Learning Module Networks

Eran Segal   ERAN@CS.STANFORD.EDU
Computer Science Department
Stanford University
Stanford, CA 94305-9010, USA

Dana Pe'er   DPEER@GENETICS.MED.HARVARD.EDU
Genetics Department
Harvard Medical School
Boston, MA 02115, USA

Aviv Regev   AREGEV@CGR.HARVARD.EDU
Bauer Center for Genomic Research
Harvard University
Cambridge, MA 02138, USA

Daphne Koller   KOLLER@CS.STANFORD.EDU
Computer Science Department
Stanford University
Stanford, CA 94305-9010, USA

Nir Friedman   NIR@CS.HUJI.AC.IL
Computer Science & Engineering
Hebrew University
Jerusalem, 91904, Israel

Editor: Tommi Jaakkola

Abstract

Methods for learning Bayesian networks can discover dependency structure between observed variables. Although these methods are useful in many applications, they run into computational and statistical problems in domains that involve a large number of variables. In this paper,[1] we consider a solution that is applicable when many variables have similar behavior. We introduce a new class of models, *module networks*, that explicitly partition the variables into modules, so that the variables in each module share the same parents in the network and the same conditional probability distribution. We define the semantics of module networks, and describe an algorithm that learns the modules' composition and their dependency structure from data. Evaluation on real data in the domains of gene expression and the stock market shows that module networks generalize better than Bayesian networks, and that the learned module network structure reveals regularities that are obscured in learned Bayesian networks.

---

1. A preliminary version of this paper appeared in the Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence, 2003 (UAI '03).

1. Introduction

Over the last decade, there has been much research on the problem of learning Bayesian networks from data (Heckerman, 1998), and successfully applying it both to density estimation, and to discovering dependency structures among variables. Many real-world domains, however, are very complex, involving thousands of relevant variables. Examples include modeling the dependencies among expression levels (a rough indicator of activity) of all the genes in a cell (Friedman *et al.*, 2000a; Lander, 1999) or among changes in stock prices. Unfortunately, in complex domains, the amount of data is rarely enough to robustly learn a model of the underlying distribution. In the gene expression domain, a typical data set describes thousands of variables, but at most a few hundred instances. In such situations, statistical noise is likely to lead to spurious dependencies, resulting in models that significantly overfit the data.

Moreover, if our goal is structure discovery, such domains pose additional challenges. First, due to the small number of instances, we are unlikely to have much confidence in the learned structure (Pe'er *et al.*, 2001). Second, a Bayesian network structure over thousands of variables is typically highly unstructured, and therefore very hard to interpret.

In this paper, we propose an approach to address these issues. We start by observing that, in many large domains, the variables can be partitioned into sets so that, to a first approximation, the variables within each set have a similar set of dependencies and therefore exhibit a similar behavior. For example, many genes in a cell are organized into *modules*, in which sets of genes required for the same biological function or response are co-regulated by the same inputs in order to coordinate their joint activity. As another example, when reasoning about thousands of NASDAQ stocks, entire sectors of stocks often respond together to sector-influencing factors (e.g., oil stocks tend to respond similarly to a war in Iraq).

We define a new representation called a *module network*, which explicitly partitions the variables into *modules*. Each module represents a set of variables that have the same statistical behavior, i.e., they share the same set of parents and local probabilistic model. By enforcing this constraint on the learned network, we significantly reduce the complexity of our model space as well as the number of parameters. These reductions lead to more robust estimation and better generalization on unseen data. Moreover, even if a modular structure exists in the domain, it can be obscured by a general Bayesian network learning algorithm which does not have an explicit representation for modules. By making the modular structure explicit, the module network representation provides insight about the domain that are often be obscured by the intricate details of a large Bayesian network structure.

A module network can be viewed simply as a Bayesian network in which variables in the same module share parents and parameters. Indeed, probabilistic models with shared parameters are common in a variety of applications, and are also used in other general representation languages, such as *dynamic Bayesian networks* (Dean and Kanazawa, 1989), *object-oriented Bayesian Networks* (Koller and Pfeffer, 1997), and *probabilistic relational models* (Koller and Pfeffer, 1998; Friedman *et al.*, 1999a). (See Section 8 for further discussion of the relationship between module networks and these formalisms.) In most cases, the shared structure is imposed by the designer of the model, using prior knowledge about the domain. A key contribution of this paper is the design of a learning algorithm that directly searches for and finds sets of variables with similar behavior, which are then defined to be a module.

We describe the basic semantics of the module network framework, present a Bayesian scoring function for module networks, and provide an algorithm that learns both the assignment of variables

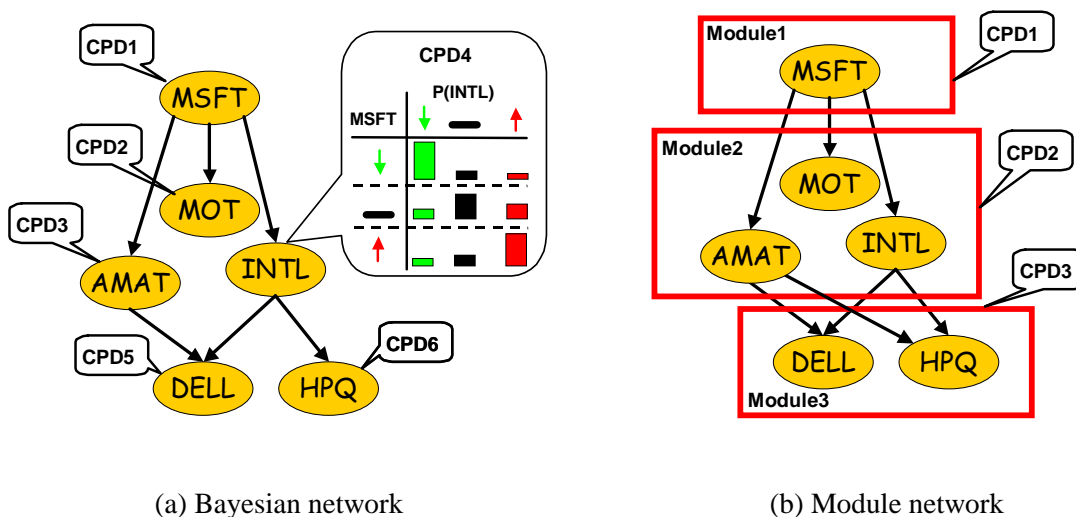(a) Bayesian network   (b) Module network

Figure 1: (a) A simple Bayesian network over stock price variables; the stock price of Intel (*INTL*) is annotated with a visualization of its CPD, described as a different multinomial distribution for each value of its influencing stock price Microsoft (*MSFT*). (b) A simple module network; the boxes illustrate modules, where stock price variables share CPDs and parameters. Note that in a module network, variables in the same module have the same CPDs but may have different descendants.

to modules and the probabilistic model for each module. We evaluate the performance of our algorithm on two real data sets, in the domains of gene expression and the stock market. Our results show that our learned module network generalizes to unseen test data much better than a Bayesian network. They also illustrate the ability of the learned module network to reveal high-level structure that provides important insights.

2. The Module Network Framework

We start with an example that introduces the main idea of module networks and then provide a formal definition. For concreteness, consider a simple toy example of modeling changes in stock prices. The Bayesian network of Figure 1(a) describes dependencies between different stocks. In this network, each random variable corresponds to the change in price of a single stock. For illustration purposes, we assume that these random variables take one of three values: 'down', 'same' or 'up', denoting the change during a particular trading day. In our example, the stock price of Intel (*INTL*) depends on that of Microsoft (*MSFT*). The *conditional probability distribution (CPD)* shown in the figure indicates that the behavior of Intel's stock is similar to that of Microsoft. That is, if Microsoft's stock goes up, there is a high probability that Intel's stock will also go up and vice versa. Overall, the Bayesian network specifies a CPD for each stock price as a stochastic function of its parents. Thus, in our example, the network specifies a separate behavior for each stock.

The stock domain, however, has higher order structural features that are not explicitly modeled by the Bayesian network. For instance, we can see that the stock price of Microsoft (*MSFT*) in-

fluences the stock price of all of the major chip manufacturers — Intel (*INTL*), Applied Materials (*AMAT*), and Motorola (*MOT*). In turn, the stock price of computer manufacturers Dell (*DELL*) and Hewlett Packard (*HPQ*), are influenced by the stock prices of their chip suppliers — Intel and Applied Materials. An examination of the CPDs might also reveal that, to a first approximation, the stock price of all chip making companies depends on that of Microsoft and in much the same way. Similarly, the stock price of computer manufacturers that buy their chips from Intel and Applied Materials depends on these chip manufacturers' stock and in much the same way.

To model this type of situation, we might divide stock price variables into groups, which we call *modules*, and require that variables in the same module have the same probabilistic model; that is, all variables in the module have the same set of parents and the same CPD. Our example contains three modules: one containing only Microsoft, a second containing chip manufacturers Intel, Applied Materials, and Motorola, and a third containing computer manufacturers Dell and HP (see Figure 1(b)). In this model, we need only specify three CPDs, one for each module, since all the variables in each module share the same CPD. By comparison, six different CPDs are required for a Bayesian network representation. This notion of a module is the key idea underlying the module network formalism.

We now provide a formal definition of a module network. Throughout this paper, we assume that we are given a domain of random variables $X = \{X_1, \ldots, X_n\}$. We use $Val(X_i)$ to denote the domain of values of the variable X_i.

As described above, a module represents a set of variables that share the same set of parents and the same CPD. As a notation, we represent each module by a *formal variable* that we use as a placeholder for the variables in the module. A *module set* C is a set of such formal variables $\mathbf{M}_1, \ldots, \mathbf{M}_K$. As all the variables in a module share the same CPD, they must have the same domain of values. We represent by $Val(\mathbf{M}_j)$ the set of possible values of the formal variable of the j'th module.

A module network relative to C consists of two components. The first defines a template probabilistic model for each module in C; all of the variables assigned to the module will share this probabilistic model.

Definition 1 *A* module network template $\mathcal{T} = (S, \theta)$ *for* C *defines, for each module* $\mathbf{M}_j \in C$:

- *a set of parents* $\mathbf{Pa_{M_j}} \subset X$;
- *a conditional probability distribution template* $P(\mathbf{M}_j \mid \mathbf{Pa_{M_j}})$ *which specifies a distribution over* $Val(\mathbf{M}_j)$ *for each assignment in* $Val(\mathbf{Pa_{M_j}})$.

We use S *to denote the dependency structure encoded by* $\{\mathbf{Pa_{M_j}} : \mathbf{M}_j \in C\}$ *and* θ *to denote the parameters required for the CPD templates* $\{P(\mathbf{M}_j \mid \mathbf{Pa_{M_j}}) : \mathbf{M}_j \in C\}$. ∎

In our example, we have three modules M_1, M_2, and M_3, with $\mathbf{Pa_{M_1}} = \emptyset$, $\mathbf{Pa_{M_2}} = \{MSFT\}$, and $\mathbf{Pa_{M_3}} = \{AMAT, INTL\}$.

The second component is a module assignment function that assigns each variable $X_i \in X$ to one of the K modules, $\mathbf{M}_1, \ldots, \mathbf{M}_K$. Clearly, we can only assign a variable to a module that has the same domain.

Definition 2 *A* module assignment function *for* C *is a function* $\mathcal{A} : X \to \{1, \ldots, K\}$ *such that* $\mathcal{A}(X_i) = j$ *only if* $Val(X_i) = Val(\mathbf{M}_j)$. ∎

In our example, we have that $\mathcal{A}(MSFT) = 1$, $\mathcal{A}(MOT) = 2$, $\mathcal{A}(INTL) = 2$, and so on.

A module network defines a probabilistic model by using the formal random variables $\mathbf{M}_j$ and their associated CPDs as templates that encode the behavior of all of the variables assigned to that module. Specifically, we define the semantics of a module network by "unrolling" a Bayesian network where all of the variables assigned to module $\mathbf{M}_j$ share the parents and conditional probability template assigned to $\mathbf{M}_j$ in $\mathcal{T}$. For this unrolling process to produce a well-defined distribution, the resulting network must be acyclic. Acyclicity can be guaranteed by the following simple condition on the module network:

Definition 3 *Let $\mathcal{M}$ be a triple $(C, \mathcal{T}, \mathcal{A})$, where C is a module set, $\mathcal{T}$ is a module network template for C, and $\mathcal{A}$ is a module assignment function for C. $\mathcal{M}$ defines a directed* module graph $\mathcal{G}_{\mathcal{M}}$ *as follows:*

- *the nodes in $\mathcal{G}_{\mathcal{M}}$ correspond to the modules in C;*
- *$\mathcal{G}_{\mathcal{M}}$ contains an edge $\mathbf{M}_j \rightarrow \mathbf{M}_k$ if and only if there is a variable $X \in X$ so that $\mathcal{A}(X) = j$ and $X \in \mathbf{Pa}_{\mathbf{M}_k}$.*

We say that $\mathcal{M}$ is a module network *if the module graph $\mathcal{G}_{\mathcal{M}}$ is acyclic.* ∎

For example, for the module network of Figure 1(b), the module graph has the structure $\mathbf{M}_1 \rightarrow \mathbf{M}_2 \rightarrow \mathbf{M}_3$.

