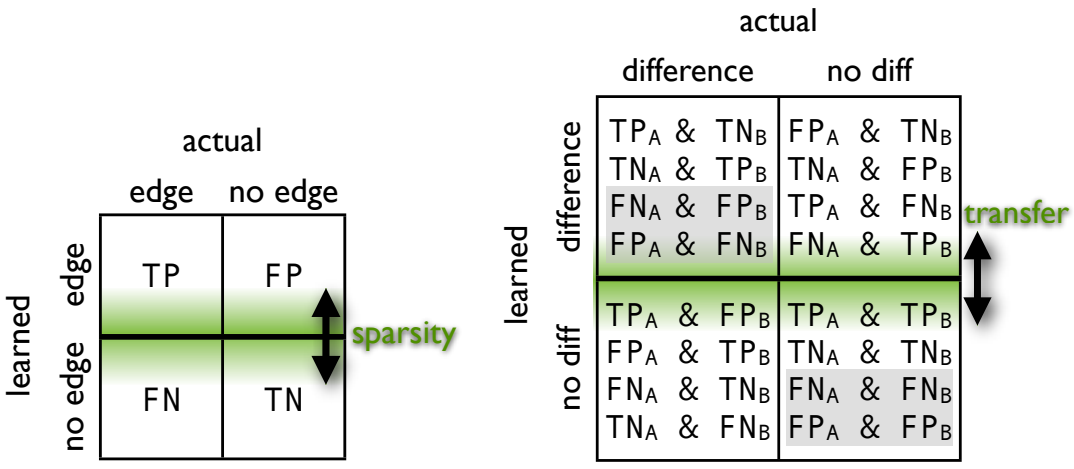
This chapter demonstrates the novel use of *transfer learning* techniques to control the precision-recall tradeoff in differential network analysis, and shows that these techniques can dramatically improve the quality of the learned differences. Our approach is to use transfer learning to bias the learned dependency networks for the different tasks to be similar. The more heavily we enforce this bias, the fewer differences will be learned between tasks. We show that true differences, those that are well supported by the data, will tend to require a higher bias to be eliminated, while spurious differences will be eliminated even with a small bias.

There does not appear to be any existing research that investigates the use of transfer learning for obtaining high quality differences or providing a mechanism to control the precision-recall tradeoff in differential network analysis. In Danaher, Wang, and Witten (2011), low recall of differences learned on synthetic data is mentioned, but not further explored. Most related to this chapter, a recent paper (Mohan et al., 2012) explores techniques for biasing learning such that the dependency networks differ in a limited number of variables. They show that if the differences match their

assumption, then they are better able to recover the true networks. Some approaches use bootstrap procedures to identify the most stable edges in each network before making comparisons (Bergmann, Ihmels, and Barkai, 2004). Rather than learning a network for each task independently, a few methods have been proposed that specifically look for commonalities and differences (Roy, Werner-Washburne, and Lane, 2011; Zhang et al., 2009). However, these methods do not consider controlling the number of differences learned except through computationally expensive bootstrapping procedures. The transfer learning approach proposed in this chapter is far less computationally expensive than bootstrapping procedures and gives higher precision differences.

The methods explored in this chapter introduce several improvements of practical importance over previous chapters. In previous chapters, we assume a discrete directed graphical model (Bayesian networks), however the results on fMRI data were not consistent across validation folds. The raw data is continuous and so using a continuous model would take advantage of that. Undirected models can be less computationally demanding than directed models because there is no acyclicity constraint. Gaussian graphical models have both of these advantages, and are able to scale to networks with thousands of nodes (as opposed to about a hundred). One major disadvantage is that the Gaussian graphical model assumes that data are produced by a multivariate Gaussian, which is not accurate in fMRI data. We handle this problem by using a non-parametric model of correlation instead of Gaussian, as detailed later in this chapter.

In graphical lasso network structure learning, the objective function is convex and the solution space is piecewise continuous with a discontinuity at the points where each element of the of the precision matrix transition from zero to non-zero (Meinshausen and Bühlmann, 2006). Like Chapter 3, graphical lasso learns a single MAP estimate of the solution, but unlike Chapter 3, this is an exact solution rather than the result of heuristic search. Even so, given limited data the optimal solution may change given a different sample of data from the same distribution. This is a serious problem when comparing two networks learned from two different (yet similar) distributions,



(a) Edges (b) Differences

Figure 5.1: Confusion matrix of possibilities for identifying (a) edges in a single network; and (b) differences between two networks A and B. Note that the gray shaded regions are situations where an edge could be identified correctly as different between the two networks, but the direction of the difference would be reverse from the truth; or a non-difference is correctly identified but for the wrong reason.

because there is no way to gauge which network differences are most likely to be real and which may be due to sample noise.

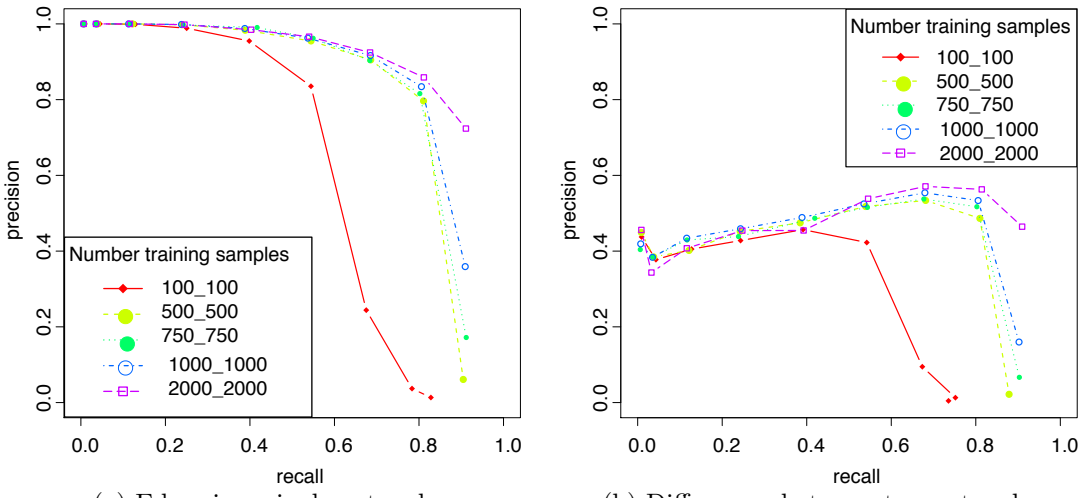
5.2 Naive Approach: Learning Independently then Comparing

Biological data comprises hundreds or thousands of variables, but often only hundreds of samples. The data is noisy and there may be some model mismatch. We must assume that there will be errors when trying to learn dependency networks from data. When the networks are learned in isolation, errors in identifying edges are likely to be made independently, which in turn introduces spurious differences between learned networks. To illustrate this issue, we first look at the precision-recall tradeoff for identifying edges in individual networks; next we look at how errors in individual networks complicate identifying differences; and then we conclude this section with a discussion of how to control the precision-recall tradeoff for differences.

Figure 5.1a shows the possible scenarios for identifying each dependency, or edge, in a confusion matrix. Ideally, all edges would be identified as true positives (TP) or true negatives (TN), but we know that this is not possible given limited data and so there will also be some false positives (FP) and false negatives (FN). Sparse network learning algorithms, such as graphical lasso, aim to learn few edges (Meinshausen and Bühlmann, 2006) by increasing the degree of sparsity of the learned network. The learning algorithm has no control over the vertical line separating the actual edges from non-edges. However, by changing the sparsity parameter, the algorithm effectively moves the boundary between the *learned* edges and non-edges (shown as the horizontal line highlighted in green in Figure 5.1a). Assuming the algorithm is able to identify edges with better than random probability, then *precision* (TP/(TP+FP)) will increase with sparsity; meanwhile, the *recall* (TP/(TP + FN)) will decrease.

It is tempting to find the ideal sparsity level before comparing the networks. However, we find that there is no such ideal setting. We created two sets of data generated from synthetic multivariate Gaussian networks. Each of these networks has 1,000 variables and 1,000 edges. Among those edges, the networks have about 80% of their edges in common and about 20% different. We generate data from these networks, then learn graphical structures and compare our learned edges within each network to ground truth. The precision-recall tradeoff for identifying edges is apparent as we adjust the strength of the sparsity penalty (see Figure 5.2a).

No matter the setting of the sparsity parameter, there will be errors, and when errors are made *independently* for each dataset, there will be many fake differences. Figure 5.2b shows just how bad the precision is for identifying *differences* in dependencies between these two networks. Changing the sparsity parameter has some effect on the recall of these differences, and some on the precision. However, the precision is never better than 0.6 for differences, even with larger amounts of data when the identification of individual edges is quite confident (precision is over 0.9 for edges alone). Figure 5.1b shows the possibilities for identifying differences. Clearly, getting edges correct in both networks is less likely than getting an edge correct in a single network, making it even more difficult to learn differences than it is to learn indi-



(a) Edges in a single network

(b) Differences between two networks

Figure 5.2: Precision versus recall

vidual edges. No matter where we are on the edge precision-recall tradeoff for each individually-learned network, like Figure 5.2a, we will always identify a large number of fake differences.

All is not lost! Figure 5.1 provides a clue about how to reduce the number of differences learned. To increase the precision of learned differences, we need to be able to adjust the horizontal boundary between learned differences and non-differences. For learning edges in individual networks, we were able to do this through the setting of the sparsity penalty. For learning multiple networks with few differences, we need to bias the networks to be similar to each other, which is exactly what transfer learning does. When learning multiple networks with transfer learning, we decrease the number of differences learned by increasing the transfer strength parameter. Now edges in both networks will tend to either be chosen together or left out together. In essence, only the differences that are strongly supported by the data will survive this bias giving us high confidence that those edges that are identified as differences are very likely to be true differences.

5.3 Transfer Learning for Differential Networks

Clearly, if we hope to identify high-precision differences then networks cannot be learned in isolation and then compared to each other. Networks must be inferred together to facilitate comparison. To achieve this, we borrow a technique developed for learning multiple networks via transfer learning, also called multitask learning (MTL) (Chiquet, Grandvalet, and Ambroise, 2011; Danaher, Wang, and Witten, 2011; Guo et al., 2011). Specifically, we employ an MTL graphical lasso objective function that explicitly controls the number of differences learned.

This section covers information about learning an individual network with the Gaussian graphical lasso objective; learning non-parametric graphical lasso with the transelliptical model; and learning multiple networks with transfer learning.

5.3.1 Gaussian Graphical Models

First, we review the key steps of the standard model for learning a single network. Gaussian graphical models (GGMs) infer a network of conditional dependencies from multivariate data by approximating the inverse covariance matrix with a sparse solution (Meinshausen and Bühlmann, 2006). If X is a p-dimensional Gaussian random variable $X \sim \mathcal{N}(0, \Sigma)$, then $\Theta = \Sigma^{-1}$ is the precision matrix. Entries in the precision matrix are partial correlations, i.e. θ_{ij} is the correlation of variables X_i and X_j given all other variables X_m, $m \neq i, j$. A value of $\theta_{ij} = 0$ implies conditional independence of X_i and X_j. Therefore, the precision matrix can be interpreted as an undirected network where nodes are the variables in the precision matrix and edges connect variables with non-zero partial correlations.

Given the dense covariance matrix, $\hat{\Sigma}$, estimated from the data, the learning objective for a single network is:

$$\widehat{\Theta} = \arg\max_{\Theta \succ 0} \log\det\Theta - \mathrm{tr}(\hat{\Sigma}\Theta) - \lambda\|\Theta\|_1 \ . \tag{5.1}$$

The parameter λ, $0 \leq \lambda \leq 1$, controls the degree of sparsity. Varying this parameter affects the precision-recall tradeoff of identifying dependencies. Similar to adjusting

the sensitivity of classification algorithm (Fawcett, 2004), it is informative to inspect the networks inferred at various values of λ to see how edges appear/disappear along the precision-recall curve.

5.3.2 Transelliptical Graphical Models

Next, we review how transelliptical grapical models relax the Gaussian assumption and give further details about the non-parametric that we use in place of the covariance matrix. We have seen that often in real data, extreme values in just a few samples, like 1% of the data, can produce a large number of Gaussian correlations that do not exist when those samples are not present. Transelliptical models replace the Gaussian covariance matrix with a non-parametric correlation matrix which is far less sensitive to extreme values (Liu, Han, and Zhang, 2012). We use the Kendall's tau correlation based on rank-statistics. This is a non-parametric measure of correlation between pairs of variables. Let $\mathbf{x}^1, \ldots, \mathbf{x}^m \in \mathbb{R}^d$ be m observations of the random vector $\mathbf{X}$. Then, for two variables, X_j and X_k, the Kendall's tau correlation statistic is defined as,

$$\hat{\tau}_{jk} = \frac{2}{m(m-1)} \sum_{1 \leq i < i' \leq m} \text{sign}(x_j^i - x_j^{i'})(x_k^i - x_k^{i'}) \ . \tag{5.2}$$

Liu, Han, and Zhang (2012) shows that the standard Gaussian covariance estimator, $\hat{\Sigma}$, can be estimated using a transformation on the Kendall's tau estimator, so that

$$\hat{\Sigma}_{jk} = \sin\left(\frac{\pi}{2}\hat{\tau}_{jk}\right) \cdot I(j \neq k) + I(j = k) \ , \tag{5.3}$$

where I is the indicator function.

To learn the transelliptical graphical model, we simply insert this sample correlation estimator matrix, $\hat{\Sigma}$, into the usual graphical lasso objective function. This simple change in estimator makes the learning significantly more robust to outliers and non-Gaussianity, without any significant loss in accuracy even when data is truly Gaussian (Liu, Han, and Zhang, 2012). In preliminary experiments, we verified this finding empirically with synthetic data. Furthermore, on our real data, the large number of spurious correlations learned with the Gaussian model made comparisons

between networks nearly impossible. With the same data and learning algorithm, but with this Kendall's tau correlation statistic, we acheived far more interpretable results. We strongly recommend using this non-parametric model whenever analyzing real data.

5.3.3 Joint Graphical Models with Transfer

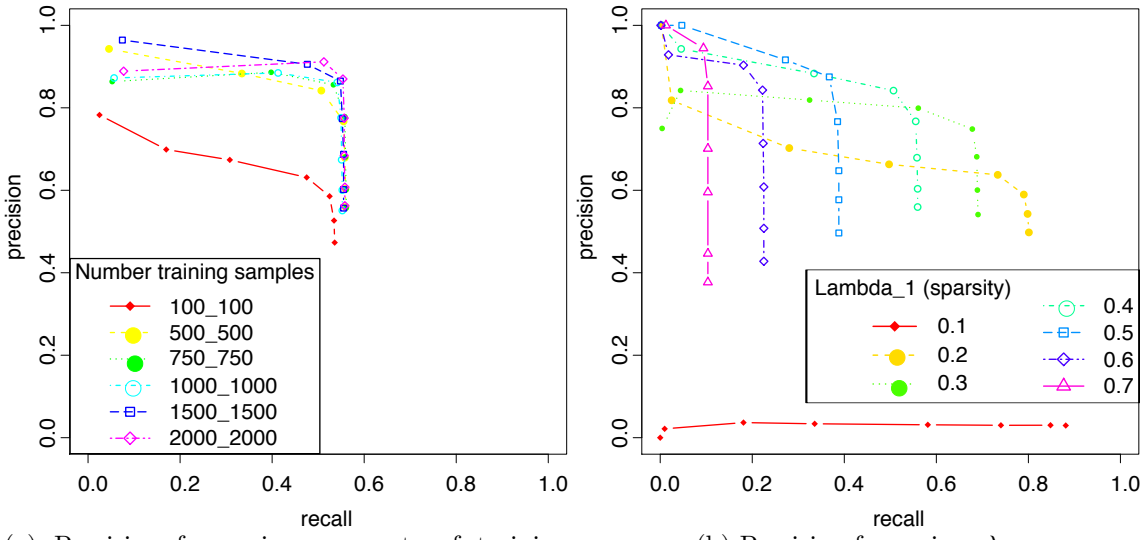
To learn multiple graphical models from multiple sets of data, we use the joint graphical lasso algorithm which incorporates a transfer bias term to encourage the learned networks to be similar (Danaher, Wang, and Witten, 2011). If we have K classes of data, we will estimate $\hat{\Sigma}^k$, $k \in \{1, \ldots, K\}$, for each set of data and learn a sparse precision matrix, $\widehat{\Theta}^k$, for each class of data.

$$\max_{\Theta^k \succ 0, \forall k} \sum_{k=1}^{K} \Big[\log\det \Theta^k - \mathrm{tr}(\hat{\Sigma}^k \Theta^k) \Big] - \lambda_1(1-\lambda_2) \sum_{i,j,k} |\theta_{ij}^k| - \lambda_1 \lambda_2 \sum_{i \neq j} \sum_{k} (\theta_{ij}^k)^2 \quad (5.4)$$

The parameter λ_1 controls the degree of sparsity in much the same way as the λ parameter in the single task case. There is also a parameter λ_2, $0 \leq \lambda_2 \leq 1$, that controls the number of differences learned among the tasks. When $\lambda_2 = 0$, the objective is the same as several independent single-task learning problems. When $\lambda_2 = 1$, the structures learned are identical. Therefore, this parameter can be used to limit the number of differences learned. More importantly, the only differences that will survive this penalty term are those that are highly supported by the data.

