An integrated approach to validating ChIP-Seq using A* Lasso for Sparse Bayesian Network Learning

Jing Xiang
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
jingx@cs.cmu.edu

Abstract

Mapping the transcription network is integral to our understanding of the transcriptional regulation in cells. Recently, ChIP-Seq experiments have become popular to determine the regulatory relationships between transcription factors (TF) and its target genes by detecting physical binding. However, these experiments are noisy and many binding events are not actually functional. Instead of doing further laboratory experiments which are expensive and not realistic for humans, we propose to estimate a sparse Bayesian network structure of the TFs and target genes, and detect SNPs that perturb the network. These SNP perturbations provide evidence of functional bindings. To learn the sparse Bayesian network, we present A* lasso, a single stage method that recovers the optimal sparse Bayesian network structure by solving a single optimization problem with A* search algorithm that uses lasso in its scoring system. Our approach substantially improves the computational efficiency of the well-known exact methods based on dynamic programming. In addition, to make the method more practical for settings that have a large number of variables, we suggest a heuristic scheme that dramatically reduces computational time without substantially compromising the quality of solutions. We apply our method to integrate SNP, expression and ChIP-Seq data. We detect several influential SNPs in our network, and are able to provide evidence for TF binding and characterize the effect on the surrounding network.

1 Introduction

Transcription is an important control point for the cell to regulate complex molecular pathways and elucidating the transcription network is a focus of active research for both computational and experimental areas. Many computational approaches have been developed that harness the information in expression data to estimate these networks [30, 19]. However, these methods depend mostly on the correlation of expression levels across genes, and thus the direction of the interaction cannot be resolved from expression data alone. Recently, experimental procedures such as chromatin immunoprecipitation sequencing (ChIP-Seq) have become popular for providing physical evidence for the binding of a transcription factor (TF) to its target gene, thus providing directional information for parts of the network.

Unfortunately, it is not possible to construct the functional transcription network from ChIP-seq data alone. ChIP-seq data is noisy and many of the binding events detected are false positives and therefore not functional. Thus, it becomes necessary to validate the binding between a TF and target. There are experimental approaches to validation such as knock down of the factor or gene by mutation and more recently by RNA interference, which silences the gene by mRNA degradation [17, 4, 28]. Knock down studies validate these binding events by perturbing the expression of a TF and observing the effect on the expression of its target genes. However, these experimental approaches have several disadvantages. For instance, laboratory experiments are expensive and time consuming on a large scale. They also pose a challenge for studying humans. While knock down studies can be performed in cell lines, they cannot be performed *in vivo* for ethical reasons.

We propose to substitute knock down experiments by collecting population SNP data and using network structure learning to detect how they perturb the transcription network. We collect SNP and expression data from a population, and allow natural genetic variation between individuals to serve as the source of perturbations. Specifically, genetic mutations that affect TF activity will then influence the expression of its target genes, providing evidence to conclude that the TF to target bindings are functional.

We use a Bayesian network of genes conditioned on SNPs to model the transcription network under SNP perturbations. We choose to model our network as a directed network using Bayesian networks. This is compatible with ChIP-seq data because candidate edges have a natural direction since TFs bind to target genes. Because we are only interested in learning the SNP perturbations of genes, and we are not interested in the structure between the SNPs, we estimate a Bayesian network model conditioned on all the SNPs. We use the candidate edges from ChIP-seq as a starting point, and we learn the transcription network that contains only the functional edges from SNP and expression data of a population of individuals. By learning this network, we will identify 1) the SNPs that perturb the transcription network, 2) the TFs that were directly perturbed by the SNPs, 3) the structure of the transcription network, and 4) the candidate edges between TF and target that are functional.

To estimate the network structure, we present A* lasso that learns a Bayesian network in high-dimensional space. A* lasso finds the optimal network structure with a sparse set of parents, a quality that is appropriate for biological networks. To scale the method to large networks, we suggest a heuristic scheme that dramatically reduces computational time without significantly compromising the quality of solutions.

We apply our method to the ENCODE project ChIP-seq data, SNP data from 1000 genomes, and expression data created for the HapMap project to elucidate the functional TF to target regulatory relationships from candidate interactions in ChIP-seq data. The SNP and expression data were collected from lymphoblastoid cell lines, the same cell line used to collect the ChIP-seq data. Unlike previous approaches, our method is designed to directly incorporate the ChIP-seq candidate edges, and validate them by integrating SNP and expression data in a single analysis. In addition, by analyzing the resulting transcription network, we can characterize the effect of SNP perturbations in detail. We are able to identify the direct regulation between TF and targets, as well as the indirect effects of the SNP perturbations on downstream genes.

2 Background on Bayesian Network Structure Learning

Bayesian networks have been popular tools for representing the probability distribution over a large number of variables. However, learning a Bayesian network structure from data has been known to be an NP-hard problem [1] because of the constraint that the network structure has to be a directed acyclic graph (DAG). Many of the exact methods that have been developed for recovering the optimal structure are computationally expensive and require exponential computation time [23, 12]. Approximate methods based on heuristic search are more computationally efficient, but they recover a suboptimal structure. In this paper, we address the problem of learning a Bayesian network structure for continuous variables in a high-dimensional space and propose an algorithm that recovers the exact solution with less computation time than the previous exact algorithms, and with the flexibility of further reducing computation time without a significant decrease in accuracy.

Many of the existing algorithms are based on scoring each candidate graph and finding a graph with the best score, where the score decomposes for each variable given its parents in a DAG. Although methods may differ in the scoring method that they use (e.g., MDL [14], BIC [22], and BDe [7]), most of these algorithms, whether exact methods or heuristic search techniques, have a two-stage learning process. In Stage 1, candidate parent sets for each node are identified while ignoring the DAG constraint. Then, Stage 2 employs various algorithms to search for the best-scoring network structure that satisfies the DAG constraint by limiting the search space to the candidate parent sets from Stage 1. For Stage 1, methods such as sparse candidate [5], max-min parents children [26], and total conditioning [18] algorithms have been previously proposed. For Stage 2, exact methods based on dynamic programming [12, 23] and A* search algorithm [29] as well as inexact methods such as heuristic search technique [26] and linear programming formulation [11] have been developed. These approaches have been developed primarily for discrete variables, and regardless of whether exact or inexact methods are used in Stage 2, Stage 1 involves exponential computation time and space.

For continuous variables, L_1-regularized Markov blanket (L1MB) [21] was proposed as a two-stage method that uses lasso to select candidate parents for each variable in Stage 1 and performs heuristic search for DAG structure and variable ordering in Stage 2. Although a two-stage approach can reduce the search space by pruning candidate parent sets in Stage 1, Huang *et al.* [10] observed that applying lasso in Stage 1 as in L1MB is likely to miss the true parents in a high-dimensional setting, thereby limiting the quality of the solution in Stage 2. They proposed the sparse Bayesian network (SBN) algorithm that formulates the problem of Bayesian network structure learning as a single-stage optimization problem and transforms it into a lasso-type optimization to obtain an approximate solution. Then, they applied a heuristic search to refine the solution as a post-processing step.

In this paper, we propose a new algorithm, called A* lasso, for learning a sparse Bayesian network structure with continuous variables in high-dimensional space. Our method is a single-stage algorithm that finds the optimal network structure with a sparse set of parents while ensuring the DAG constraint is satisfied. We first show that a lasso-based scoring method can be incorporated within dynamic programming (DP). While previous approaches based on DP required identifying the exponential number of candidate parent sets and their scores for each variable in Stage 1 before applying DP in Stage 2 [12, 23], our approach effectively combines the score computation in Stage 1 within Stage 2 via lasso optimization. Then, we present A* lasso which significantly prunes the search space of DP by incorporating the A* search algorithm [20], while guaranteeing the optimality of the solution. Since in practice, A* search can still be expensive compared to heuristic methods, we explore heuristic schemes that further limit the search space of A* lasso. We demonstrate in our experiments that this heuristic approach can substantially improve the computation time without significantly compromising the quality of the solution, especially on large Bayesian networks.

3 Problem Setup for Bayesian Network Structure Learning

A Bayesian network is a probabilistic graphical model defined over a DAG G with a set of $p = |V|$ nodes $V = \{v_1, \ldots, v_p\}$, where each node v_j is associated with a random variable X_j [13]. The probability model associated with G in a Bayesian network factorizes as $p(X_1, \ldots, X_p) = \prod_{j=1}^{p} p(X_j|\text{Pa}(X_j))$, where $p(X_j|\text{Pa}(X_j))$ is the conditional probability distribution for X_j given its parents Pa(X_j) with directed edges from each node in Pa(X_j) to X_j in G. We assume continuous random variables and use a linear regression model for the conditional probability distribution of each node $X_j = \text{Pa}(X_j)'\boldsymbol{\beta}_j + \epsilon$, where $\boldsymbol{\beta}_j = \{\beta_{jk}\text{'s for } X_k \in \text{Pa}(X_j)\}$ is the vector of unknown parameters to be estimated from data and ϵ is the noise distributed as $\sim N(0, 1)$.

