Learning the Structure of Related Tasks

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 a formalism for learning related Bayes Net structures that takes advantage of the similarity between tasks by biasing toward learning 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 synthetic 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 dependency graph also proved to be a very useful data analysis tool. For example Friedman et al. [2] used Bayes Nets learned from gene expression level data to discover regulatory interactions between genes for one type of yeast.

In this paper we focus on simultaneously learning Bayes Net structures for multiple problems. Multi-task learning [3, 4, 5] suggests that if different problems are related, learning structures for the problems together should provide an advantage because what is learned for one problem may be useful for the other problems. For example, suppose that gene expression data is available for a number of different species of yeast. It is reasonable to assume that the regulatory interactions between genes for all yeast are similar, but not identical. Learning that there is an interaction between two genes for one species of yeast should provide evidence for interaction between the two genes in other species of yeast.

The paper is organized as follows: the next section gives an overview of Bayes Net structure learning. Section 3 describes one approach to simultaneously learning Bayes Nets for multiple data sets. Results of experiments with this method are presented in Section 4.

2 Learning Bayesian Networks 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]. Given the DAG G the joint probability distribution $P(X)$ can be factored according to:

$$P(X_1, X_2, ..., X_n|G, \theta) = \prod_{i=1}^{n} P(X_i|Pa(X_i), \theta) \tag{1}$$

where $Pa(X_i)$ are the parents of X_i in G. The parents $Pa(X_i)$ together with X_i are called the family of the node X_i. Thus 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 understand and can be used to gain better understanding of the problem.

Given a dataset $D = \{x^1, x^2, ...x^m\}$,where each x^i is a complete assignment of the variables $X_1, X_2, ..., X_n$, it is possible to learn both the structure G and the parameters θ. In this paper we will be concerned only with structure learning.

In the Bayesian learning paradigm, one is trying to estimate the posterior probability of the structure given the data: $P(G|D)$. Applying Bayes rule we get:

$$P(G|D) \propto P(G)P(D|G) \tag{2}$$

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 use the uninformative (uniform) prior that assigns the same probability to every structure. This prior is rarely accurate, but often is used for convenience. If a complete ordering on the nodes in G can be specified such that all the parents of a node precede it in the ordering, then a prior can be assessed by specifying the probability that each of the $n(n-1)/2$ possible edges is present in the correct structure [6]. Alternately, when there is access to a network believed to be close to the correct one (e.g. from an expert), $P(G)$ can also be specified by penalizing by a constant factor each difference between G and the given network [7].

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

$$P(D|G) = \int P(D|G, \theta)P(\theta|G)d\theta \tag{3}$$

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 efficiently computed in closed form [8].

Treating $P(G|D)$ as a score, one can search for a high scoring network using an heuristic search algorithm [8]. Greedy search, for example, starts from an initial structure, evaluates the score of all the *neighbors* of that structure and moves to the neighbor that has the highest score, provided its score is higher than the score of the current structure. The search terminates when we reach a structure that has a higher score than all of 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 structures every time. 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 have seen how to learn a Bayes Net for a given 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?

Formally, given the i.i.d 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 an ordered set of structures $(G_1, ..., G_k)$. As in Section 2, the posterior probability of a configuration given the data can be computed as:

$$P(G_1, ..., G_k|D_1, ..., D_k) \propto P(G_1, ..., G_k)P(D_1, ..., D_k|G_1, ..., G_k) \tag{4}$$

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:

$$\begin{aligned} &P(D_1, ..., D_k|G_1, ..., G_k) = \\ &= \int P(D_1, ..., D_k|G_1, ..., G_k, \theta_1, ..., \theta_k)P(\theta_1, ..., \theta_k|G_1, ..., G_k)d\theta_1...d\theta_k \\ &= \int P(\theta_1, ..., \theta_k|G_1, ..., G_k) \prod_{p=1}^{k} P(D_p|G_p, \theta_p)d\theta_1...d\theta_k \end{aligned} \tag{5}$$