We can now define the semantics of a module network:

Definition 4 *A module network $\mathcal{M} = (C, \mathcal{T}, \mathcal{A})$ defines a* ground Bayesian network $\mathcal{B}_{\mathcal{M}}$ *over X as follows: For each variable $X_i \in X$, where $\mathcal{A}(X_i) = j$, we define the parents of X_i in $\mathcal{B}_{\mathcal{M}}$ to be $\mathbf{Pa}_{\mathbf{M}_j}$, and its conditional probability distribution to be $P(\mathbf{M}_j \mid \mathbf{Pa}_{\mathbf{M}_j})$, as specified in $\mathcal{T}$. The distribution associated with $\mathcal{M}$ is the one represented by the Bayesian network $\mathcal{B}_{\mathcal{M}}$.* ∎

Returning to our example, the Bayesian network of Figure 1(a) is the ground Bayesian network of the module network of Figure 1(b).

Using the acyclicity of the module graph, we can now show that the semantics for a module network is well-defined.

Proposition 5 *The graph $\mathcal{G}_{\mathcal{M}}$ is acyclic if and only if the dependency graph of $\mathcal{B}_{\mathcal{M}}$ is acyclic.*

Proof: The proof follows from the direct correspondence between edges in the module graph and edges in the ground Bayesian network. Consider some edge $X_i \rightarrow X_j$ in $\mathcal{B}_{\mathcal{M}}$. By definition of the module graph, we must have an edge $\mathbf{M}_{\mathcal{A}(X_i)} \rightarrow \mathbf{M}_{\mathcal{A}(X_j)}$ in the module graph. Thus, any cyclic path in $\mathcal{B}_{\mathcal{M}}$ corresponds directly to a cyclic path in the module graph, proving one direction of the theorem. The proof in the other direction is slightly more subtle. Assume that there exists a cyclic path $\mathbf{p} = (\mathbf{M}_1 \rightarrow \mathbf{M}_2 \ldots \mathbf{M}_l \rightarrow \mathbf{M}_1)$ in the module graph. By definition of the module graph, if $\mathbf{M}_i \rightarrow \mathbf{M}_{i+1}$ there is a variable X_i with $\mathcal{A}(X_i) = \mathbf{M}_i$ that is a parent of X_{i+1}, for each $i = 1, \ldots, l - 1$. By construction, it follows that there is an arc $X_i \rightarrow X_{i+1}$ in $\mathcal{B}_{\mathcal{M}}$. Similarly, there is a variable X_l with $\mathcal{A}(X_l) = \mathbf{M}_l$ that is a parent of $\mathbf{M}_1$. And so, we conclude that $\mathcal{B}_{ModNet}$ contains a cycle $X_1 \rightarrow X_2 \rightarrow \ldots X_l \rightarrow X_1$, proving the other direction of the theorem ∎

Corollary 6 *For any module network $\mathcal{M}$, $\mathcal{B}_{\mathcal{M}}$ defines a coherent probability distribution over X.*

As we can see, a module network provides a succinct representation of the ground Bayesian network. In a realistic version of our stock example, we might have several thousand stocks. A Bayesian network in this domain needs to represent thousands of CPDs. On the other hand, a module network can often represent a good approximation of the domain using a model with only few dozen CPDs.

3. Data Likelihood and Bayesian Scoring

We now turn to the task of learning module networks from data. Recall that a module network is specified by a set of modules C, an assignment function $\mathcal{A}$ of nodes to modules, the parent structure S specified in $\mathcal{T}$, and the parameters θ for the local probability distributions $P(\mathbf{M}_j \mid \mathbf{Pa}_{\mathbf{M}_j})$. We assume in this paper that the set of modules C is given, and omit reference to it from now on. We note that, in the models we consider in this paper, we do not associate properties with specific modules and thus only the number of modules is of relevance to us. However, in other settings (e.g., in cases with different types of random variables) we may wish to distinguish between different module types. Such distinctions can be made within the module network framework through more elaborate prior probability functions that take the module type into account.

One can consider several learning tasks for module networks, depending on which of the remaining aspects of the module network specification are known. In this paper, we focus on the most general task of learning the network structure and the assignment function, as well as a Bayesian posterior over the network parameters. The other tasks are special cases that can be derived as a by-product of our algorithm.

Thus, we are given a training set $\mathcal{D} = \{\mathbf{x}[1], \ldots, \mathbf{x}[M]\}$, consisting of M instances drawn independently from an unknown distribution $P(\mathcal{X})$. Our primary goal is to learn a module network structure and assignment function for this distribution. We take a *score-based approach* to this learning task. In this section, we define a scoring function that measures how well each candidate model fits the observed data. We adopt the Bayesian paradigm and derive a Bayesian scoring function similar to the Bayesian score for Bayesian networks (Cooper and Herskovits, 1992; Heckerman *et al.*, 1995). In the next section, we consider the algorithmic problem of finding a high scoring model.

3.1 Likelihood Function

We begin by examining the *data likelihood* function

$$L(\mathcal{M} : \mathcal{D}) = P(\mathcal{D} \mid \mathcal{M}) = \prod_{m=1}^{M} P(\mathbf{x}[m] \mid \mathcal{T}, \mathcal{A}).$$

This function plays a key role both in the parameter estimation task and in the definition of the structure score.

As the semantics of a module network is defined via the ground Bayesian network, we have that, in the case of complete data, the likelihood decomposes into a product of *local likelihood functions*, one for each variable. In our setting, however, we have the additional property that the variables in a module share the same local probabilistic model. Hence, we can aggregate these local likelihoods, obtaining a decomposition according to modules.

More precisely, let $\mathbf{X}^j = \{X \in \mathcal{X} \mid \mathcal{A}(X) = j\}$, and let $\theta_{\mathbf{M}_j \mid \mathbf{Pa}_{\mathbf{M}_j}}$ be the parameters associated with the CPD template $P(\mathbf{M}_j \mid \mathbf{Pa}_{\mathbf{M}_j})$. We can decompose the likelihood function as a product of
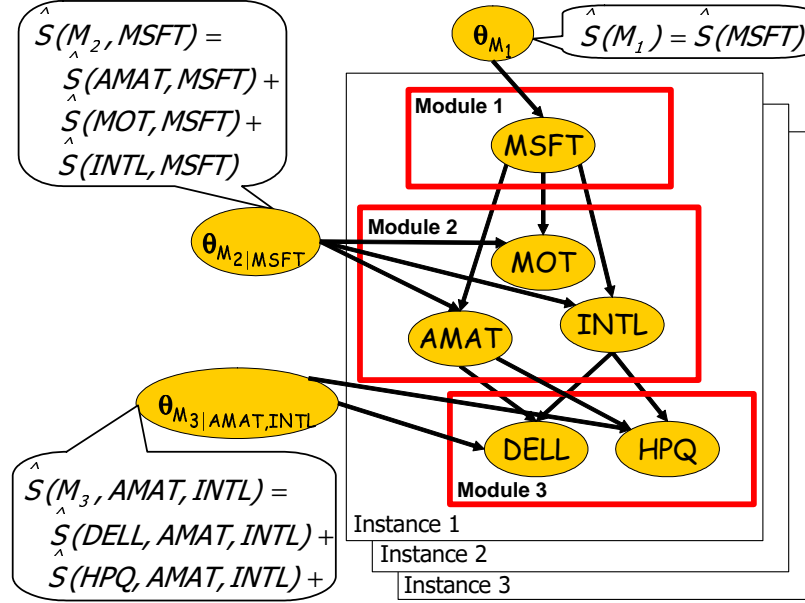
Figure 2: Shown is a plate model for three instances of the module network example of Figure 1(b). The CPD template of each module is connected to all variables assigned to that module (e.g. $\theta_{M_2|MSFT}$ is connected to *AMAT*, *MOT*, and *INTL*). The sufficient statistics of each CPD template are the sum of the sufficient statistics of each variable assigned to the module and the module parents.

module likelihoods, each of which can be calculated independently and depends only on the values of $\mathbf{X}^j$ and $\mathbf{Pa}_{\mathbf{M}_j}$, and on the parameters $\theta_{\mathbf{M}_j|\mathbf{Pa}_{\mathbf{M}_j}}$:

$$
\begin{aligned}
&L(\mathcal{M} : \mathcal{D}) \\
&= \prod_{j=1}^{K} \left[\prod_{m=1}^{M} \prod_{X_i \in \mathbf{X}^j} P(x_i[m] \mid \mathbf{pa}_{\mathbf{M}_j}[m], \theta_{\mathbf{M}_j|\mathbf{Pa}_{\mathbf{M}_j}}) \right] \\
&= \prod_{j=1}^{K} L_j(\mathbf{Pa}_{\mathbf{M}_j}, \mathbf{X}^j, \theta_{\mathbf{M}_j|\mathbf{Pa}_{\mathbf{M}_j}} : \mathcal{D}).
\end{aligned}
\tag{1}
$$

If we are learning conditional probability distributions from the exponential family (e.g., discrete distribution, Gaussian distributions, and many others), then the local likelihood functions can be reformulated in terms of *sufficient statistics* of the data. The sufficient statistics summarize the relevant aspects of the data. Their use here is similar to that in Bayesian networks (Heckerman, 1998), with one key difference. In a module network, all of the variables in the same module share the same parameters. Thus, we pool all of the data from the variables in $\mathbf{X}^j$, and calculate our statistics based on this pooled data. More precisely, let $S_j(M_j, \mathbf{Pa}_{\mathbf{M}_j})$ be a sufficient statistic function for the CPD $P(M_j \mid \mathbf{Pa}_{\mathbf{M}_j})$. Then the value of the statistic on the data set $\mathcal{D}$ is

$$
\hat{S}_j = \sum_{m=1}^{M} \sum_{X_i \in \mathbf{X}^j} S_j(x_i[m], \mathbf{pa}_{\mathbf{M}_j}[m]).
\tag{2}
$$

For example, in the case of networks that use only multinomial table CPDs, we have one sufficient statistic function for each joint assignment $x \in Val(\mathbf{M}_j), \mathbf{u} \in Val(\mathbf{Pa}_{\mathbf{M}_j})$, which is $\eta\{X_i[m] = x, \mathbf{pa}_{\mathbf{M}_j}[m] = \mathbf{u}\}$ — the indicator function that takes the value 1 if the event $(X_i[m] = x, \mathbf{Pa}_{\mathbf{M}_j}[m] = \mathbf{u})$ holds, and 0 otherwise. The statistic on the data is

$$\hat{S}_j[x, \mathbf{u}] \quad = \quad \sum_{m=1}^{M} \sum_{X_i \in \mathbf{X}^j} \eta\{X_i[m] = x, \mathbf{Pa}_{\mathbf{M}_j}[m] = \mathbf{u}\}.$$

Given these sufficient statistics, the formula for the module likelihood function is:

$$L_j(\mathbf{Pa}_{\mathbf{M}_j}, \mathbf{X}^j, \theta_{\mathbf{M}_j | \mathbf{Pa}_{\mathbf{M}_j}} : \mathcal{D}) = \prod_{x, \mathbf{u} \in Val(\mathbf{M}_j, \mathbf{Pa}_{\mathbf{M}_j})} \theta_{x|\mathbf{u}}^{\hat{S}_j[x, \mathbf{u}]}.$$

This term is precisely the one we would use in the likelihood of Bayesian networks with multinomial table CPDs. The only difference is that the vector of sufficient statistics for a local likelihood term is pooled over all the variables in the corresponding module.

For example, consider the likelihood function for the module network of Figure 1(b). In this network we have three modules. The first consists of a single variable and has no parents, and so the vector of statistics $\hat{S}[\mathbf{M}_1]$ is the same as the statistics of the single variable $\hat{S}[MSFT]$. The second module contains three variables; thus, the sufficient statistics for the module CPD is the sum of the statistics we would collect in the ground Bayesian network of Figure 1(a):

$$\hat{S}[\mathbf{M}_2, MSFT] = \hat{S}[AMAT, MSFT] + \hat{S}[MOT, MSFT] + \hat{S}[INTL, MSFT].$$

Finally,

$$\hat{S}[\mathbf{M}_3, AMAT, INTL] = \hat{S}[DELL, AMAT, INTL] + \hat{S}[HPQ, AMAT, INTL].$$

An illustration of the decomposition of the likelihood and the associated sufficient statistics using the plate model is shown in Figure 2.