Given a dataset $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_p]$, where $\mathbf{x}_j$ is a vector of n observations for random variable X_j, our goal is to estimate the graph structure G and the parameters $\boldsymbol{\beta}_j$'s jointly. We formulate this problem as that of obtaining a sparse estimate of $\boldsymbol{\beta}_j$'s, under the constraint that the overall graph structure G should not contain directed cycles. Then, the nonzero elements of $\boldsymbol{\beta}_j$'s indicate the presence of edges in G. We obtain an estimate of Bayesian network structure and parameters by minimizing the negative log likelihood of data with sparsity enforcing L_1 penalty as follows:

$$\min_{\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_p} \sum_{j=1}^{p} \parallel \mathbf{x}_j - \mathbf{x}_{-j}'\boldsymbol{\beta}_j \parallel_2^2 + \lambda \sum_{j=1}^{p} \parallel \boldsymbol{\beta}_j \parallel_1 \quad \text{s.t. } G \in \text{DAG}, \tag{1}$$

where $\mathbf{x}_{-j}$ represents all columns of $\mathbf{X}$ excluding $\mathbf{x}_j$, assuming all other variables are candidate parents of node v_j. Given the estimate of $\boldsymbol{\beta}_j$'s, the set of parents for node v_j can be found as the support of $\boldsymbol{\beta}_j$, $S(\boldsymbol{\beta}_j) = \{v_i|\beta_{ji} \neq 0\}$. The λ is the regularization parameter that determines the amount of sparsity in $\boldsymbol{\beta}_j$'s and can be determined by cross-validation. We notice that if the acyclicity constraint is ignored, Equation (1) decomposes into individual lasso estimations for each node:

$$\text{LassoScore}(v_j|V\backslash v_j) = \min_{\boldsymbol{\beta}_j} \parallel \mathbf{x}_j - \mathbf{x}_{-j}'\boldsymbol{\beta}_j \parallel_2^2 + \lambda \parallel \boldsymbol{\beta}_j \parallel_1,$$

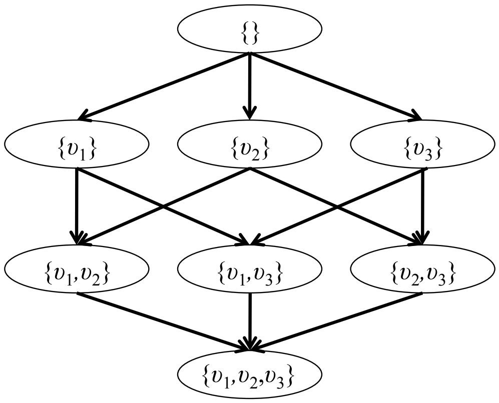

Figure 1: Search space of variable ordering for three variables $V = \{v_1, v_2, v_3\}$.

where $V \backslash v_j$ represents the set of all nodes in V excluding v_j. The above lasso optimization problem can be solved efficiently with the shooting algorithm [6]. However, the main challenge in optimizing Equation (1) arises from ensuring that the $\boldsymbol{\beta}_j$'s satisfy the DAG constraint.

4 A* Lasso for Bayesian Network Structure Learning

4.1 Dynamic Programming with Lasso

The problem of learning a Bayesian network structure that satisfies the constraint of no directed cycles can be cast as that of learning an optimal ordering of variables [13]. Once the optimal variable ordering is given, the constraint of no directed cycles can be trivially enforced by constraining the parents of each variable in the local conditional probability distribution to be a subset of the nodes that precede the given node in the ordering. We let $\Pi^V = [\pi_1^V, \ldots, \pi_{|V|}^V]$ denote an ordering of the nodes in V, where π_j^V indicates the node $v \in V$ in the jth position of the ordering, and $\Pi_{\prec v_j}^V$ denote the set of nodes in V that precede node v_j in ordering Π^V.

Algorithms based on DP have been developed to learn the optimal variable ordering for Bayesian networks [25]. These approaches are based on the observation that the score of the optimal ordering of the full set of nodes V can be decomposed into (a) the optimal score for the first node in the ordering, given a choice of the first node and (b) the score of the optimal ordering of the nodes excluding the first node. The optimal variable ordering can be constructed by recursively applying this decomposition to select the first node in the ordering and to find the optimal ordering of the set of remaining nodes $U \in V$. This recursion is given as follows, with an initial call of the recursion with $U = V$:

$$\text{OptScore}(U) = \min_{v_j \in U} \text{OptScore}(U \backslash v_j) + \text{BestScore}(v_j | V \backslash U) \tag{2}$$

$$\pi_1^U = \operatorname*{argmin}_{v_j \in U} \text{OptScore}(U \backslash v_j) + \text{BestScore}(v_j | V \backslash U), \tag{3}$$

where BestScore$(v_j | V \backslash U)$ is the optimal score of v_j under the optimal choice of parents from $V \backslash U$.

In order to obtain BestScore$(v_j | V \backslash U)$ in Equations (2) and (3), for the case of discrete variables, many previous approaches enumerated all possible subsets of V as candidate sets of parents for node v_j to precompute BestScore$(v_j | V \backslash U)$ in Stage 1 before applying DP in Stage 2 [12, 23]. While this approach may perform well in a low-dimensional setting, in a high-dimensional setting, a two-stage method is likely to miss the true parent sets in Stage 1, which in turn affects the performance of Stage 2 [10]. In this paper, we consider the high-dimensional setting and present a single-stage method that applies lasso to obtain BestScore$(v_j | V \backslash U)$ within DP as follows:

$$\begin{aligned} \text{BestScore}(v_j | V \backslash U) &= \text{LassoScore}(v_j | V \backslash U) \\ &= \min_{\boldsymbol{\beta}_j, S(\boldsymbol{\beta}_j) \subseteq V \backslash U} \parallel \mathbf{x}_j - \mathbf{x}_{-j}{}' \boldsymbol{\beta}_j \parallel_2^2 + \lambda \parallel \boldsymbol{\beta}_j \parallel_1 . \end{aligned}$$

The constraint $S(\boldsymbol{\beta}_j) \subseteq V \backslash U$ in the above lasso optimization can be trivially maintained by setting the β_{jk} for $v_k \in U$ to 0 and optimizing only for the other β_{jk}'s. When applying the recursion in Equations (2) and (3), DP takes advantage of the overlapping subproblems to prune the search space of orderings, since the problem of computing OptScore(U) for $U \subseteq V$ can appear as a subproblem of scoring orderings of any larger subsets of V that contain U.

The problem of finding the optimal variable ordering can be viewed as that of finding the shortest path from the start state to the goal state in a search space given as a subset lattice. The search space consists of a set of states, each of which is associated with one of the $2^{|V|}$ possible subsets of nodes in V. The start state is the empty set $\{\}$ and the goal state is the set of all variables V. A valid move in this search space is defined from a state for subset Q_s to another state for subset $Q_{s'}$, only if $Q_{s'}$ contains one additional node to Q_s. Each move to the next state corresponds to adding a node at the end of the ordering of the nodes in the previous state. The cost of such a move is given by BestScore$(v|Q_s)$, where $v = Q_{s'} \backslash Q_s$. Each path from the start state to the goal state gives one possible ordering of nodes. Figure 1 illustrates the search space, where each state is associated with a Q_s. DP finds the shortest path from the start state to the goal state that corresponds to the optimal variable ordering by considering all possible paths in this search space and visiting all $2^{|V|}$ states.

4.2 A* Lasso for Pruning Search Space

As discussed in the previous section, DP considers all $2^{|V|}$ states in the subset lattice to find the optimal variable ordering. Thus, it is not sufficiently efficient to be practical for problems with more than 20 nodes. On the other hand, a greedy algorithm is computationally efficient because it explores a single variable ordering by greedily selecting the most promising next state based on BestScore$(v|Q_s)$, but it returns a suboptimal solution. In this paper, we propose A* lasso that incorporates the A* search algorithm [20] to construct the optimal variable ordering in the search space of the subset lattice. We show that this strategy can significantly prune the search space compared to DP, while maintaining the optimality of the solution.

When selecting the next move in the process of constructing a path in the search space, instead of greedily selecting the move, A* search also accounts for the estimate of the future cost given by a heuristic function $h(Q_s)$ that will be incurred to reach the goal state from the candidate next state. Although the exact future cost is not known until A* search constructs the full path by reaching the goal state, a reasonable estimate of the future cost can be obtained by ignoring the directed acyclicity constraint. It is well-known that A* search is guaranteed to find the shortest path if the heuristic function $h(Q_s)$ is *admissible* [20], meaning that $h(Q_s)$ is always an underestimate of the true cost of reaching the goal state. Below, we describe an admissible heuristic for A* lasso.