If we make the simplifying assumption that the parameters of different networks are independent apriori (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:

$$P(G_1, ..., G_k|D_1, ..., D_k) \propto P(G_1, ..., G_k) \prod_{p=1}^{k} P(D_p|G_p) \tag{6}$$

While assuming parameter independence between related tasks is counterintuitive, this assumption is needed to make structure learning more efficient (see Section 3.3). It is important to note that this is not a restrictive assumption, i.e. it will not force the learned parameters to be independent, it only prevents multi-task learning from taking advantage of the similarities between the parameters of the Bayes Nets for different tasks. After the structures for all tasks have been learned, this assumption could be dropped in order to learn more accurate parameters. For simplicity, we also assume that all tasks have the same attributes $X_1, ..., X_n$. Everything that follows can be easily extended to handle tasks that do not have all attributes in common.

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 apriori (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 we are convinced that the structures for all the different tasks should be identical, then the prior $P(G_1, ..., G_k)$ should put zero probability on any configuration $(G_1, ..., G_k)$ that contains structures that are not identical. In this case we can efficiently learn the same structure for all the tasks by creating a new data set that has the 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 the parent of all the other nodes. The common structure for all the tasks is exactly this learned structure, with the node TSK and all the arcs connected to it removed.

[1] This is different from pooling the data, which would mean that not only the structures, but also the parameters for all tasks are 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 independently penalize each arc (X_i, X_j) that is present in one structure but not in the other by a constant $\delta_{ij} \in [0, 1]$:

$$P(G_1, G_2) = Z_\delta P(G_1) P(G_2) \prod_{\substack{(X_i, X_j) \in \\ G_1 \Delta G_2}} (1 - \delta_{ij}) \tag{7}$$

Assuming that the penalty for each arc is the same, only one parameter, δ, needs to be specified. If $\delta = 0$ the structures are independent. If $\delta = 1$ only configurations with identical structures have probability mass. 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 kind of structure prior that is appropriate for the task at hand. Z_δ is a normalization factor that is absorbed in the proportionality constant of equation 6.

In the case of more than two tasks an intuitive prior is:

$$P(G_1, ..., G_k) = Z \left(\prod_{1 \le i < j \le k} P(G_i, G_j) \right)^{\frac{1}{k-1}} \tag{8}$$

where $P(G_i, G_j)$ is the prior described above.

3.2 Greedy Structure Learning

As in the single task case, treating $P(G_1, ..., G_k | D_1, ..., D_k)$ as a score, we can search for a high scoring configuration using some heuristic search algorithm. If we choose to use greedy search for example, we start from a 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? Let the extended neighborhood of a single structure be its neighborhood as defined in Section 2 plus the structure itself. Then the neighborhood of a configuration could be defined as the cross-product of the extended neighborhoods of each structure in the configuration. In other words, every structure in the configuration can either stay unchanged or can change by deleting or reversing an existing arc or adding an arc in either direction between two nonadjacent nodes, with the constraint that the resulting structure must remain a DAG.

This definition of neighborhood is appealing because in the special cases when the structures for all the tasks are considered independent (zero penalty for diverging structures), and when they are required to be identical (infinite penalty for diverging structures), multi-task structure learning will produce identical results with the specialized algorithms described in Section 3.1. Unfortunately the size of such neighborhoods is of order of $O((n^2)^k)$ for k tasks and n attributes.

Another intuitive way to define the neighborhood of a configuration is to take all the configurations obtained by changing (add, remove, or reverse) an arc between the *same* two variables for all structures in the configuration. Given this more restrictive definition, the size of a neighborhood is $O(n^2 3^k)$, still exponential in the number of tasks, but only quadratic in the number of nodes. In this paper we will use this definition of neighborhood.

The number of configurations in a neighborhood can be quite large for large n or k, so computing the score for all the configurations can be expensive. In practice however, since we are only interested in the best scoring configuration in a neighborhood, we do not need to compute the scores of the configurations that have no chance of having the best score, so only a small fraction of configurations needs to be evaluated (see Section 3.3).

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. Under some assumptions, however, only fraction of the scores need to be computed.