5.4 Experiments with Synthetic Data

We use synthetic networks and data to test whether our approach correctly identifies true differences between related networks. To create a synthetic dataset, we generate a network with 1,000 Gaussian variables and 1,000 undirected edges. Then, the endpoint of each edge is re-wired with some probability to a different node, creating a different network with edges in common with the first one. The values for the partial correlations associated with each edge are drawn independently from a normal



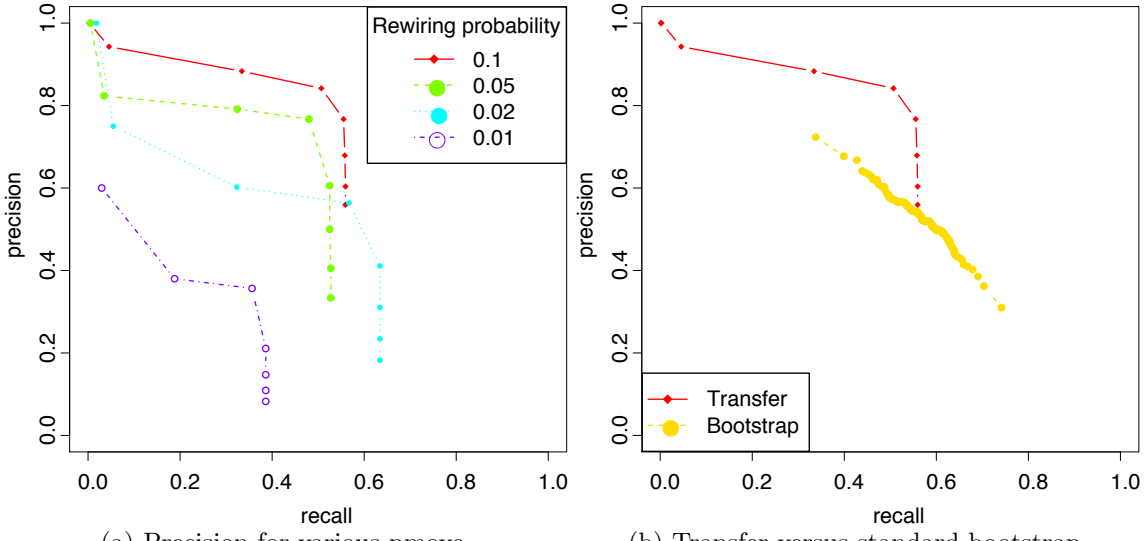
(a) Precision for various amounts of training data.

(b) Precision for various λ_1.

Figure 5.3: Precision versus recall of learned differences, on synthetic networks.

distribution, then scaled to ensure the positive semi-definiteness of the partial correlation matrix Θ. Data are generated by drawing 500 samples for each task from the distributions $\mathcal{N}(0, \Theta^{-1})$ for each Θ. Results are averaged across 10 trials.

We demonstrate the effectiveness of the algorithm by measuring the rate of learned



(a) Precision for various pmove.

(b) Transfer versus standard bootstrap.

Figure 5.4: Precision versus recall of learned differences, on synthetic networks.

edge differences that actually exist in the true networks (precision) against the rate of true differences that are learned (recall). The results of the experiments are depicted in Figure 5.3. In all of these plots λ_2 is varied to produce the various points of precision/recall. Figure 5.3a shows the precision recall graphs for various training set sizes with $\lambda_1 = 0.4$. As λ_2 is increased, the precision increases from around 0.6 at the rightmost point of each curve ($\lambda_2 = 0$, learning each network independently) to almost 1 in the left region of the curve corresponding to large amounts of transfer.

While the tradeoff between precision and recall for learned differences is mainly controlled by the λ_2 parameter, λ_1 also has an effect on the differences learned because the sparsity of the networks controls which edges are present in each task. Figure 5.3b shows the precision/recall tradeoff for various values of λ_1. Lower values of λ_1 (denser networks) produce more differences and increase the recall of differences. However, learning overly dense networks can make it impossible to discern the true differences from the false ones even with high levels of transfer.

We also vary the rewiring probability when generating the true networks, varying the number of true differences. Figure 5.4a shows the precision-recall graphs for differences across different values of the rewiring probability. As the number of true differences decreases, it gets harder to identify them and performance decreases, but increasing the amount of transfer (increasing λ_2) still leads to higher precision.

We also compare to using bootstrap procedures to identify higher confidence differences. For the bootstrap procedure, we generate a bootstrap sample of the data and train independent graphical models on that data. We repeat this 400 times. Then for each edge, we calculate the frequency of it appearing in one task but not the other. This is the bootstrap probability $P_B(e)$ that the edge e is different between tasks. For a given cut-off probability, c, we consider all edges with probability $P_B(e) \geq c$ as inferred differences. We then calculate the precision/recall based on the true differences. The bootstrap graphs are generated by varying c from 0 to 1 so that at $c = 0$ any difference that appeared in any bootstrap is considered a difference, and at $c = 1$ only differences that appear in all 400 bootstraps are considered differences.

Figure 5.4b shows the comparison between the transfer algorithm and the boot-

strap method when 500 training samples are available for each task. Our transfer method dominates the precision-recall curve compared to bootstrapping. Bootstrapping does not produce high precision differences, although it gives some exploration over a limited range of the precision/recall curve. Interestingly, the bootstrapping method becomes increasingly computationally expensive depending on the degree of precision (or recall) desired. The highest point in the bootstrap curve shown occurs when $c = 1$ and there are $\sim$100 differences ($\sim$30 of them false) that occur in all 400 bootstraps. Therefore, to push the precision ratio higher than 0.7, we would need to run more bootstrap samples until most of those false differences do not appear in at least one of the new bootstraps (while hoping that some true differences remain). Bootstrapping becomes computationally quite expensive.

5.5 Case Studies on Real Data

In this section we present three real case studies where domain experts performed differential dependency network analysis in a neuroimaging study, an ovarian cancer study, and a pancreatic cancer study.

5.5.1 Addressing Gaussian Model Mismatch

On both types of real data (plasma protein concentrations and fMRI activity), we find that the multivariate Gaussian model is highly sensitive to outliers. An extreme value in as little as a single sample could produce hundreds of weak correlations that are not identified when that sample is removed from the training data. Comparing networks with this model-mismatch is near impossible. The problem was detected because the transfer bias eliminated most of these spurious edges, but left few real differences to explore. Furthermore, knowing that such a large portion of discovered edges are false gives us little confidence in any of the learned edges. A straightforward solution to this problem is to use a rank-statistic to estimate correlations, such as the Kendall's tau statistic described earlier (Liu, Han, and Zhang, 2012). With this modification, we

are able to produce meaningful network models for differential dependency analysis.

5.5.2 False Discovery Rate

When applying a machine learning algorithm on real data, we would like to perform a quantitative evaluation of the results. The quantitative evaluation of the results is difficult in real usage scenarios because there is no ground truth to compare against so true precision and recall can not be calculated. In these cases, a common approach is to estimate the false discovery rate (FDR) of the algorithm. When the transfer strength is increased, the objective function guarantees that fewer differences will be learned. We need to show that as the total number of differences decreases, the remaining differences contain proportionally more real differences than spurious differences until there are no spurious differences. False discoveries can be measured by removing the differences between classes through scrambling of the data. The marginal distribution of each class will remain the same, but there will be no true differences between the classes. Any differences identified by the algorithm are therefore false.

We employ two approaches for estimating the false discovery. In the first approach we take data from one task and randomly partition that data into two synthetic tasks. These two synthetic tasks of data are created from a single class, and so any differences learned between them are false differences. To mimic the original data as much as possible, we create a bootstrap sample from the artificially partitioned tasks to make task-specific datasets that are the same size as the original dataset. We perform multiple random partitions and draw bootstrap samples to create several training datasets from all original classes and take the average number of differences identified by the algorithm as an estimate of the expected number of false discoveries made in the original problem.

The second FDR estimation approach is to pool the data from all classes together, then randomly split the data into synthetic tasks with the same number of instances per task as the original classes. There should not be any differences between the dependency structures of these newly generated synthetic tasks, so any difference identified

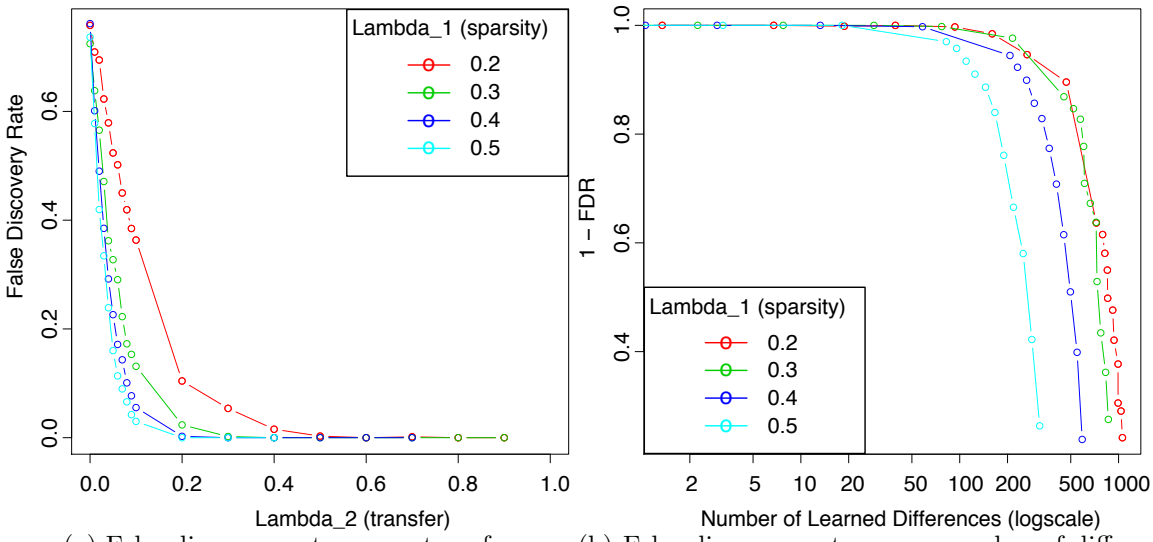
by the algorithm is a false discovery. We perform the splitting procedure multiple times and take the average number of differences identified by the algorithm as an estimate of the expected number of false discoveries made on the original problem.

Neither FDR approach is perfect and each has its own set of assumptions. Both tend to underestimate the true false discovery rate, as tested empirically on synthetic data. The second approach, in which samples from all classes are mixed together, produces a distribution of data for each model that should truly be a mixture model from both classes. Dependencies will be weakened and may be harder to identify in either artificial task, complicating the false discovery estimation. The first approach avoids this problem by creating artificial tasks that are assumed to be generated from an underlying true dependency network, with exactly the same dependencies in all artificial tasks. However, for each false discovery experiment, we only have data from a single class and those samples must be bootstrapped up to the sample size of the original problem. Empirically, the two methods perform similarly. The first method is used on the fMRI data presented next, and the second method is used on the protein data.

5.5.3 Accelerated Learning fMRI Study

Functional magnetic resonance imaging (fMRI) measures the activity level in regions of the brain while a subject is in the scanner. The network of partial correlations among regions of interest (ROI) in the brain, is called a functional brain network because it indicates which regions of the brain have activity patterns that appear to be exchanging information with each other. A common question is whether these dependencies are different in subjects under different conditions.

Using data from the Accelerated Learning fMRI Study, we want to see how brain regions interact before and after a person learns a new skill (Clark et al., 2012). In this study, subjects are asked to identify concealed objects in still images taken from a virtual reality environment. Initially, all subjects are considered Novice, that is they are not significantly better than random at identifying images with concealed objects.



(a) False discovery rate versus transfer

(b) False discovery rate versus number of differences

Figure 5.5: Accelerated learning fMRI study false discovery rate.

fMRI data are collected from these subjects while performing this identification task. Then, subjects are trained until they reach a level of Intermediate (midway between chance and perfect) competency. At this point, fMRI data are again collected while performing the identification task. In total, we have data from 12 subjects at the Novice stage and 4 at the Intermediate stage. For each subject, there are 1056 samples of brain activity from 116 regions of interest (ROIs) in the brain. The ROIs are defined by the AAL atlas (Tzourio-Mazoyer et al., 2002). Our goal is to identify dependencies among the ROIs that are different in the Intermediate stage from the Novice stage. Looking at the networks (rather than the activity levels of individual ROIs) shows us which ROIs are most critical for performing a cognitive task (Clark et al., 2012).

Our false discovery experiments with fMRI data show that without transfer, there are always differences learned between the stages. As the transfer strength is increased, these known false differences disappear quickly. We compare this behavior against what happens when we perform similar bootstrap sampling from the two classes of data that we would like to compare. We see that many more differences are learned and these differences do not disappear so quickly as the transfer strength is increased. Figure 5.5a illustrates this phenomenon by showing the ratio of fake differences (those

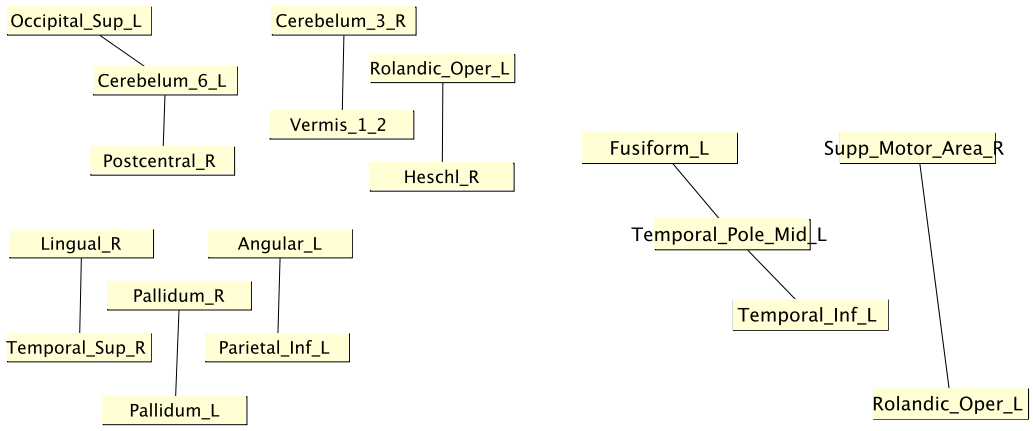
learned between sets of data from the same population) to the differences learned between sets of data from the two populations of interest. At the left end of the plot, when there is no transfer, the number of fake differences learned is about 80% the number of potentially-real differences between the two classes. As the transfer strength parameter increases, this percentage drops, indicating that the rate of decrease of the fake differences is faster than that of the potentially-real differences.

Figure 5.5b shows the tradeoff between estimated precision (1-FDR) and the number of differences found (analogous to the precision recall curves in Figure 5.3). There are edge differences between Novice and Intermediate that are more resistant to transfer bias than the differences between two sets of samples from the same class. We are therefore more confident that these Novice vs Intermediate edge differences represent true differences than those found without transfer. However, we are not sure what percentage of these edges are expected to be real because our estimate of the false discovery rate may be low.

Learned Brain Networks

Figure 5.6 shows the networks learned for the two classes of data (Novice and Intermediate). Figure 5.6a shows the edges that are learned in Novice but not Intermediate. Figure 5.6b shows the edges that are learned in Intermediate but not Novice.