As usual, the decomposition of the likelihood function allows us to perform maximum likelihood or MAP parameter estimation efficiently, optimizing the parameters for each module separately. The details are standard (Heckerman, 1998), and are thus omitted.

3.2 Priors and the Bayesian Score

As we discussed, our approach for learning module networks is based on the use of a Bayesian score. Specifically, we define a model score for a pair $(\mathcal{S}, \mathcal{A})$ as the posterior probability of the pair, integrating out the possible choices for the parameters θ. We define an assignment prior $P(\mathcal{A})$, a structure prior $P(\mathcal{S} \mid \mathcal{A})$ and a parameter prior $P(\theta \mid \mathcal{S}, \mathcal{A})$. These describe our preferences over different networks *before* seeing the data. By Bayes' rule, we then have

$$P(\mathcal{S}, \mathcal{A} \mid \mathcal{D}) \propto P(\mathcal{A})P(\mathcal{S} \mid \mathcal{A})P(\mathcal{D} \mid \mathcal{S}, \mathcal{A}),$$

where the last term is the *marginal likelihood*

$$P(\mathcal{D} \mid \mathcal{S}, \mathcal{A}) = \int P(\mathcal{D} \mid \mathcal{S}, \mathcal{A}, \theta)P(\theta \mid \mathcal{S})d\theta.$$

We define the Bayesian score as the log of $P(\mathcal{S}, \mathcal{A} \mid \mathcal{D})$, ignoring the normalization constant

$$\text{score}(\mathcal{S}, \mathcal{A} : \mathcal{D}) = \log P(\mathcal{A}) + \log P(\mathcal{S} \mid \mathcal{A}) + \log P(\mathcal{D} \mid \mathcal{S}, \mathcal{A}). \tag{3}$$

As with Bayesian networks, when the priors satisfy certain conditions, the Bayesian score decomposes. This decomposition allows to efficiently evaluate a large number of alternatives. The same general ideas carry over to module networks, but we also have to include assumptions that take the assignment function into account. Following is a list of conditions on the prior required for the decomposability of the Bayesian score in the case of module networks:

Definition 7 *Let $P(\theta, S, \mathcal{A})$ be a prior over assignments, structures, and parameters.*

- *$P(\theta, S, \mathcal{A})$ is* globally modular *if*

$$P(\theta \mid S, \mathcal{A}) = P(\theta \mid S),$$

  and

$$P(S, \mathcal{A}) \propto \rho(S)\kappa(\mathcal{A})C(\mathcal{A}, S),$$

  where $\rho(S)$ and $\kappa(\mathcal{A})$ are non-negative measures over structures and assignments, and $C(\mathcal{A}, S)$ is a constraint indicator function that is equal to 1 if the combination of structure and assignment is a legal one (i.e., the module graph induced by the assignment $\mathcal{A}$ and structure S is acyclic), and 0 otherwise.

- *$P(\theta \mid S)$ satisfies* parameter independence *if*

$$P(\theta \mid S) = \prod_{j=1}^{K} P(\theta_{\mathbf{M}_j | \mathbf{Pa}_{\mathbf{M}_j}} \mid S).$$

- *$P(\theta \mid S)$ satisfies* parameter modularity *if*

$$P(\theta_{\mathbf{M}_j | \mathbf{Pa}_{\mathbf{M}_j}} \mid S_1) = P(\theta_{\mathbf{M}_j | \mathbf{Pa}_{\mathbf{M}_j}} \mid S_2).$$

  for all structures S_1 and S_2 such that $\mathbf{Pa}_{\mathbf{M}_j}^{S_1} = \mathbf{Pa}_{\mathbf{M}_j}^{S_2}$.

- *$\rho(S)$ satisfies* structure modularity *if*

$$\rho(S) = \prod_{j} \rho_j(S_j),$$

  where S_j denotes the choice of parents for module $\mathbf{M}_j$ and ρ_j is a non-negative measure over these choices.

- *$\kappa(\mathcal{A})$ satisfies* assignment modularity *if*

$$\kappa(\mathcal{A}) = \prod_{j} \kappa_j(\mathcal{A}_j),$$

  where $\mathcal{A}_j$ denote is the choice of variables assigned to module $\mathbf{M}_j$ and κ_j is a non-negative measure over these choices. ∎

Global modularity implies that the prior can be thought of as a combination of three components — a parameter prior that depends on the network structure, a structure prior, and an assignment prior. Clearly the last two components cannot be independent, as the the assignment and the structure together must define a legal network. However, global modularity implies that these two priors are "as independent as possible". The legality requirement, which is encoded by the indicator function $C(\mathcal{A}, \mathcal{S})$ ensures that only legal assignment/structure pairs have a non-zero probability. Other than this constraint, the preferences over structures and over assignments are specified separately.

Parameter independence and parameter modularity are the natural analogues of standard assumptions in Bayesian network learning (Heckerman *et al.*, 1995). Parameter independence implies that $P(\theta \mid \mathcal{S})$ is a product of terms that parallels the decomposition of the likelihood in Equation (1), with one prior term per local likelihood term L_j. Parameter modularity states that the prior for the parameters of a module M_j depends only on the choice of parents for M_j and not on other aspects of the structure.

Finally, structure modularity and assignment modularity imply that the structure an assignments priors are products of local terms that encode preferences over parents and variable assignments separately for each module.

As for the standard conditions on Bayesian network priors, the conditions we define are not universally justified, and one can easily construct examples where we would want to relax them. However, they simplify many of the computations significantly, and are therefore useful even if they are only a rough approximation. Moreover, the assumptions, although restrictive, still allow broad flexibility in our choice of priors. For example, we can encode preference (or restrictions) on the assignments of particular variables to specific modules. In addition, we can also encode preference for particular module sizes.

For priors satisfying the assumptions of Definition 7, we can prove the decomposability property of the Bayesian score for module networks:

Theorem 8 *Let $P(\theta, \mathcal{S}, \mathcal{A})$ be a prior satisfying the assumptions of Definition 7. Then, the Bayesian score decomposes into local* module scores*:*

$$\text{score}(\mathcal{S}, \mathcal{A} \; : \; \mathcal{D}) = \sum_{j=1}^{K} \text{score}_{\mathbf{M}_j}(\mathbf{Pa}_{\mathbf{M}_j}, \mathcal{A}(\mathbf{X}^j) \; : \; \mathcal{D}),$$

where

$$
\begin{aligned}
\text{score}_{\mathbf{M}_j}(\mathbf{U}, \mathbf{X} \; : \; \mathcal{D}) \;\; = \;\; & \log \int L_j(\mathbf{U}, \mathbf{X}, \theta_{\mathbf{M}_j|\mathbf{U}} : \mathcal{D}) P(\theta_{\mathbf{M}_j} \mid \mathbf{U}) d\theta_{\mathbf{M}_j|\mathbf{U}} \\
& + \log \rho_j(\mathbf{U}) + \log \kappa_j(\mathbf{X}).
\end{aligned}
\tag{4}
$$

Proof Recall that we defined the Bayesian score of a module network as:

$$\text{score}(\mathcal{S}, \mathcal{A} \; : \; \mathcal{D}) = \log P(\mathcal{D} \mid \mathcal{S}, \mathcal{A}) + \log P(\mathcal{S}, \mathcal{A}).$$

Using *global modularity*, *structure modularity* and *assignment modularity* assumptions of Definition 7, $\log P(\mathcal{S}, \mathcal{A})$ decomposes by modules, resulting in the second and third terms Equation (4) that capture the preferences for the parents of module $\mathbf{M}_j$ and the variables assigned to it. Note that we can ignore the normalization constant of the prior $P(\mathcal{S}, \mathcal{A})$. For the first term of Equation (4), we

can write:

$$
\begin{aligned}
\log P(\mathcal{D} \mid \mathcal{S}, \mathcal{A}) &= \log \int P(\mathcal{D} \mid \mathcal{S}, \mathcal{A}, \theta) P(\theta \mid \mathcal{S}, \mathcal{A}) d\theta \\
&= \log \prod_{i=1}^{K} \int L_j(\mathbf{U}, \mathbf{X}, \theta_{\mathbf{M}_j \mid \mathbf{U}} : \mathcal{D}) P(\theta_{\mathbf{M}_j} \mid \mathbf{U}) d\theta_{\mathbf{M}_j \mid \mathbf{U}} \\
&= \sum_{i=1}^{K} \log \int L_j(\mathbf{U}, \mathbf{X}, \theta_{\mathbf{M}_j \mid \mathbf{U}} : \mathcal{D}) P(\theta_{\mathbf{M}_j} \mid \mathbf{U}) d\theta_{\mathbf{M}_j \mid \mathbf{U}},
\end{aligned}
$$

where in the second step we used the likelihood decomposition of Equation (1) and the global modularity, parameter independence, and parameter modularity assumptions of Definition 7. ∎

As we shall see below, the decomposition of the Bayesian score plays a crucial rule in our ability to devise an efficient learning algorithm that searches the space of module networks for one with high score. The only question is how to evaluate the integral over $\theta_{\mathbf{M}_j}$ in $\mathrm{score}_{\mathbf{M}_j}(\mathbf{U}, \mathbf{X} : \mathcal{D})$. This depends on the parametric forms of the CPD and the form of the prior $P(\theta_{\mathbf{M}_j} \mid \mathcal{S})$. Usually we choose priors that are *conjugate* to the parameter distributions. Such a choice leads to closed form analytic formula of the value of the integral as a function of the sufficient statistics of $L_j(\mathbf{Pa}_{\mathbf{M}_j}, \mathbf{X}^j, \theta_{\mathbf{M}_j \mid \mathbf{Pa}_{\mathbf{M}_j}} : \mathcal{D})$. For example, using Dirichlet priors with multinomial table CPDs leads to the following formula for the integral over $\theta_{\mathbf{M}_j}$:

$$
\begin{aligned}
&\log \int L_j(\mathbf{U}, \mathbf{X}, \theta_{\mathbf{M}_j \mid \mathbf{U}} : \mathcal{D}) P(\theta_{\mathbf{M}_j} \mid \mathbf{U}) d\theta_{\mathbf{M}_j \mid \mathbf{U}} = \\
&\sum_{\mathbf{u} \in \mathbf{U}} \log \frac{\Gamma(\sum_{v \in Val(\mathbf{M}_j)} \alpha_j[v, \mathbf{u}])}{\Gamma(\sum_{v \in Val(\mathbf{M}_j)} \hat{S}_j[v, \mathbf{u}] + \alpha_j[v, \mathbf{u}])} \prod_{v \in Val(\mathbf{M}_j)} \frac{\Gamma(\hat{S}_j[v, \mathbf{u}] + \alpha_j[v, \mathbf{u}])}{\Gamma(\alpha_j[v, \mathbf{u}])},
\end{aligned}
$$

where $\hat{S}_j[v, \mathbf{u}]$ is the sufficient statistics function as defined in Equation (2), and $\alpha_j[v, \mathbf{u}]$ is the hyperparameter of the Dirichlet distribution given the assignment $\mathbf{u}$ to the parents $\mathbf{U}$ of $\mathbf{M}_j$. We note that in the above formula we have also made use of the *local parameter independence* assumption on the form of the prior (Heckerman, 1998), which states that the prior distribution for the different values of the parents are independent:

$$
P(\theta_{\mathbf{M}_j \mid \mathbf{Pa}_{\mathbf{M}_j}} \mid \mathcal{S}) = \prod_{\mathbf{u} \in Val(\mathbf{Pa}_{\mathbf{M}_j})} P(\theta_{\mathbf{M}_j \mid \mathbf{u}} \mid \mathcal{S}).
$$

4. Learning Algorithm

Given a scoring function over networks, we now consider how to find a high scoring module network. This problem is a challenging one, as it involves searching over two combinatorial spaces simultaneously — the space of structures and the space of module assignments. We therefore simplify our task by using an iterative approach that repeats two steps: In one step, we optimize a dependency structure relative to our current assignment function, and in the other, we optimize an assignment function relative to our current dependency structure.

4.1 Structure Search Step

The first type of step in our iterative algorithm learns the structure $\mathcal{S}$, assuming that $\mathcal{A}$ is fixed. This step involves a search over the space of dependency structures, attempting to maximize the score defined in Equation (3). This problem is analogous to the problem of structure learning in Bayesian networks. We use a standard heuristic search over the combinatorial space of dependency structures (Heckerman *et al.*, 1995). We define a search space, where each state in the space is a legal parent structure, and a set of operators that take us from one state to another. We traverse this space looking for high scoring structures using a search algorithm such as greedy hill climbing.