While exploring the search space, A* search algorithm assigns a score $f(Q_s)$ to each state s and its corresponding subset Q_s of variables for which the ordering has been determined. A* search algorithm computes this score $f(Q_s)$ as the sum of the cost $g(Q_s)$ that has been incurred so far to reach the current state from the start state and an estimate of the cost $h(Q_s)$ that will be incurred to reach the goal state from the current state:

$$f(Q_s) = g(Q_s) + h(Q_s). \tag{4}$$

More specifically, given the ordering Π^{Q_s} of variables in Q_s that has been constructed along the path from the start state to the state for Q_s, the cost that has been incurred so far is defined as

$$g(Q_s) = \sum_{v_j \in Q_s} \text{LassoScore}(v_j | \Pi^{Q_s}_{\prec v_j}) \tag{5}$$

and the heuristic function for the estimate of the future cost to reach the goal state is defined as:

$$h(Q_s) = \sum_{v_j \in V \backslash Q_s} \text{LassoScore}(v_j | V \backslash v_j) \tag{6}$$

Note that the heuristic function is admissible, or an underestimate of the true cost, since the constraint of no directed cycles is ignored and each variable in $V \backslash Q_s$ is free to choose any variables in V as its parents, which lowers the lasso objective value.

When the search space is a graph where multiple paths can reach the same state, we can further improve efficiency if the heuristic function has the property of *consistency* in addition to admissibility.

A consistent heuristic always satisfies $h(Q_s) \leq h(Q_{s'}) + \text{LassoScore}(v_k|Q_s)$, where $\text{LassoScore}(v_k|Q_s)$ is the cost of moving from state Q_s to state $Q_{s'}$ with $\{v_k\} = Q_{s'} \backslash Q_s$. Consistency ensures that the first path found by A* search to reach the given state is always the shortest path to that state [20]. This allows us to prune the search when we reach the same state via a different path later in the search. The following proposition states that our heuristic function is consistent.

Proposition 1 *The heuristic in Equation (6) is consistent.*

Proof For any successor state $Q_{s'}$ of Q_s, let $v_k = Q_{s'} \backslash Q_s$.

$$\begin{aligned} h(Q_s) &= \sum_{v_j \in V \backslash Q_s} \text{LassoScore}(v_j|V \backslash v_j) \\ &= \sum_{v_j \in V \backslash Q_s, v_j \neq v_k} \text{LassoScore}(v_j|V \backslash v_j) + \text{LassoScore}(v_k|V \backslash v_k) \\ &\leq h(Q_{s'}) + \text{LassoScore}(v_k|Q_s), \end{aligned}$$

where $\text{LassoScore}(v_k|Q_s)$ is the true cost of moving from state Q_s to $Q_{s'}$. The inequality above holds because v_k has fewer parents to choose from in $\text{LassoScore}(v_k|Q_s)$ than in $\text{LassoScore}(v_k|V \backslash v_k)$. Thus, our heuristic in Equation (6) is consistent. ■

Given a consistent heuristic, many paths that go through the same state can be pruned by maintaining an *OPEN* list and a *CLOSED* list during A* search. In practice, the *OPEN* list can be implemented with a priority queue and the *CLOSED* list can be implemented with a hash table. The *OPEN* list is a priority queue that maintains all the intermediate results $(Q_s, f(Q_s), g(Q_s), \Pi^{Q_s})$'s for a partial construction of the variable ordering up to Q_s at the frontier of the search, sorted according to the score $f(Q_s)$.

During search, A* lasso pops from the *OPEN* list the partial construction of ordering with the lowest score $f(Q_s)$, visits the successor states by adding another node to the ordering Π^{Q_s}, and queues the results onto the *OPEN* list. Any state that has been popped by A* lasso is placed in the *CLOSED* list. The states that have been placed in the *CLOSED* list are not considered again, even if A* search reaches these states through different paths later in the search.

The full algorithm for A* lasso is given in Algorithm 1. As in DP with lasso, A* lasso is a single-stage algorithm that solves lasso within A* search. Every time A* lasso moves from state Q_s to the next state $Q_{s'}$ in the search space, $\text{LassoScore}(v_j|\Pi^{Q_s}_{\prec v_j})$ for $\{v_j\} = Q_{s'} \backslash Q_s$ is computed with the shooting algorithm and added to $g(Q_s)$ to obtain $g(Q_{s'})$. The heuristic score $h(Q_{s'})$ can be precomputed as $\text{LassoScore}(v_j|V \backslash v_j)$ for all $v_j \in V$ for a simple look-up during A* search.

```
**Input** : $\mathbf{X}$, $V$, $\lambda$
**Output**: Optimal variable ordering $\Pi^V$
Initialize *OPEN* to an empty queue;
Initialize *CLOSED* to an empty set;
Compute LassoScore($v_j|V\backslash v_j$) for all $v_j \in V$;
*OPEN*.insert(($Q_s = \{\}, f(Q_s) = h(\{\}), g(Q_s) = 0, \Pi^{Q_s} = [\,]$));
**while** *true* **do**
    $(Q_s, f(Q_s), g(Q_s), \Pi^{Q_s}) \leftarrow$ *OPEN*.pop();
    **if** *h($Q_s$) = 0* **then**
        Return $\Pi^V \leftarrow \Pi^{Q_s}$;
    **end**
    **foreach** $v \in V\backslash Q_s$ **do**
        $Q_{s'} \leftarrow Q_s \cup \{v\}$;
        **if** $Q_{s'} \notin$ *CLOSED* **then**
            Compute LassoScore($v|Q_s$) with lasso shooting algorithm;
            $g(Q_{s'}) \leftarrow g(Q_s) +$ LassoScore($v|Q_s$);
            $h(Q_{s'}) \leftarrow h(Q_s) -$ LassoScore($v|V\backslash v$);
            $f(Q_{s'}) \leftarrow g(Q_{s'}) + h(Q_{s'})$;
            $\Pi^{Q_{s'}} \leftarrow [\Pi^{Q_s}, v]$;
            *OPEN*.insert($L = (Q_{s'}, f(Q_{s'}), g(Q_{s'}), \Pi^{Q_{s'}})$);
            *CLOSED* $\leftarrow$ *CLOSED* $\cup\{Q_{s'}\}$;
        **end**
    **end**
**end**
```

Algorithm 1: A* lasso for learning Bayesian network structure

A* lasso finds the optimal ordering because A* search is optimal given an admissible heuristic. It is possible that A* lasso can find multiple optimal orderings, because there exist many networks that are equivalent due to equivalence classes. However, if we fix the variable ordering to one of the optimal orderings, then the problem is convex because our objective is strongly convex. As a result, we will get a global optima.

4.3 Heuristic Schemes for A* Lasso to Improve Scalability

Although A* lasso substantially prunes the search space compared to DP, it is not sufficiently efficient for large graphs, because it still considers a large number of states in the exponentially large search space. One simple strategy for further pruning the search space would be to limit the size of the priority queue in the *OPEN* list, forcing A* lasso to discard less promising intermediate results first. In this case, limiting the queue size to one is equivalent to a greedy algorithm with a scoring function in Equation (4). In our experiments, we found that such a naive strategy substantially reduced the quality of solutions because the best-scoring intermediate results tend to be the results at the early stage of the exploration. They are at the shallow part of the search space near the start state because the admissible heuristic underestimates the true cost.

Instead, given a limited queue size, we propose to distribute the intermediate results to be discarded across different depths/layers of the search space. For example, given the depth of the search space $|V|$, if we need to discard k intermediate results, we discard $k/|V|$ intermediate results at each depth. In our experiments, we found that this heuristic scheme substantially improves the computation time of A* lasso with a small reduction in the quality of the solution. We also considered other strategies such as inflating heuristics [15] and pruning edges in preprocessing with lasso, but such strategies substantially reduced the quality of solutions.

4.4 Bayesian Network Structure Learning for Conditional Model

In some cases, it may not be necessary to estimate the network structure for all the random variables in our domain. Specifically, for a conditional model, we only need to estimate the network structure of a subset of the variables conditioned on the rest. We now extend our formulation to include two types of nodes, $W = \{w_1, \cdots, w_q\}$ and the previously defined V. Let Z_m be the random variable associated with each node w_m, where $Z_1, \cdots, Z_q$ denotes the random variables that can be conditioned on automatically. Then, the modified probability model for G in the Bayesian network factorizes as $p(X_1, \ldots, X_p | Z_1, \cdots, Z_q) = \prod_{j=1}^{p} p(X_j | \text{Pa}(X_j))$ where $p(X_j | \text{Pa}(X_j))$ is the conditional probability distribution for X_j given its parents Pa(X_j). The key difference is that the parents of X_j, Pa(X_j), can include $Z_1, \cdots, Z_q$ in addition to X_{-j}.

During the A* Lasso search algorithm, the set of nodes W are added to the node ordering $\Pi^{V \cup W}$ in the beginning because they are being conditioned on. Thus, in applications where you can divide your variables into these distinct groups, further reductions in computational time are possible since only part of the structure must be learned.