In order to gain evidence that we are learning good networks, we look at the pathways activated by the visual exercise in both stages of learning (Novice and Intermediate). These results show that for both stages, groups of brain regions are found that share information, which correlate well with sensory-motor pathways found in humans (Figure 5.7). This includes the occipito-parietal dorsal visual pathway that computes the location of objects, the occipito-temporal ventral pathway that determines the identity of objects, collections of frontal and cingulate regions that help to make decisions about responses, as well as separate cerebellar and middle temporal networks, along with other smaller networks of brain regions (Mishkin, Ungerleider, and Macko, 1983). With learning to identify hidden objects in this task,

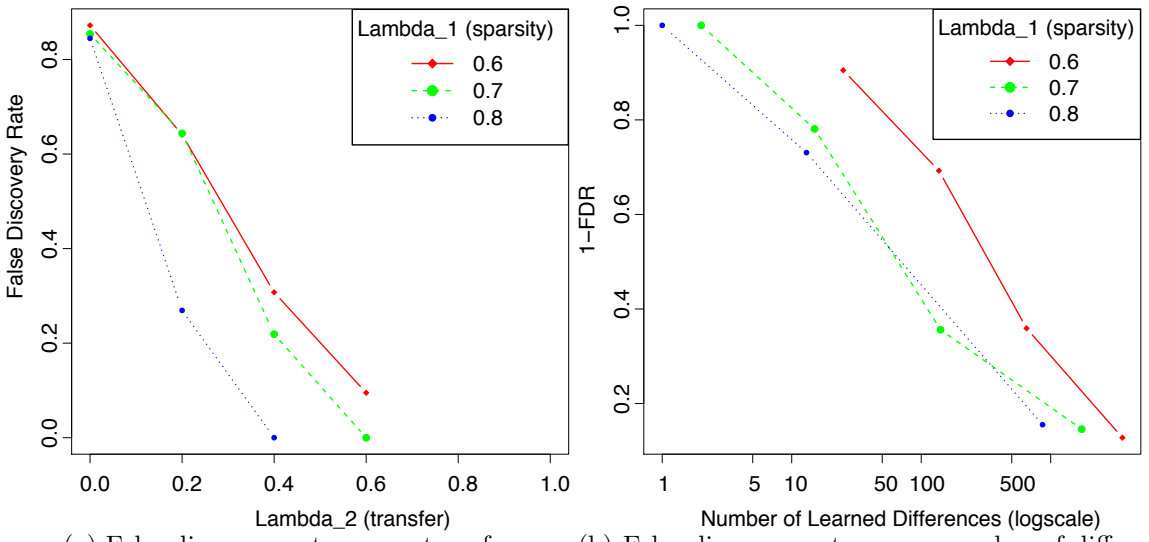


(a) Novice but not Intermediate (b) Intermediate but not Novice

Figure 5.6: Differential dependency networks learned from Accelerated Learning fMRI Study with $\lambda_1 = 0.5$ and $\lambda_2 = 0.2$.

it was found that portions of the ventral pathway increased in strength, suggesting that learning resulted in greater information flow among regions that specialize in visual object identification.

Figure 5.7: Network of dependencies shared among Novice and Intermediate stages of the Accelerated Learning study.



(a) False discovery rate versus transfer

(b) False discovery rate versus number of differences

Figure 5.8: Ovarian cancer study false discovery rate.

5.5.4 Ovarian Cancer

The ovarian cancer study uses data from a cohort of 247 patients. The study contains 114 patients diagnosed with ovarian cancer, and 133 controls. Each patient had a blood sample taken prior to the diagnosis, and plasma concentrations of 858 proteins were measured using SOMAmer technology (Gold et al., 2010). SOMAmer is a multiplexed method of identifying proteins based on aptamers. Aptamers are specialized molecules that bind to individual proteins, allowing the concentrations of the bound proteins to be measured. Yet, due to the specialization of SOMAmer to particular proteins, it can be expensive to measure many proteins. For research purposes, this expense is reasonable and so we can learn large networks to better understand the disease. Another goal is to develop inexpensive, reliable blood tests that can be given to patients to quickly diagnose disease. Therefore, the dependencies that are associated with cancer may be used to reduce the number of proteins necessary for diagnosis.

Figure 5.8a plots the false discovery rate (FDR), calculated as described in Section 5.5, as a function of the transfer parameter λ_2 for different settings of the sparsity parameter λ_1. Looking at $\lambda_2 = 0$, which corresponds to the standard approach of

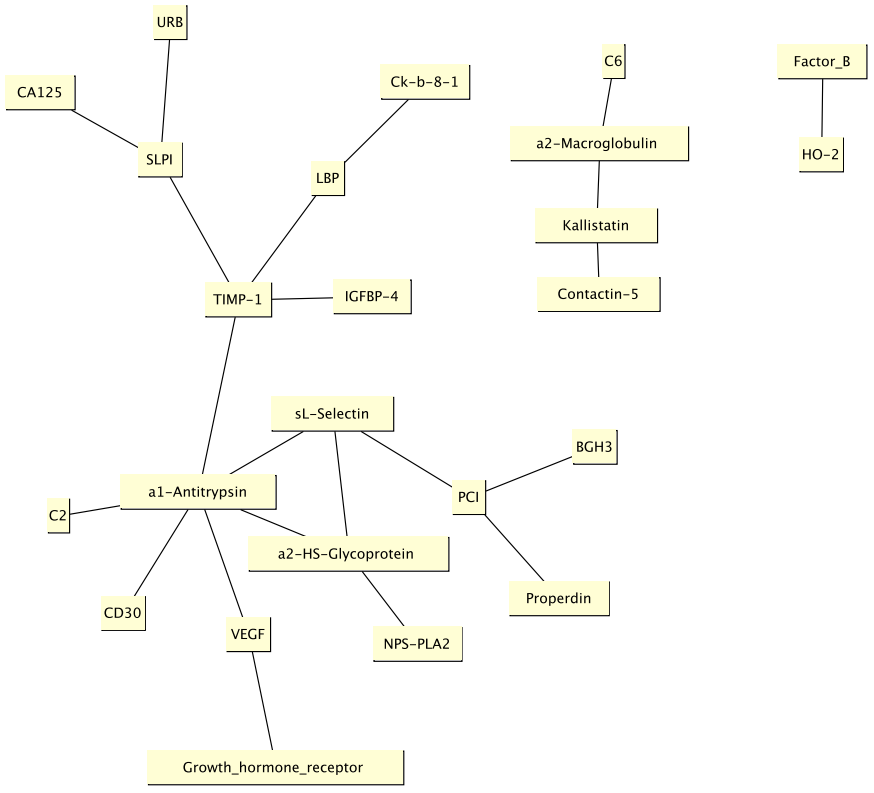


Figure 5.9: Ovarian cancer difference network with transfer bias. These dependencies exist in the *cancer* class but not in the *control* class.

learning the structures independently and comparing them, we see that the estimated false discovery rate is almost 0.9; that is, nine out of ten differences found are estimated to be false. This level of false discovery clearly makes impossible any analysis of the found differences and renders the results pretty much useless.

As the transfer rate is increased, the estimated FDR steadily decreases for all setting of the sparsity parameter, reaching levels below 0.1 which is very acceptable in biological applications. As in the synthetic data, this improvement in precision comes at the cost of lower recall. Analogous to the precision recall curves in Figure 5.3, Figure 5.8b shows the tradeoff between estimated precision (1-FDR) and the number of differences found. Without transfer (right end of the graph) there are a large number differences found, but almost 90% of these are false. As the amount of transfer is increased significantly fewer differences are found, but we have much higher confidence that the differences that remain are real.

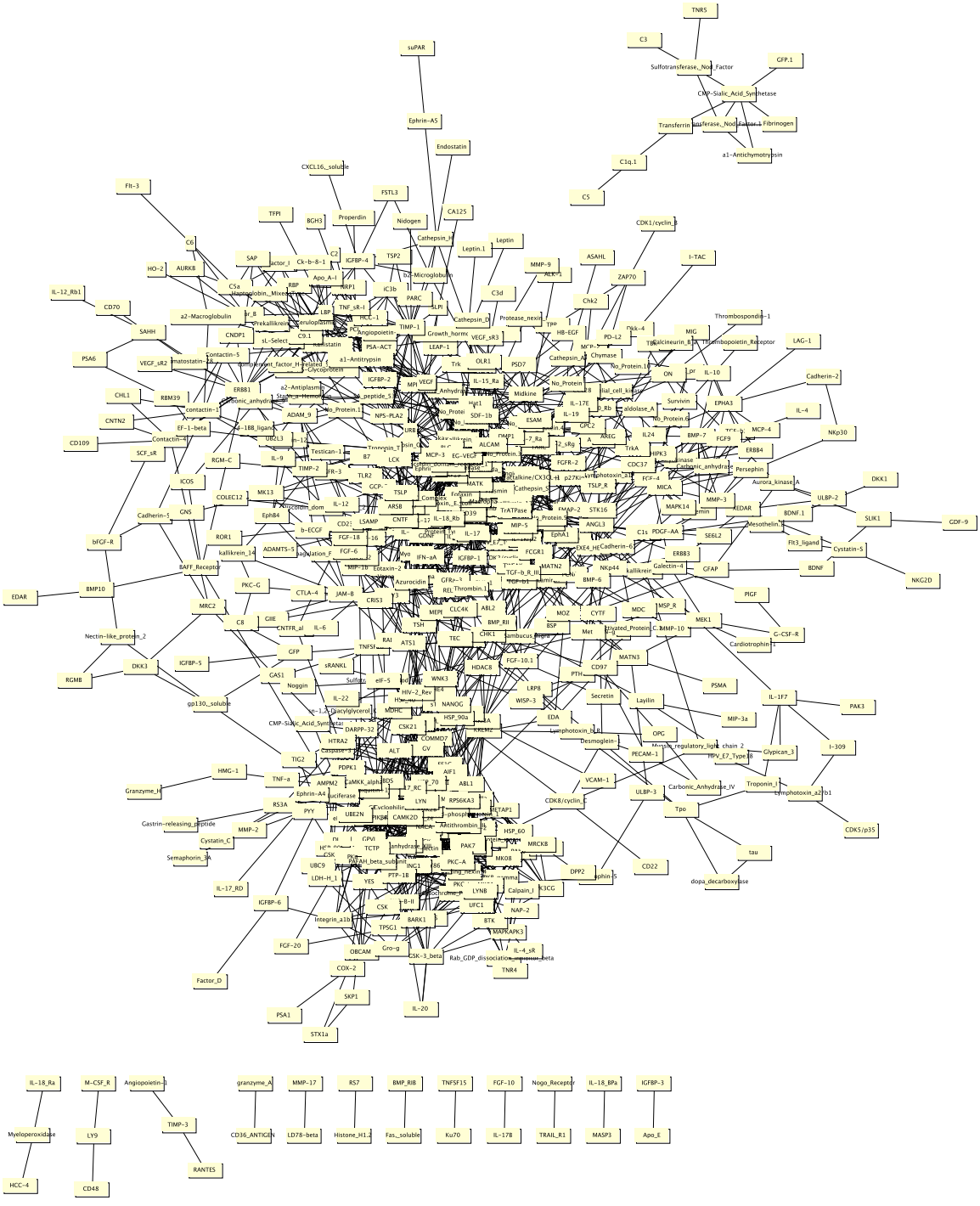


Figure 5.10: Ovarian cancer difference network without transfer bias. These dependencies exist in the *cancer* class but not in the *control* class.

Figure 5.9 shows the differential dependency network between the cancer and control populations for a sparsity setting of $\lambda_1 = 0.6$ and a transfer setting of $\lambda_2 = 0.6$.

Immune response proteins	Inflammatory response proteins	Coagulation and complement proteins	Proteins involved in extracellular matrix	Endopeptidase inhibitor proteins
a2-Macroglobulin	a2-Macroglobulin	a2-Macroglobulin	TIMP1	SLPI
C2	Ck-b-8-1	C2	TIMP1	TIMP1
C6	GHR	C6	URB	a2-Macroglobulin
Ck-b-8-1	LBP	Factor B	a1-Antitrypsin	a2-HS-Glycoprotein
Factor B	a1-Antitrypsin	a1-Anti-trypsi	BGH3	a1-Antitrypsin
Properdin	TIMP-1	PCI	VGEF	Kallistatin
GHR	CD30	TIMP-1		PCI
LBP	VEGF	CD30		
sL-Selectin	a2-HS-Glycoprotein	VEGF		
a1-Antitrypsin				
TIMP-1				
VEGF				
CA-125				

Table 5.1: Proteins associated with ovarian cancer.

Every edge in this network represents a dependency that is present in the cancer population but not in the control population. For contrast, Figure 5.10 shows the differential dependency network obtained without transfer ($\lambda_1 = 0.6$, $\lambda_2 = 0$). To ensure that this network presents relevant biological information, we run a standard enrichment analysis using DAVID (Huang, Sherman, and Lempicki, 2008) on the 24 proteins that appear in the figure, and asked collaborators that have extensive expertise in cancer biology to analyze the results. The enrichment analysis shows that the following functional clusters are significantly enriched. An enriched functional cluster means that there are significantly more members of that cluster present in the query list than it would be expected from the random background distribution: endopeptidase inhibitor, inflammatory response, complement and coagulation and extracellular matrix (see Table 5.1). Our collaborators, the oncology domain experts, report:

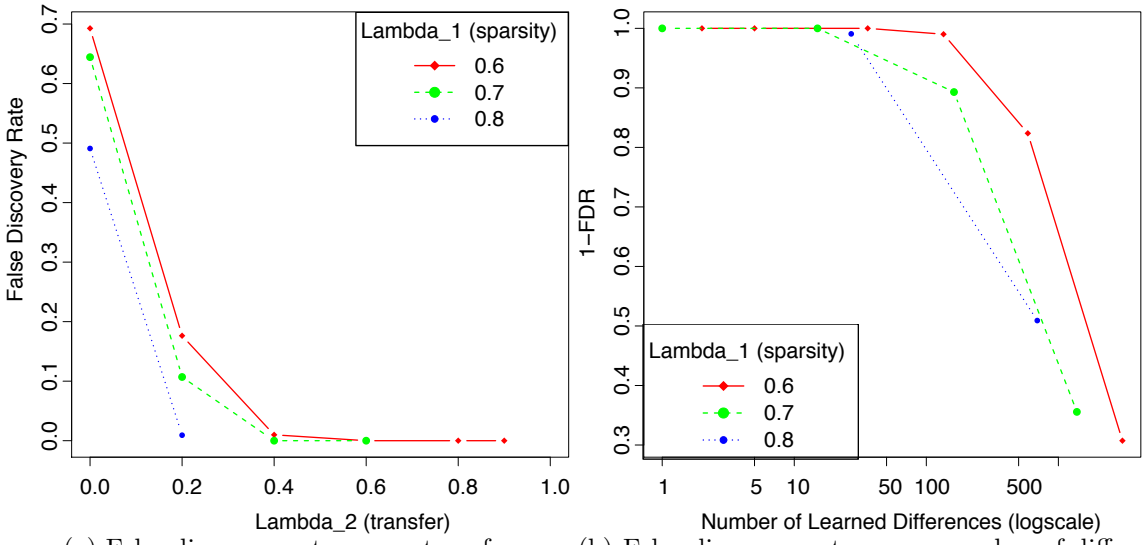
> This is consistent with what is known about the biology of ovarian cancer. The body's reaction to ovarian cancer includes stimulation of both the adaptive (antibodies, cellular immunity) and innate (complement, inflammation) immune systems. In fact, the new, foreign entity (ovarian cancer) which stimulates these responses also creates a new milieu in which tumor

> mutations are selected for when they help the cancer evade these immune responses (Wang et al., 2005). Ovarian cancers (as well as many other cancers) also tend to induce a hypercoagulable state, which involve coagulation and complement proteins. Endopeptidases play essential roles in hemostatis and signal transduction. Changes in the extracellular matrix is also a key process as cancer cells escape the primary tumor and metastasize. [...] Many of the proteins [in Figure 5.9] have been associated with cancer in general and with ovarian cancer in particular. For instance CA125 is a well known and clinically used ovarian cancer marker, SLPI has been shown to be over-expressed in gastric, lung and ovarian cancers, accelerating metastasis (Choi et al., 2011),VEGF is involved in the growth of blood vessels (tumors require heavy vascularization to grow), IGFBP4 has been associated with a number of cancers, including ovarian (Walker et al., 2007). (Oyen et al., 2013)

5.5.5 Pancreatic Cancer

The pancreatic cancer study uses data from a cohort of 469 patients (239 cases and 230 controls) from two sites. As with the ovarian cancer study, each patient had a blood sample taken prior to the diagnosis, and plasma concentrations of 858 proteins were measured from this blood sample using SOMAmer technology.

Figure 5.11a shows the estimated FDR as a function of the transfer parameter (λ_2) for various sparsity levels. At $\lambda_2 = 0$ the estimated FDR is lower than in the ovarian cancer case, probably due to the extra available data, but it is still the case that at least half of the learned differences are false, making any interpretation of the differential network problematic. As the transfer parameter is increased, the estimated false discovery rate quickly drops to zero. Figure 5.11b shows the tradeoff between estimated precision and the number of learned differences. Compared to the ovarian cancer results in Figure 5.8b, a larger number of differences can be identified at high estimated precision levels (1-FDR), such as 0.9 and above.



(a) False discovery rate versus transfer

(b) False discovery rate versus number of differences learned

Figure 5.11: Pancreatic cancer study false discovery rate.

Figure 5.12 shows the differential dependency network between the cancer and control populations for the sparsity setting $\lambda_1 = 0.6$ and the transfer setting $\lambda_2 = 0.6$, with the node labels showing the functional descriptions in lieu of the protein names. Our collaborators, oncology domain experts, provided us with a promising interpretation of the learned proteins:

> The differential dependency network shows proteins that are linked with the endocrine pancreas (e.g. endosomal insulin protease, insulin sensitivity regulator, protein regulating secretion of hormones by pancreas) and with the exocrine pancreas (e.g, HDL, LDL, IDL proteins, bile dependent digestive enzyme), as well as proteins that are associated with cancer and cancer related processes (e.g. tumor cell lysis receptor, mesothelial tumor differentiation antigen, down regulator of p53, endoplasmic reticulum chaperone). An enrichment analysis finds the following processes to be significantly enriched: extracellular matrix, lipid transport and cell adhesion. These processes are very relevant to the pancreatic and cancer biology. As mentioned above, changes in the extracellular matrix are involved in cancer cells escaping the primary tumor and metastasizing.