In many cases, an obvious choice of local search operators involves steps of adding or removing a variable X_i from a parent set $\mathbf{Pa}_{\mathbf{M}_j}$. (Note that edge reversal is not a well-defined operator for module networks, as an edge from a variable to a module represents a one-to-many relation between the variable and all of the variables in the module.) When an operator causes a parent X_i to be added to the parent set of module $\mathbf{M}_j$, we need to verify that the resulting module graph remains acyclic, relative to the current assignment $\mathcal{A}$. Note that this step is quite efficient, as acyclicity is tested on the module graph, which contains only K nodes, rather than on the dependency graph of the ground Bayesian network, which contains n nodes (usually $n \gg K$).

Also note that, as in Bayesian networks, the decomposition of the score provides considerable computational savings. When updating the dependency structure for a module $\mathbf{M}_j$, the module score for another module $\mathbf{M}_k$ does not change, nor do the changes in score induced by various operators applied to the dependency structure of $\mathbf{M}_k$. Hence, after applying an operator to $\mathbf{Pa}_{\mathbf{M}_j}$, we need only update the change in score for those operators that involve $\mathbf{M}_j$. Moreover, only the delta score of operators that add or remove a parent from module $\mathbf{M}_j$ need to be recomputed after a change to the dependency structure of module $\mathbf{M}_j$, resulting in additional savings. This is analogous to the case of Bayesian network learning, where after applying a step that changes the parents of a variable X, we only recompute the delta score of operators that affect the parents of X.

Overall, if the maximum number of parents per module is d, the cost of evaluating each operator applied to the module is, as usual, at most $O(Md)$, for accumulating the necessary sufficient statistics. The total number of structure update operators is $O(Kn)$, so the cost of computing the delta-scores for all structure search operators requires $O(KnMd)$. This computation is done at the beginning of each structure learning phase. During the structure learning phase, each step to the parent set of module $\mathbf{M}_j$ requires that we re-evaluate at most n operators (one for each existing or potential parent of $\mathbf{M}_j$), at a total cost of $O(nMd)$.

4.2 Module Assignment Search Step

The second type of step in our iteration learns an assignment function $\mathcal{A}$ from data. This type of step occurs in two places in our algorithm: once at the very beginning of the algorithm, in order to initialize the modules, and once at each iteration, given a module network structure $\mathcal{S}$ learned in the previous structure learning step.

4.2.1 MODULE ASSIGNMENT AS CLUSTERING

In this step, our task is as follows: Given a fixed structure $\mathcal{S}$ we want to find $\mathcal{A} = \operatorname{argmax}_{\mathcal{A}'} \operatorname{score}_{\mathbf{M}}(\mathcal{S}, \mathcal{A}' : \mathcal{D})$. Interestingly, we can view this task as a clustering problem. A module consists of a set of variables that have the same probabilistic model. Thus, for a given instance, two different variables in the same module define the same probabilistic model, and therefore should have similar behavior.

Input:
    D // Data set
    $\mathcal{A}_0$ // Initial assignment function
    $\mathcal{S}$ // Given dependency structure
Output:
    $\mathcal{A}$ // improved assignment function
Sequential-Update
    $\mathcal{A} = \mathcal{A}_0$
    Loop
        For $i = 1$ to n
            For $j = 1$ to K
                $\mathcal{A}' = \mathcal{A}$ except that $\mathcal{A}'(X_i) = j$
                If $\langle \mathcal{G}_{\mathcal{M}}, \mathcal{A}' \rangle$ is cyclic, **continue**
                If $\mathrm{score}(\mathcal{S}, \mathcal{A}' : \mathcal{D}) > \mathrm{score}(\mathcal{S}, \mathcal{A} : \mathcal{D})$
                    $\mathcal{A} = \mathcal{A}'$
    Until no reassignments to any of $X_1, \ldots X_n$
    Return $\mathcal{A}$

Figure 3: Outline of sequential algorithm for finding the module assignment function

We can therefore view the module assignment task as the task of clustering variables into sets, so that variables in the same set have a similar behavior across all instances.

For example, in our stock market example, we would cluster stocks based on the similarity of their behavior over different trading days. Note that in a typical application of a clustering algorithm (e.g., k-means or the AutoClass algorithm of Cheeseman *et al.* (1988)) to our data set, we would cluster data instances (trading days) based on the similarity of the variables characterizing them. Here, we view instances as features of variables, and try to cluster variables. (See Figure 5.)

However, there are several key differences between this task and the typical formulation of clustering. First, in general, the probabilistic model associated with each cluster has structure, as defined by the CPD template associated with the cluster (module). Moreover, our setting places certain constraints on the clustering, so that the resulting assignment function will induce a legal (acyclic) module network.

4.2.2 MODULE ASSIGNMENT INITIALIZATION

In the initialization phase, we exploit the clustering perspective directly, using a form of hierarchical agglomerative clustering that is tailored to our application. Our clustering algorithm uses an objective function that evaluates a partition of variables into modules by measuring the extent to which the module model is a good fit to the features (instances) of the module variables. This algorithm can also be thought of as performing *model merging* (as in (Elidan and Friedman, 2001; Cheeseman *et al.*, 1988)) in a simple probabilistic model.

In the initialization phase, we do not yet have a learned structure for the different modules. Thus, from a clustering perspective, we consider a simple naive Bayes model for each cluster, where the distributions over the different features within each cluster are independent and have a separate parameterization. We begin by forming a cluster for each variable, and then merge two clusters whose probabilistic models over the features (instances) are similar.

¿From a module network perspective, the naive Bayes model can be obtained by introducing a dummy variable U that encodes training instance identity — $u[m] = m$ for all m. Throughout our clustering process, each module will have $\mathbf{Pa}_{\mathbf{M}_i} = \{U\}$, providing exactly the effect that, for each variable X_i, the different values $x_i[m]$ have separate probabilistic models. We then begin by creating n modules, with $\mathcal{A}(X_i) = i$. In this module network, each instance and each variable has its own local probabilistic model.

We then consider all possible legal module mergers (those corresponding to modules with the same domain), where we change the assignment function to replace two modules j_1 and j_2 by a new module $j_{1,2}$. This step corresponds to creating a cluster containing the variables X_{j_1} and X_{j_2}. Note that, following the merger, the two variables X_{j_1} and X_{j_2} now must share parameters, but each instance still has a different probabilistic model (enforced by the dependence on the instance ID U). We evaluate each such merger by computing the score of the resulting module network. Thus, the procedure will merge two modules that are similar to each other across the different instances. We continue to do these merge steps until we construct a module network with the desired number of modules, as specified in the original choice of C.

4.2.3 MODULE REASSIGNMENT

In the module reassignment step, the task is more complex. We now have a given structure S, and wish to find $\mathcal{A} = \mathrm{argmax}_{\mathcal{A}'}\mathrm{score}_{\mathbf{M}}(S, \mathcal{A}' : \mathcal{D})$. We thus wish to take each variable X_i, and select the assignment $\mathcal{A}(X_i)$ that provides the highest score.

At first glance, we might think that we can decompose the score across variables, allowing us to determine independently the optimal assignment $\mathcal{A}(X_i)$ for each variable X_i. Unfortunately, this is not the case. Most obviously, the assignments to different variables must be constrained so that the module graph remains acyclic. For example, if $X_1 \in \mathbf{Pa}_{\mathbf{M}_i}$ and $X_2 \in \mathbf{Pa}_{\mathbf{M}_j}$, we cannot simultaneously assign $\mathcal{A}(X_1) = j$ and $\mathcal{A}(X_2) = i$. More subtly, the Bayesian score for each module depends non-additively on the sufficient statistics of all the variables assigned to the module. (The log-likelihood function is additive in the sufficient statistics of the different variables, but the log marginal likelihood is not.) Thus, we can only compute the delta score for moving a variable from one module to another given a *fixed* assignment of the other variables to these two modules.

We therefore use a sequential update algorithm that reassigns the variables to modules one by one. The idea is simple. We start with an initial assignment function $\mathcal{A}^0$, and in a "round-robin" fashion iterate over all of the variables one at a time, and consider changing their module assignment. When considering a reassignment for a variable X_i, we keep the assignments of all other variables fixed and find the optimal legal (acyclic) assignment for X_i relative to the fixed assignment. We continue reassigning variables until no single reassignment can improve the score. An outline of this algorithm appears in Figure 3

The key to the correctness of this algorithm is its sequential nature: Each time a variable assignment changes, the assignment function as well as the associated sufficient statistics are updated before evaluating another variable. Thus, each change made to the assignment function leads to a legal assignment which improves the score. Our algorithm terminates when it can no longer improve the score. Hence, it converges to a local maximum, in the sense that no single assignment change can improve the score.

The computation of the score is the most expensive step in the sequential algorithm. Once again, the decomposition of the score plays a key role in reducing the complexity of this computation:

Input:
    D // Data set
    K // Number of modules
Output:
    $\mathbf{M}$ // A module network
Learn-Module-Network
    $\mathcal{A}_0 =$ cluster $\mathcal{X}$ into K modules
    $\mathcal{S}_0 =$ empty structure
    Loop $t = 1, 2, \ldots$ until convergence
        $\mathcal{S}_t =$ Greedy-Structure-Search$(\mathcal{A}_{t-1}, \mathcal{S}_{t-1})$
        $\mathcal{A}_t =$ Sequential-Update$(\mathcal{A}_{t-1}, \mathcal{S}_t)$;
    Return $\mathbf{M} = (\mathcal{A}_t, \mathcal{S}_t)$

Figure 4: Outline of the *module network* learning algorithm. Greedy-Structure-Search successively applies operators that change the structure as long as each such operator results in a legal structure and improves the module network score

When reassigning a variable X_i from one module $\mathbf{M}_{old}$ to another $\mathbf{M}_{new}$, only the local scores of these modules change. The module score of all other modules remains unchanged. The rescoring of these two modules can be accomplished efficiently by subtracting X_i's statistics from the sufficient statistics of $\mathbf{M}_{old}$ and adding them to those of $\mathbf{M}_{new}$. Thus, assuming that we have precomputed the sufficient statistics associated with every pair of variable X_i and module $\mathbf{M}_j$, the cost of recomputing the delta-score for an operator is $O(s)$, where s is the size of the table of sufficient statistics for a module. The only operators whose delta-scores change are those involving reassignment of variables to/from these two modules. Assuming that each module has approximately $O(n/K)$ variables, and we have at most K possible destinations for reassigning each variable, the total number of such operators is generally linear in n. Thus, the cost of each reassignment step is approximately $O(ns)$. In addition, at the beginning of the module reassignment step, we must initialize all of the sufficient statistics at a cost of $O(Mnd)$, and compute all of the delta-scores at a cost of $O(nK)$.

4.3 Algorithm Summary

To summarize, our algorithm starts with an initial assignment of variables to modules. In general, this initial assignment can come from anywhere, and may even be a random guess. We choose to construct it using the clustering-based idea described in the previous section. The algorithm then iteratively applies the two steps described above: learning the module dependency structures, and reassigning variables to modules. These two steps are repeated until convergence, where convergence is defined by a score improvement of less than some fixed threshold Δ between two consecutive learned models. An outline of the module network learning algorithm is shown in Figure 4.

