Inductive Transfer for Bayesian Network Structure Learning

Alexandru Niculescu-Mizil
Department of Computer Science
Cornell University
Ithaca, NY 14853
alexn@cs.cornell.edu

Rich Caruana
Department of Computer Science
Cornell University
Ithaca, NY 14853
caruana@cs.cornell.edu

Abstract

We consider the problem of learning Bayes Net structures for related tasks. We present an algorithm for learning Bayes Net structures that takes advantage of the similarity between tasks by biasing learning toward similar structures for each task. Heuristic search is used to find a high scoring *set* of structures (one for each task), where the *score* for a set of structures is computed in a principled way. Experiments on problems generated from the ALARM and INSURANCE networks show that learning the structures for related tasks using the proposed method yields better results than learning the structures independently.

1 Introduction

Bayes Nets [1] provide a compact, intuitive description of the dependency structure of a domain by using a directed acyclic graph to encode probabilistic dependencies between variables. This intuitive encoding of the dependency structure makes Bayes Nets appealing in expert systems where expert knowledge can be encoded through hand-built dependency graphs. Acquiring expertise from humans, however, is difficult and expensive, so significant research has focused on learning Bayes Nets from data. The learned dependency graph provides useful information about a problem and is often used as a data analysis tool. For example Friedman et al. used Bayes Nets learned from gene expression level data to discover regulatory interactions between genes for a species of yeast [2].

Until now, Bayes Net structure learning research has focused on learning the dependency graph for one problem in isolation. In many situations, however, data is available for multiple related problems. In these cases, inductive transfer [3, 4, 5] suggests that it may be possible to learn more accurate dependency graphs by *transferring* information between problems. For example, suppose that we want to learn the gene regulatory structure for a number of yeast species. Since the regulatory structures are very similar, learning that there is an interaction between two genes in one species of yeast should provide evidence for the existence of the same interaction in the other species.

In this paper we present an algorithm for learning the Bayesian Network structure for multiple problems simultaneously. We start with an overview of Bayes Net structure learning for a single problem, then describe a new multitask structure learning algorithm in Section 3. The empirical evaluation in Section 4 shows that more accurate dependency graphs can be learned via inductive transfer compared to learning the Bayes Net structure for each problem in isolation.

2 Learning Bayes Nets from Data

A Bayesian Network $\mathcal{B} = \{G, \theta\}$ that encodes the joint probability distribution of a set of n random variables $X = \{X_1, X_2, ..., X_n\}$ is specified by a directed acyclic graph (DAG) G and a set of conditional probability functions parametrized by θ [1]. The Bayes Net *structure*, G, encodes the probabilistic dependencies in the data: the presence of an edge between two variables means that there exists a direct dependency between them. An appealing feature of Bayes Nets is that the dependency graph G is easy to interpret and can be used to aid understanding the problem domain.

Given a dataset $D = \{x^1, ..., x^m\}$ where each x^i is a complete assignment of variables $X_1, ..., X_n$, it is possible to learn both the structure G and the parameters θ [6, 7]. Following the Bayesian paradigm, the posterior probability of the structure given the data is estimated via Bayes rule:

$$P(G|D) \propto P(G)P(D|G) \tag{1}$$

The prior $P(G)$ indicates the belief before seeing any data that the structure G is correct. If there is no reason to prefer one structure over another, one should assign the same probability to all structures. This uninformative (uniform) prior is rarely accurate, but often is used for convenience. If there exists a known ordering on the nodes in G such that all the parents of a node precede it in the ordering, a prior can be assessed by specifying the probability that each of the $n(n-1)/2$ possible arcs is present in the correct structure [8]. Alternately, when there is access to a structure believed to be close to the correct one (e.g. from an expert), $P(G)$ can be specified by penalizing each difference between G and the given structure by a constant factor [9].

The marginal likelihood, $P(D|G)$, is computed by integrating over all possible parameter values:

$$P(D|G) = \int P(D|G,\theta)P(\theta|G)d\theta \tag{2}$$

When the local conditional probability distributions are from the exponential family, the parameters θ_i are mutually independent, we have conjugate priors for these parameters, and the data is complete, $P(D|G)$ can be computed in closed form [7].

Treating $P(G|D)$ as a score, one can search for a high scoring network using heuristic search [7]. Greedy search, for example, starts from an initial structure, evaluates the score of all the *neighbors* of that structure and moves to the neighbor with the highest score. The search terminates when the current structure is better than all its neighbors. Because it is possible to get stuck in a local minima, this procedure usually is repeated a number of times starting from different initial structures. A common definition of the *neighbors* of a structure G is the set of all the DAGs that can be obtained by removing or reversing an existing arc in G, or by adding an arc that is not present in G.