5 Simulation Study

We perform simulation studies in order to evaluate the accuracy of the estimated structures and measure the computation time of our method. We created several small networks under 20 nodes and obtained the structure of several benchmark networks between 27 and 56 nodes from the Bayesian Network Repository (the left-most column in Table 1). In addition, we used the tiling technique [27] to generate two networks of approximately 300 nodes so that we could evaluate our method on larger graphs. Given the Bayesian network structures, we set the parameters $\boldsymbol{\beta}_j$ for each conditional probability distribution of node v_j such that $\beta_{jk} \sim \pm Uniform[l, u]$ for predetermined values for u and l if node v_k is a parent of node v_j and $\beta_{jk} = 0$ otherwise. We then generated data from each Bayesian network by forward sampling with noise $\epsilon \sim N(0, 1)$ in the regression model, given the true variable ordering. All data were mean-centered.

We compare our method to several other methods including DP with lasso for an exact method, L1MB for heuristic search, and SBN for an optimization-based approximate method. We downloaded the software implementations of L1MB and SBN from the authors' website. For L1MB,

Table 1: Comparison of computation time of different methods

Dataset (Nodes)	DP	A* lasso	A* Qlimit 1000	A* Qlimit 200	A* Qlimit 100	A* Qlimit 5	L1MB	SBN
Dsep (6)	0.20 (64)	0.14 (15)	– (–)	– (–)	– (–)	0.17 (11)	2.65	8.76
Asia (8)	1.07 (256)	0.26 (34)	– (–)	– (–)	– (–)	0.22 (12)	2.79	8.9
Bowling (9)	2.42 (512)	0.48 (94)	– (–)	– (–)	– (–)	0.23 (13)	2.85	8.75
Inversetree (11)	8.44 (2048)	1.68 (410)	– (–)	1.8 (423)	1.16 (248)	0.2 (16)	3.03	8.56
Rain (14)	1216 (1.60e4)	76.64 (2938)	64.38 (1811)	13.97 (461)	7.88 (270)	1.67 (17)	12.26	10.19
Cloud (16)	1.6e4 (6.6e4)	137.36 (2660)	108.39 (1945)	26.16 (526)	9.92 (244)	2.14 (19)	4.72	14.56
Funnel (18)	4.2e4 (2.6e5)	1527.0 (2.3e4)	88.87 (2310)	25.19 (513)	11.53 (248)	2.73 (21)	4.76	10.08
Galaxy (20)	1.3e5 (1.0e6)	2.40e4 (8.2e4)	110.05 (3093)	27.59 (642)	12.02 (323)	3.03 (23)	6.59	11.0
Factor (27)	– (–)	– (–)	1389.7 (3912)	125.91 (801)	59.92 (397)	3.96 (30)	9.04	13.91
Insurance (27)	– (–)	– (–)	2874.2 (3448)	442.65 (720)	202.9 (395)	16.31 (33)	10.96	29.45
Water (32)	– (–)	– (–)	2397.0 (3442)	301.67 (687)	130.71 (343)	12.14 (38)	32.73	14.96
Mildew (35)	– (–)	– (–)	3928.8 (3737)	802.76 (715)	339.04 (368)	29.3 (36)	15.25	116.33
Alarm (37)	– (–)	– (–)	2732.3 (3426)	384.87 (738)	158.0 (378)	12.42 (42)	7.91	39.78
Barley (48)	– (–)	– (–)	10766.0 (4072)	1869.4 (807)	913.46 (430)	109.14 (52)	23.25	483.33
Hailfinder (56)	– (–)	– (–)	9752.0 (3939)	2580.5 (816)	1058.3 (390)	112.61 (57)	44.36	826.41

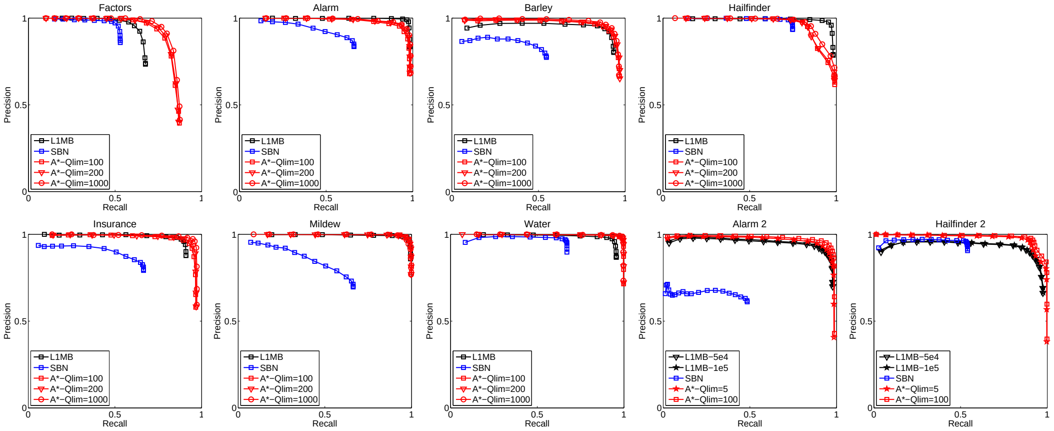

Figure 2: Precision/recall curves for the recovery of skeletons of benchmark Bayesian networks.

we increased the authors' recommended number of evaluations 2500 to 10 000 in Stage 2 heuristic search for all networks except the two larger networks of around 300 nodes (Alarm 2 and Hailfinder 2), where we used two different settings of 50 000 and 100 000 evaluations. We also evaluated A* lasso with the heuristic scheme with the queue sizes of 5, 100, 200, and 1000.

DP, A* lasso, and A* lasso with a limited queue size require a selection of the regularization parameter λ with cross-validation. In order to determine the optimal value for λ, for different values of λ, we trained a model on a training set, performed an ordinary least squares re-estimation of the non-zero elements of $\boldsymbol{\beta}_j$ to remove the bias introduced by the L_1 penalty, and computed prediction errors on the validation set. Then, we selected the value of λ that gives the smallest prediction error as the optimal λ. We used a training set of 200 samples for relatively small networks with under 60 nodes and a training set of 500 samples for the two large networks with around 300 nodes. We used a validation set of 500 samples. For L1MB and SBN, we used a similar strategy to select the regularization parameters, while mainly following the strategy suggested by the authors and in their software implementation.

We present the computation time for the different methods in Table 1. For DP, A* lasso, and A* lasso with limited queue sizes, we also record the number of states visited in the search space in parentheses in Table 1. All methods were implemented in Matlab and were run on computers with 2.4 GHz processors. We used a dataset generated from a true model with $\beta_{jk} \sim \pm Uniform[1.2, 1.5]$. It can be seen from Table 1 that DP considers all possible states $2^{|V|}$ in the search space that grows exponentially with the number of nodes. It is clear that A* lasso visits significantly fewer states

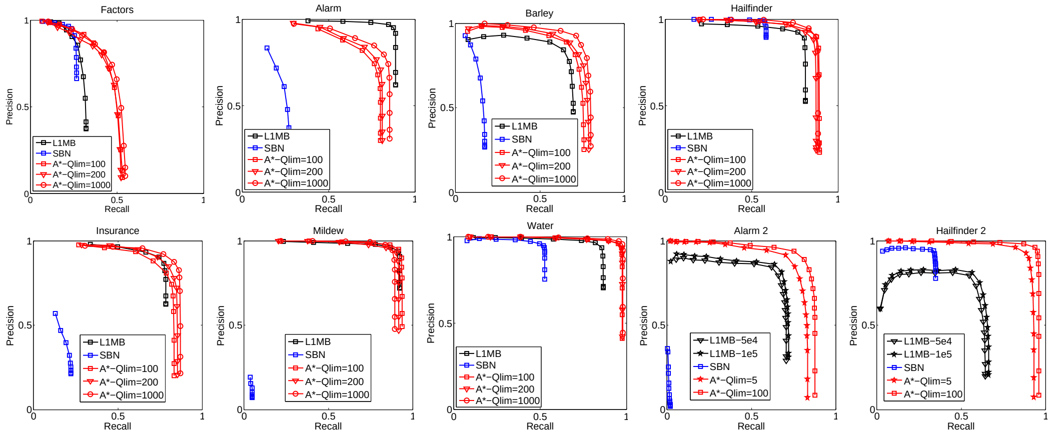

Figure 3: Precision/recall curves for the recovery of v-structures of benchmark Bayesian networks.

than DP, visiting about 10% of the number of states in DP for the funnel and galaxy networks. We were unable to obtain the computation time for A* lasso and DP for some of the larger graphs in Table 1 as they required significantly more time. Limiting the size of the queue in A* lasso reduces both the computation time and the number of states visited. For smaller graphs, we do not report the computation time for A* lasso with limited queue size, since it is identical to the full A* lasso. We notice that the computation time for A* lasso with a small queue of 5 or 100 is comparable to that of L1MB and SBN.

In general, we found that the extent of pruning of the search space by A* lasso compared to DP depends on the strengths of edges ($\boldsymbol{\beta}_j$ values) in the true model. We applied DP and A* lasso to datasets of 200 samples generated from each of the networks under each of the three settings for the true edge strengths, $\pm Uniform[1.2, 1.5]$, $\pm Uniform[1, 1.2]$, and $\pm Uniform[0.8, 1]$. As can be seen from the computation time and the number of states visited by DP and A* lasso in Table 2, as the strengths of edges increase, the number of states visited by A* lasso and the computation time tend to decrease. The results in Table 2 indicate that the efficiency of A* lasso is affected by the signal-to-noise ratio.