Throughout this section we make the following assumptions: 1) the set $\mathcal{N} = \mathcal{G}_1 \times ... \times \mathcal{G}_k$ is generated by the cross-product of k sets of candidate structures, one for each task; 2) the parameters for each task are mutually independent apriori so the score of a configuration has the form in equation 6; 3) the prior over configurations has the form in equation 8.

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

$$S_{\mathcal{N}}(\mathcal{C}_i) = \left(\prod_{1 \le p < q \le i} P(G_p, G_q) \right)^{\frac{1}{k-1}} \left(\prod_{p=1}^{i} P(D_p | G_p) \right) \left(\prod_{q=i+1}^{k} Best_q \right) \tag{9}$$

where $Best_q = \max_{G_q \in \mathcal{G}_q} P(D_q | G_q)$. 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[2]. 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 configuration that need to be explored. In our experiments, using this pruning, we only need evaluate 2-4% of the configurations.

There are other heuristic search techniques that can be applied in order to make finding the best scoring configuration even faster. For example one could use A^* search with the scores of the partial configurations as the evaluation function. Another option is to return "almost" the best configuration by stopping the search early. In conclusion, even if in the worst case it takes exponential time, in practice it is often the case that the best neighboring configuration can be found much faster.

Moreover, because a configuration score has the form in equation 6, we can precompute the marginal likelihoods for all structures in $\mathcal{G}_1, ..., \mathcal{G}_k$, avoiding this expensive computation at every step of the search. In fact precomputing all the marginal likelihoods, which is required whether we do multitask learning or not, took 3 - 7 times longer than computing the best neighboring configuration [3].

4 Experimental Results

We evaluate the performance of multi-task structure learning on a set of synthetically created problems. Using the ALARM [9] and INSURANCE [10] networks, we created five related tasks by starting with the original network and deleting arcs with probability P_{del}. If 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 highly correlated parameters whose structures can be made more or less similar by varying P_{del} (For $P_{del} = 0$ all the structures are

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

[3]The implementation we used for this results did not use AD-trees. Using AD-trees would significantly reduce the time to precompute the marginal likelihoods.

identical). We also generated two more problems with similar structures but independent parameters, ALARM-IND and INSURANCE-IND, by starting, for each of the five tasks, with random parameters instead of the original ones. For all experiments we used ten trials to estimate performance. For each trial, in addition to varying the train and test sets, also we constructed different Bayes Nets. This way we show the expected performance over the entire class of problems that can be constructed using this methodology.

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[4] 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 ones. Structures are learned with the greedy multi-task learning algorithm described in Section 3.2, using the more restricted definition of a neighborhood. The greedy search is initialized with the solution found by single-task learning. We use the prior over configurations described in equation 8 with the same penalty for all arcs.

Figures 2 through 5 show the average percent reduction in loss, in terms of KL-divergence and edit distance, achieved by multi-task learning over single-task learning for a training set of 1000 points. On the x-axis we vary the penalty parameter of the prior. The 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 graphs for all values of P_{del} have similar behavior. 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 to high, however, the algorithm loses the ability to find true differences between the tasks and the performance drops. The trends in the graphs are exactly as expected. As the tasks become more similar, 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. Also 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), and consequently have smaller effect on KL-divergence. 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 proves that the assumption of apriori parameter independence made in Section 3 does not hurt the performance of multi-task learning. However, if we were able to also take advantage of the similarity between the parameters of the different tasks, we could presumably obtain even better performance. It is an open question how to relax the parameter independence assumption while still maintaining computational efficiency.