3 Learning from Related Tasks

In the previous section we reviewed how to learn a Bayes Net for a single task. What if instead of a single task we have a number of related tasks (e.g., gene expression data for a number of related species) and we want to learn a Bayes Net structure for each of them?

Given k data-sets, $D_1, ..., D_k$, we want to simultaneously learn the structures of the Bayes Nets $\mathcal{B}_1 = \{G_1, \theta_1\}$, ..., $\mathcal{B}_k = \{G_k, \theta_k\}$. We will use the term *configuration* to refer to a set of structures $(G_1, ..., G_k)$. From Bayes rule, the posterior probability of a configuration given the data is:

$$\begin{aligned} &P(G_1, ..., G_k|D_1, ..., D_k) \propto \\ &\quad \propto \quad P(G_1, ..., G_k)P(D_1, ..., D_k|G_1, ..., G_k) \end{aligned} \tag{3}$$

The marginal likelihood $P(D_1, ..., D_k|G_1, ..., G_k)$ is computed by integrating over all parameter values for all the k networks:

$$\begin{aligned} &P(D_1, ..., D_k|G_1, ..., G_k) = \\ &= \int P(D_1, ..., D_k|G_1, ..., G_k, \theta_1, ..., \theta_k) \times \\ &\quad P(\theta_1, ..., \theta_k|G_1, ..., G_k)d\theta_1...d\theta_k \\ &= \int P(\theta_1, ..., \theta_k|G_1, ..., G_k)\prod_{p=1}^{k} P(D_p|G_p, \theta_p)d\theta_1...d\theta_k \end{aligned} \tag{4}$$

If we make the parameters of different networks independent *a priori* (i.e. $P(\theta_1, ..., \theta_k|G_1, ..., G_k) = P(\theta_1|G_1)...P(\theta_k|G_k)$), the marginal likelihood is just the product of the marginal likelihoods of each data set given its network structure. In this case the posterior can be written as:

$$P(G_1, .., G_k|D_1, .., D_k) \propto P(G_1, .., G_k)\prod_{p=1}^{k} P(D_p|G_p) \tag{5}$$

Making the parameters independent *a priori* is unfortunate and contradicts the intuition that related tasks should have related parameters, but it is needed in order to make structure learning efficient (see Section 3.3). It is important to note that this is not a restriction on the model. Unlike the Naive Bayes model for example, where the attribute independence assumption actually restricts the class of models that can be learned, here the learned parameters will be correlated if such correlation is present in the data. The downside of making the parameters independent *a priori* is that it prevents multi-task structure learning from taking advantage of the similarities between parameters of different tasks during structure learning phase. After the structures have been learned, however, such similarities could be leveraged to learn more accurate parameters. Finding ways to allow for some *a priori* parameter dependence while still maintaining computational efficiency is an interesting direction for future work.

3.1 The Prior

The prior knowledge of how related the different tasks are and how similar their structures should be is encoded in the prior $P(G_1, ..., G_k)$. If there is no reason to believe that the structures for each task should be related, then $G_1, ..., G_k$ should be made independent *a priori* (i.e. $P(G_1, ..., G_k) = P(G_1) \cdot ... \cdot P(G_k)$). In this case the structure-learning can be done independently for each task using the corresponding data set.

At the other extreme, if the structures for all the different tasks should be identical, the prior $P(G_1, ..., G_k)$ should put zero probability on any configuration that contains non-identical structures. In this case one can efficiently learn the same structure for all tasks by creating a new data set with attributes $X_1, ..., X_n, TSK$, where TSK encodes the task the case is coming from.[1] Then learn the structure for this new data set under the restriction that TSK is always

[1]This is different from pooling the data, which would mean that not only the structures, but also the parameters for all tasks will be identical.

the parent of all the other nodes. The common structure for all the tasks is exactly the learned structure, with the node TSK and all the arcs connected to it removed. This approach, however, does not easily generalize to the case where tasks have only partial overlap in their attributes. The algorithm proposed below avoids this problem, while computing the same solution when structures are forced to be identical.

Between these two extremes, the prior should encourage finding similar network structures for the tasks. The prior can be seen as penalizing structures that deviate from each other, so that deviation will occur only if it is supported by enough evidence in the data.