> Related to the extracellular matrix, cell adhesion is also a key process that regulates the migration (spreading) of cancer cells through the body and the destruction of the histological structure in cancerous tissues (Hirohashi and Kanai, 2005). The lipid transport is related to the exocrine pancreatic function (Lopez-Candales et al., 1993). (Oyen et al., 2013)

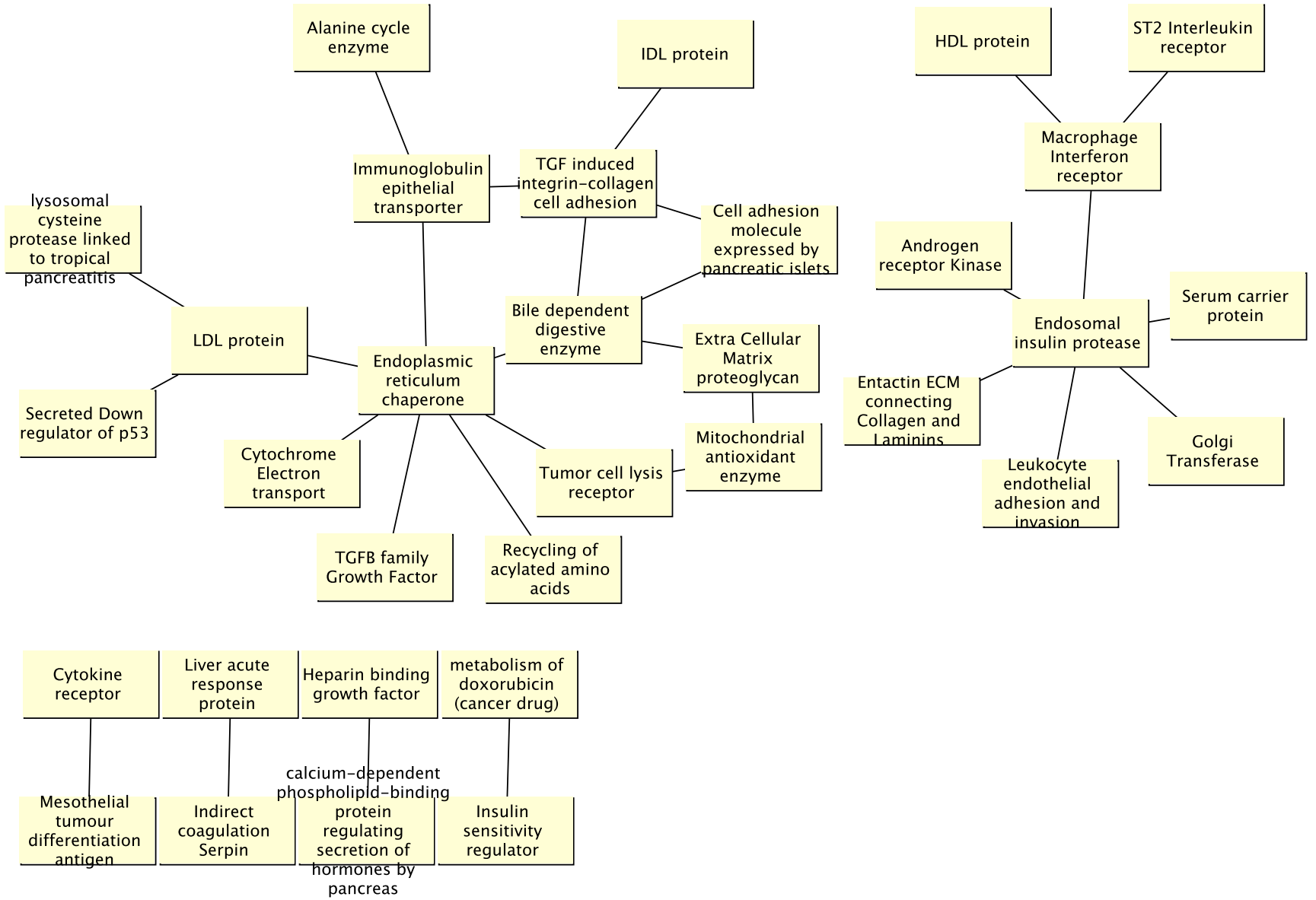


Figure 5.12: Pancreatic cancer difference network with some transfer. Protein families reported rather than individual proteins. These dependencies exist in the *cancer* class but not the *control* class.

5.6 Conclusion

Differential analysis of dependency networks of multivariate data allows domain experts to uncover and understand the differences between related populations and the processes that are generating these differences. Such questions arise in many domains including biology, medicine, and neuroscience. This chapter demonstrates that the traditional approach of learning the dependency networks for each task independently and then comparing them is prone to having high false discovery rates. Controlling the confidence, or precision, of differences is important for scientific data discovery. Therefore, we have shown that transfer learning algorithms provide this type of control over the number of differences learned. Previous chapters (and related papers) demonstrate that learned networks benefit from transfer learning in terms of probability distribution matching or fit to validation data. This research, further demonstrates the power of transfer learning for making discoveries not only about the shared sub-structures of networks, but also the differences.

We demonstrated empirically that this transfer learning approach achieves higher precision than existing methods for learning differential dependency networks. At the same time, the presented approach yields better performance than the significantly more expensive bootstrapping procedures. Through three real case studies, domain experts used the proposed techniques to uncover compelling evidence of biological processes involved in cancer and human learning. These case studies provide further evidence that human control over transfer bias guides solutions towards those of most interest to the end-user.

Chapter 6

Interactive Exploration of Hyper-Parameters

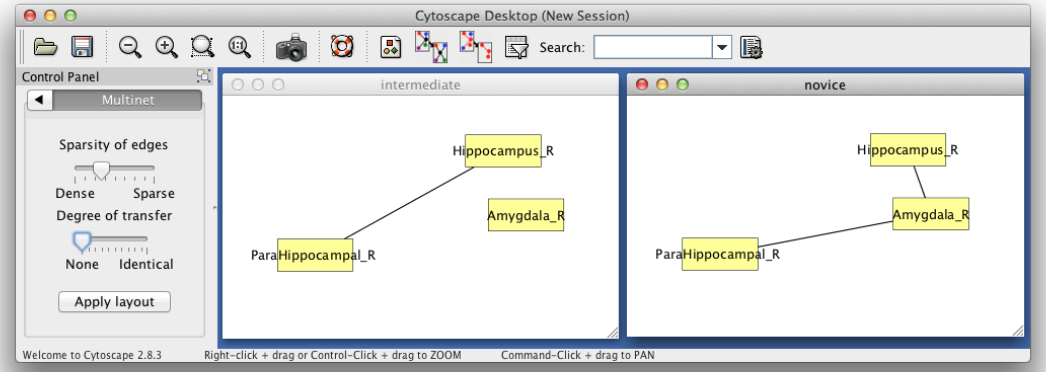
We have shown that domain knowledge about the strength of transfer shapes the space of solutions to find models that are of interest to the domain expert. We can give the domain expert the ability to see how different choices of algorithm hyper-parameters, including transfer strength, affect the solution. In order to give the domain expert (or any end-user) such control, the learning algorithm must be able to receive feedback from a user and update the solution in realtime. Furthermore, a reasonable mechanism for providing feedback must exist such that the control given to the user is intuitive enough to use without being a machine learning expert. This control should also be simple enough that the algorithm will be able to incorporate such feedback quickly. To meet these goals, we propose to present a visual display of learned networks and their differences to the user, along with control knobs to adjust the number of edges learned in each task and the number of differences learned among pairs of tasks. As the user provides feedback about whether these numbers should move up or down, the learning algorithm adjusts its own hyper-paramaters, re-calculates the solution and presents the resulting visualization of the solution to the user. This approach gives an interactive visualization of the space of solutions.

6.1 Motivation

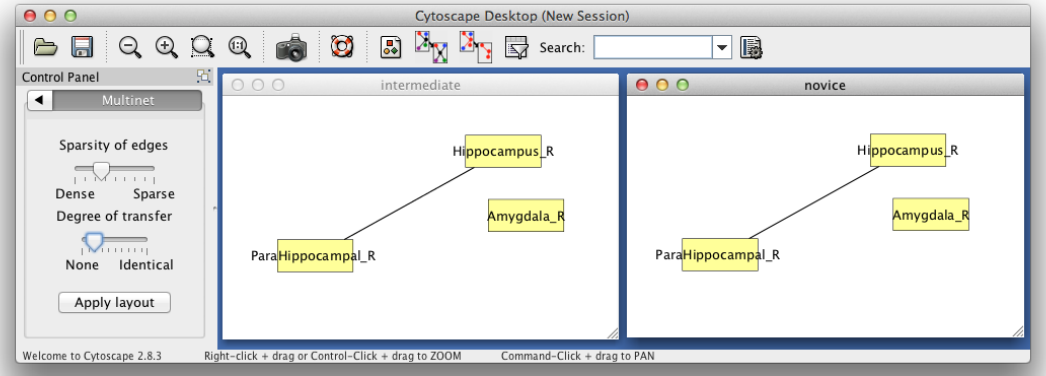
We are interested in identifying patterns of dependency among variables in related sets of data. Graphical models encode a sparse dependency structure that is easily visualized to give insight about the relationships among variables in a system. In particular, we are interested in comparing these dependency structures for multiple related datasets, for example sub-populations in a study, or different experimental conditions. In earlier chapters, this dissertation has shown that multitask graphical models address this problem (Danaher, Wang, and Witten, 2011; Niculescu-Mizil and Caruana, 2007). Multiple graphs are learned simultaneously, producing models that are similar, except where the data strongly supports differences, easing comparison (see example in Figure 6.1). Hyper-parameters determine how similar the learned graphs are and how many edges are present in each graph. Typically these hyper-parameters are determined through trial and error after examining the learned graphs or through a computationally expensive grid search by optimization with respect to holdout data (Meinshausen and Bühlmann, 2006; Van Allen and Greiner, 2000). The choice of which criteria to optimize becomes a model selection problem in itself (see in particular the discussion section of Van Allen and Greiner (2000)). We propose giving the user interactive control so that they can explore the space of possible solutions and more quickly find good values for the hyper-parameters.

Multitask graph structure learning is a promising direction for knowledge discovery in many scientific domains. However, there remain issues of practical concern; namely, the exploration of the solution space for different settings of hyper-parameters. 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). Both parameters are difficult to tune automatically from the data. Much research has gone into optimizing the sparsity parameter (Maslov and Sneppen, 2002) without a clear resolution to the problem.

Solutions to the model selection problem generally fall into two categories. The



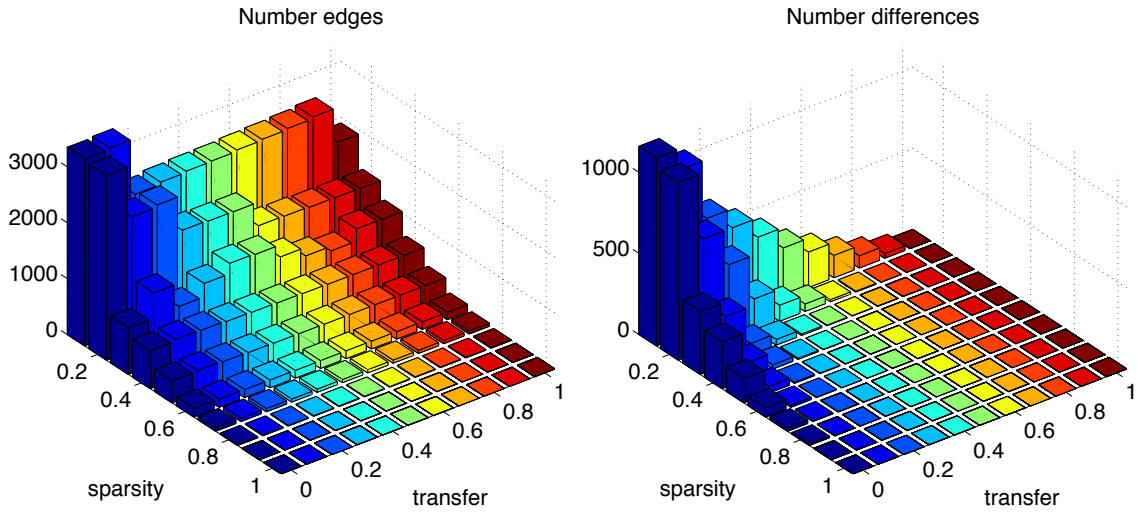
(a) Independently learned graphs



(b) Graphs learned with some transfer

Figure 6.1: Example of a sub-graph learned from neuroimaging data. When the graphs are learned independently, the connections are different. However, with even a little bit of transfer bias encouraging them to be similar, the differences disappear, suggesting that the likelihood of that difference being real is not very strong.

first is to use some form of complexity penalty, such as an information criterion or minimum description length. This penalty term is chosen a priori to fit assumptions about the problem domain and the ideal solution (Van Allen and Greiner, 2000). The primary problem with this approach is that when there is limited data or the assumptions are incorrect, the criterion may perform arbitrarily badly in terms of generalization performance, but it is not always easy to tell if this is the case. The second solution is to use some form of grid search over the space of parameters. The

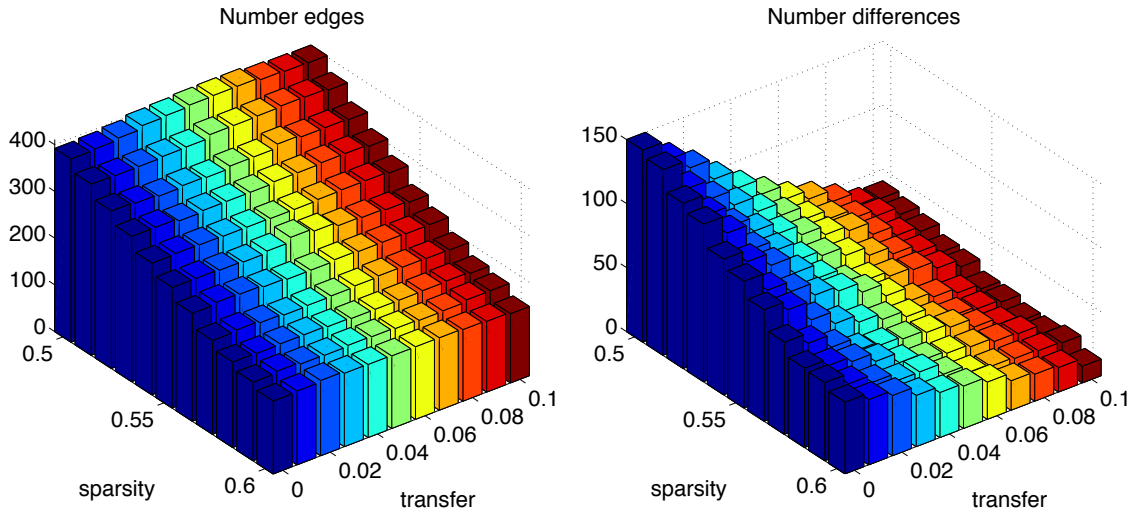


(a) Number of learned edges (b) Number of learned differences

Figure 6.2: Accelerated learning study: summary statistics about learned network models for various values of sparsity and transfer hyper-parameters.

model is then selected by choosing the solution that best fits some choice of optimal performance, most typically a fit to hold-out validation data (Niculescu-Mizil and Caruana, 2007; Oyen and Lane, 2012). This approach works well for optimizing generalization performance (Van Allen and Greiner, 2000) but may not fit assumptions about the simplicity of the underlying process that generated the data. It also suffers from the problem of reducing the amount of available training data because some data must be reserved for validation. The reduction of training data means that the solution will have higher variance than if all of the training data were used. In practice, the ideal model in a grid search is often chosen because it "looks right" (Danaher, Wang, and Witten, 2011). Bootstrapping procedures, such as stability selection (Liu, Roeder, and Wasserman, 2010), also fall into this grid-search category. Any grid search is computationally expensive because the learning algorithm must be run many times. Bootstrapping methods incur even more computational cost because the learning algorithm is run over various sample sets. Moreover, learning the best model requires that the best hyper-parameters are included in the grid search.