Each of these two steps — structure update and assignment update — is guaranteed to either improve the score or leave it unchanged. The following result therefore follows immediately:

Theorem 4.1: The iterative module network learning algorithm converges to a local maximum of score$(\mathcal{S}, \mathcal{A} : \mathcal{D})$.
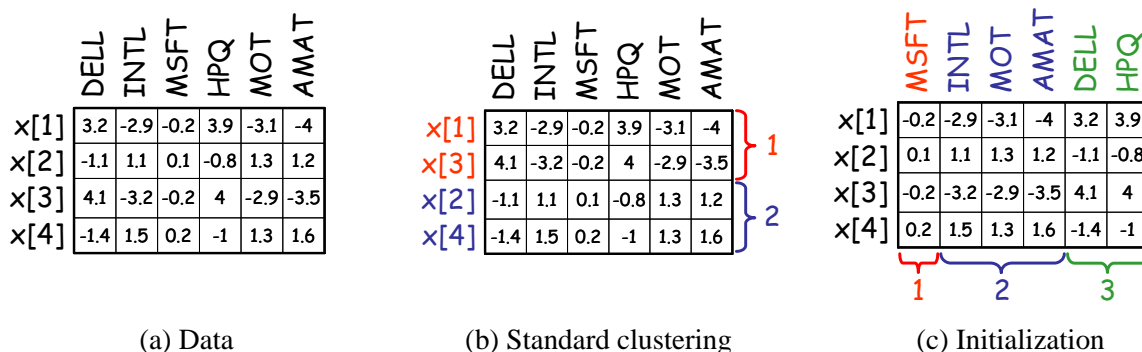
	DELL	INTL	MSFT	HPQ	MOT	AMAT
x[1]	3.2	-2.9	-0.2	3.9	-3.1	-4
x[2]	-1.1	1.1	0.1	-0.8	1.3	1.2
x[3]	4.1	-3.2	-0.2	4	-2.9	-3.5
x[4]	-1.4	1.5	0.2	-1	1.3	1.6

(a) Data

	DELL	INTL	MSFT	HPQ	MOT	AMAT	
x[1]	3.2	-2.9	-0.2	3.9	-3.1	-4	1
x[3]	4.1	-3.2	-0.2	4	-2.9	-3.5	
x[2]	-1.1	1.1	0.1	-0.8	1.3	1.2	2
x[4]	-1.4	1.5	0.2	-1	1.3	1.6	

(b) Standard clustering

	MSFT	INTL	MOT	AMAT	DELL	HPQ
x[1]	-0.2	-2.9	-3.1	-4	3.2	3.9
x[2]	0.1	1.1	1.3	1.2	-1.1	-0.8
x[3]	-0.2	-3.2	-2.9	-3.5	4.1	4
x[4]	0.2	1.5	1.3	1.6	-1.4	-1

| 1 | 2 | 3 |

(c) Initialization

Figure 5: Relationship between the module network procedure and clustering. Finding an assignment function can be viewed as a clustering of the variables whereas clustering typically clusters instances. Shown is sample data for the example domain of Figure 1, where the rows correspond to instances and the columns correspond to variables. (a) Data. (b) Standard clustering of the data in (a). Note that $x[2]$ and $x[3]$ were swapped to form the clusters. (c) Initialization of the assignment function for the module network procedure for the data in (a). Note that variables were swapped in their location to reflect the initial assignment into three modules.

We note that both the structure search step and the module reassignment step are done using simple greedy hill-climbing operations. As in other settings, this approach is liable to get stuck in local maxima. We attempt to somewhat compensate for this limitation by initializing the search at a reasonable starting point, but local maxima are clearly still an issue. An additional strategy that would help circumvent some maxima is the introduction of some randomness into the search (e.g., by random restarts or simulated annealing), as is often done when searching complex spaces with multi-modal target functions.

5. Learning with Regression Trees

We now briefly review the family of conditional distributions we use in the experiments below. Many of the domains suited for module network models contain continuous valued variables, such as gene expression or price changes in the stock market. For these domains, we often use a conditional probability model represented as a *regression tree* (Breiman *et al.*, 1984). For our purposes, a regression tree T for $P(X \mid \mathbf{U})$ is defined via a rooted binary tree, where each *node* in the tree is either a *leaf* or an *interior node*. Each interior node is labeled with a test $U < u$ on some variable $U \in \mathbf{U}$ and $u \in \mathbb{R}$. Such an interior node has two outgoing *arcs* to its children, corresponding to the outcomes of the test (true or false). The tree structure T captures the *local* dependency structure of the conditional distribution. The parameters of T are the distributions associated with each leaf. In our implementation, each leaf ℓ is associated with a univariate Gaussian distribution over values of X, parameterized by a mean μ_ℓ and variance σ_ℓ^2. An example of a regression tree CPD is shown in Figure 6. We note that, in some domains, Gaussian distributions may not be the appropriate choice of models to assign at the leaves of the regression tree. In such cases, we can apply transforma-
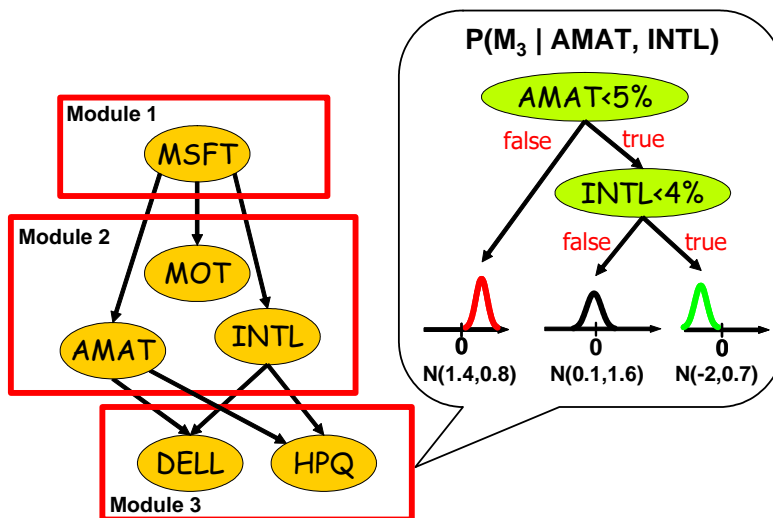
Figure 6: Example of a regression tree with univariate Gaussian distributions at the leaves for representing the CPD $P(\mathbf{M}_3 \mid AMAT, INTL)$, associated with $\mathbf{M}_3$. The tree has internal nodes labeled with a test on the variable (e.g. $AMAT < 5\%$). Each univariate Gaussian distribution at a leaf is parameterized by a mean and a variance. The tree structure captures the local dependency structure of the conditional distributions. In the example shown, when $AMAT \geq 5\%$, then the distribution over values of variables assigned to $\mathbf{M}_3$ will be Gaussian with mean 1.4 and standard deviation 0.8 regardless of the value of $INTL$.

tions to the data to make it more appropriate for modeling by Gaussian distributions, or use other continuous or discrete distributions at the leaves.

To learn module networks with regression-tree CPDs, we must extend our previous discussion by adding another component to $\mathcal{S}$ that represents the trees $T_1, \ldots, T_K$ associated with the different modules. Once we specify these components, the above discussion applies with several small differences. These issues are similar to those encountered when introducing decision trees to Bayesian networks (Chickering *et al.*, 1997; Friedman and Goldszmidt, 1998), so we discuss them only briefly.

Given a regression tree T_j for $P(\mathbf{M}_j \mid \mathbf{Pa}_{\mathbf{M}_j})$, the corresponding sufficient statistics are the statistics of the distributions at the leaves of the tree. For each leaf ℓ in the tree, and for each data instance $\mathbf{x}[m]$, we let $\ell_j[m]$ denote the leaf reached in the tree given the assignment to $\mathbf{Pa}_{\mathbf{M}_j}$ in $\mathbf{x}[m]$. The module likelihood decomposes as a product of terms, one for each leaf ℓ. Each term is the likelihood for the Gaussian distribution $\mathcal{N}\left(\mu_\ell; \sigma_\ell^2\right)$, with the usual sufficient statistics for a Gaussian distribution.

Given a regression tree T_j for $P(\mathbf{M}_j \mid \mathbf{Pa}_{\mathbf{M}_j})$, the corresponding sufficient statistics are the statistics of the distributions at the leaves of the tree. For each leaf ℓ in the tree, and for each data instance $\mathbf{x}[m]$, we let $\ell_j[m]$ denote the leaf reached in the tree given the assignment to $\mathbf{Pa}_{\mathbf{M}_j}$ in $\mathbf{x}[m]$. The module likelihood decomposes as a product of terms, one for each leaf ℓ. Each term is the likelihood for

the Gaussian distribution $\mathcal{N}\left(\mu_\ell; \sigma_\ell^2\right)$, with the sufficient statistics for a Gaussian distribution.

$$
\begin{aligned}
\hat{S}_{j,\ell}^0 &= \sum_m \sum_{X_i \in \mathbf{X}^j} \eta\{\ell_j[m] = \ell\}, \\
\hat{S}_{j,\ell}^1 &= \sum_m \sum_{X_i \in \mathbf{X}^j} \eta\{\ell_j[m] = \ell\}x_i, \\
\hat{S}_{j,\ell}^2 &= \sum_m \sum_{X_i \in \mathbf{X}^j} \eta\{\ell_j[m] = \ell\}x_i^2.
\end{aligned}
\tag{5}
$$

The local module score further decomposes into independent components, one for each leaf ℓ. Here, we use a Normal-Gamma prior (DeGroot, 1970) for the distribution at each leaf: Letting $\tau_\ell = 1/\sigma_\ell^2$ stand for the precision at leaf ℓ, we define: $P(\mu_\ell, \tau_\ell) = P(\mu_\ell \mid \tau_\ell)P(\tau_\ell)$, where $P(\tau_\ell) \sim \Gamma(\alpha_0, \beta_0)$ and $P(\mu_\ell \mid \tau_\ell) \sim \mathcal{N}\left(\mu_0; (\lambda_0 \tau_\ell)^{-1}\right)$, where we assume that all leaves are associated with the same prior. Letting $\hat{S}_{j,\ell}^i$ be defined as in Equation (5), we have that the component of the log marginal likelihood associated with a leaf ℓ of module j is given by:

$$
-\frac{1}{2}\hat{S}_{j,\ell}^0 \log(2\pi) + \frac{1}{2}\log\left(\frac{\lambda_0}{\lambda_0 + \hat{S}_{j,\ell}^0}\right) + \log\left(\Gamma(\alpha_0 + \frac{1}{2}\hat{S}_{j,\ell}^0)\right)
$$
$$
-\log\left(\Gamma(\alpha_0)\right) + \alpha_0 \log(\beta_0) - \left(\alpha_0 + \frac{1}{2}\hat{S}_{j,\ell}^0\right)\log(\beta),
$$

where

$$
\beta = \beta_0 + \frac{1}{2}\left(\hat{S}_{j,\ell}^2 - \frac{(\hat{S}_{j,\ell}^1)^2}{\hat{S}_{j,\ell}^0}\right) + \frac{\hat{S}_{j,\ell}^0 \lambda_0 \left(\frac{\hat{S}_{j,\ell}^1}{\hat{S}_{j,\ell}^0} - \mu_0\right)^2}{2(\lambda_0 + \hat{S}_{j,\ell}^0)}.
$$

When performing structure search for module networks with regression-tree CPDs, in addition to choosing the parents of each module, we must also choose the associated tree structure. We use the search strategy proposed by Chickering *et al.* (1997), where the search operators are leaf splits. Such a *split* operator replaces a leaf in a tree T_j with an internal node with some test on a variable U. The two branches below the newly created internal node point to two new leaves, each with its associated Gaussian. This operator must check for acyclicity, as it implicitly adds U as a parent of $\mathbf{M}_j$.

When performing the search, we consider splitting each possible leaf on each possible parent U and each value u. As always in regression-tree learning, we do not have to consider all real values u as possible split points; it suffices to consider values that arise in the data set. Moreover, under an appropriate choice of prior (i.e., an independent prior for each leaf), regression-tree learning provides another level of score decomposition: The score of a particular tree is a sum of scores for the leaves in the tree. Thus, a split operation on one leaf in the tree does not affect the score component of another leaf, so that operators applied to other leaves do not need to re-evaluated.

6. Experimental Results

We evaluated our module network learning procedure on synthetic data and on two real data sets — gene expression data, and stock market data. In all cases, our data consisted solely of continuous values. As all of the variables have the same domain, the definition of the module set reduces simply
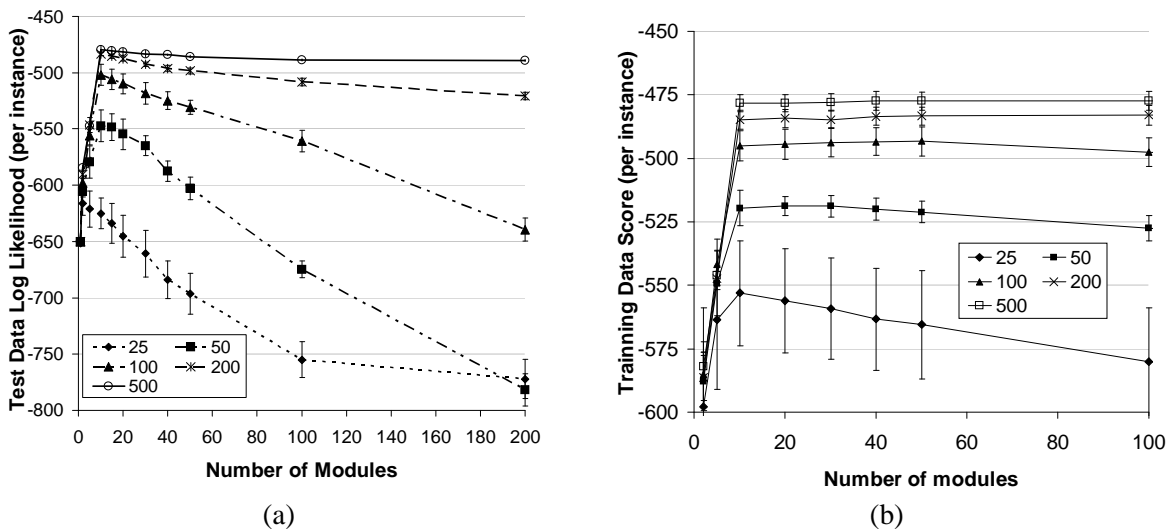
Figure 7: Performance of learning from synthetic data as a function of the number of modules and training set size. The *x*-axis corresponds to the number of modules, each curve corresponds to a different number of training instances, and each point shows the mean and standard deviations from the 10 sampled data sets. (a) Log-likelihood per instance assigned to held-out data. (b) Average score per instance on the training data.

to a specification of the total number of modules. We used regression trees as the local probability model for all modules, and uniform priors for $\rho(\mathcal{S})$ and $\kappa(\mathcal{A})$. For structure search, we used beam search, using a lookahead of three splits to evaluate each operator. When learning Bayesian networks, as a comparison, we used precisely the same structure learning algorithm, simply treating each variable as its own module.