One way to generate such a prior for two structures is to penalize each arc (X_i, X_j) that is present in one structure but not in the other by a constant $\delta \in [0, 1]$:

$$P(G_1, G_2) = Z_\delta \cdot (P(G_1)P(G_2))^{\frac{1}{1+\delta}} \prod_{\substack{(X_i,X_j)\in \\ G_1 \Delta G_2}} (1-\delta) \quad (6)$$

where Z_δ is a normalization factor that is absorbed in the proportionality constant of equation 5, and $G_1 \Delta G_2$ represents the symmetric difference between the edge sets of the two DAGs. If $\delta = 0$ then $P(G_1, G_2) = P(G_1)P(G_2)$, so the structures are learned independently. If $\delta = 1$ then $P(G_1, G_2) = \sqrt{P(G)P(G)} = P(G)$ for $G_1 = G_2 = G$ and $P(G_1, G_2) = 0$ for $G_1 \neq G_2$, leading to learning identical structures for all tasks. For δ between 0 and 1, the higher the penalty, the higher the probability of more similar structures. The advantage of this prior is that $P(G_1)$ and $P(G_2)$ can be any structure priors that are appropriate for the task at hand. If a variable, X_i, is present in one structure but not in the other, then any arc that has X_i as one of its extremities should not incur any penalty. A generalization to more than two tasks is:

$$P(G_1, ..., G_k) = Z_{\delta,k} \prod_{1\le s\le k} P(G_s)^{\frac{1}{1+(k-1)\delta}} \times$$
$$\times \prod_{1\le s<t\le k} \left(\prod_{\substack{(X_i,X_j)\in \\ G_s \Delta G_t}} (1-\delta) \right)^{\frac{1}{k-1}} \quad (7)$$

The exponent $1/(1+(k-1)\delta)$ is used to transition smoothly between the case where structures should be independent (i.e. $P(G_1, ..., G_k) = (P(G_1)...P(G_k))^1$ for $\delta = 0$) and the case where structures should be identical (i.e. $P(G, .., G) = (P(G)...P(G))^{1/k}$ for $\delta = 1$). The exponent $1/(k-1)$ is used because each edge for each individual structure is involved in $k-1$ terms (one for each other structure).

This prior can be easily generalized by using different penalties for different edges (e.g. if certain edges should not change between tasks then the penalty on those edges should be 1), and/or different penalties between different tasks. There are, of course, other priors that encourage finding similar networks for each task in different ways. Comparisons to other such priors is a direction for future work.

3.2 Greedy Structure Learning

Treating $P(G_1, ..., G_k|D_1, ..., D_k)$ as a score, we can search for a high scoring configuration using an heuristic search algorithm. If we choose to use greedy search for example, we start from an initial configuration, compute the scores of the neighboring configurations, then move to the configuration that has the highest score. The search ends when no neighboring configuration has a higher score than the current one.

One question remains: what do we mean by the neighborhood of a configuration? An intuitive definition of a neighbor is the configuration obtained by modifying a single arc in a single DAG in the configuration, such that the resulting graph is still a DAG. With this definition, the size of the neighborhood of a configuration is $O(k * n^2)$ for k tasks and n variables. Unfortunately, this definition creates a lot of local minima in the search space. Consider for example the case where there is a strong belief that the structures should be similar (i.e. the penalty parameter of the prior, δ, is near one resulting in a prior probability near zero when the structures in the configuration differ). In this case it would be difficult to take any steps in the greedy search since modifying a single edge for a single DAG would make it different from the other DAGs, resulting in a very low posterior probability (score).

To correct this problem, we define the neighborhood of a configuration to be the set of all configurations obtained by selecting two nodes, and for each structure in the configuration, adding, removing, reversing, or leaving unchanged the arc between the two selected nodes, under the restriction that the resulting structure remains a DAG. It is easy to see that there is a path between any two configurations, so the search space is connected. Given this definition, the size of a neighborhood is $O(n^2 3^k)$, which is exponential in the number of tasks, but only quadratic in the number of nodes.[2] In the case where all the learned structures are required to be identical (infinite penalty for diverging structures) multi-task learning, with this definition of neighborhood, will find the same structures as the specialized algorithm described in Section 3.1.

[2]The restriction that changes, if any, have to occur between the same nodes in all the structures could be dropped, but this would lead to a neighborhood that is exponential in both n and k. Considering the assumption that the structures should be similar, such a restriction is not inappropriate.

3.3 Searching for the Best Configuration

At each iteration, the greedy procedure described in the previous section must find the best scoring configuration from a set $\mathcal{N}$ of configurations. In the naive approach the score of every configuration in $\mathcal{N}$ is computed and the configuration with the highest score is selected. Since the size of $\mathcal{N}$ can get large for large n or k, this naive approach can be expensive. Under the following assumptions, however, it is possible to evaluate only a fraction of the scores in order to find the highest one: 1) the parameters for each task are mutually independent *a priori* so the score of a configuration has the form in equation 5; 2) the prior over configurations has the form in equation 7.