Table 2: A* lasso computation time under different edge strengths $\boldsymbol{\beta}_j$'s

Dataset (Nodes)	**(1.2,1.5)**	**(1,1.2)**	**(0.8,1)**
Dsep (6)	0.14 (15)	0.14 (16)	0.17 (30)
Asia (8)	0.26 (34)	0.23 (37)	0.29 (59)
Bowling (9)	0.48 (94)	0.49 (103)	0.54 (128)
Inversetree (11)	1.68 (410)	2.09 (561)	2.25 (620)
Rain (14)	76.64 (2938)	66.93 (2959)	97.26 (4069)
Cloud (16)	137.36 (2660)	229.12 (7805)	227.43 (8858)
Funnel (18)	1526.7 (22930)	2060.2 (33271)	3744.4 (40644)
Galaxy (20)	24040 (82132)	66710 (168492)	256490 (220821)

In order to evaluate the accuracy of the Bayesian network structures recovered by each method, we make use of the fact that two Bayesian network structures are indistinguishable if they belong to the same equivalence class, where an equivalence class is defined as the set of networks with the same skeleton and v-structures. The skeleton of a Bayesian network is defined as the edge connectivities ignoring edge directions and a v-structure is defined as the local graph structure over three variables, with two variables pointing to the other variables (i.e., $A \rightarrow B \leftarrow C$). We evaluate the performance of the different methods by comparing the estimated network structure with the true network structure in terms of skeleton and v-structures and computing the precision and recall.

The precision/recall curves for the skeleton and v-structures of the models estimated by the different methods are shown in Figures 2 and 3, respectively. Each curve was obtained as an average over the results from 30 different datasets for the two large graphs (Alarm 2 and Hailfinder 2) and

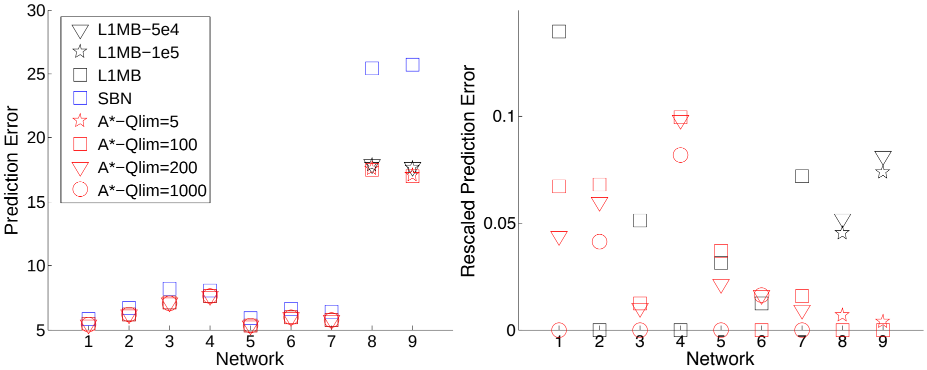

Figure 4: Prediction errors for benchmark Bayesian networks.The right figure is rescaled for clarity. The x-axis labels indicate different benchmark Bayesian networks for 1: Factors, 2: Alarm, 3: Barley, 4: Hailfinder, 5: Insurance, 6: Mildew, 7: Water, 8: Alarm 2, and 9: Hailfinder 2.

from 50 different datasets for all the other Bayesian networks. All data were simulated under the setting $\beta_{jk} \sim \pm Uniform[0.4, 0.7]$. For the benchmark Bayesian networks, we used A* lasso with different queue sizes, including 100, 200, and 1000, whereas for the two large networks (Alarm 2 and Hailfinder 2) that require more computation time, we used A* lasso with queue size of 5 and 100. As can be seen in Figures 2 and 3, all methods perform relatively well on identifying the true skeletons, but find it significantly more challenging to recover the true v-structures. We find that although increasing the size of queues in A* lasso generally improves the performance, even with smaller queue sizes, A* lasso outperforms L1MB and SBN in most of the networks. While A* lasso with a limited queue size preforms consistently well on smaller networks, it significantly outperforms the other methods on the larger graphs such as Alarm 2 and Hailfinder 2, even with a queue size of 5 and even when the number of evaluations for L1MB has been increased to 50 000 and 100 000. This demonstrates that while limiting the queue size in A* lasso will not guarantee the optimality of the solution, it still reduces the computation time of A* lasso dramatically without substantially compromising the quality of the solution. In addition, we compare the performance of the different methods in terms of prediction errors on independent test datasets in Figure 4. We find that the prediction errors of A* lasso are consistently lower even with a limited queue size.

6 Transcriptional Network Learning using A* Lasso

To validate ChIP-Seq binding events between TFs and targets computationally, we want to detect SNP perturbations of the transcriptional network. To achieve this, we must first learn the transcriptional network of TFs and targets conditioned on SNPs. Thus, we use A* lasso to learn a sparse Bayesian network structure for a conditional model. This requires integrating ChIP-Seq, SNP and expression data to estimate the network. In the following sections we described the preprocessing procedures as well as how the experiments were performed.

6.1 Data Sets

A collection of ChIP-Seq, SNP and microarray expression data was assembled from the the Encyclopedia of DNA Elements (ENCODE) project [3], 1000 Genomes Project [2] and Hap Map 3 population gene expression arrays [16, 24] respectively. These data sets were integrated and A* lasso was used to learn a sparse Bayesian network.

The binding sites of each ChIP-Seq experiment from ENCODE were obtained from the Transcription Factor ChIP-seq Uniform Peaks for Human Genome Build 37 (hg19) and further processing included identifying gene annotations and matching them to the genes in the expression data. We retrieved experiments of 76 unique transcription factors from the GM12878 lymphoblastoid cell line. The peaks had already been called allowing us to directly extract the location and signal intensity of the binding sites. We then extracted all binding sites between -20 kb of the transcription start site and +20 kb the transcription end site of a gene. This region allows us to include promoters, exons, introns and nearby enhancers both upstream and downstream. The transcription start and end sites as well as gene annotations were taken from Human Build 37 (GenBank Assembly ID: GCA_000001405.1). Originally, there were 23,429 unique targets across of the TFs. After selecting only those target genes that also have expression data, 15,971 targets remained.

We used microarray expression data originally collected for Hap Map 3 individuals (E-MTAB-264 and E-MTAB-198). Even though the expression data was collected from HapMap 3 samples, many of them were also used for the 1000 Genomes Project. We used these overlap individuals, which span seven populations and total 423 samples (Table 6.1). The platform for the expression data was the Illumina Human-6 v2 Expression BeadChip. Thus, to resolve the difference in naming conventions between the RefSeq gene annotations and the Illumina reporter names, we use DAVID Bioinformatics Resources 6.7 [8, 9] to convert Illumina ID's. Any duplicate genes were replaced with the entry with the highest variance. After filtering out genes with low variance, we had 9634 targets remaining from 15,971. The remaining 30 genes from the expression data set that were not ChIP-seq targets were included to make a total of 9,664 genes.

Pop. Code	Pop. Description	# Samples with SNP and Expression data
CEU	European	73
CHB	Chinese	77
JPT	Japanese	72
LWK	Luhya in Kenya	80
MEX	Mexican	42
YRI	Yoruba in Nigeria	79
Total		423

Table 3: Number of samples that had both microarray expression and 1,000 genomes SNP data available.

We extracted samples and relevant SNPs from the 1000 Genomes Project Phase I Release. We started with the integrated variant call set based on both low coverage and exome whole genome sequence data which consisted of 1,092 individuals and 38.2 million SNPs. As previously mentioned, we used only the individuals that had corresponding expression data. We extracted the SNPs that intersected with ChIP-Seq binding sites and filtered for those SNPs that had a minor allele frequency (MAF) of at least 0.05. This left us with 102 326 SNPs.

6.2 A* Lasso Experimental Setup

The three forms of data: ChIP-Seq, SNP and expression were integrated to perform a single A* Lasso network learning experiment. For this task, we used the conditional model for Bayesian network structure learning discussed in section 4.4. The set of nodes V are the genes that were assayed using expression and the set of nodes W are the SNPs. The data $\boldsymbol{X}$ consists of the expression and SNP data after mean centering.