6.1 Synthetic Data

As a basic test of our procedure in a controlled setting, we used synthetic data generated by a known module network. This gives a known ground truth to which we can compare the learned models. To make the data realistic, we generated synthetic data from a model that was learned from the gene expression data set described below. The generating model had 10 modules and a total of 35 variables that were a parent of some module. From the learned module network, we selected 500 variables, including the 35 parents. We tested our algorithm's ability to reconstruct the network using different numbers of modules; this procedure was run for training sets of various sizes ranging from 25 instances to 500 instances, each repeated 10 times for different training sets.

We first evaluated the generalization to unseen test data, measuring the likelihood ascribed by the learned model to 4500 unseen instances. The results, summarized in Figure 7(a), show that, for all training set sizes, except the smallest one with 25 instances, the model with 10 modules performs the best. As expected, models learned with larger training sets do better; but, when run using the correct number of 10 modules, the gain of increasing the number of data instances beyond 100 samples is small and beyond 200 samples is negligible.
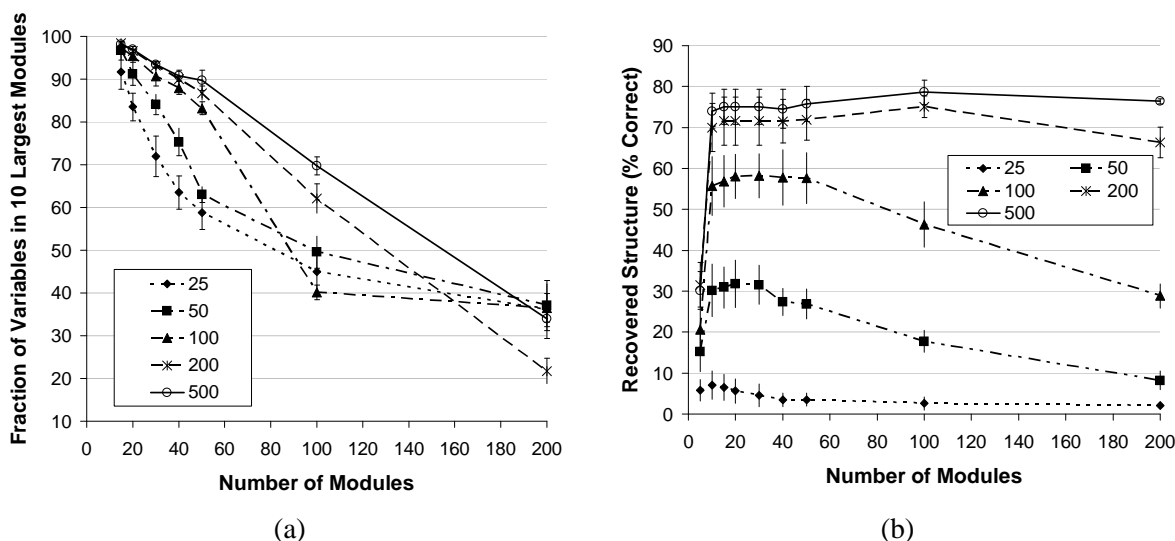
575

Figure 8: (a) Fraction of variables assigned to the 10 largest modules. (b) Average percentage of correct parent-child relationships recovered (fraction of parent-child relationships in the true model recovered in the learned model) when learning from synthetic data for models with various number of modules and different training set sizes. The *x*-axis corresponds to the number of modules, each curve corresponds to a different number of training instances, and each point shows the mean and standard deviations from the 10 sampled data sets.

To test whether we can use the score of the model to select the number of modules, we also plotted the score of the learned model on the training data (Figure 7(b)). As can be seen, when the number of instances is small (25 or 50), the model with 10 modules achieves the highest score and for a larger number of instances, the score does not improve when increasing the number of modules beyond 10. Thus, these results suggest that we can select the number of modules by choosing the model with the smallest number of modules from among the highest scoring models.

A closer examination of the learned models reveals that, in many cases, they are almost a 10-module network. As shown in Figure 8(a), models learned using 100, 200, or 500 instances and up to 50 modules assigned ≥ 80% of the variables to 10 modules. Indeed, these models achieved high performance in Figure 7(a). However, models learned with a larger number of modules had a wider spread for the assignments of variables to modules and consequently achieved poor performance.

Finally, we evaluated the model's ability to recover the correct dependencies. The total number of parent-child relationships in the generating model was 2250. For each model learned, we report the fraction of correct parent-child relationships it contains. As shown in Figure 8(b), our procedure recovers 74% of the true relationships when learning from a data set with 500 instances. Once again, we see that, as the variables begin fragmenting over a large number of modules, the learned structure contains many spurious relationships. Thus, our results suggest that, in domains with a modular structure, statistical noise is likely to prevent overly detailed learned models such as Bayesian networks from extracting the commonality between different variables with a shared behavior.

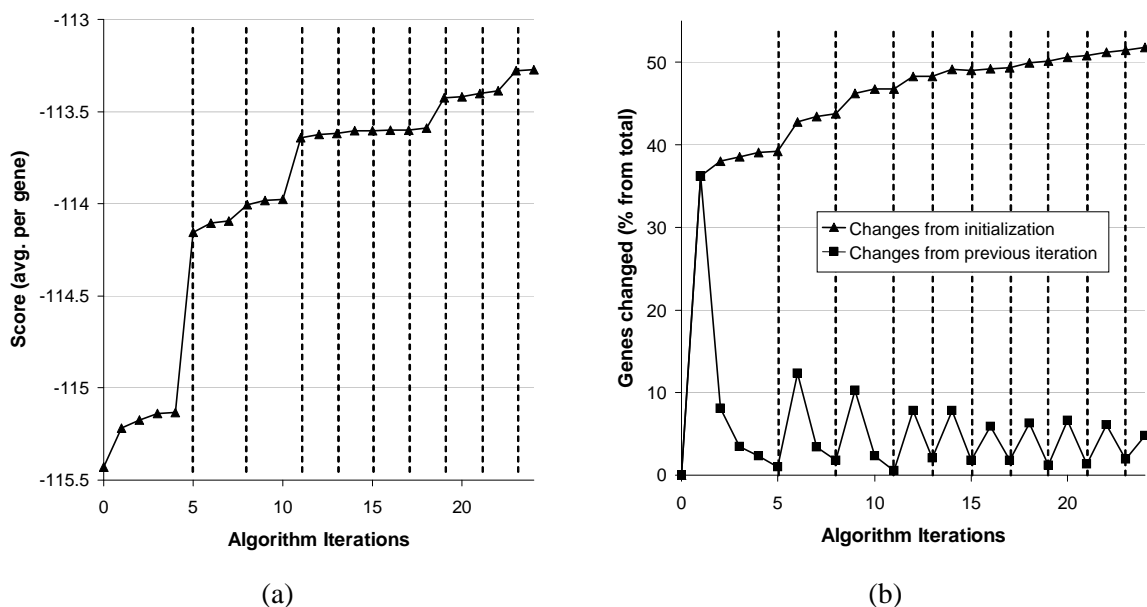(a)   (b)

Figure 9: (a) Score of the model (normalized by the number of variables/genes) across the iterations of the algorithm for a module network learned with 50 modules on the gene expression data. Iterations in which the structure was changed are indicated by dashed vertical lines. (b) Changes in the assignment of genes to modules for the module network learned in (a) across the iterations of the algorithm. Shown are both the total changes compared to the initial assignment (triangles) and the changes compared to the previous iteration (squares).

6.2 Gene Expression Data

We next evaluated the performance of our method on a real world data set of gene expression measurements. A *microarray* measures the activity level (mRNA expression level) of thousands of genes in the cell in a particular condition. We view each experiment as an instance, and the expression level of each measured gene as a variable (Friedman *et al.*, 2000a). In many cases, the coordinated activity of a group of genes is controlled by a small set of *regulators*, that are themselves encoded by genes. Thus, the activity level of a regulator gene can often predict the activity of the genes in the group. Our goal is to discover these modules of co-regulated genes, and their regulators.

We used the expression data of Gasch *et al.* (et al., 2000), which measured the response of yeast to different stress conditions. The data consists of 6157 genes and 173 experiments. In this domain, we have prior knowledge of which genes are likely to play a regulatory role (e.g., based on properties of their protein sequence). Consequently, we restricted the possible parents to 466 yeast genes that may play such a role. We then selected 2355 genes that varied significantly in the data and learned a module network over these genes. We also learned a Bayesian network over this data set.
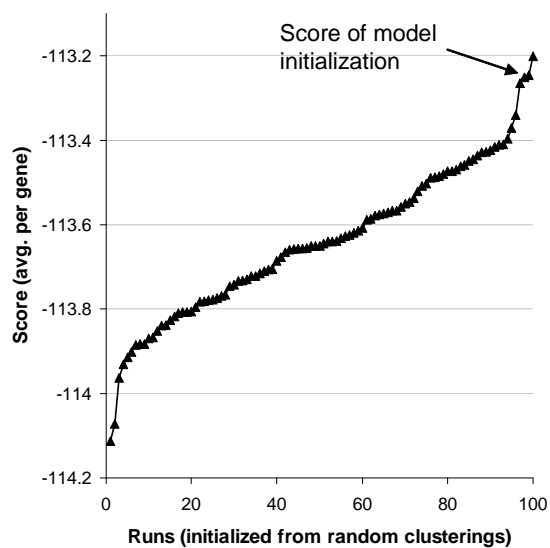
Figure 10: Score of 100 module networks (normalized by the number of variables/genes) each learned with 50 modules from a random clustering initialization, where the runs are sorted according to their score. The score of a module network learned using the deterministic clustering initialization described in Section 4.2 is indicated by a pointed arrow.

6.2.1 STATISTICAL EVALUATION

We first examined the behavior of the learning algorithm on the training data when learning a module network with 50 modules. This network converged after 24 iterations (of which nine were iterations in which the structure of the network changed). To characterize the trajectory of the algorithm, we plot in Figure 9 its improvement across the iterations, measured as the score on the training data, normalized by the number of genes (variables). To obtain a finer-grained picture, we explicitly show structure learning steps, as well as each pass over the variables in the module reassignment step. As can be seen in Figure 9(a), the model score improves nicely across these steps, with the largest gains in score occurring in iterations in which the structure was changed (dotted lines in Figure 9(a)). Figure 9(b) demonstrates how the algorithm changes the assignments of genes to modules, with 1221 of the 2355 (51.8%) genes changing their assignment upon convergence, and the largest assignment changes occurring immediately after structure modification steps.

As for most local search algorithms, initialization is an key component: A bad initialization can cause the algorithm to get trapped in a poor local maximum. As we discussed in Section 4.2, we initialize the assignment function using a clustering program. The advantage of a simple deterministic initialization procedure is that it is computationally efficient, and results in reproducible behavior. We evaluated this proposed initialization by comparing the results to module networks initialized randomly. We generated 100 random assignments of variables to modules, and learned a module network starting from each initialization. We compared the model score of the network learned using our deterministic initialization, and the 100 networks initialized randomly. A plot of

these sorted scores is shown in Figure 10. Encouragingly, the score for the network initialized using our procedure was better than 97/100 of the runs initialized from random clusters, and the 3/100 runs that did better are only incrementally better.

We evaluated the generalization ability of different models, in terms of log-likelihood of test data, using 10-fold cross validation. In Figure 11(a), we show the difference between module networks of different size and the baseline Bayesian network, demonstrating that module networks generalize much better to unseen data for almost all choices of number of modules.

6.2.2 BIOLOGICAL EVALUATION

As we discussed in the introduction, a common goal in learning a network structure is to reveal structural properties of the underlying distribution. This goal is definitely an important one in the biological domain, where we want to discover both sets of co-regulated genes, and the regulatory mechanism governing their behavior. We therefore evaluated the ability of our module network learning procedure to reveal known biological properties of this domain.