Let a partial configuration of order l, $\mathcal{C}_l = (G_1, .., G_l)$, be a configuration where only the structures for the first l tasks are specified and the rest of $k - l$ structures are not specified. We say that a configuration $\mathcal{C}$ matches a partial configuration $\mathcal{C}_l$ if the structures for the first l tasks in $\mathcal{C}$ are the same as the structures in $\mathcal{C}_l$. Let the score of a partial configuration be:

$$S_{\mathcal{N}}(\mathcal{C}_l) = Z_{\delta,k} \prod_{1\leq s<t\leq l} \left(\prod_{\substack{(X_i,X_j)\in \\ G_s \Delta G_t}} (1-\delta) \right)^{\frac{1}{k-1}} \times \quad (8)$$
$$\times \prod_{1\leq p\leq l} P(G_p)^{\frac{1}{1+(k-1)\delta}} P(D_p|G_p) \prod_{l+1\leq p\leq k} Best_q$$

where $Best_q = \max\{P(G_q)^{\frac{1}{1+(k-1)\delta}} P(D_q|G_q)\}$. It is easy to see that the score of a partial configuration is an upper bound on the score of any configuration that matches it.[3] When searching for the best scoring structure in $\mathcal{N}$, we do not explore any configuration matching a partial configuration with lower score than the current best configuration. This significantly reduces the number of partial configurations (and consequently complete configurations) that need to be explored. In our experiments, using this pruning, we only need to evaluate 2-4% of the partial configurations, resulting in computational savings of 96-98%.

Another source of computational savings is the precomputation of the individual marginal likelihoods. With the definition of a neighborhood we are using, a neighboring configuration will have, in each of the k components, one of the $2n^2$ or fewer individual DAGs that differ by exactly one edge from the current DAG in the respective component. Each of these $2n^2$ (or fewer) DAGs are present in about 3^{k-1} neighboring configurations. Since a configuration score has the form in equation 5, the marginal likelihoods for the individual DAGs, $P(D_i|G_i)$, can be reused, thus reducing by a factor of about 3^{k-1} the expense of computing the marginal likelihoods of the neighboring configurations.In our experiments, precomputing all the marginal likelihoods, which is required whether we do multi-task or single-task learning, took 3 - 7 times longer than finding the best neighboring configuration.[4] It is also worth mentioning that both the prior and the likelihood are decomposable, so evaluating the score of the neighboring configurations requires only local computations.

[3]It is possible to get a tighter upper bound, but we will use this one for simplicity.

4 Experimental Results

We evaluate the performance of multi-task structure learning using the ALARM [10] and INSURANCE [11] Bayes Nets. For each problem, we create five related tasks by starting with the original network and deleting arcs with probability P_{del}. The structures of the five tasks can be made more or less similar by varying P_{del} (For $P_{del} = 0$ all structures are identical). We create four sets of related tasks. For two of them, ALARM and INSURANCE, we start with the parameters of the original networks for all five tasks. When an arc is deleted, the parameters of the network are recomputed by integrating over the deleted parent, so that the dependency between the child and the remaining parents is unchanged. This yields five related tasks with correlated parameters. For the other two sets, ALARM-IND and INSURANCE-IND, we start with random parameters for each task instead of the original ones, and apply the same procedure as above when an arc is deleted. This way we create five tasks with similar structures but independent parameters.

We also experiment with a qualitatively different way of generating related tasks, ALARM-COMP. We split the ALARM network in 4 components: nodes 1-7 in the first component, nodes 9-14, 21 and 34 in the second, nodes 8,27-31, 36 and 37 in the third and the rest in the fourth component. For each of the five tasks, we randomly change the structure and parameters of one or two of the components, while keeping the rest of the Bayes net (including parameters) unchanged. This way parts of the structures are shared between tasks while other parts are completely unrelated (see Figure 7).

All reported results are averages over ten trials. For each trial, in addition to varying the train and test sets, we also construct different target Bayes Nets. This way we show the expected performance over the entire class of problems that can be constructed using the methods above.

The goal is to recover as closely as possible the structure of all five Bayes Nets. We measure performance both in terms of average edit distance[5] between the true structures and learned structures, and in terms of average empirical KL-divergence (computed on a large test set) between the distributions encoded by the true networks and the learned

[4]The implementation we used for these results did not use AD-trees. Using AD-trees would significantly reduce the time to pre-compute the marginal likelihoods.

[5]Edit distance measures how many edits (arc additions, deletions or reversals) are needed to get from one structure to the other.