To illustrate the problem of grid search, we take a closer look at the Accelerated Learning fMRI Study from the previous chapter (Section 5.5.3). We assigned the



(a) Number of learned edges

(b) Number of learned differences

Figure 6.3: Accelerated learning study: summary statistics about learned network models for a fine grid of values of sparsity and transfer hyper-parameters.

sparsity hyper-parameter to 10 values evenly spaced in the range $[0.1, 1]$ and we assigned the transfer hyper-parameter to 11 values evenly spaced in the range $[0, 1]$. All combinations of sparsity and transfer settings were run through the multitask graphical lasso algorithm. The results from that grid are the solutions that were displayed to the domain expert in the previous chapter. Here, we take a broader look at the results from all 110 settings of these parameters. Figure 6.2a 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 6.2b 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 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 6.3a gives the number of individual edges learned in each networks while Figure 6.3b 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. However, run such a fine grid over the entire range of possible values would be prohibitively expensive. Therefore, the typical workflow is to first run a coarse grid over the entire space of hyper-parameter values. Then, upon inspection of the results, a finer grid is run over a range of values that promise the most interesting solutions.

In some applications, if there is a robustly optimal setting of the hyper-parameters, then it may be possible to find it faster via interactive exploration than through naive exhaustive grid search. On the other hand, there may not be a single best solution, particularly given limited data and the goal of scientific discovery of patterns. The goal, then, is not so much to find the single best solution, but to provide all meaningful solutions to the user. In this case, grid search is effective at producing many solutions. However, a grid that is evenly spaced in hyper-parameter space often produces uneven exploration of the solution space. In practice the hyper-parameter grid search is performed iteratively: results are visually inspected, then several new hyper-parameter values in areas of interest are applied, and the cycle is repeated until the results explore the space of solutions of interest. However, the user will most likely want to see solutions from a narrow range of parameter values that are difficult to guess a priori as would be necessary for naive (non-interactive) grid search. Interaction provides a means to efficiently explore the space of solutions.

We want to provide an interactive exploration of learned graphical models. Our goal is to incorporate this into an interactive visualization system, pictured in Figure 6.4. A single setting of the hyper-parameters does not give the full picture that domain scientists may want to see. Therefore, we propose a graphical structure learn-

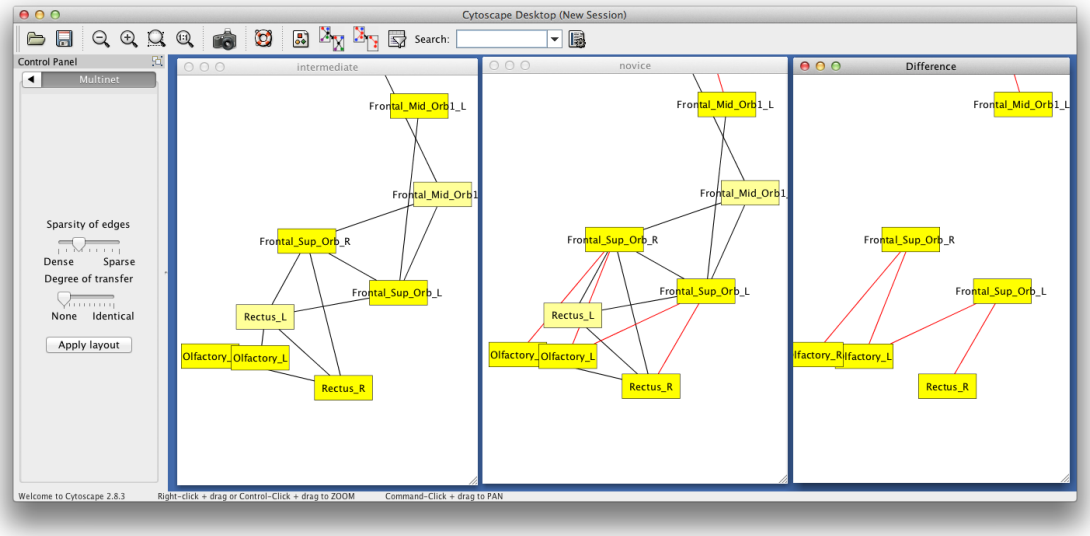


Figure 6.4: Interactive multi-graph visualization. Our system consists of the following components: a visual display of multiple learned graphs, user controls to increase/decrease the number of edges in each graph, user controls to increase/decrease the degree of similarity among pairs of graphs, efficient update of learned graphs in response to user controls.

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

6.2 Current Approach and 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. The plugin interface allows us to customize the display for comparison of multiple graphs (see Figure 6.4). We have also incorporated sliders that allow a user to modify the sparsity and degree of transfer among networks. Originally, these sliders simply looked up the pre-computed graphs learned from a list of parameter values. The user did not have any control over the granularity of the slider, and furthermore, changing a 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. This requires an algorithm to estimate the necessary change in the hyper-parameter values that will effect the result requested.

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 (Amershi et al., 2011) and gives the user control over the objective function (Kapoor et al., 2012). 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.

Graph structure learning is an unsupervised learning domain and so there may not be an optimal parameter setting. In Meinshausen and Bühlmann (2006) they show that even the oracle value of hyper-parameters does not guarantee optimal performance, instead they recommend 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 (Niculescu-Mizil and Caruana, 2007; Oyen and Lane, 2012) and subjective human-selection being another choice (Danaher, Wang, and Witten, 2011). Yet, as argued in the previous chapter, 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 (Chang et al., 2009). 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.

6.3 Updating Learned Graphs with User Feedback

Similarly to all problems considered in this dissertation, we have several sets of data, D_k for $k \in \{1, 2, \ldots, K\}$, for which we will 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. Unique to this chapter, therefore, 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 as already shown. 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.

A graph, G_k, is a compact representation of a joint distribution, with a set of vertices V and a set of edges E. In this chapter, we represent the set of edges in all of the K graphs with $\mathbf{E}$, an $|E| \times K$ binary matrix. Each entry E_{ik} represents the presence or absence of the edge i 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 binary matrices $\mathcal{S} = \{S_1, \ldots, S_K\}$ that

correspond to the graphs $\mathcal{G}$ (explained in further detail later).

6.3.1 Sketch of Interactive Approach

Our interactive approach, therefore, alternates between learning the graph structure $\mathbf{E} = f(\mathcal{D}, \Lambda)$ and learning the 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 independently ($\lambda_2 = 0$) 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, $\mathcal{S}$, we compute the necessary Λ to achieve the change requested (details in the next section). Using the computed Λ, we re-learn the graphs, $\mathcal{G}$, and update the visualization, allowing the user to further interact until satisfied with the solution.

6.3.2 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. The number of edges in each graph and the amount of similarity can be affected by both hyper-parameters, so we must estimate 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 $\mathbf{E}$ ($|E| \times K$). Each entry, S_{ik}, indicates the user's desire to see the presence or absence of edge i in task k. Through this representation, we can move the learned structures $\mathbf{E}$ in the direction of the user preferences $\mathbf{S}$ by finding an appropriate adjustment to Λ.

An example will help illustrate how the user representation works. Consider the example where a user wishes to see fewer differences between tasks a and b. Let the

currently existing set of edges in graph a be A and the set of edges currently existing 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 $\mathbf{E}$, i.e. $S_{ek} = E_{ek} \ \forall e \notin U$ and $\forall k$.

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

- **Fewer edges in task i:**
  Assign $S_{ei} = 0 \quad \forall \ e \in E$.
  Assign $S_{ek} = G_{ek} \quad \forall \ e \in E$ and $\forall \ k \neq i$.

- **More edges in task i:**
  Assign $S_{ei} = 1 \quad \forall \ e \in E$.
  Assign $S_{ek} = G_{ek} \quad \forall \ e \in E$ and $\forall \ k \neq i$.

- **Fewer edges in task i that are not in task j:**
  Define set $U = \{e \in E \ \big| \ G_{ei} = 1 \wedge G_{ej} = 0\}$.
  Assign $S_{ei} = 0 \quad \forall \ e \in U$.
  Assign $S_{ei} = G_{ei} \quad \forall \ e \in \{E \setminus U\}$.
  Assign $S_{ek} = G_{ek} \quad \forall \ e \in E$ and $\forall \ k \neq i$.

- **More edges in task i that are not in task j:**
  Define set $U = \{e \in E \ \big| \ G_{ei} = 0 \wedge G_{ej} = 0\}$.
  Assign $S_{ei} = 1 \quad \forall \ e \in U$.
  Assign $S_{ei} = G_{ei} \quad \forall \ e \in \{E \setminus U\}$.
  Assign $S_{ek} = G_{ek} \quad \forall \ e \in E$ and $\forall \ k \neq i$.

6.3.3 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 **E**.

$$g(\Lambda) = \sum_{k=1}^{K} \sum_{e \in E} (S_{ek} - G_{ek}(\Lambda))^2 \tag{6.1}$$

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 6.2.

$$\begin{aligned} \nabla_\Lambda g &= -2 \sum_{k=1}^{K} \sum_{e \in E} (S_{ek} - G_{ek}(\Lambda)) \cdot \nabla_\Lambda G_{ek}(\Lambda) \\ &= -2 \cdot \mathbf{J}_\Lambda(\vec{G}) \cdot (\vec{S} - \vec{G}) \end{aligned} \tag{6.2}$$

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.

$$\Lambda^{\text{new}} = \Lambda - \eta \cdot \nabla_\Lambda g \tag{6.3}$$

6.3.4 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 this chapter, 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 user may never notice the delay.

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).

6.4 Exploration of Multitask Bayesian Networks

First, we will review the Bayesian discovery of Bayesian networks algorithms, 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.

6.4.1 Preliminaries

Bayesian structure discovery produces a posterior estimate of the likelihood of each edge in a Bayesian network. For multitask Bayesian networks, there will be a posterior estimate of the expectation of each edge in each task organized 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 $v \rightarrow i$ exists (and $f_i(\pi_i) = 1$) iff $v \in \pi_i$, 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. For a given ordering, the parents of a node i must precede i in the order.

$$w_{ek} = \sum_{\prec} P(\prec) P(f^{(k)}, \mathcal{D} | \prec) \tag{6.4}$$

Koivisto and Sood (2004) give a relatively efficient method for exactly calculating each w_e for single-task learning. Their 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 $C(m)$ that depends on the number of samples m. The maximum number of parents allowed for any node is typically fixed to a small natural number, r. Therefore, there are $O(N^{r+1})$ of these functions to calculate for a total computational complexity of $O(N^{r+1}C(M))$.

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, all of the α functions can be computed in $O(N2^N)$ 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 $N!$ 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 $O(N2^N)$.

The total computational complexity for a single task is $O(N2^N + N^{r+1}C(m))$. This is the exact calculation of the posterior. For large networks, roughly $N > 30$, the exponential term is intractable. In these cases, we can use MCMC to approximate the sum over orders. 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.

Previously, we showed how to replace the single-task prior bias $P(\pi_i)$ with a transfer bias $P(\pi_i^{(k)}, \pi_i^{(j)})$ that share information among tasks k and j. In terms of the

three-step method of Koivisto and Sood (2004), this means replacing their β functions with:

$$\begin{aligned}\beta_{ki}(\pi_i, \lambda_2) =& f_i(\pi_i^{(k)})P(x_i^{(k)}|\pi_i^{(k)})P(\pi_i^{(k)}, \pi_i^{(j)}) \\ =& f_i(\pi_i^{(k)})P(x_i^{(k)}|\pi_i^{(k)}) \times \frac{1}{(K-1)(4-\lambda_2)^{|U_i|}} \times \\ & \left[\sum_{j \neq k} \sum_{\pi_i^{(j)} \subseteq U_i} P(x_i^{(j)}|\pi_i^{(j)})(1-\lambda_2)^{\Delta(\pi_i^{(k)}, \pi_i^{(j)})}\right] .\end{aligned} \tag{6.5}$$

Under single-task learning, we assume each β function takes time $C(m)$ to compute. Under transfer learning, there is a now a sum over parent sets for each task, therefore the computational complexity is $O(KN^r)$ for each β function. There are $KN \cdot N^r$ of these functions to calculate. This gives a total computational complexity for all multitask β functions of $O(K^2N^{2r+1})$.

Once the multitask β functions are calculated, the rest of the posterior estimate can be calculated using existing algorithms, such as the exact computation (Koivisto and Sood, 2004; Parviainen and Koivisto, 2009) or MCMC approximations (Niinimaki, Parviainen, and Koivisto, 2011).

6.4.2 Efficient Computation of Transfer Bias

Unfortunately, this sum can be a rather large polynomial in N. However, we can store intermediate calculations that will 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.

$$\begin{aligned}\sum_{\pi_i^{(j)} \subseteq U_i} P(x_i^{(j)}|\pi_i^{(j)})(1-\lambda_2)^{\Delta(\pi_i^{(k)}, \pi_i^{(j)})} &= \sum_{\delta=0}^{r} \sum_{\pi_i^{(j)}|\Delta(\pi_i^{(k)}, \pi_i^{(j)})=\delta} P(x_i^{(j)}|\pi_i^{(j)})(1-\lambda_2)^{\delta} \\ &= \sum_{\delta=0}^{r} (1-\lambda_2)^{\delta} \sum_{\pi_i^{(j)}|\Delta(\pi_i^{(k)}, \pi_i^{(j)})=\delta} P(x_i^{(j)}|\pi_i^{(j)})\end{aligned} \tag{6.6}$$

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:

$$\gamma_{ki\delta}(\pi_i, \delta) = \sum_{j \neq k} \sum_{\pi_i^{(j)} | \Delta(\pi_i^{(k)}, \pi_i^{(j)}) = \delta} P(x_i^{(j)} | \pi_i^{(j)}) \quad \text{for all } \pi_i \subseteq V - \{i\},\ \delta \in \mathbb{Z},\ 0 \leq \delta \leq r \ . \tag{6.7}$$

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: KrN^{r+1}, one for every family in every task for every value of δ. The calculation of all of these γ functions is $O(K^2 r N^{2r+1} C(m))$.

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.

$$\beta_{ki}(\pi_i, \lambda_2) = \frac{f_i(\pi_i^{(k)}) P(x_i^{(k)} | \pi_i^{(k)})}{(K-1)(4-\lambda_2)^{|U_i|}} \cdot \sum_{\delta=0}^{r} (1-\lambda_2)^{\delta} \gamma_{ki\delta}(\pi_i, \delta) \tag{6.8}$$

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

6.4.3 Thresholding for graphs

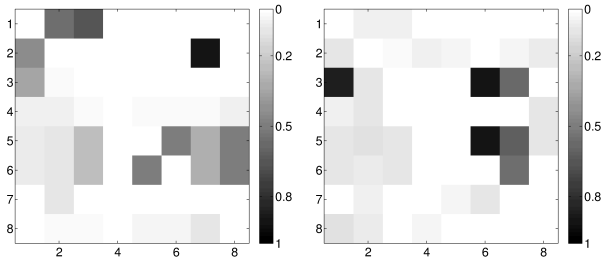
The feature probabilities, w_{ek}, learned from Equation 6.4 can be organized into square matrices W_k for each task k representing the directed edges of a network. Figure 6.5 shows an example of these learned feature posterior probabilities.

In order to display graphs to the user (see Figure 6.6), 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:

$$G_{ek} = \frac{1}{1 + \exp[-\beta(w_{ek} - \lambda_1)]} \ . \tag{6.9}$$

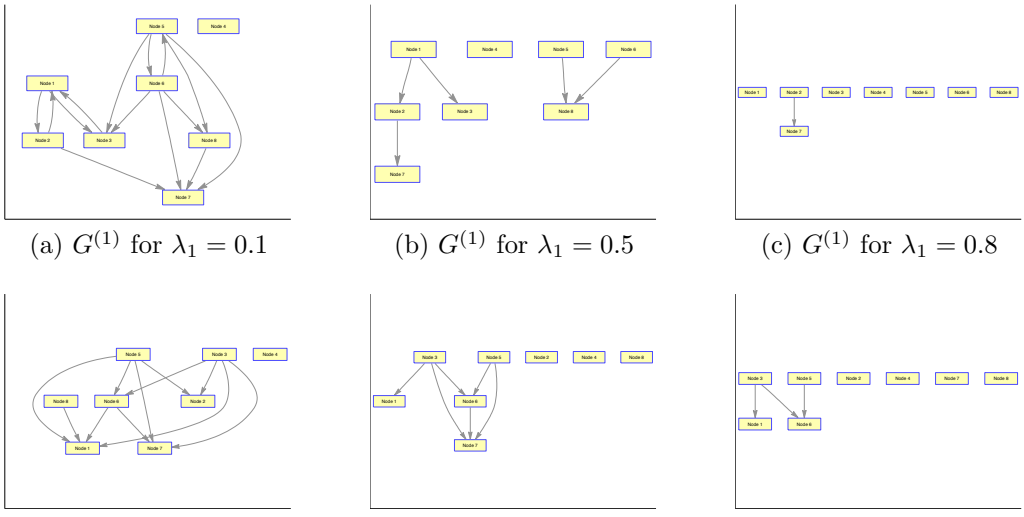
For sufficiently large values of $\beta > 1$ this is equivalent to a hard threshold at λ_1.