Unfortunately, when applying multi-task structure learning to a real problem, a good value for the penalty parameter of the prior is not usually known apriori. Here we take a very 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 1 shows the KL-Divergence and edit distance performance for single task learning and multi-task learning for the four problems when P_{del} is set to 0.05 for a training set of 1000 cases and a validation set of 50 cases. The penalty parameter for multi-task learning is selected as described above. Single-task learning uses both the training and the validation set to learn both the structure and the parameters of the Bayes

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

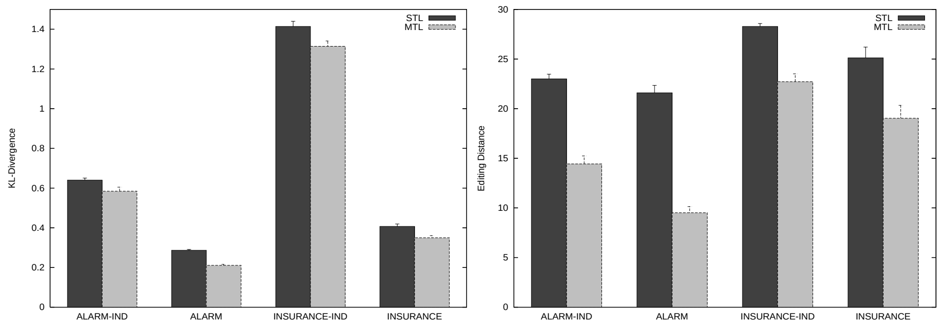

Figure 1: KL-Divergence (left) and edit distance (right) performance for STL and MTL

Nets. The figure shows that multi-task learning reduces the KL-divergence of 7% - 26% and the number of incorrect arcs in the learned structures by 20% - 55% when compared to structures learned separately. All differences between multi-task and single-task learning are .95 significant according to paired T-tests. In fact, multi-task learning never had worse performance than single-task learning on any trial for any of the four problems.

For a qualitative perspective, figure 6 shows the true structures, the structures learned by multi-task learning and single-task learning for the five tasks (one per line) on one trial of the ALARM-IND problem. The figure clearly shows that multi-task learning is able to to find more accurate structures by taking advantage of the similarity between the five different tasks, while still preserving some of the true differences between the five tasks (e.g. the edge $15 \rightarrow 32$).

5 Discussion and Related Work

We believe this is the first algorithm for multi-task Bayesian Network structure learning. 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, 11, 12]. For an overview of learning Bayes Nets from data see [8, 13].

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 [14], obtaining confidence measures on the structural features of the configurations via bootstrap analysis [15], and structure learning from incomplete datasets via the structural EM algorithm [16]. Other extensions such as obtaining a sample from the posterior distribution via MCMC methods are more problematic: with MCMC all neighboring configurations have to be scored at each step, not only the best one. MCMC might still be tractable if we use the methods in Section 3.3 to avoid examining low scoring configurations that will have a minor impact on the results. Another open question is whether we can relax the assumption that the parameters of the Bayes Nets for the different related tasks are independent apriori. This assumption is clearly not true in practice, and relaxing it might 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 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 arcs in the other structures. When this assumption is true, learning the structures together provides an advantage over learning a structure for each task individually. Similarity between learned structures is controlled via a prior. The posterior probability of a set of structures given

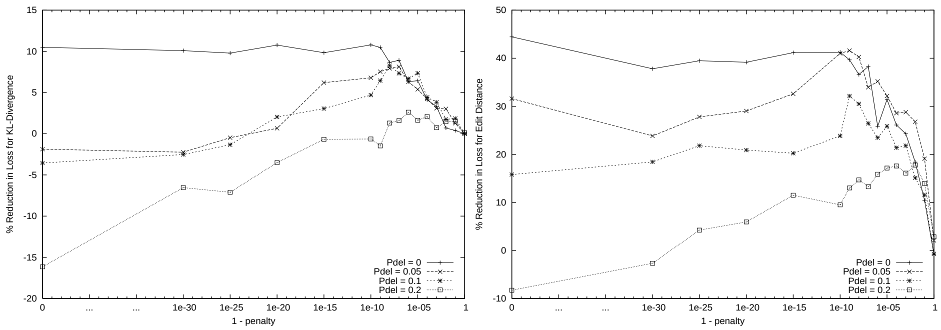

Figure 2: Reduction in KL-Divergence (left) and edit distance (right) for ALARM-IND