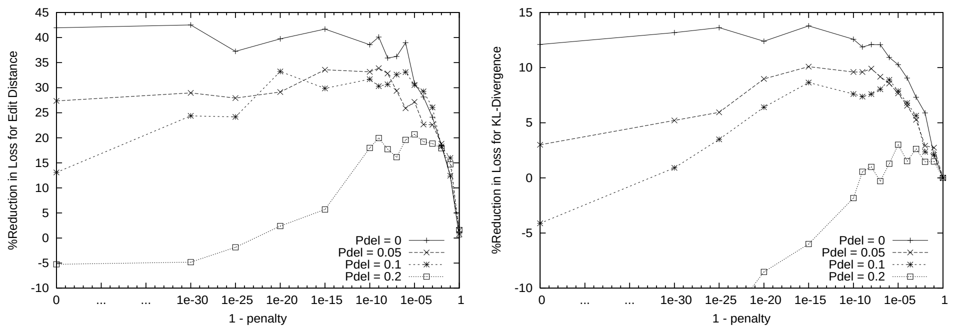


Figure 1: Reduction in edit distance (left) and KL-Divergence (right) for ALARM-IND

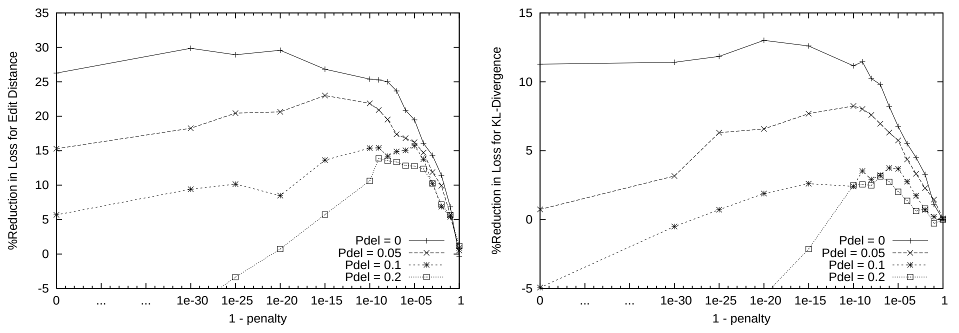


Figure 2: Reduction in edit distance (left) and KL-Divergence (right) for INSURANCE-IND

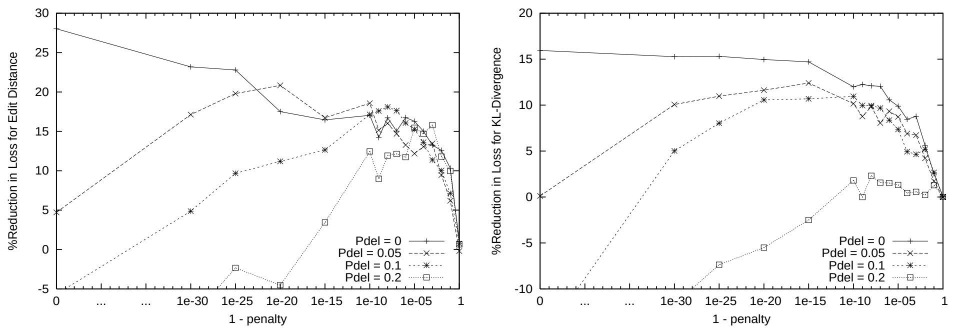


Figure 3: Reduction in edit distance (left) and KL-Divergence (right) for INSURANCE

ones. We compare multi-task structure learning (MTL) to single-task structure learning (STL) and learning identical structures for all tasks via the algorithm presented in Section 3.1 (IDENTICAL). Single-task structure learning uses greedy hill-climbing with tabu lists to learn the structure of each task independently of the others.[6] The greedy search is initialized with the empty structure. Multi-task structure learning uses the greedy algorithm described in Section 3.2 with the prior over configurations from equation 7, and uninformative prior on the individual structures. The greedy MTL search is initialized with the solution found by single-task learning.[7] For all methods, after learning the structures, the Bayes net parameters are learned using Bayesian updating (see e.g. [6])

Figures 1 - 3 show the average percent reduction in loss, in terms of edit distance and KL-divergence, achieved by multi-task learning (MTL) over single-task learning (STL) for a training set of 1000 points on the ALARM-IND, INSURANCE-IND and INSURANCE problems. (The graphs for the ALARM problem look similar, and are not included due to space constraints.) On the x-axis we vary the penalty parameter of the prior on a log-scale.[8] The

[6]Learning identical structures and single-task structure learning can be viewed as learning an augmented naive Bayesian network and a Bayesian multi-net [12] respectively, where the "class" of each example is the task it belongs to . Unlike in the usual setting, however, here we are not interested in predicting to which task an example belongs to. We are only interested in recovering accurate network structures for each task.

[7]Initializing MTL search with the STL solution does not provide an advantage to MTL, but makes the search more efficient.