When comparing the similarities and differences among a set of graphs, it is helpful to be able to control for the number of differences between the graphs. By encouraging



(a) $W^{(1)}$ (b) $W^{(2)}$

Figure 6.5: Estimated posterior likelihoods for two tasks with $\lambda_2 = 0$. There are 8 variables, and therefore 8×7 possible directed edges, which we have organized into a weighted adjacency matrix.

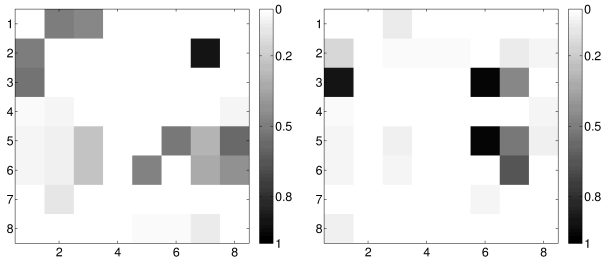


(a) $G^{(1)}$ for $\lambda_1 = 0.1$ (b) $G^{(1)}$ for $\lambda_1 = 0.5$ (c) $G^{(1)}$ for $\lambda_1 = 0.8$

(d) $G^{(2)}$ for $\lambda_1 = 0.1$ (e) $G^{(2)}$ for $\lambda_1 = 0.5$ (f) $G^{(2)}$ for $\lambda_1 = 0.8$

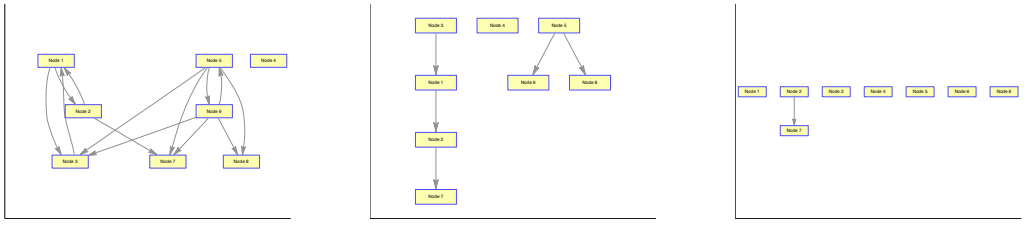
Figure 6.6: By thresholding at λ_1, we obtain graphs $G^{(k)}$ from the weighted adjacency matrices $W^{(k)}$.

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 6.7 and 6.8). The λ_2 parameter controls the amount of similarity bias.

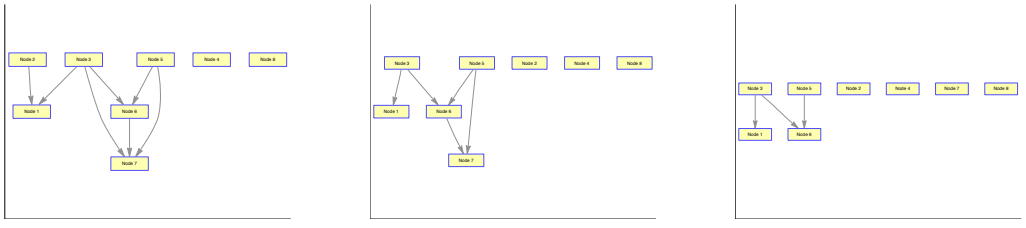


(a) $W^{(1)}$ (b) $W^{(2)}$

Figure 6.7: 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.



(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 6.8: By thresholding at λ_1, we obtain graphs $G^{(k)}$ from the weighted adjacency matrices $W^{(k)}$ with $\lambda_2 > 0$.

6.5 Numerical Estimation of Hyper-Parameters

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.

6.5.1 Estimation of Λ for Multitask Bayesian Networks

To re-learn graphs after getting feedback from the user, we need to calculate the Jacobian $\nabla_\Lambda G_{ek}(\Lambda)$ in Equation 6.2. The partial derivative with respect to λ_1 is fairly straightforward.

$$\begin{aligned}\frac{\partial}{\partial\lambda_1}G_{ek} &= \frac{-\beta e^{-\beta(w_{ek}-\lambda_1)}}{1+e^{-\beta(w_{ek}-\lambda_1)}} \\ &= -\beta e^{-\beta(w_{ek}-\lambda_1)}G_{ek}\end{aligned} \tag{6.10}$$

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.

$$\begin{aligned}\frac{\partial}{\partial\lambda_2}G_{ek} =& -\beta e^{-\beta(w_{ek}(\lambda_2)-\lambda_1)}G_{ek}\sum_{\prec}\sum_{\pi_e^{(k)}\subseteq U_e} f_e(G^{(k)})P(x_e^{(k)}|\pi_e^{(k)})\times \\ &\left[\sum_{\pi_e^{(j)}\subseteq U_e} P(x_e^{(j)}|\pi_e^{(j)})\frac{-\Delta(1-\lambda_2)^{\Delta-1}+|U_i|(1-\lambda_2)^{\Delta}(4-\lambda_2)^{-1}}{(4-\lambda_2)^2}\right] \\ =& -\beta e^{-\beta(w_{ek}(\lambda_2)-\lambda_1)}G_{ek}\sum_{\prec}\sum_{\pi_e^{(k)}\subseteq U_e} f_e(G^{(k)})P(x_e^{(k)}|\pi_e^{(k)})\times \\ &\left[\sum_{\pi_e^{(j)}\subseteq U_e} P(x_e^{(j)}|\pi_e^{(j)})\frac{(1-\lambda_2)^{\Delta_{ikj}}}{(4-\lambda_2)^2}\cdot\left(\frac{|U_i|}{4-\lambda_2}-\frac{\Delta_{ikj}}{1-\lambda_2}\right)\right]\end{aligned} \tag{6.11}$$

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

$$\begin{aligned}\frac{\partial}{\partial\lambda_2}G_{ek} =& -\beta e^{-\beta(w_{ek}(\lambda_2)-\lambda_1)}G_{ek}\sum_{\prec}\sum_{\pi_e^{(k)}\subseteq U_e} f_e(G^{(k)})P(x_e^{(k)}|\pi_e^{(k)})\times \\ &\left[\sum_{\delta=0}^{r}\frac{(1-\lambda_2)^{\delta}}{(4-\lambda_2)^2}\cdot\left(\frac{|U_i|}{4-\lambda_2}-\frac{\delta}{1-\lambda_2}\right)\gamma_{ke\delta}(\pi_e^{(k)},\delta)\right]\end{aligned} \tag{6.12}$$

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.

$$\sum_{e,k}|S_{ek}-G_{ek}(\Lambda^{\text{new}})| - \sum_{e,k}|S_{ek}-G_{ek}(\Lambda)| = -1 \tag{6.13}$$

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

6.6 Discussion

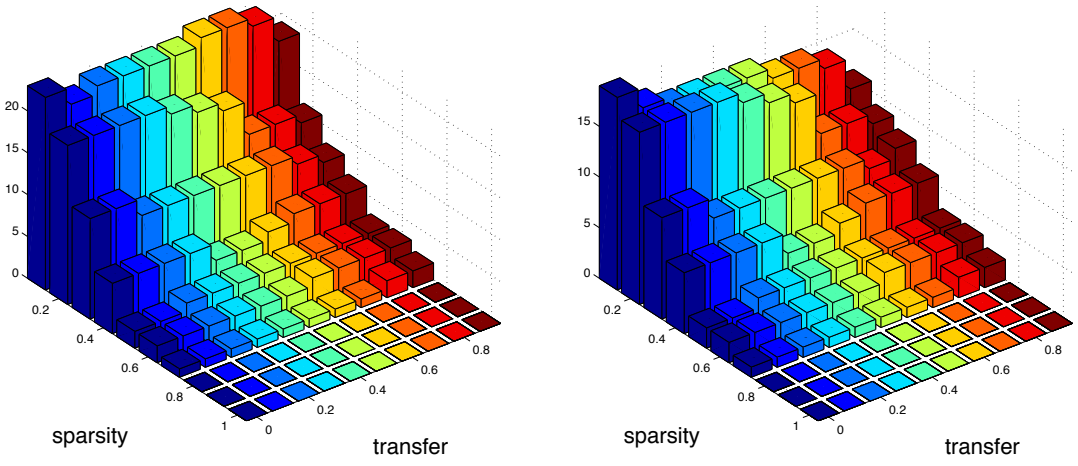
There is strong motivation for creating an interactive human-in-the-loop algorithms for exploring comparative dependency networks. Here we discuss our initial findings on benchmark networks, share case studies on real data and then suggest directions for future work.

6.6.1 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 (Lauritzen and Spiegelhalter, 1988). 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.

Using two tasks, and starting with an initial value for Λ, we learn networks G. Then simulated feedback responses, S, are given. For each of these feedback matrices, we track the movement in Λ to investigate the effect of 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 6.9 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.

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. Yet, subjective measures of interestingness should be gauged by a human with knowledge



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

Figure 6.9: Modified *asia* networks: summary statistics about learned network models for various values of sparsity and transfer hyper-parameters.

about the domain. 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.

6.6.2 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 sparsity and transfer. 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 found that different domain experts were 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.

6.6.3 Future Work

The concept of interactive network comparison is compelling. This chapter provides a method for creating interactive algorithms. These algorithms remain a computationally challenging problem. The example of Bayesian posterior distributions on multiple Bayesian networks given in this chapter, 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.

As seen in the previous chapter, graphical lasso scales to large networks much better than Bayesian networks. Therefore, we would like to apply the proposed in-

teractive method to multitask graphical lasso. However, the graphical lasso objective with respect to λ_1 and λ_2 is discontinuous; therefore, the gradient (Equation 6.2) 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 (Efron et al., 2004; Schmidt, Niculescu-Mizil, and Murphy, 2007). However, there is not any such algorithm for multitask network learning. Multitask learning involves two regularization penalties, therefore it is a surface of regularization that needs to be computed.

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. 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. Woking 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 chapter 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, and then to see whether the human users

or the automated approaches are more effective at finding interesting solutions.

6.7 Conclusions

This chapter presents an interactive system for learning multiple networks, displaying them to the user, gathering feedback from the user to interactively adjust the hyper-parameters of the learning algorithm and explore the space of solutions. This approach is general enough to apply to various types of transfer structure learning algorithms. The specific details for estimating hyper-parameters and quickly updating solutions will be specific to each algorithm. The concept of interactive control over hyper-parameters is even more broadly applicable to many problems in unsupervised learning. The goal is to be able to display a solution to the user, and provide a means for the user to direct exploration of the solution space. Interactive methods like this have been shown to perform quantifiably better in supervised learning domains. In unsupervised domains, it may be more difficult to quantify performance, but this is precisely why it should be even more beneficial to the user to explore the space of solutions. Many applications for interactive machine learning are imaginable. In practice, all machine learning applications involve some form of interaction between looking at results and adjusting the algorithm to produce better results. Automating this interactive process allows domain scientists and other end-users to work more efficiently to discover patterns in their data.

Chapter 7

Conclusions

The comparison of multiple dependency networks is a challenging machine learning problem with the potential to have impact in diverse scientific domains. To be of practical use, the domain expert must be able to guide the machine learning objective toward solutions that provide answers to comparative queries about patterns in the data. This dissertation provides a general framework for incorporating domain knowledge in multitask network structure learning algorithms. Typically machine learning algorithms have optimized the fit of learned models to data. Yet, many solutions may be nearly equivalent in this regard and knowledgeable scientists have questions that are more specific about the features of the models themselves. In order to explore these questions, end-users must be able to give feedback to the learning algorithm. A specific example on real data is given concerning learning high-confidence differences among learned network models. Finally, a complete system for learning models, displaying solutions, gathering feedback from a user and updating models is provided. Altogether, this dissertation provides a means to interactively explore and compare network models learned from multiple related datasets.

7.1 Discussion

Specific contributions of this dissertation to the field of multiple network structure learning include the incorporation of human knowledge about task relationships. This

human knowledge improves the quality of models learned by the transfer learning algorithm while shaping the topology of the solution space to facilitate comparisons among learned network structures. Learning multiple solutions through Bayesian posterior distribution modeling or for various settings of hyper-parameters give domain experts a comprehensive picture of possible patterns in the data, aiding discovery of insight into the underlying processes that produced the data. An interactive human-in-the-loop machine learning algorithm is the culmination of this project; allowing a human to guide the machine learning objective function while exploring the space of solutions.

Prior knowledge about task-relatedness improves transfer. Transfer learning algorithms were created as data-driven methods for improving the robustness of learned models in the face of limited data. However, common scenarios of multiple related data sets include tasks that are not all equally related. Often there is domain knowledge or meta-information about the relationships among tasks. A few examples of these scenarios exist in the literature, including network structures that change over time (Husmeier, Dondelinger, and Lèbre, 2010). Structures should only be directly related to those structures that are adjacent in time. Such straightforward domain knowledge has not been easy to incorporate into multitask learning algorithms for networks structure learning. Therefore, the first contribution of this dissertation is to provide a general framework for incorporating prior knowledge about task-relatedness in multitask network structure learning algorithms. This framework is a generalization of the specific case of standard multitask learning in which all pairs of tasks are equally related. The framework also generalizes specific cases that already exist in the literature, particularly for networks that evolve over time.

Using this general framework, we develop a task-relatedness aware multitask Bayesian network structure learning algorithm. The task-relatedness metric allows more flexibility in the objective function and empirically, we show that it learns more accurate network structures than standard multitask structure learning. Furthermore, on real neuroimaging data, we show that the task-relatedness aware framework learns

models that better fit validation data, even though the true task-relatedness metric is unknown. In summary, transfer learning can be improved by the use of domain knowledge about the relative relatedness of pairs of tasks.

The degree of transfer strength, as set by a human, shapes the topology of the solution space. We give several transfer learning approaches that shape the topology of the solution space. The first approach is the multitask Bayesian network structure learning algorithm developed under the general task-relatedness aware multitask (TRAM) framework. The algorithm performed at least as well as the naive multitask learning algorithm on synthetic data specifically designed to test the accuracy of the algorithms. However, what is even more compelling, is the practicality of using TRAM on real data. In real data, there is no guarantee that the given task-relatedness metric accurately reflects the true similarity of the generative networks. Yet, the task-relatedness metric provides guidance to the learning algorithm about the topology of solutions that are of interest to the end-user. We demonstrated this phenomenon with neuroimaging data from schizophrenia patients on various types of medication. In this case, neuroscientists are interested in finding changes in functional brain connectivity due to the mental illness (unmitigated by medication) and then changes due to specific types of medication. The task-relatedness metric allows for such comparisons to take place that would not be so straightforward under the standard multitask learning model or the naive approach of learning each network independently.

Other methods of determining task-relatedness may better be able to better fit the data; however, they would not provide information to the domain expert about how the functional brain network responds to various types of medication. By this example, we show that the standard data-driven approach of machine learning objectives do not always capture the pattern discovery objective of the end-user. The objectives of the end-user can be incorporated into the learning objective of the algorithm through Bayesian prior distributions or biases. These biases, in conjunction with the standard fit to data objective, shape the solution space to reflect the query of the end-user

while still finding patterns supported by the data.

Tractable incorporation of prior knowledge in Bayesian posterior distribution learning. Bayesian posterior distributions over the space of solutions provide invaluable information about all possible dependencies in the data. Solutions that give a general sense of all possibilities, rather than just a single optimal solution, are of particular interest to domain scientists. This dissertation provides a tractable method for incorporating transfer bias into Bayesian posterior distribution learning algorithms for Bayesian network structure learning. We show how to use this transfer bias to learn multiple related Bayesian networks with the assumption that learned networks should have many edges in common. Empirical results demonstrate that this transfer bias reduces the posterior likelihood of spurious edges for all tasks. For the scientific problems considered in this dissertation, this is an important contribution for learning the full picture of possible solutions, rather than just a single point solution.

The theorem provided in Chapter 4 for incorporating order-modular structure bias could also be used by structure learning algorithms with prior knowledge about the existence of particular structural features. Other algorithms have been proposed for such purposes that use the slower, less-convergent search procedures over structure space rather than order space (Grzegorczyk and Husmeier, 2008). The theorem given here provides a means for incorporating structural knowledge into algorithms that operate in order space.

Transfer learning produces higher confidence differences than existing approaches. A common question asked by domain scientists is how dependencies are different between related tasks. For example, which protein correlations in patients' blood proteins are associated with cancer? Attempts at answering questions like this have produced various unprincipled ad-hoc approaches in domain literature. Machine learning approaches have centered on two approaches: discriminative models and bootstrapping procedures. Discriminative models do not really answer the

question of which dependencies exist in each task, as they do not learn dependencies that may have actually produced the data. Bootstrapping procedures have serious computational costs and we show that their ability to produce high-confidence differences is limited. Somewhat surprisingly, transfer learning has not been previously proposed as a solution to this problem, yet we show that existing transfer learning algorithms perform quite well at producing high-precision differences among learned networks. The intuition is that by biasing network structures to be similar, only those differences that are strongly supported by the data will survive.