ChIP-seq experiments provide a set of candidate edges. During learning, we select functional edges from the set of candidate edges and include them in the Bayesian network. We consider edges to be functional if ChIP-seq detected a binding event from the TF to the target gene, and the target gene expression is changed. To accomplish this, we encode the candidate edges in the heuristic function h instead of using the naive heuristic from the simulation experiments. Recall, that when we previously computed the heuristic function, the constraint of no directed cycles was ignored and the node to be added was allowed to choose any nodes in V as its parents. Here, we can do better than simply allowing the node to choose any nodes in $V \cup W$. Using information from ChIP-Seq, we know which transcription factors bind to which target genes. Thus, we can identify candidate parents V_j' for each gene v_j. In addition, the SNPs most likely to influence the expression of a particular gene are the SNPs that are associated with that gene and those that are associated with its candidate parents, so we can also identify candidate parents W_j' which is a subset of W. For a particular node v_j, we allow it to choose any parents from $V_j' \cup W_j'$. This results in a heuristic function that is still admissible because we are ignoring the DAG constraint, but it is a better estimate of the distance to the goal state because it incorporates the ChIP-Seq binding information.

The ChIP-seq binding data outlines the general structure of the network. The TFs have directed edges to other genes, these genes can be other TFs or target genes, and all the genes are conditioned on SNPs. Given this general structure, it makes sense to first learn the structure of the TFs using A* lasso, and then add target genes. To reduce the size of the problem, we divided the target genes into groups which were selected based on a hierarchical clustering of the gene expression data. After determining the structure of TFs, we then added a group of target genes and continued running the algorithm. Each group is run separately resulting in a reduction in computational time. We used queue limits of 10, 20, 50, 100, and 200 and a range of λ's between 0.004 and 1. We cross-validated with a 80/20 split of the data and selected the best parameters based on test error.

7 ChIP-Seq Validation Results

We use the Bayesian network structure learning method A* lasso to integrate ChIP-Seq, expression and genotype data to allow us to computationally validate ChIP-Seq experiments. Determining which binding events are functional cannot be done simply by examining the results of ChIP-Seq experiments. However, ChIP-seq provides an important starting point as it identifies the binding relationships of TFs and targets. To find a functional binding, we need to perturb the expression of the TF and observe that it has an effect on the expression of its target genes. The expression of the TF can be altered by genetic mutations, which can be detected in the genotype data. The TF, targets and SNPs form a network and our objective is to identify genetic mutations that perturb the network. If we can find these perturbations, then we can provide evidence that the binding event is functional.

To establish that binding events from ChIP-Seq experiments are functional, we use A* lasso, which finds genetic mutations that perturb the network, as edges from a SNP to a gene. A mutation can have two potential effects. First, it can alter the concentration of the TF. For example, in Case 1a (Fig. 5A), the mutation occurs in the promoter region of the TF A which interferes with the binding of its regulator TF B. This in turn affects the concentration of TF A resulting in a global effect on all of TF A's targets. These changes in concentration will be reflected in the expression data and A* lasso will find all the edges in gray. In addition, if the mutation occurs in the promoter of a target gene, then only the expression of the specific target will be affected (Fig. 5B). In Case 1b, A* lasso will only discover the local effect. Another possible effect of a mutation is that it changes the binding affinity of the TF itself such as in Case 2 (Fig. 5C). This can happen if the mutation occurs in the coding region of the TF, and slightly alters the amino acid sequence of the TF. While the TF is still expressed at the same level, it may not have the same affinity for its targets. In this case, A* lasso will learn edges between the SNP and the target genes. However, no edges will be estimated between the TF and the targets because the expression of the TF and targets are no longer correlated. All these scenarios can be extracted directly from the network that we estimate, providing evidence for functional binding events.

We now compare the transcriptional network of TFs and targets that we learned with the network consisting of candidate edges from ChIP-Seq. Here, we focus on the network that was generated from applying A* lasso to one of the groups of genes in the data set. This group had 71 TFs, 203 targets genes and 6 other genes from the expression data for a total of 280 genes and 2,161 SNPs. Figure 6 shows a global view of the network that was generated. Each element in the figure at row i and column j shows the relationship between gene i and gene j. A white square denotes no relationship. If the square is gray, then gene i (which is always a transcription factor) binds to gene j according to ChIP-Seq experiments. Black indicates that gene i binds to gene j and there is a directed edge from gene i to gene j in the estimated network. As you can see from Figure 6, a sparse set of the binding events have been estimated as edges in our network suggesting that very few ChIP-Seq binding events are actually functional.

During processing of ChIP-Seq data, peaks are called and a signal value is assigned indicating average enrichment for the region. In order to test whether this signal value was related to the presence of a functional binding, we performed a 2-sample t-test comparing the signal values of all the gray squares indicating binding only, with the signal values of all the black squares corresponding to binding and a corresponding directed edge. The t-test did not reject the null hypothesis with p-value = 0.1903. This suggests that higher signal values do not necessarily indicate functional binding.

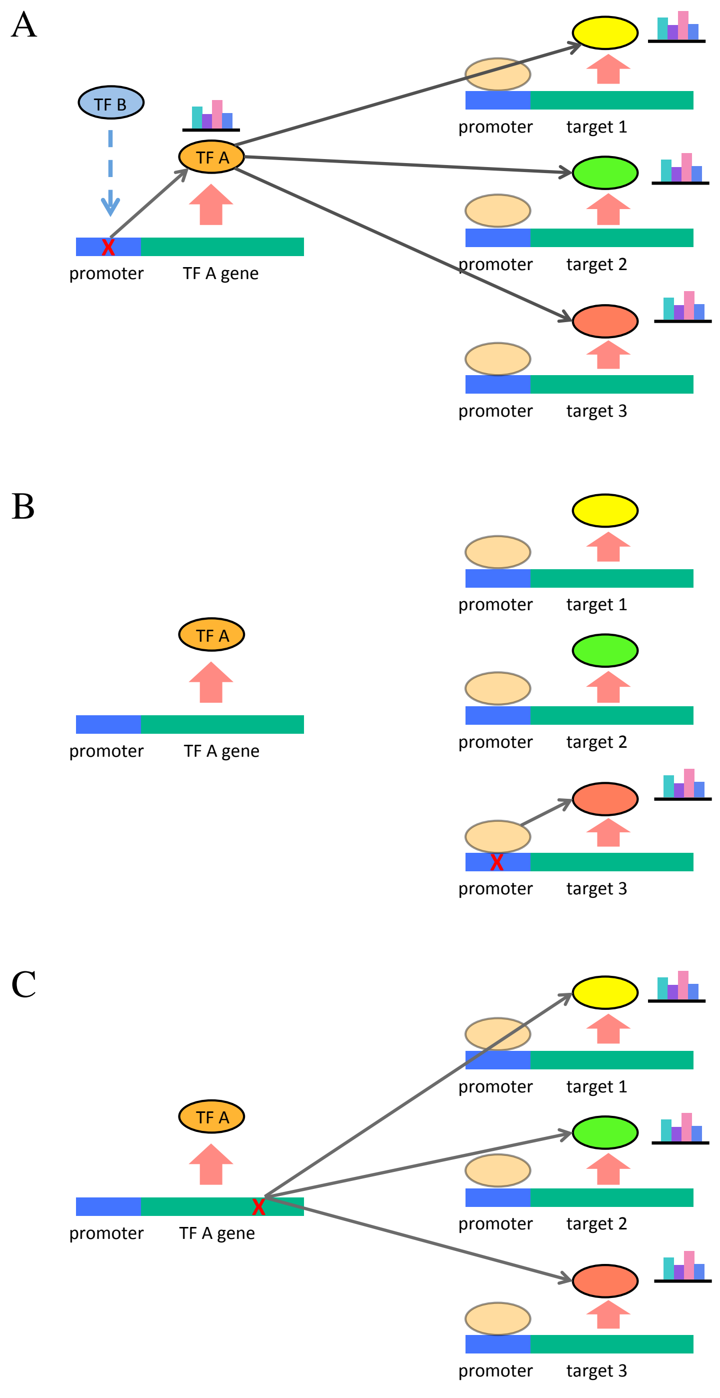

Figure 5: Scenarios illustrating how SNP perturbations can affect the expression of TFs and targets in the transcriptional network. A: A mutation in the promoter of TF A changes the ability of TF B to bind. This alters the concentration of TF A and this effect cascades to its target genes (Case 1a). B: The mutation occurs in the promoter of a target gene and thus only has a local effect on its associated gene (Case 1b). C: In this setting, the mutation occurs in the exon of the TF and affects the ability of that TF to bind to its targets. The expression of the targets are affected even though the expression of the TF remains stable (Case 2).