[8]The log-scale is needed because we are working in the prob-

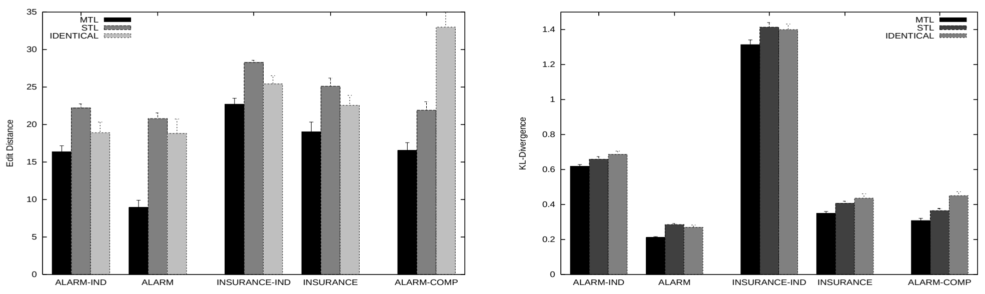


Figure 4: Edit distance (left) and KL-Div (right) for STL, learning identical structures and MTL

higher the penalty, the more similar the learned structures will be, with all the structures being identical for a penalty of one (left end of graphs). Each line in the figure corresponds to a particular value of P_{del}.

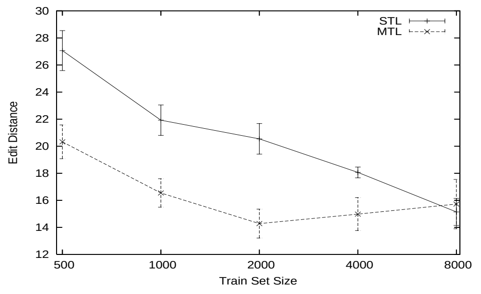
The trends in the graphs are exactly as expected. For all values of P_{del}, as the penalty increases, the performance increases because the learning algorithm takes into account information from the other tasks when deciding whether to add a new arc or not. If the penalty is too high, however, the algorithm loses the ability to find true differences between tasks and the performance drops. As the tasks become more similar (lower values of P_{del}), the best performance is obtained at higher penalties. Also as the tasks become more similar, more information can be extracted from the related tasks, so usually multi-task learning provides more benefit. As expected, multi-task structure learning provides a larger improvement in edit distance than in KL-divergence. This happens because multi-task structure learning helps to correctly identify the arcs that encode weaker dependencies (or independences) which have a smaller effect on KL-divergence. The edges that encode strong dependencies that have the biggest effect on KL-divergence can be easily learned without help from the other tasks. Multi-task learning provides similar benefits whether the tasks have highly correlated parameters (ALARM and INSURANCE problems) or independent parameters (ALARM-IND and INSURANCE-IND problems). This shows that making the parameters independent *a priori* (see Section 3) does not hurt the performance of multi-task learning. However, if we were able to take advantage of the similarity between the parameters of the different tasks, we could presumably improve performance even further. It is an open question how to relax the *a priori* parameter independence requirement while still maintaining computational efficiency.

When applying multi-task structure learning to a real problem, a good value for the penalty parameter of the prior is not usually known *a priori*. Here we take a simple approach to finding it: we learn Bayes Nets for a number of different values of the penalty parameter and pick the networks corresponding to the penalty parameter that gives the highest average log likelihood on a small independent validation set. We then relearn only the parameters of these Bayes Nets using both the training and the validation set. More involved approaches for setting the penalty parameter are possible and might yield even better performance. Figure 4 shows the edit distance and KL-Divergence performance for single task learning (STL), learning identical networks via the algorithm presented in Section 3.1, and multi-task learning (MTL) for the five problems (P_{del} is set to 0.05 for ALARM-IND, ALARM, INSURANCE-IND and INSURANCE). The training set has 1000 cases, with a validation set of 50 cases for selecting the penalty parameter for MTL. Single-task learning and identical structure learning use both the training and the validation sets to learn both the structure and the parameters of the Bayes Nets. The figure shows that multi-task learning yields a 6%-21% reduction in KL-divergence and a 11% - 52% reduction in edit distance when compared to learning identical structures for all tasks. All differences except for edit distance on ALARM-IND and INSURANCE problems are .95 significant according to paired T-tests. When compared to single-task learning multi-task learning reduces the KL-divergence 6% - 26% and the number of incorrect arcs in the learned structures by 20% - 57%. All differences are .95 significant. Since the five tasks for the ALARM, INSURANCE, and ALARM-COMP problems share a large number of their parameters, simply pooling the data might work well. However, this is not the case. Except for the ALARM problem, where it achieves about the same edit distance as learning identical structures (18.6), pooling the data has much worse performance both in terms of edit distance and in terms of KL-divergence.