Furthermore, the level of confidence in learned edges is directly controlled by the transfer strength parameter. For different end-users or various data sets being investigated, the desired number and confidence of these identified differences can vary. We show that giving the end-user the ability to see which dependencies change with the transfer strength enables them to make discoveries that are relevant to their field of research.

User feedback about solutions encoded as a learning objective. Building off of the findings about incorporating domain knowledge and giving control to the user in exploring solutions, we provide a new machine learning workflow with a human-in-the-loop. To achieve this workflow, we incorporate domain knowledge into the learning objective, give the end-user a full picture of possible solutions to their question, and give the end-user the ability to vary the confidence of learned solutions. This human-guided process is in contrast to existing machine learning workflows in which there is no feedback loop. Typically, data and a learning objective is provided to the machine learning algorithm and a solution (or many solutions) is produced. If the end-user wants to change the learning objective, in the traditional workflow, then the machine learning algorithm would need to be re-run. That workflow generally requires that the domain expert and the machine learning programmer discuss the changes to be made, a process that could take days or weeks. Automating this feedback loop requires that the end-user can see solutions, provide feedback, and get updated solutions in realtime.

Chapter 6 provides a system for collecting and incorporating such feedback from the end-user. Based on the current network structures learned, a user provides feedback about increasing or decreasing the number of edges or differences included in the solution. This is a relatively intuitive change that a human can suggest, and we show that it can be represented in a learning objective to be optimized by the machine learning algorithm.

Interactive system for visualizing results, gathering user feedback and updating solutions. This dissertation provides an interactive multiple network visualization tool. The fully automated tool displays a solution to the multiple network structure learning algorithm. The visualization of the solution eases comparative observations of dependency patterns in the data. The end-user can request changes to the solution, such as increasing or decreasing the number of edges or differences in the solution. This request is encoded as feedback from the user and incorporated into a machine learning objective function. Using this feedback, new hyper-parameters are estimated that produce the results required by the user. These new hyper-parameters are used in the multitask network structure learning algorithm to update the solution, producing a result that fits both the data and the user's desires.

Currently, this general interactive approach has been applied to the problem of multitask Bayesian network structure learning. These models do not scale well to large networks. Future plans include applying this general approach for interactively learning multiple graphical lasso models. These models scale to larger size networks, however estimating the updated hyper-parameters is computationally challenging and so this is not a trivial problem to solve.

7.2 Future Work

Transfer learning has been shown empirically to provide higher confidence in learned edges as well as difference between tasks. However, quantifying the level of this confidence is difficult because the edges are not independent of each other. Methods

for quantifying confidence generally resort to computationally expensive bootstrapping procedures. Bootstrapping also reduces the amount of training data available for learning models, thereby reducing the estimation power. Other ways for measuring confidence on the same data that is used to train the data may be possible, but must be carefully considered in light of the fact that features are not independent of each other. The calculation of confidence intervals would greatly increase the value of transfer learning algorithms for practitioners.

As presented in this dissertation, the task-relatedness aware multitask learning framework assumes that there is some form of prior knowledge that encodes the relatedness of pairs of tasks. In real data, we have shown that such information is often available. However, the task-relatedness metric of the framework could be treated as an unobserved variable to be learned. Such a data-driven approach would be interesting to see if the learned model will better fit the data, although the learning procedure will be considerably more expensive. Learning the task-relatedness metric directly from data could be an interesting solution in itself if it gives domain scientists insight into the relationships among various sets of data.

This dissertation concentrated on real data in the areas of neuroimaging and plasma protein analysis. Some issues are common to both of these areas, such as the non-Gaussian nature of the data, while other issues are unique to each data set. Many other applications that involve comparisons among dependencies in multivariate data are possible. Such areas of research include gene expression networks, social influence networks, infectious disease pathways, and others. Hopefully, the algorithms presented here can be applied directly to these other applications, but it would be interesting to explore any issues that are unique to other data sets.

7.3 Closing Statement

Complex data analysis depends not only on the data but also on the question being asked. This is a fundamental concept in statistical data analysis, yet machine learning algorithms have tended to focus primarily on fitting the data with little regard to

the question being asked. We show that for learning multiple network structures, the questions asked by domain scientists are rarely answered sufficiently by presenting only a single solution based on optimal fit to data. Rather, this dissertation provides an interactive dialog between the end-user and the machine learning algorithm. Machine learning ensures that all solutions fit the data well. Meanwhile, the end-user ensures that solutions reflect answers to questions of comparative analysis among the learned networks. The methods presented incorporate human desires into machine learning algorithms for the discovery of similarities and differences in dependency networks from multiple related data sets.

Appendix A
Multitask Bayesian Discovery Proofs

A.1 Extended proof of Theorem 1

Apply the chain rule and marginalize over graph structure to get Eq 4.6. Then use the modularity properties on each term in the product.

$$
\begin{aligned}
P(f^{(1)}, \mathcal{D}|\prec) &= \sum_{G^{(1)}\subseteq\prec} f(G^{(1)})P(D^{(1)}|G^{(1)}) \sum_{G^{(2)}\subseteq\prec} P(D^{(2)}|G^{(2)})P(G^{(1)}, G^{(2)}|\prec) \\
&= \sum_{\pi_1^{(1)}\subseteq U_1} \cdots \sum_{\pi_n^{(1)}\subseteq U_n} \prod_{i=1}^{n} \left[f_i(\pi_i^{(1)})P(x_i^{(1)}|\pi_i^{(1)}) \right] \times \\
&\quad \times \left[\sum_{\pi_1^{(2)}\subseteq U_1} \cdots \sum_{\pi_n^{(2)}\subseteq U_n} \prod_{i=1}^{n} P(x_i^{(2)}|\pi_i^{(2)})P(\pi_i^{(1)}, \pi_i^{(2)}|U_i) \right]
\end{aligned}
$$

Factor terms in the structure bias as follows.

$$\sum_{\pi_1^{(2)} \subseteq U_1} \cdots \sum_{\pi_n^{(2)} \subseteq U_n} \prod_{i=1}^{n} P(x_i^{(2)}|\pi_i^{(2)}) P(\pi_i^{(1)}, \pi_i^{(2)}|U_i) =$$

$$= \sum_{\pi_1^{(2)} \subseteq U_1} \cdots \sum_{\pi_{n-1}^{(2)} \subseteq U_{n-1}} \prod_{i=1}^{n-1} P(x_i^{(2)}|\pi_i^{(2)}) P(\pi_i^{(1)}, \pi_i^{(2)}|U_i) \times$$

$$\times \left[\sum_{\pi_n^{(2)} \subseteq U_n} P(x_n^{(2)}|\pi_n^{(2)}) P(\pi_n^{(1)}, \pi_n^{(2)}|U_n) \right]$$

$$= \left[\sum_{\pi_1^{(2)} \subseteq U_1} P(x_1^{(2)}|\pi_1^{(2)}) P(\pi_1^{(1)}, \pi_1^{(2)}|U_1) \right] \times$$

$$\times \left[\sum_{\pi_2^{(2)} \subseteq U_2} P(x_2^{(2)}|\pi_2^{(2)}) P(\pi_2^{(1)}, \pi_2^{(2)}|U_2) \right] \times \cdots$$

$$\cdots \times \left[\sum_{\pi_n^{(2)} \subseteq U_n} P(x_n^{(2)}|\pi_n^{(2)}) P(\pi_n^{(1)}, \pi_n^{(2)}|U_n) \right]$$

$$= \prod_{i=1}^{n} \sum_{\pi_i^{(2)} \subseteq U_i} P(x_i^{(2)}|\pi_i^{(2)}) P(\pi_i^{(1)}, \pi_i^{(2)}|U_i)$$

We now have:

$$P(f^{(1)}, \mathcal{D}|\prec) = \sum_{\pi_1^{(1)} \subseteq U_1} \cdots \sum_{\pi_n^{(1)} \subseteq U_n} \prod_{i=1}^{n} f_i(\pi_i^{(1)}) P(x_i^{(1)}|\pi_i^{(1)}) \times$$

$$\times \left[\sum_{\pi_i^{(2)} \subseteq U_i} P(x_i^{(2)}|\pi_i^{(2)}) P(\pi_i^{(1)}, \pi_i^{(2)}|U_i) \right]$$

If we perform a similar factoring to that above, we get

$$P(f^{(1)}, \mathcal{D}|\prec) = \prod_{i=1}^{n} \sum_{\pi_i^{(1)} \subseteq U_i} f_i(\pi_i^{(1)}) P(x_i^{(1)}|\pi_i^{(1)}) \left[\sum_{\pi_i^{(2)} \subseteq U_i} P(x_i^{(2)}|\pi_i^{(2)}) P(\pi_i^{(1)}, \pi_i^{(2)}|U_i) \right]$$

A.2 Normalization constant for structure bias

Calculation of the normalization constant requires summing over all possible combinations of parent sets $(\pi_i^{(1)}, \pi_i^{(2)})$. We accomplish this by fixing parent set $\pi_i^{(1)}$ and counting how many parent sets $\pi_i^{(2)}$ will give $\Delta_{ikj} = 0$ ($\pi_i^{(2)}$ can contain any parents from the set $\{U_i \setminus \pi_i^{(1)}\}$ but *none* from $\pi_i^{(1)}$); then how many $\pi_i^{(2)}$ will give $\Delta_{ikj} = 1$ ($\pi_i^{(2)}$ can contain any parents from the set $\{U_i \setminus \pi_i^{(1)}\}$ and *exactly one* from $\pi_i^{(1)}$); etc, up to the maximum of $\Delta_{ikj} = |\pi_i^{(1)}|$. This sum turns out to be a binomial expansion, and so we can write it in closed form. Next, we perform a similar expansion of the sum over parent sets $\pi_i^{(1)}$ that have size $|\pi_i^{(1)}| = 0$, and $|\pi_i^{(1)}| = 1$, etc up to the maximum $|\pi_i^{(1)}| = |U_i|$. This sum also turns out to be a binomial expansion and therefore can be simplified into a closed form.

$$
\begin{aligned}
Z &= \sum_{\pi_i^{(1)} \subseteq U_i} \sum_{\pi_i^{(2)} \subseteq U_i} (1-\lambda)^{\Delta_{i12}} \\
&= \sum_{\pi_i^{(1)} \subseteq U_i} \left[\sum_{\{\pi_i^{(2)} \subseteq U_i \,\big|\, \Delta_{i12}=0\}} 1 + \sum_{\{\pi_i^{(2)} \subseteq U_i \,\big|\, \Delta_{i12}=1\}} (1-\lambda) + \cdots + \sum_{\{\pi_i^{(2)} \subseteq U_i \,\big|\, \Delta_{i12}=|\pi_i^{(1)}|\}} (1-\lambda)^{|\pi_i^{(1)}|} \right] \\
&= \sum_{\pi_i^{(1)} \subseteq U_i} \left[\binom{|\pi_i^{(1)}|}{0} \frac{2^{|U_i|}}{2^{|\pi_i^{(1)}|}} + \binom{|\pi_i^{(1)}|}{1} \frac{2^{|U_i|}(1-\lambda)}{2^{|\pi_i^{(1)}|}} + \cdots + \binom{|\pi_i^{(1)}|}{|\pi_i^{(1)}|} \frac{2^{|U_i|}(1-\lambda)^{|\pi_i^{(1)}|}}{2^{|\pi_i^{(1)}|}} \right] \\
&= \sum_{\pi_i^{(1)} \subseteq U_i} \sum_{j=0}^{|\pi_i^{(1)}|} \binom{|\pi_i^{(1)}|}{j} 2^{|U_i| - |\pi_i^{(1)}|} (1-\lambda)^j \\
&= \sum_{\pi_i^{(1)} \subseteq U_i} 2^{|U_i| - |\pi_i^{(1)}|} \sum_{j=0}^{|\pi_i^{(1)}|} \binom{|\pi_i^{(1)}|}{j} (1-\lambda)^j \\
&= 2^{|U_i|} \sum_{\pi_i^{(1)} \subseteq U_i} 2^{-|\pi_i^{(1)}|} (1+1-\lambda)^{|\pi_i^{(1)}|} \\
&= 2^{|U_i|} \sum_{\pi_i^{(1)} \subseteq U_i} \left(\frac{2-\lambda}{2} \right)^{|\pi_i^{(1)}|}
\end{aligned}
$$

$$
\begin{aligned}
&= 2^{|U_i|} \sum_{\pi_i^{(1)} \subseteq U_i} \left(1 - \frac{\lambda}{2}\right)^{|\pi_i^{(1)}|} \\
&= 2^{|U_i|} \left[\sum_{\{\pi_i^{(1)} \subseteq U_i \,\big|\, |\pi_i^{(1)}|=0\}} 1 + \sum_{\{\pi_i^{(1)} \subseteq U_i \,\big|\, |\pi_i^{(1)}|=1\}} \left(1 - \frac{\lambda}{2}\right) + \cdots + \sum_{\{\pi_i^{(1)} \subseteq U_i \,\big|\, |\pi_i^{(1)}|=|U_i|\}} \left(1 - \frac{\lambda}{2}\right)^{|U_i|} \right] \\
&= 2^{|U_i|} \left[\binom{|U_i|}{0} + \binom{|U_i|}{1} \left(1 - \frac{\lambda}{2}\right) + \ldots + \binom{|U_i|}{|U_i|} \left(1 - \frac{\lambda}{2}\right)^{|U_i|} \right] \\
&= 2^{|U_i|} \sum_{j=0}^{|U_i|} \binom{|U_i|}{j} \left(1 - \frac{\lambda}{2}\right)^{j} \\
&= 2^{|U_i|} \left(2 - \frac{\lambda}{2}\right)^{|U_i|} \\
&= (4 - \lambda)^{|U_i|}
\end{aligned}
$$

A.3 Integration of Bayesian model average

We integrate over all values of the parameter λ to obtain the Bayesian model average over all possible structure priors, thus eliminating the need to select a point-estimate for λ. We use an uninformative, uniform prior for λ, i.e. $p(\lambda|U_i) = 1$ for $0 \leq \lambda \leq 1$.

$$
\begin{aligned}
P(\pi_i^{(k)}, \pi_i^{(j)}|U_i) &= \int_0^1 P(\pi_i^{(k)}, \pi_i^{(j)}|U_i, \lambda) p(\lambda|U_i) d\lambda \\
&= \int_0^1 \frac{(1-\lambda)^{\Delta_{ikj}}}{(4-\lambda)^{|U_i|}} p(\lambda|U_i) d\lambda \\
&= \int_0^1 \frac{(1-\lambda)^{\Delta_{ikj}}}{(4-\lambda)^{|U_i|}} d\lambda
\end{aligned}
$$

Next, we use an identity given by Euler in 1748 (Bailey, 1935). If β is the beta function and ${}_2\mathrm{F}_1$ is the ordinary hypergeometric function, then

$$
\int_0^1 x^{b-1}(1-x)^{c-b-1}(1-zx)^{-a} dx = \beta(b, c-b) {}_2\mathrm{F}_1(a, b; c; z) \qquad \text{for } \Re(c) > \Re(b) > 0
$$

Let $x = \lambda$, $a = |U_i|$, $b = 1$, $c = \Delta_{ikj} + 2$, and $z = 1/4$. Then the condition,

$\Delta_{ikj} + 2 > 1 > 0$, holds for any $\Delta_{ikj} \geq 0$ which is the valid range for Δ_{ikj} and:

$$\int_0^1 \lambda^0 (1-\lambda)^{\Delta_{ikj}} (1-\lambda/4)^{-|U_i|} d\lambda = \beta(1, \Delta_{ikj}+1)\, {}_2\mathrm{F}_1(|U_i|, 1; \Delta_{ikj}+2; 1/4)$$

$$\int_0^1 \frac{4^{|U_i|}(1-\lambda)^{\Delta_{ikj}}}{(4-\lambda)^{|U_i|}} d\lambda = \left(\frac{0!\Delta_{ikj}!}{(\Delta_{ikj}+1)!}\right) {}_2\mathrm{F}_1(|U_i|, 1; \Delta_{ikj}+2; 1/4)$$

$$\int_0^1 \frac{(1-\lambda)^{\Delta_{ikj}}}{(4-\lambda)^{|U_i|}} d\lambda = \left(\frac{1}{4^{|U_i|}(\Delta_{ikj}+1)}\right) {}_2\mathrm{F}_1(|U_i|, 1; \Delta_{ikj}+2; 1/4)$$

giving the result:

$$P(\pi_i^{(k)}, \pi_i^{(j)} | U_i) = \frac{{}_2\mathrm{F}_1(|U_i|, 1; \Delta_{ikj}+2; 1/4)}{4^{|U_i|}(\Delta_{ikj}+1)}$$

We only need the solution to this formula for a limited number of combinations of integer values of Δ_{ikj} and $|U_i|$, $0 \leq \Delta_{ikj} \leq |U_i| < n$, where n is the number of variables in the network. Therefore we pre-compute a lookup table of the necessary values using the GNU Scientific Library hypergeometric function solver.