Figure 7 visualizes the SNPs influencing the transcriptional network. Similar to Figure 7, each element (i, j) shown as a black stripe denotes whether there is a directed edge from the SNP to the

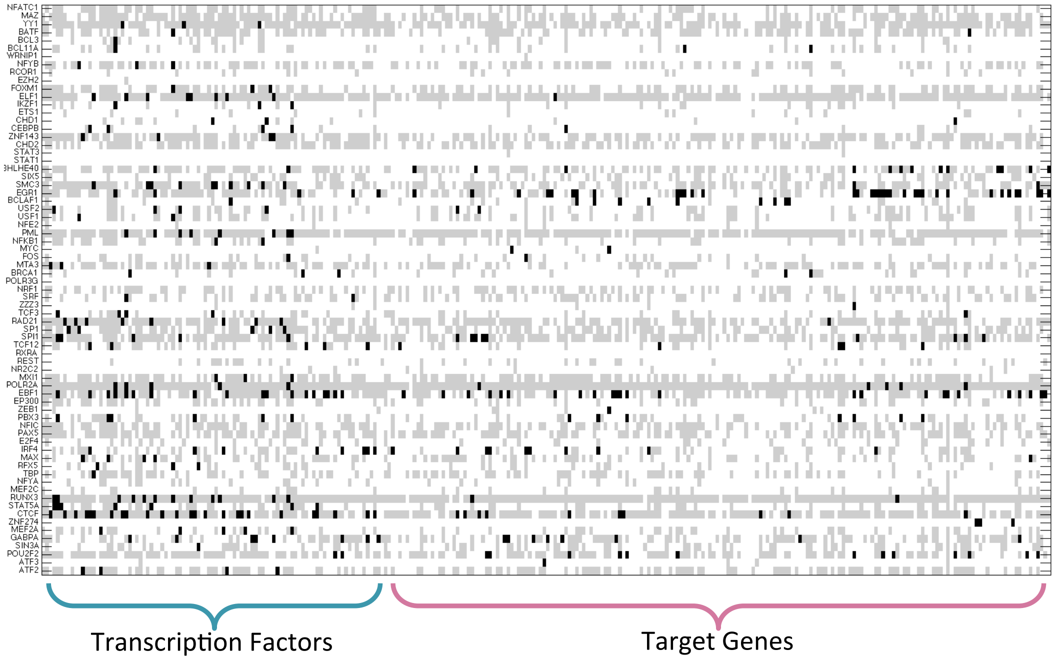

Figure 6: Estimated network of genes. The y-axis has transcription factor names. This figure shows the binding of these TFs to the targets on the x-axis. Note that targets can also be TFs as shown on the left.

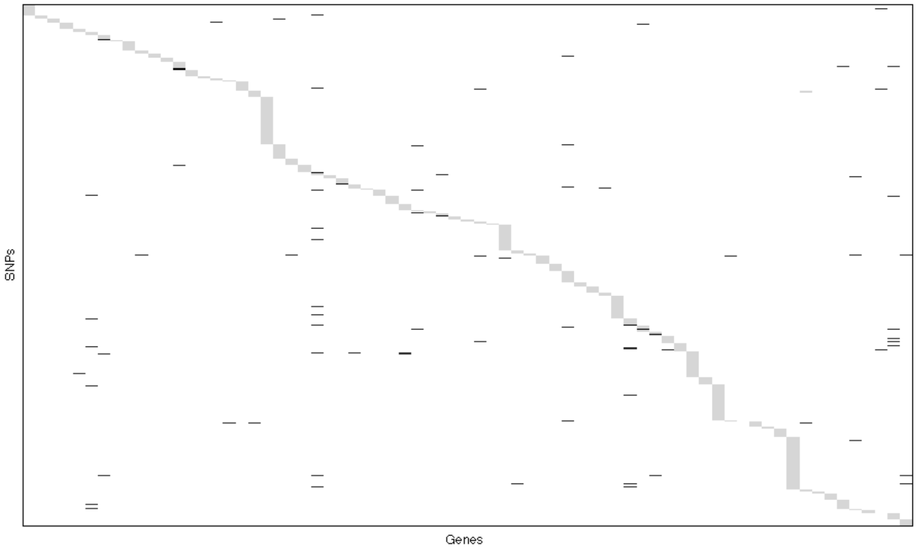

Figure 7: Estimated network of SNPs to genes.

target gene. If a SNP to target edge overlaps with the gray region, then it is a *cis* SNP which means that the SNP is close to the target gene. All other SNPs are *trans*, that is, they are located far away from the gene that it affects and potentially on another chromosome. In total there were 39 unique *cis* SNPs and 100 unique *trans* SNPs. We then focused on *cis* SNPs and examined where they were located. Figure 7 shows the distribution of where the SNPs are located with respect to their corresponding gene. SNPs that were classified as being in the promoter region were located within 2000 bp of the transcription start site. The majority of SNPs are located in the intergenic regions

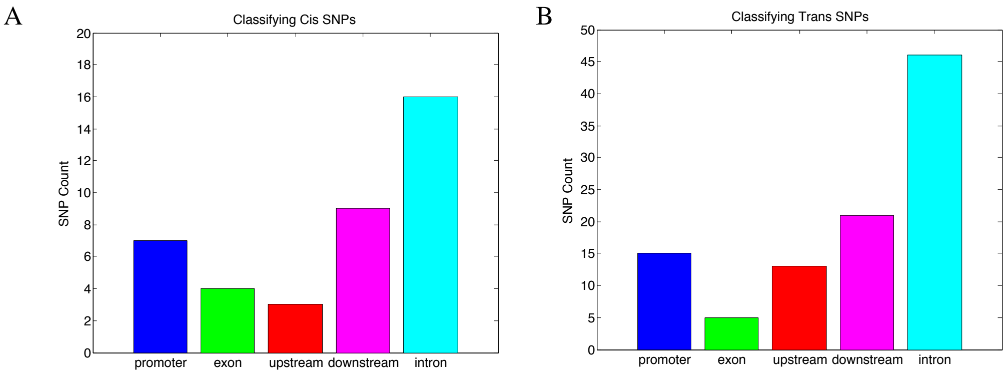

Figure 8: Distribution of the locations of *cis* SNPs and *trans* SNPs.

such as the promoter regions, introns, upstream and downstream and not in the coding regions called exons. This is to be expected since TFs generally bind to functional elements outside the exons. For the purposes of this study, we will focus on the promoter regions. However, TFs can also bind to enhancers which can be located upstream, downstream and within introns of the genes and is the subject of future work.

After examining the SNPs in the promoter regions, we found that 7 of them are associated with target genes (Table. 7), not transcription factors. They are examples of Case 1b) from Figure 5B where the SNP has a local effect on a particular gene. The other four SNPS are associated with TFs: EP300, ZEB1, TBP and EGR1. The first three have very few targets in the network. However, EGR1 has edges directed to 40 unique target genes in the network and we found that EGR1 actually binds to the promoter region in 55% of them. This shows that we can extract the scenario described in Case 1a (Fig. 5A) from the network. In addition, this suggests that this SNP affects the expression of EGR1 which then affects the expression of many genes downstream.

Chr	SNP ID	Associated target gene	Binding TF
12	rs2617834	YEATS4	MTA3
12	rs2617835	YEATS4	MTA3
4	rs769236	BBS7	POLR2A
22	rs61342075	EP300	POLR2A
10	rs1576050	ZEB1	POLR2A
6	rs2179373	TBP	SIN3A
5	rs3813321	EGR1	POLR2A

Table 4: *Cis* SNPs located in the promoter region of their associated target gene.

The *trans* SNPs were distributed similar to the *cis* SNPs. For *trans* SNPs, we are interested in those that occur in the exon of the associated gene which will always be a transcription factor in our setting. There were 5 SNPs discovered by the network. First, we had to confirm that the TF product of the associated gene actually binds to the target gene. Out of 5 SNPs, 4 SNPs satisfied this condition and are listed in Table 7. Thus, these SNPs fit the situation in Case 2 from Figure 5C where the mutation in the exon affects the affinity of the TF product causing a change in the expression of it's targets. Further analysis of whether this SNP is a synonymous or non-synonymous SNP should be carried out to define its function. Note that with the exception of TCF3 and EGR1, there are no edges between the TF and the target, only the SNP and the target. Thus, without estimating this network which detects SNP perturbations, this type of interaction could not be discovered. For instance, a correlation network of the expression data would miss these interactions.

In many cases, TFs do not act alone but regulate in pairs or small groups. That is, cofactors must be present for a functional binding to occur (Fig. 9). If we take the matrix shown in Figure 6 and cluster the rows and the columns, then we can identify TFs that work together as vertical blocks in the Figure 10. One interesting cofactor pair that we discovered is GABPA and EGR1. They both regulate the target genes GOLPH3, MAN1A2, and USP33. In addition, they have a common parent

SNP ID	Associated TF	target gene	Binding location on target gene
rs2000882	PAX5	USF1	promoter
rs2000882	PAX5	EGR1	upstream
rs2000882	PAX5	MYC	downstream
rs2000882	PAX5	B4GALT6	intron
rs41275834	TCF3	EGR1	promoter
rs41275834	IRF4	ATF3	intron
rs41275834	NFATC1	BCL11A	upstream

Table 5: *Trans* SNPs found in the exon of their associated TF.

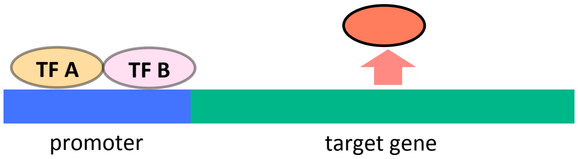

Figure 9: A pair of cofactors bind to activate transcription.