Figure 5 shows the performance of single and multi-task learning as the train set size varies from 500 to 8000 cases (MTL uses 5% of the training points as a validation set to select the penalty parameter). On average MTL needs a quarter as much data as STL to achieve the same edit distance. As discussed before, the improvement in KL-divergence is smaller and represents about a 20% savings in sample size. As expected, the benefit from MTL decreases with increasing sample size.

Figure 6 shows the edit distance and KL-divergence for one task of the ALARM problem as the number of related tasks varies from zero (single task learning) to ten. With more re-

ability space so $1 - \delta$ needs to change by orders of magnitude for the effects to be noticeable.



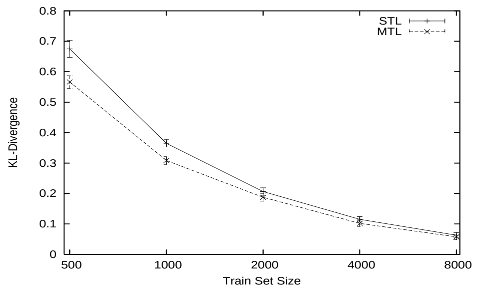
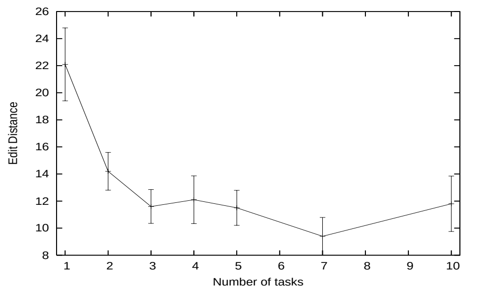


Figure 5: Edit distance (left) and KL-Divergence (right) vs. train set size for ALARM-COMP.



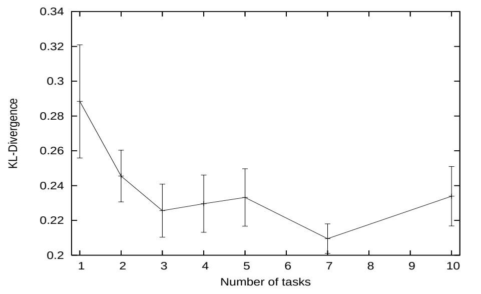


Figure 6: Edit distance (left) and KL-Divergence (right) vs. number of tasks related tasks for ALARM.

lated tasks, there is more information that multi-task learning can exploit and the performance increases. Once there are enough tasks (three for this problem), there is little benefit to having more, and the performance plateaus.

For a qualitative perspective, Figure 7 shows the true structures and the structures learned by MTL and STL for the five tasks (one per column) on one trial of the ALARM-COMP problem. The figure clearly shows that multi-task learning finds more accurate structures by taking advantage of the similarity between the five tasks, while still preserving some of the true differences between them.

5 Discussion and Related Work

We believe this is the first multi-task Bayesian Network structure learning algorithm. The work most closely related to ours is Baxter's [4] which provides a Bayesian interpretation of multi-task learning. Other work in multi-task learning includes [3, 5, 13, 14]. For an overview of learning Bayes Nets from data see [6, 7, 15, 16].

In this paper, we use heuristic search in the space of network structures. Some straightforward extensions are greedy search in the space of equivalence classes [17], obtaining confidence measures on the structural features of the configurations via bootstrap analysis [18], and structure learning from incomplete datasets via the structural EM algorithm [19]. Other extensions such as obtaining a sample from the posterior distribution via MCMC methods might be more problematic. Because of the larger search space, MCMC methods might not converge in reasonable time. Evaluating different MCMC schemes is a direction for future work. Another open question is whether we can relax the requirement that the parameters of the Bayes Nets for the different related tasks are independent *a priori*. Relaxing this requirement might further improve the performance of multi-task learning since the task would be able to share not only the structures but also the parameters, thus having more opportunities for inductive transfer.

6 Conclusions

We present a method for learning the Bayes Net structures of related tasks. The approach assumes that the structures of related tasks are similar: the presence or absence of arcs in some of the structures provides evidence for the presence or absence of those same arcs in the other structures. When this assumption is true, learning the structures together yields an advantage over learning a structure for each task individually. Similarity between learned structures is controlled via a prior. Experiments with perturbed ALARM and INSURANCE networks show that learning related structures simultaneously yields a reduction in KL-Divergence of 6% - 26% and reduces the number of incorrect arcs in the learned structures by 20% - 57% when compared to structures learned separately.

Acknowledgments

Thanks to Joe Halpern, John Platt and the anonymous reviewers for helpful comments. Thanks to Kevin Murphy and Philippe Leray for making the BNT and respectively BNT Structure Learning packages available. This work was supported by NSF Awards 0412930 and 0347318.