We evaluated a learned module network with 50 modules, where we selected 50 modules due to the biological plausibility of having, on average, 40–50 genes per module. First, we examined whether genes in the same module have shared functional characteristics. To this end, we used annotations of the genes' biological functions from the Saccharomyces Genome Database (Cherry *et al.*, 1998). We systematically evaluated each module's gene set by testing for significantly enriched annotations. Suppose we find l genes with a certain annotation in a module of size N. To check for enrichment, we calculate the *hypergeometric p-value* of these numbers — the probability of finding that many genes of that annotation in a random subset of N genes. For example, the "protein folding" module contains 10 genes, 7 of which are annotated as protein folding genes. In the whole data set, there are only 26 genes with this annotation. The p-value of this annotation, that is, the probability of choosing 7 or more genes in this category by choosing 10 random genes, is less than 10^{-12}. As there are a large number of possible annotations, there is a nontrivial probability that some will be enriched simply by chance. We therefore corrected these p-values using the standard Bonferroni correction for independent multiple hypotheses (Savin, 1980). Our evaluation showed that, of the 50 modules, 42 (resp. 20) modules had at least one significantly enriched annotation with a p-value less than 0.005 (resp. less than 10^{-6}). Furthermore, the enriched annotations reflect the key biological processes expected in our data set. We used these annotations to label the modules with meaningful biological names. A comparison of the overall enrichments of the modules learned by module networks to the enrichments obtained for clusters using AutoClass is shown in Figure 11(b), indicating that there are many annotations that are much more significantly enriched in module networks.

We can use these annotations to reason about the dependencies between different biological processes at the module level. For example, we find that the *cell cycle* module, regulates the *histone* module. The cell cycle is the process in which the cell replicates its DNA and divides, and it is indeed known to regulate histones — key proteins in charge of maintaining and controlling the DNA structure. Another module regulated by the cell cycle module is the *nitrogen catabolite repression (NCR)* module, a cellular response activated when nitrogen sources are scarce. We find that the *NCR* module regulates the *amino acid metabolism*, *purine metabolism* and *protein synthesis* modules, all representing nitrogen-requiring processes, and hence likely to be regulated by the *NCR* module.

(a) Test-data generalization (Expression)   (b) Annotation enrichment (Expression)
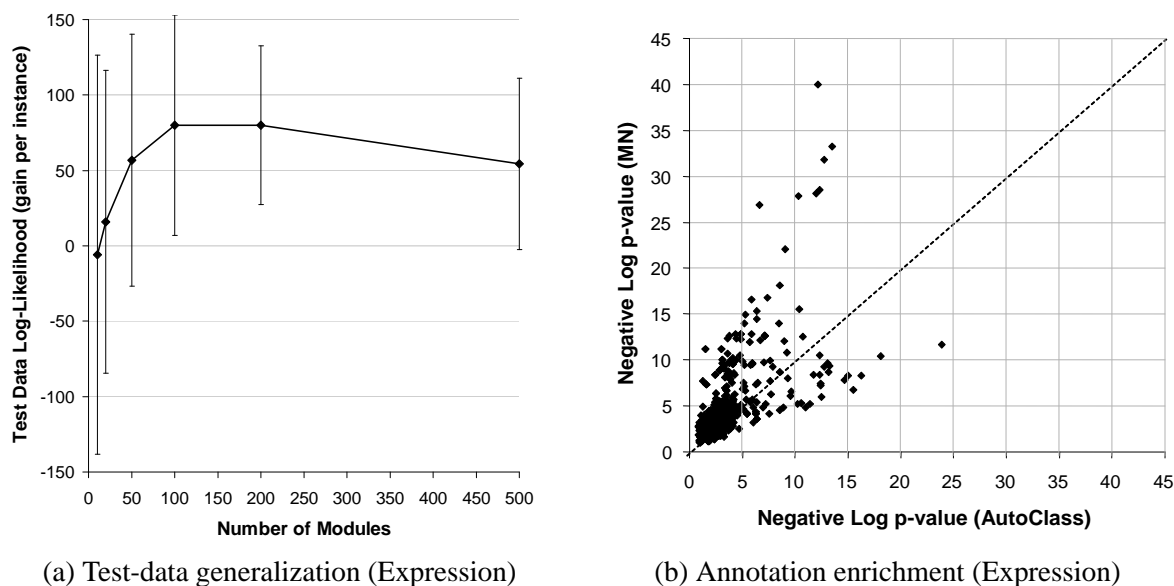
Figure 11: (a) Comparison of generalization ability of module networks learning with different numbers of modules on the gene expression data set. The *x*-axis denotes the number of modules. The *y*-axis denotes the difference in log-likelihood on held out data between the learned module network and the learned Bayesian network, averaged over 10 folds; the error bars show the standard deviation. (b) Comparison of the enrichment for annotations of functional annotations between the modules learned using the module network procedure and the clusters learned by the AutoClass clustering algorithm (Cheeseman *et al.*, 1988) applied to the variables. Each point corresponds to an annotation, and the *x* and *y* axes are the negative log *p*-values of its enrichment for the two models.

These examples demonstrate the insights that can be gleaned from a higher order model, and which would have been obscured in the unrolled Bayesian network over 2355 genes.

6.3 Stock Market Data

In a very different application, we examined a data set of NASDAQ stock prices. We collected stock prices for 2143 companies, in the period 1/1/2002–2/3/2003, covering 273 trading days (data was obtained from http://finance.yahoo.com). We took each stock to be a variable, and each instance to correspond to a trading day, where the value of the variable is the log of the ratio between that day's and the previous day's closing stock price. This choice of data representation focuses on the relative changes to the stock price, and eliminates the magnitude of the price itself (which depends on such irrelevant factors as the number of outstanding shares). As potential controllers, we selected 250 of the 2143 stocks, whose average trading volume was the largest across the data set.

As with gene expression data, we used cross validation to evaluate the generalization ability of different models. As we can see in Figure 12(a), module networks perform significantly better than Bayesian networks in this domain.
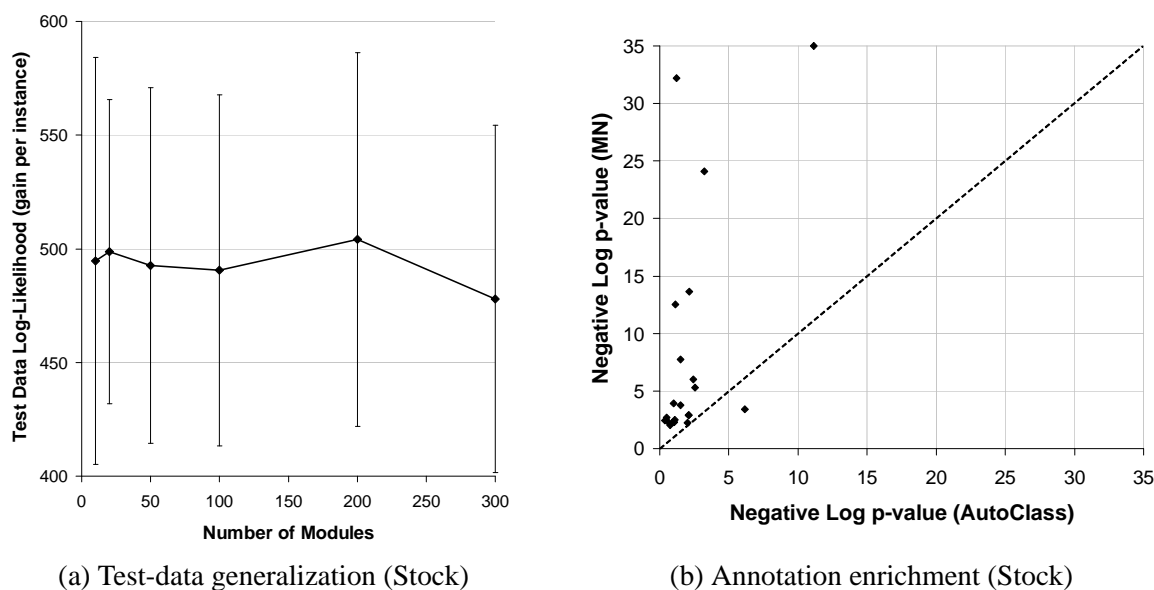
(a) Test-data generalization (Stock)　　　　　(b) Annotation enrichment (Stock)

Figure 12: (a) Comparison of generalization ability of module networks learning with different numbers of modules on the stock data set. The *x*-axis denotes the number of modules. The *y*-axis denotes the difference in log-likelihood on held out data between the learned module network and the learned Bayesian network, averaged over 10 folds; the error bars show the standard deviation. (b) Comparison of the enrichment for annotations of sectors between the modules learned using the module network procedure and the clusters learned by the AutoClass clustering algorithm (Cheeseman *et al.*, 1988) applied to the variables. Each point corresponds to an annotation, and the *x* and *y* axes are the negative log *p*-values of its enrichment for the two models.

To test the quality of our modules, we measured the enrichment of the modules in the network with 50 modules for annotations representing various sectors to which each stock belongs (based on sector classifications from http://finance.yahoo.com). We found significant enrichment for 21 such annotations, covering a wide variety of sectors. We also compared these results to the clusters of stocks obtained from applying the popular probabilistic clustering algorithm AutoClass (Cheeseman *et al.*, 1988) to the data. Here, as we described above, each instance corresponds to a stock and is described by 273 random variables, each representing a trading day. In 20 of the 21 cases, the enrichment was far more significant in the modules learned using module networks compared to the one learned by AutoClass, as can be seen in Figure 12(b).

Finally, we also looked at the structure of the module network, and found several cases where the structure fit our (limited) understanding of the stock domain. Several modules corresponded primarily to high tech stocks. One of these, consisting mostly of software, semi-conductor, communication, and broadcasting services, had as its two main predictors Molex, a large manufacturer of electronic, electrical and fiber optic interconnection products and systems, and Atmel, specializing in design, manufacturing and marketing of advanced semiconductors. Molex was also the parent for another module, consisting primarily of software, semi-conductor, and medical equip-

ment companies; this module had as additional parents Maxim, which develop integrated circuits, and Affymetrix, which designs and develops gene microarray chips. In this, as in many other cases, the parents of a module are from similar sectors as the stocks in the module.

7. Related Work

Module networks are related to several other approaches, including plates Buntine (1994), hierarchical Bayesian models DeGroot (1970), *object-oriented Bayesian networks* (OOBNs) (Koller and Pfeffer, 1997) and to the framework of *probabilistic relational models* (PRMs) (Koller and Pfeffer, 1998; Friedman *et al.*, 1999a).

Both plates and hierarchical Bayesian approaches allow us to represent models where objects in the same class share parameters. Plate models also allow objects to share the same parent set. In many ways, they allow a more expressive dependency structure than module networks, as they allow a richly structured hierarchical set of variables, determined by the nested plate structure. However, variables in one plate can only depend on variables in an enclosing plate. Thus, plate models are not sufficiently expressive to encode the inter-module dependencies in a module-network. Hierarchical Bayesian models are more expressive than module networks in that they allow parameters of different variables to be statistically related but not necessarily equal. However, hierarchical Bayesian approaches are not a language that includes structure as well as parameters, so that an additional representation layer would have to be added to provide a framework similar to module networks. One can easily extend module networks with ideas from the hierarchical Bayesian framework, allowing the parameters of different variables in the same module to be correlated but not necessarily equal. Most importantly, neither plates nor the hierarchical Bayesian framework have provided a method that allows us to learn automatically which subsets of variables share parameters.

OOBNs and PRMs extend Bayesian Networks to a setting involving multiple related objects, and allow the attributes of objects of the same class to share parameters and dependency structure. One can view the module network framework as a restriction of these frameworks, where we have one object for every variable X_i, with a single attribute corresponding to the value of X_i. Each module can be viewed as a class, so that the variables in a single module share the same probabilistic model. As the module assignments are not known in advance, module networks correspond most closely to the variant of these frameworks where there is *type uncertainty* — uncertainty about the class assignment of objects. However, despite this high-level similarity, the module network framework differs in certain key points from both OOBNs and PRMs, with significant impact on the learning task.

In OOBNs, objects in the same class must have the same internal structure and parameterization, but can depend on different sets of variables (as specified in the mapping of variables in an object's interface to its actual inputs). By contrast, in a module network, all of the variables in a module (class) must have the same specific parents. This assumption greatly reduces the size and complexity of the hypothesis space, leading to a more robust learning algorithm. On the other hand, this assumption requires that we be careful in making certain steps in the structure search, as they have more global effects than on just one or two variables. Due to these differences, we cannot simply apply an OOBN structure-learning algorithm, such as the one proposed by Langseth and Nielsen (2003), to such complex, high-dimensional domains.