CTCF which can suggest an explanation of how their expression can be altered together. After examining the binding regions, we found that the two bind to the same region in each gene. For example, they both bind to the exon in GOLPH3, the promoter in MAN1A2 and upstream region in USP33. GOLPH3 encodes a peripheral membrane protein of the Golgi stack, the protein products of both MAN1A2 and USP22 also reside in the Golgi apparatus suggesting that these cofactors are collaborating to regulate protein trafficking in the Golgi. Thus, the network between genes is also useful in elucidating functional relationships between TFs and their targets.

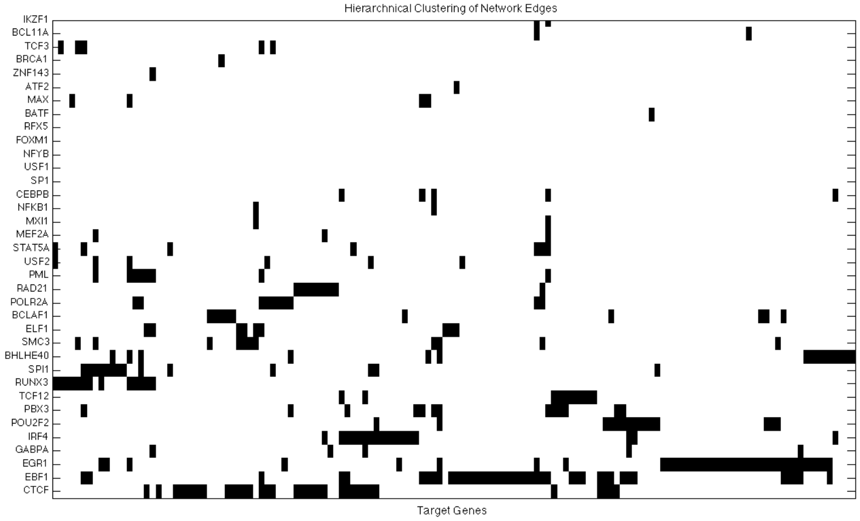

Figure 10: The edges between the TFs and the target genes are shown here after hierarchical clustering on both the x and y-axes.

8 Discussion and Conclusions

We have presented an approach to validating ChIP-Seq experiments whereby we estimate a Bayesian network that provides the evidence for functional bindings. By analyzing how SNPs interact with the genes in the network, we can establish which relationships are likely functional. In addition, we describe A* lasso which is a method to optimally estimate a sparse Bayesian network given data. To reduce computational time, we develop a heuristic scheme that allows network estimation to scale without severely compromising the quality of solutions. We extend the model to a conditional model which allows us to model SNPs as nodes that are always conditioned on.

In this work, we chose to model the transcriptional network using Bayesian networks because the interactions have a natural directionality. The edges that we learn from a TF to a target often match a binding event. In addition, estimated edges that occur from targets to other genes represent how targets influence downstream genes. Despite the limitation that Bayesian networks generally do not model feedback loops, they are still useful for constructing the transcriptional network that we need to resolve the functional binding events and their downstream effects.

We describe how to apply A* lasso to validating ChIP-seq and how to assimilate the binding information into the heuristic function, thereby guiding the search closer to the best network. We then present results of the network estimation and show how to extract subnetworks that validate ChIP-seq binding events. In our analysis, we found that very few of the candidate edges given by ChIP-seq were actually selected by our network. As the 1000 Genomes Project releases more samples as part of Phase III, we will probably be able to validate more candidate edges. Currently, with limited data, it is possible that important genetic variations are not represented in the population. In addition, because we filter out rare variants ($\text{MAF} < 0.05$), more samples will allow us to include a portion of these SNPs in our analysis.

Future work will carefully address enhancers, binding elements that are located far from the promoter region. We would also like to do motif analysis on the binding sites to assemble more supporting evidence. In addition, we will explore how to do inference on the model such that we can find indirect effects of the functional bindings.

References

[1] David Maxwell Chickering. Learning Bayesian networks is NP-complete. In *Learning from data*, pages 121–130. Springer, 1996.

[2] 1000 Genomes Project Consortium et al. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422):56–65, 2012.

[3] ENCODE Project Consortium et al. An integrated encyclopedia of dna elements in the human genome. *Nature*, 489(7414):57–74, 2012.

[4] Andrew G Fraser, Ravi S Kamath, Peder Zipperlen, Maruxa Martinez-Campos, Marc Sohrmann, and Julie Ahringer. Functional genomic analysis of c. elegans chromosome i by systematic rna interference. *Nature*, 408(6810):325–330, 2000.

[5] Nir Friedman, Iftach Nachman, and Dana Peér. Learning Bayesian network structure from massive datasets: the "Sparse Candidate" algorithm. In *Proceedings of the Fifteenth conference on Uncertainty in Artificial Intelligence*, pages 206–215. Morgan Kaufmann Publishers Inc., 1999.

[6] Wenjiang J Fu. Penalized regressions: the bridge versus the lasso. *Journal of Computational and Graphical Statistics*, 7(3):397–416, 1998.

[7] David Heckerman, Dan Geiger, and David M Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20(3):197–243, 1995.

[8] Da Wei Huang, Brad T Sherman, and Richard A Lempicki. Systematic and integrative analysis of large gene lists using david bioinformatics resources. *Nature protocols*, 4(1):44–57, 2008.

[9] Da Wei Huang, Brad T Sherman, and Richard A Lempicki. Bioinformatics enrichment tools: paths toward the comprehensive functional analysis of large gene lists. *Nucleic acids research*, 37(1):1–13, 2009.

[10] Shuai Huang, Jing Li, Jieping Ye, Adam Fleisher, Kewei Chen, Teresa Wu, and Eric Reiman. A sparse structure learning algorithm for Gaussian Bayesian network identification from high-dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1328–1342, 2013.

[11] Tommi Jaakkola, David Sontag, Amir Globerson, and Marina Meila. Learning Bayesian network structure using LP relaxations. In *Proceedings of the Thirteenth International Conference on Artificial intelligence and Statistics (AISTATS)*, 2010.

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

[13] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[14] Wai Lam and Fahiem Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational intelligence*, 10(3):269–293, 1994.

[15] Maxim Likhachev, Geoff Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems (NIPS)*, 16, 2003.

[16] Stephen B Montgomery, Micha Sammeth, Maria Gutierrez-Arcelus, Radoslaw P Lach, Catherine Ingle, James Nisbett, Roderic Guigo, and Emmanouil T Dermitzakis. Transcriptome genetics using second generation sequencing in a caucasian population. *Nature*, 464(7289):773–777, 2010.

[17] Aidas Nasevicius and Stephen C Ekker. Effective targeted gene knockdownin zebrafish. *Nature genetics*, 26(2):216–220, 2000.

[18] Jean-Philippe Pellet and André Elisseeff. Using Markov blankets for causal structure learning. *The Journal of Machine Learning Research*, 9:1295–1342, 2008.

[19] Dana Peer, Aviv Regev, Gal Elidan, and Nir Friedman. Inferring subnetworks from perturbed expression profiles. *Bioinformatics*, 17(suppl 1):S215–S224, 2001.

[20] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 74. Prentice hall Englewood Cliffs, 1995.

[21] Mark Schmidt, Alexandru Niculescu-Mizil, and Kevin Murphy. Learning graphical model structure using L1-regularization paths. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 1278, 2007.

[22] Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[23] Ajit Singh and Andrew Moore. Finding optimal Bayesian networks by dynamic programming. Technical Report 05-106, School of Computer Science, Carnegie Mellon University, 2005.

[24] Barbara E Stranger, Stephen B Montgomery, Antigone S Dimas, Leopold Parts, Oliver Stegle, Catherine E Ingle, Magda Sekowska, George Davey Smith, David Evans, Maria Gutierrez-Arcelus, et al. Patterns of cis regulatory variation in diverse human populations. *PLoS genetics*, 8(4):e1002639, 2012.

[25] Marc Teyssier and Daphne Koller. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *Proceedings of the Twentieth conference on Uncertainty in Artificial Intelligence*, pages 584–590, 2005.

[26] Ioannis Tsamardinos, Laura E Brown, and Constantin F Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.

[27] Ioannis Tsamardinos, Alexander Statnikov, Laura E Brown, and Constantin F Aliferis. Generating realistic large Bayesian networks by tiling. In *the Nineteenth International FLAIRS conference*, pages 592–597, 2006.

[28] Troy W Whitfield, Jie Wang, Patrick J Collins, E Christopher Partridge, Shelley Force Aldred, Nathan D Trinklein, Richard M Myers, Zhiping Weng, et al. Functional analysis of transcription factor binding sites in human promoters. *Genome Biol*, 13(9):R50, 2012.

[29] Changhe Yuan, Brandon Malone, and Xiaojian Wu. Learning optimal Bayesian networks using A* search. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, pages 2186–2191. AAAI Press, 2011.

[30] Xiang Zhang, Wei Cheng, Jennifer Listgarten, Carl Kadie, Shunping Huang, Wei Wang, and David Heckerman. Learning transcriptional regulatory relationships using sparse graphical models. *PloS one*, 7(5):e35762, 2012.