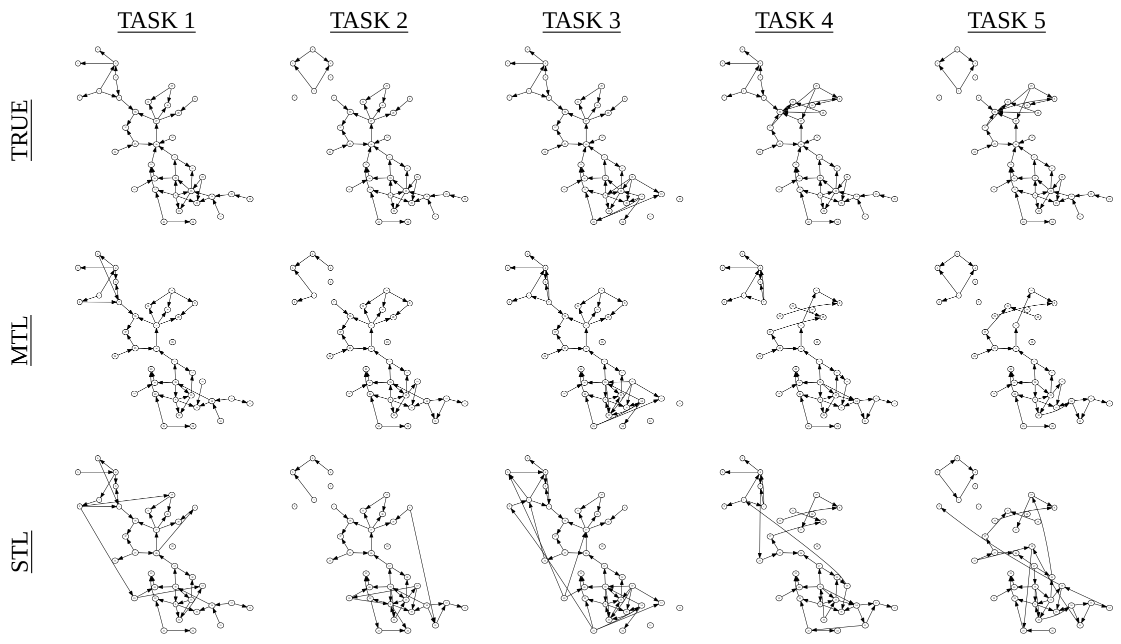


Figure 7: The true structures and structures learned by MTL and STL for ALARM-COMP

References

[1] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[2] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using bayesian networks to analyze expression data. *J. Comput. Biol.*, 7(3-4):601–620, 2000.

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

[4] J. Baxter. A bayesian/information theoretic model of learning to learn via multiple task sampling. *Mach. Learn.*, 28(1):7–39, 1997.

[5] S. Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, 1996.

[6] G. Cooper and E. Hersovits. A bayesian method for the induction of probabilistic networks from data. *Maching Learning*, 9:309–347, 1992.

[7] D. Heckerman. A tutorial on learning with bayesian networks. *Learning in graphical models*, pages 301–354, 1999.

[8] W. Buntine. Theory refinement on bayesian networks. In *Proc. 7th Conference on Uncertainty in Artificial Intelligence (UAI '91)*. 1991.

[9] D. Heckerman, A. Mamdani, and M.P. Wellman. Real-world applications of Bayesian networks. *Communications of the ACM*, 38(3):24–30, 1995.

[10] I.A. Beinlich, H.J. Suermondt, R.M. Chavez, and G.F. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the Second European Conference on Artificial Intelligence in Medicine*, 1989.

[11] J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29, 1997.

[12] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian Network Classifiers. *Machine Learning*, 29(2):131–163, 1997.

[13] T. Jebara. Multi-task feature and kernel selection for svms. In *ICML '04: Twenty-first international conference on Machine learning*, 2004.

[14] N. D. Lawrence and J. C. Platt. Learning to learn with the informative vector machine. In *ICML '04: Twenty-first international conference on Machine learning*, 2004.

[15] W. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Trans. On Knowledge and data Engineering*, 8:195–210, 1996.

[16] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. The MIT Press, Cambridge, MA, second edition, 2000.

[17] D. Chickering. Learning equivalence classes of Bayesian network structures. In *Proc. 12th Conference on Uncertainty in Artificial Intelligence (UAI'96)*, 1996.

[18] N. Friedman, M. Goldszmidt, and A. J. Wyner. Data analysis with bayesian networks: A bootstrap approach. In *Proc. 15th Conference on Uncertainty in Artificial Intelligence*, 1999.

[19] N. Friedman. The Bayesian structural EM algorithm. In *Proc. 14th Conference on Uncertainty in Artificial Intelligence (UAI '98)*, 1998.