In PRMs, the probabilistic dependency structure of the objects in a class is determined by the relational structure of the domain (e.g., the *Cost* attribute of a particular car object might depend on

the *Income* attribute of the object representing this particular car's owner). In the case of module networks, there is no known relational structure to which probabilistic dependencies can be attached. Without such a relational structure, PRMs only allow dependency models specified at the class level. Thus, we can assert that the objects in one class depend on some aggregate quantity of the objects in another. We cannot, however, state a dependence on a particular object in the other class (without some relationship specified in the model). Getoor *et al.* (2000) attempt to address this issue using a class hierarchy. Their approach is very different from ours, requiring some fairly complex search steps, and is not easily applied to the types of domains considered in this paper.

To better relate the PRM approach to module networks, recall the relationship between module networks and clustering, as described in Section 4.2. As we discussed, we can view the module network learning procedure as grouping variables into clusters that share the same probabilistic dependency model. As shown in Figure 5, we are taking the data points in the (variables *x* instances) matrix, and grouping rows. As we discussed, in other settings, we often group columns (instances). In fact, in many cases, the notion of "variables" and "instances" is somewhat arbitrary. PRMs allow us to define a probabilistic model where the value of a data point depends both on properties of the rows and properties of the column. In particular, we can define a hidden attribute for either rows, columns, or both; the values of this hidden attribute would correspond to a clustering of rows, or columns, or a two-sided clustering of both rows and columns simultaneously (see Segal *et al.* (2001)).

From this perspective, the module network framework can be viewed as being closely related to a PRM where the module assignment is a hidden attribute of a row. For example, in the gene expression domain, the expression value of gene g_i in microarray a_j depends on attributes both of g_i and of a_j. The gene g_i only has one attribute, representing its module assignment. The array a_j has attributes representing the expression levels of the different regulators in the array. The expression level of gene g_i in experiment a_j then depends on all of these attributes, i.e., on the gene's module assignment and on the values of the regulators. A key difference between the PRM-based approach and our module network framework is that, in the PRM, the regulators cannot themselves participate in the probabilistic model without leading to cycles. This restriction forces us to select a relatively small set of candidate regulators in advance. Moreover, as no probabilistic dependency model is learned for regulators, this approach cannot discover compound regulatory pathways, which are often of great interest.

Overall, the module network framework places strong restrictions on the richness of the set of objects and on the dependency structures that can be represented. However, these restrictions allow us to formulate a reasonably effective algorithm for learning which variables share parameters. Although it is possible to define such algorithms for the rich representation frameworks such as plates, OOBNs, or PRMs, it remains to be seen whether such algorithms can perform effectively, given that the much larger search space can introduce both computational problems and problems related to overfitting.

8. Discussion and Conclusions

We have introduced the framework of *module networks*, an extension of Bayesian networks that includes an explicit representation of *modules* — subsets of variables that share a statistical model. We have presented a Bayesian learning framework for module networks, which learns both the partitioning of variables into modules and the dependency structure of each module. We showed

experimental results on two complex real-world data sets, each including measurements of thousands of variables, in the domains of gene expression and stock market. Our results show that our learned module networks have much higher generalization performance than a Bayesian network learned from the same data.

There are several reasons why a learned module network is a better model than a learned Bayesian network. Most obviously, parameter sharing between variables in the same module allows each parameter to be estimated based on a much larger sample. Moreover, this allows us to learn dependencies that are considered too weak based on statistics of single variables. These are well-known advantages of parameter sharing; the interesting aspect of our method is that we determine automatically which variables share parameters.

More interestingly, the assumption of shared structure significantly restricts the space of possible dependency structures, allowing us to learn more robust models than those learned in a classical Bayesian network setting. While the variables in the same module might behave according to the same model in underlying distribution, this will often not be the case in the empirical distribution based on a finite number of samples. A Bayesian network learning algorithm will treat each variable separately, optimizing the parent set and CPD for each variable in an independent manner. In the high-dimensional domains in which we are interested, there are bound to be spurious correlations that arise from sampling noise, inducing the algorithm to choose parent sets that do not reflect real dependencies, and will not generalize to unseen data. Conversely, in a module network setting, a spurious correlation would have to arise between a possible parent and a large number of other variables before the algorithm would find it worthwhile to introduce the dependency.

The module network framework, as presented here, has several important limitations, both from a modeling perspective and from the perspective of the learning algorithm.

¿From a modeling perspective, it is important to recognize that a module network is not a universally appropriate model for all domains. In particular, many domains do not have a natural organization of variables into higher level modules with common characteristics. In such domains, a module network would force variables into sharing dependency structures and CPDs and may result in poor representations of the underlying domain properties.

Even in domains where the modularity assumption is warranted, the module network models we presented here may not be ideal. In particular, the module network models we presented here allow each variable to be assigned to only one module. For instance, in the gene expression domain, this means that each gene is allowed to participate in only a single module. This assumption is not realistic biologically, as biological processes often involve partially overlapping sets of genes, so that many genes participate in more than one process. The framework presented in this paper, by restricting each gene to only one module, cannot represent such overlapping processes with different regulatory mechanisms. Recently (Segal *et al.*, 2003a; Battle *et al.*, 2004), we presented one possible extension to the module network framework presented in this paper, which allows genes to be assigned to several modules. The expression of a gene in a particular array is then modeled as a sum of its expression in each of the modules in which it participates, and each module can potentially have a different set of regulators. Clearly, this approach for "allocating" a variable and its observed signal among different modules is only one possible model, and one which is not appropriate to all settings. Other domains will likely require the development of other approaches.

Turning to the learning algorithm, one important limitation is our assumption that the number of modules is determined in advance. For instance, in the biological domain, the number of regulatory modules of an organism in an expression data set is obviously not known and thus determining

the number of modules should be part of the regulatory module discovery task. In Section 6.1 we showed that, at least in synthetic data, where the number of modules is known, we can use the score of the model to select the correct number of modules by choosing the model with the smallest number of modules from among the highest scoring models. This observation is encouraging, as it suggests that we can extend our approach to select the number of modules automatically by adding search steps that modify the number of modules and use the model score to compare models that differ in their number of modules. However, much remains to be done on the problem of proposing new modules and initializing them.

Another important limitation of the learning algorithm is the use of heuristic search to select a single module network model. As other models may have comparable (or even better) scores to that of the final model selected, a critical issue is to provide confidence estimates for the structural relationships reported by the model. This problem is common to many learning algorithms, including standard methods for Bayesian network learning, but is particularly acute when we are trying to use the learned structure for knowledge discovery, as we do in the biology domain. In this paper, we addressed this issue only indirectly, through statistical generalization tests on held out data and through the evaluation of our results relative to the to existing annotations (e.g., of stock categories in the stock market domain).

As a more direct approach, in some cases we can make use of well known methods for confidence estimation such as *bootstrap* (Efron and Tibshirani, 1993), which repeatedly learns models from resamples of the original input data and then estimates the confidence of different features of the model based on the number of times they appear in all models learned. Such an approach was adopted for estimating the confidence in features of a Bayesian network by Friedman *et al.* (1999b) and consequently applied by Friedman *et al.* (2000b) for learning fragments of regulatory networks from expression data. An alternative approach is to use Markov Chain Monte Carlo methods to sample models from the posterior given the data. It is fairly straightforward to use the Bayesian score we devised here within a Metropolis-Hastings sampling procedure Doucet *et al.* (2001) to perform model averaging Hoeting *et al.* (1999). The challenge is to design sampling strategies that lead to rapid mixing of the Markov Chain sampler. In the context of Bayesian networks, recent results (e.g., (Friedman and Koller, 2003)) use the decomposable structure of the posterior for efficient sampling. In the context of module networks, we also need to construct efficient sampling strategies over assignment functions. Recall that the space of possible assignment functions is huge, and so *a priori* it is not clear that a simple sampling procedure (e.g., mirroring our search strategy and moving one variable at each step) will mix in reasonable time. Clearly, adapting such confidence estimation approaches for our models can greatly enhance the reliability of our results but require additional development and validation.

In this paper, we focused on the statistical properties of our method. In a companion biological paper (Segal *et al.*, 2003b), we use the module network learned from the gene expression data described above to predict gene regulation relationships. There, we performed a comprehensive evaluation of the validity of the biological structures reconstructed by our method. By analyzing biological databases and previous experimental results in the literature, we confirmed that many of the regulatory relations that our method automatically inferred are indeed correct. Furthermore, our model provided focused predictions for genes of previously uncharacterized function. We performed wet lab biological experiments that confirmed the three novel predictions we tested. Thus, we have demonstrated that the module network model is robust enough to learn a good approximation of the dependency structure between 2355 genes using only 173 instances. These results show

that, by learning a structured probabilistic representation, we identify regulation networks from gene expression data and successfully address one of the central problems in analysis of gene expression data.

Acknowledgments

References

A. Battle, E. Segal, and D. Koller. Probabilistic discovery of overlapping cellular processes and their regulation using gene expression data. In *Proceedings Eighth Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, 2004.

L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth & Brooks, Monterey, CA, 1984.

W. Buntine. Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2:159–225, 1994.

P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. Autoclass: a Bayesian classification system. In *Proceedings Fifth International Conference on Machine Learning (ML)*, pages 54–64, 1988.

J. M. Cherry, C. Ball, K. Dolinski, S. Dwight, M. Harris, J. C. Matese, G. Sherlock, G. Binkley, H. Jin, S. Weng, and D. Botstein. Saccharomyces genome database. *Nucleic Acid Research*, 26:73–79, 1998. http://genome-www.stanford.edu/Saccharomyces/.

D. M. Chickering, D. Heckerman, and C. Meek. A Bayesian approach to learning Bayesian networks with local structure. In *Proceedings Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 80–89, 1997.

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

T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150, 1989.

M. H. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, New York, 1970.

A. Doucet, N. de Freitas, and N. Gordon (eds). *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.

B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, London, 1993.

G. Elidan and N. Friedman. Learning the dimensionality of hidden variables. In *Proceedings Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 144–151, 2001.

A. P. Gasch et al. Genomic expression program in the response of yeast cells to environmental changes. *Mol. Bio. Cell*, 11:4241–4257, 2000.

N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 421–460. Kluwer, Dordrecht, Netherlands, 1998.

N. Friedman and D. Koller. Being Bayesian about Bayesian network structure: A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50:95–126, 2003.

N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings Sixteenth International Conference on Artificial Intelligence (IJCAI)*, pages 1300–1309, 1999.

N. Friedman, M. Goldszmidt, and A. Wyner. Data analysis with Bayesian networks: A bootstrap approach. In *Proc. UAI*, pages 206–215, 1999.

N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using Bayesian networks to analyze expression data. *Journal of Computational Biology*, 7:601–620, 2000.

N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using Bayesian networks to analyze expression data. *Computational Biology*, 7:601–620, 2000.

L. Getoor, D. Koller, and N. Friedman. From instances to classes in probabilistic relational models. In *Proceedings of the ICML Workshop on Attribute-Value and Relational Learning*, 2000.

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

D. Heckerman. A tutorial on learning with Bayesian networks. In M. I. Jordan, editor, *Learning in Graphical Models*. Kluwer, Dordrecht, Netherlands, 1998.

J. A. Hoeting, D. Madigan, A. Raftery, and C. T. Volinsky. Bayesian model averaging: A tutorial. *Statistical Science*, 14(4), 1999.

D. Koller and A. Pfeffer. Object-oriented Bayesian networks. In *Proceedings Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 302–313, 1997.

D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proceedings National Conference on Artificial Intelligence (AAAI)*, pages 580–587, 1998.

E. Lander. Array of hope. *Nature Genetics*, 21:3–4, 1999.

H. Langseth and T. D. Nielsen. Fusion of domain knowledge with data for structural learning in object oriented domains. *Machine Learning Research*, 4:339–368, 2003.

D. Pe'er, A. Regev, G. Elidan, and N. Friedman. Inferring subnetworks from perturbed expression profiles. *Bioinformatics*, 17(Suppl 1):S215–24, 2001.

N. E. Savin. The Bonferroni and the Scheffe multiple comparison procedures. *Review of Economic Studies*, 47(1):255–73, 1980.

E. Segal, B. Taskar, A. Gasch, N. Friedman, and D. Koller. Rich probabilistic models for gene expression. *Bioinformatics*, 17(Suppl 1):S243–52, 2001.

E. Segal, A. Battle, and D. Koller. Decomposing gene expression into cellular processes. In *Proceedings Eighth Pacific Symposium on Biocomputing (PSB)*, 2003.

E. Segal, M. Shapira, A. Regev, D. Pe'er, D. Botstein, D. Koller, and N. Friedman. Module networks: Discovering regulatory modules and their condition specific regulators from gene expression data. *Nature Genetics*, 34(2):166–176, 2003.