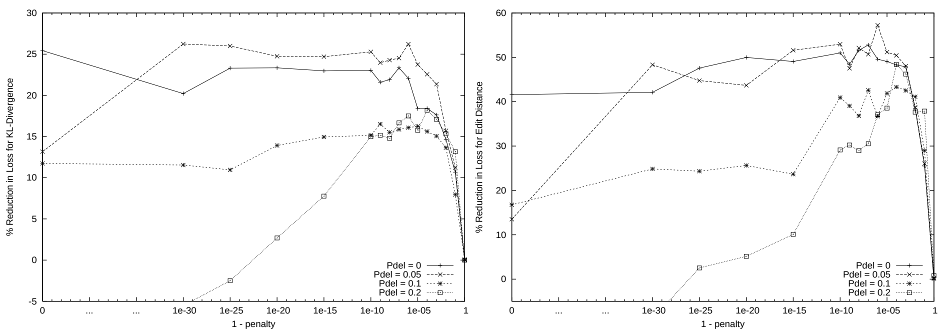

Figure 3: Reduction in KL-Divergence (left) and edit distance (right) for ALARM

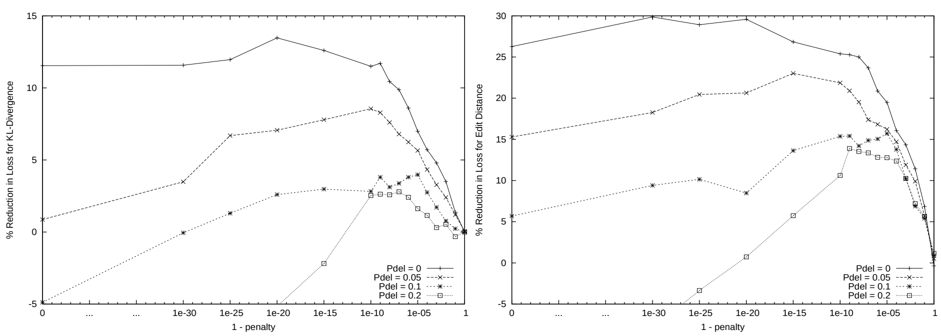

Figure 4: Reduction in KL-Div (left) and edit distance (right) for INSURANCE-IND

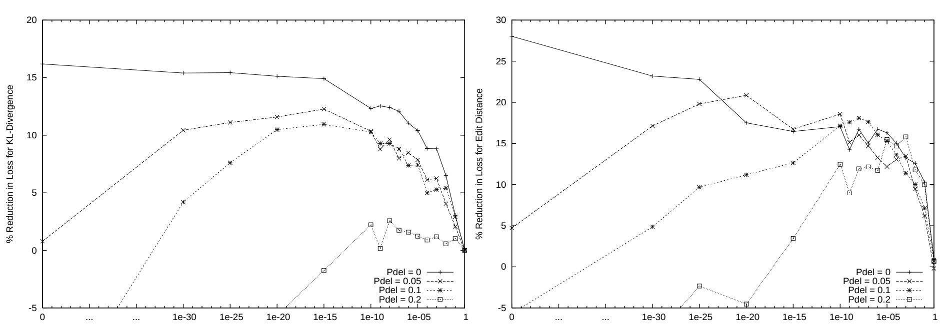

Figure 5: Reduction in KL-Divergence (left) and edit distance (right) for INSURANCE

the data is used by heuristic search to efficiently find high-scoring sets of structures. Experiments with synthetic data from perturbed ALARM and INSURANCE networks show that learning related structures simultaneously yields a reduction in KL-Divergence of 7% - 26% and reduces the number of incorrect arcs in the learned structures by 20% - 55% when compared to structures learned separately.

Acknowledgments

We thank 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.

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

[7] David Heckerman, Abe Mamdani, and Michael P. Wellman. Real-world applications of Bayesian networks. *Communications of the ACM*, 38(3):24–30, 1995.

[8] D. Heckerman. A tutorial on learning with bayesian networks, 1995.

[9] 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.

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

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

[12] 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.

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

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

[15] 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.