References

Abu-Mostafa, Y. 1995. Hints. *Neural Computation* 7:639–671.

Akaike, H. 1973. Information theory and an extension of the maximum likelihood principle. *Second International Symposium on Information Theory* 2:267–281.

Amershi, S.; Fogarty, J.; Kapoor, A.; and Tan, D. 2010. Examining multiple potential models in end-user interactive concept learning. In *Twenty-Eighth International Conference on Human Factors in Computing Systems*, 1357–1360.

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

Bailey, W. N. 1935. *Generalised Hypergeometric Series*. Cambridge, England: University Press.

Bakker, B., and Heskes, T. 2003. Task clustering and gating for Bayesian multitask learning. *Journal of Machine Learning Research* 4:83–99.

Banerjee, O.; El Ghaoui, L.; and d'Aspremont, A. 2008. Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data. *The Journal of Machine Learning Research* 9:485–516.

Baxter, J. 1997. A Bayesian / information theoretic model of learning to learn via multiple task sampling. *Machine Learning* 28(1):7–39.

Baxter, J. 2000. A model of inductive bias learning. *Journal of Artificial Intelligence Research* 12:149–198.

Beinlich, I. A.; Suermondt, H. J.; Chavez, R. M.; and Cooper, G. F. 1989. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Second European Conference on Artificial Intelligence in Medicine*, volume 38, 247–256.

Bekkerman, R.; Raghavan, H.; Allan, J.; and Eguchi, K. 2007. Interactive clustering of text collections according to a user-specified criterion. In *International Joint Conference on Artificial Intelligence*, volume 20, 684–689.

Bergmann, S.; Ihmels, J.; and Barkai, N. 2004. Similarities and differences in genome-wide expression data of six organisms. *PLoS Biology* 2(1):e9.

Bhattacharya, I.; Godbole, S.; Joshi, S.; and Verma, A. 2009. Cross-guided clustering: Transfer of relevant supervision across domains for improved clustering. In *Ninth IEEE International Conference on Data Mining*, 41–50.

Buntine, W. 1991. Theory refinement on Bayesian networks. In *Seventh Conference on Uncertainty in Artificial Intelligence*, 52–60.

Burge, J., and Lane, T. 2005. Learning class-discriminative dynamic Bayesian networks. In *Twenty-Second International Conference on Machine Learning*, 97–104.

Caruana, R. 1997. Multitask learning. *Machine Learning* 28(1):41–75.

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

Chen, T.; Yan, J.; Xue, G.; and Chen, Z. 2010. Transfer learning for behavioral targeting. In *Nineteenth International Conference on World Wide Web*, 1077–1078.

Chiquet, J.; Grandvalet, Y.; and Ambroise, C. 2011. Inferring multiple graphical structures. *Statistics and Computing* 21(4):537–553.

Choi, B.-D.; Jeong, S.-J.; Wang, G.; Park, J.-J.; Lim, D.-S.; Kim, B.-H.; Cho, Y.-I.; Kim, C.-S.; Jeong, M.-J.; et al. 2011. Secretory leukocyte protease inhibitor is

associated with MMP-2 and MMP-9 to promote migration and invasion in SNU638 gastric cancer cells. *International Journal of Molecular Medicine* 28(4):527.

Clark, V. P.; Coffman, B. A.; Mayer, A. R.; Weisend, M. P.; Lane, T. D. R.; Calhoun, V. D.; Raybourn, E. M.; Garcia, C. M.; and Wassermann, E. M. 2012. TDCS guided using fMRI significantly accelerates learning to identify concealed objects. *NeuroImage* 59(1):117–128.

Cohn, D.; Caruana, R.; and McCallum, A. 2003. Semi-supervsied clustering with user feedback. *Constrained Clustering: Advances in Algorithms, Theory, and Applications* 4(1):17–32.

Cohn, D. A.; Ghahramani, Z.; and Jordan, M. I. 1996. Active learning with statistical models. *Journal of Artificial Intelligence Research* 4:129–145.

Cooper, G. F., and Herskovits, E. 1992. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning* 9:309–347.

Cooper, G., and Yoo, C. 1999. Causal discovery from a mixture of experimental and observational data. In *Conference on Uncertainty in Artificial Intelligence*, 116–125.

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

Dasgupta, S., and Ng, V. 2009. Single data, multiple clusterings. In *Neural Information Processing Systems Workshop on Clustering: Science or Art? Towards Principled Approaches*.

Dempster, A. P. 1972. Covariance selection. *Biometrics* 157–175.

desJardins, M.; MacGlashan, J.; and Ferraioli, J. 2007. Interactive visual clustering. In *Twelfth International Conference on Intelligent User Interfaces*, 361–364.

Dondelinger, F.; Lèbre, S.; and Husmeier, D. 2010. Heterogeneous continuous dynamic Bayesian networks with flexible structure and inter-time segment information sharing. In *Twenty-Seventh International Conference on Machine Learning*.

Dredze, M.; Kulesza, A.; and Crammer, K. 2010. Multi-domain learning by confidence-weighted parameter combination. *Machine Learning* 79(1-2):123–149.

Dubey, A.; Bhattacharya, I.; and Godbole, S. 2010. A cluster-level semi-supervision model for interactive clustering. *Machine Learning and Knowledge Discovery in Databases* 409–424.

Dy, J., and Brodley, C. 2000. Visualization and interactive feature selection for unsupervised data. In *Sixth International Conference on Knowledge Discovery and Data Mining*, 360–364.

Eaton, E.; desJardins, M.; and Lane, T. 2008. Modeling transfer relationships between learning tasks for improved inductive transfer. In *European Conference on Machine Learning and Knowledge Discovery in Databases*, 317–332.

Eaton, E.; Holness, G.; and McFarlane, D. 2010. Interactive learning using manifold geometry. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 437–443.

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

Efron, B. 1982. The jackknife, the bootstrap and other resampling plans. In *Society for Industrial and Applied Mathematics*.

Epshteyn, A., and DeJong, G. 2006. Generative prior knowledge for discriminative classification. *Journal of Artificial Intelligence Research* 27(1):25–53.

Fails, J., and Olsen Jr, D. 2003. Interactive machine learning. In *Eighth International Conference on Intelligent User Interfaces*, 39–45.

Fawcett, T. 2004. ROC graphs: Notes and practical considerations for researchers. *Machine Learning* 31:1–38.

Fogarty, J.; Tan, D.; Kapoor, A.; and Winder, S. 2008. CueFlik: Interactive concept learning in image search. In *Computer Human Interface Conference on Human Factors in Computing Systems*, 29–38.

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

Friedman, N., and Yakhini, Z. 1996. On the sample complexity of learning Bayesian networks. In *Twelfth Conference on Uncertainty in Artificial Intelligence*, 274–282.

Friedman, J.; Hastie, T.; and Tibshirani, R. 2008. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics* 9(3):432–441.

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

Friston, K. J.; Holmes, A. P.; Worsley, K. J.; Poline, J.-P.; Frith, C. D.; and Frackowiak, R. S. 1994. Statistical parametric maps in functional imaging: a general linear approach. *Human Brain Mapping* 2(4):189–210.

Friston, K. J.; Harrison, L.; and Penny, W. 2003. Dynamic causal modelling. *NeuroImage* 19(4):1273–1302.

Gold, L.; Ayers, D.; Bertino, J.; Bock, C.; Bock, A.; Brody, E. N.; Carter, J.; Dalby, A. B.; Eaton, B. E.; Fitzwater, T.; et al. 2010. Aptamer-based multiplexed proteomic technology for biomarker discovery. *PloS ONE* 5(12):e15004.

Grzegorczyk, M., and Husmeier, D. 2008. Improving the structure MCMC sampler for Bayesian networks by introducing a new edge reversal move. *Machine Learning* 71(2-3):265–305.

Gu, Q., and Zhou, J. 2009. Learning the shared subspace for multi-task clustering and transductive transfer classification. In *Ninth IEEE International Conference on Data Mining*, ICDM '09, 159–168.

Guo, J.; Levina, E.; Michailidis, G.; and Zhu, J. 2011. Joint estimation of multiple graphical models. *Biometrika* 98(1):1.

Heckerman, D.; Geiger, D.; and Chickering, D. M. 1995. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning* 20(3):197–243.

Hirohashi, S., and Kanai, Y. 2005. Cell adhesion system and human cancer morphogenesis. *Cancer Science* 94(7):575–581.

Honorio, J., and Samaras, D. 2010. Multi-task learning of Gaussian graphical models. In *Twenty-Seventh International Conference on Machine Learning*.

Huang, D. W.; Sherman, B. T.; and Lempicki, R. A. 2008. Systematic and integrative analysis of large gene lists using DAVID bioinformatics resources. *Nature Protocols* 4(1):44–57.

Husmeier, D.; Dondelinger, F.; and Lèbre, S. 2010. Inter-time segment information sharing for non-homogeneous dynamic Bayesian networks. In *Advances in Neural Information Processing Systems 23*, 901–909.

Jbabdi, S.; Woolrich, M.; and Behrens, T. 2009. Multiple-subjects connectivity-based parcellation using hierarchical dirichlet process mixture models. *NeuroImage* 44(2):373 – 384.

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

Kindermann, R., and Snell, J. L. 1980. *Markov Random Fields and Their Applications*, volume 1 of *Contemporary Mathematics*. Providence, Rhode Island: American Mathematical Society.

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

Koivisto, M. 2006. Advances in exact Bayesian structure discovery in Bayesian networks. In *Twenty-Second Conference Annual Conference on Uncertainty in Artificial Intelligence*, 241–248.

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

Lancaster, J. L.; Woldorff, M. G.; Parsons, L. M.; Liotti, M.; Freitas, C. S.; Rainey, L.; Kochunov, P. V.; Nickerson, D.; Mikiten, S. A.; and Fox, P. T. 2000. Automated Talairach atlas labels for functional brain mapping. *Human Brain Mapping* 10(3):120–131.

Larochelle, H.; Erhan, D.; and Bengio, Y. 2008. Zero-data learning of new tasks. In *Twenty-Third National Conference on Artificial Intelligence*, 646–651.

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

Liu, Y.; Niculescu-Mizil, A.; Lozano, A.; and Lu, Y. 2010. Temporal graphical models for cross-species gene regulatory network discovery. In *Life Sciences Society Computational Systems Bioinformatics Conference*, volume 9, 70–81.

Liu, H.; Han, F.; and Zhang, C.-H. 2012. Transelliptical graphical models. In *Advances in Neural Information Processing Systems 25*. 809–817.

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

Lopez-Candales, A.; Bosner, M. S.; Spilburg, C. A.; and Lange, L. G. 1993. Cholesterol transport function of pancreatic cholesterol esterase: directed sterol uptake and esterification in enterocytes. *Biochemistry* 32(45):12085–12089.

Luis, R.; Sucar, L. E.; and Morales, E. F. 2009. Inductive transfer for learning Bayesian networks. *Machine Learning* 79(1-2):227–255.

Madigan, D.; York, J.; and Allard, D. 1995. Bayesian graphical models for discrete data. *International Statistical Review* 215–232.

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

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

Miller, G. 1995. WordNet: a lexical database for English. *Communications of the ACM* 38(11):39–41.

Mishkin, M.; Ungerleider, L. G.; and Macko, K. A. 1983. Object vision and spatial vision: two cortical pathways. *Trends in Neurosciences* 6:414–417.

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

Murphy, K. P. 2001. Active learning of causal Bayes net structure. Technical report.

Neumann, J.; Fox, P. T.; Turner, R.; and Lohmann, G. 2010. Learning partially directed functional networks from meta-analysis imaging data. *NeuroImage* 49(2):1372–1384.

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

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

Oyen, D., and Lane, T. 2012. Leveraging domain knowledge in multitask Bayesian network structure learning. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Oyen, D.; Niculescu-Mizil, A.; Ostroff, R.; Stewart, A.; and Clark, V. P. 2013. Controlling the precision-recall tradeoff in differential dependency network analysis. ArXiv.

Palatucci, M.; Pomerleau, D.; Hinton, G.; and Mitchell, T. 2009. Zero-shot learning with semantic output codes. In *Neural Information Processing Systems (NIPS)*, 1410–1418.

Pan, S. J., and Yang, Q. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 22(10):1345–1359.

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

Pearl, J., and Bareinboim, E. 2011. Transportability of causal and statistical relations: A formal approach. In *Twenty-Fifth National Conference on Artificial Intelligence*, 247—254.

Roy, S.; Werner-Washburne, M.; and Lane, T. 2011. A multiple network learning approach to capture system-wide condition-specific responses. *Bioinformatics* 27(13):1832–1838.

Sarty, G. E. 2007. *Computing Brain Activity Maps from fMRI Time-Series Images*. Cambridge University Press.

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

Schwarz, G. 1978. Estimating the dimension of a model. *The Annals of Statistics* 6:461–464.

Settles, B. 2009. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison.

Smith, S. M.; Miller, K. L.; Salimi-Khorshidi, G.; Webster, M.; Beckmann, C. F.; Nichols, T. E.; Ramsey, J. D.; and Woolrich, M. W. 2011. Network modelling methods for fMRI. *NeuroImage* 54(2):875–891.

Steele, E., and Tucker, A. 2009. Selecting and weighting data for building consensus gene regulatory networks. In *Advances in Intelligent Data Analysis VIII*. 190–201.

Talbot, J.; Lee, B.; Kapoor, A.; and Tan, D. S. 2009. EnsembleMatrix: Interactive visualization to support machine learning with multiple classifiers. In *Twenty-Seventh International Conference on Human Factors in Computing Systems*, 1283–1292.

Thrun, S., and O'Sullivan, J. 1996. Discovering structure in multiple learning tasks: The TC algorithm. In *International Conference on Machine Learning*, 489–497.

Thrun, S. 1996. Is learning the n-th thing any easier than learning the first? *Advances in Neural Information Processing Systems* 640–646.

Tibshirani, R. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B (Methodological)* 267–288.

Tong, S., and Koller, D. 2001. Active learning for structure in Bayesian networks. In *International Joint Conference on Artificial Intelligence*, volume 17, 863–869.

Tzourio-Mazoyer, N.; Landeau, B.; Papathanassiou, D.; Crivello, F.; Etard, O.; Delcroix, N.; Mazoyer, B.; and Joliot, M. 2002. Automated anatomical labeling of activations in SPM using a macroscopic anatomical parcellation of the MNI MRI single-subject brain. *NeuroImage* 15(1):273–289.

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

Varoquaux, G.; Gramfort, A.; Poline, J. B.; and Thirion, B. 2010. Brain covariance selection: Better individual functional connectivity models using population prior. In *Advances in Neural Information Processing Systems*.

Walker, G.; MacLeod, K.; Williams, A. R.; Cameron, D. A.; Smyth, J. F.; and Langdon, S. P. 2007. Insulin-like growth factor binding proteins IGFBP3, IGFBP4, and

IGFBP5 predict endocrine responsiveness in patients with ovarian cancer. *Clinical Cancer Research* 13(5):1438–1444.

Wang, X.; Wang, E.; Kavanagh, J. J.; and Freedman, R. S. 2005. Ovarian cancer, the coagulation pathway, and inflammation. *Journal of Translational Medicine* 3(1):25.

Ware, M.; Frank, E.; Holmes, G.; Hall, M.; and Witten, I. H. 2001. Interactive machine learning: letting users build classifiers. *International Journal of Human-Computer Studies* 55(3):281–292.

Wasserman, L., and Roeder, K. 2009. High dimensional variable selection. *Annals of Statistics* 37(5A):2178.

Werhli, A., and Husmeier, D. 2007. Reconstructing gene regulatory networks with Bayesian networks by combining expression data with multiple sources of prior knowledge. *Statistical Applications in Genetics and Molecular Biology* 6(1).

Widmer, C.; Leiva, J.; Altun, Y.; and Rätsch, G. 2010. Leveraging sequence classification by taxonomy-based multitask learning. *Research in Computational Molecular Biology* 6044:522–534.

Yu, S.; Tresp, V.; and Yu, K. 2007. Robust multi-task learning with t-processes. In *Twenty-Fourth International Conference on Machine Learning*, 1103–1110.

Yuan, M., and Lin, Y. 2007. Model selection and estimation in the Gaussian graphical model. *Biometrika* 94(1):19–35.

Zhang, J., and Zhang, C. 2010. Multitask Bregman clustering. In *Twenty-Fourth National Conference on Artificial Intelligence*.

Zhang, B.; Li, H.; Riggins, R.; Zhan, M.; Xuan, J.; Zhang, Z.; Hoffman, E.; Clarke, R.; and Wang, Y. 2009. Differential dependency network analysis to identify condition-specific topological changes in biological networks. *Bioinformatics* 25(4):526–532.

Zheng, V. W.; Pan, S. J.; Yang, Q.; and Pan, J. J. 2008. Transferring multi-device localization models using latent multi-task learning. In *Twenty-Third National Conference on Artificial Intelligence*.