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

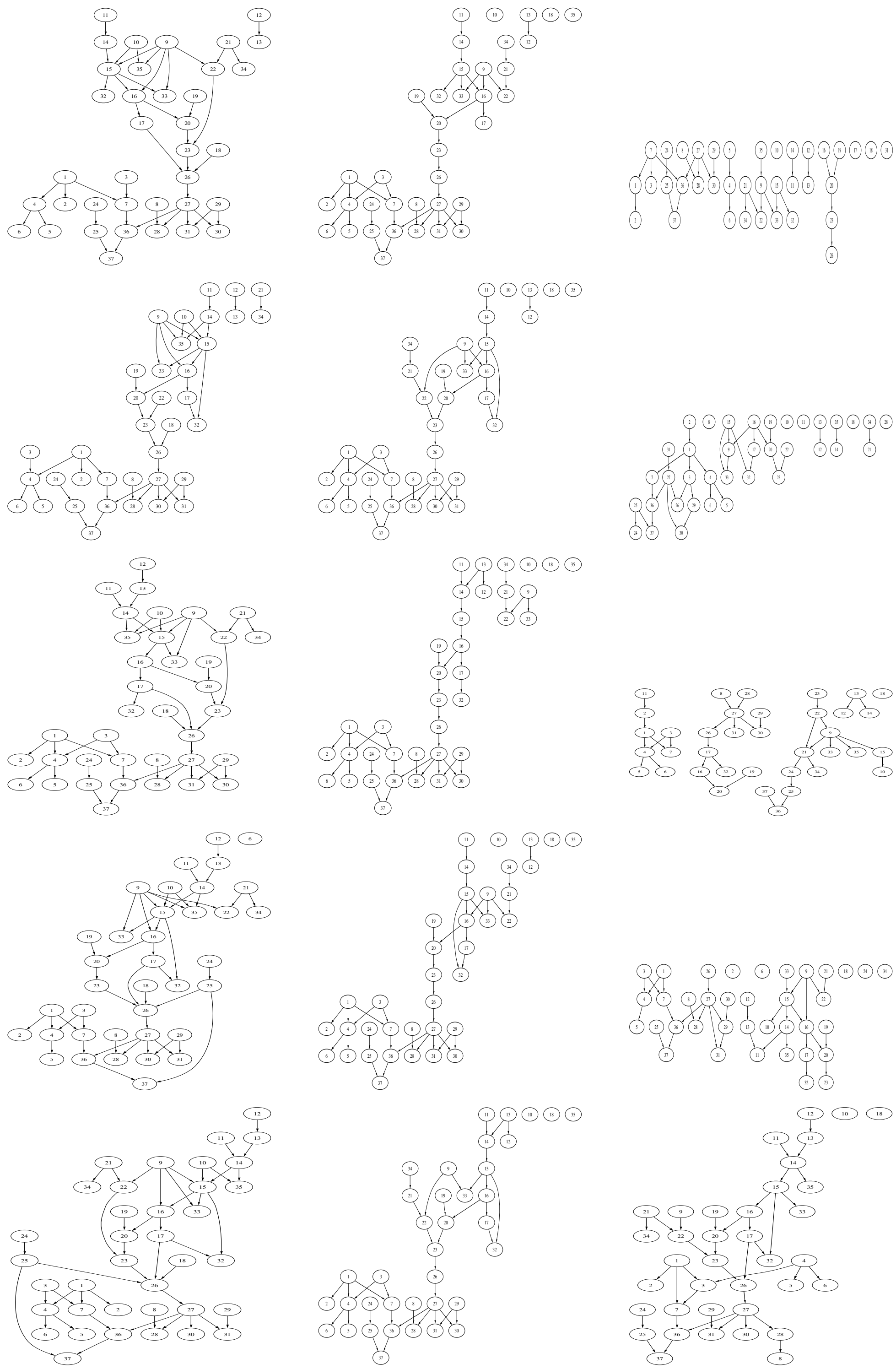

Figure 6: The true structures (left), structures learned by MTL (middle) and STL (right) for ALARM-IND