Fast Bayesian Network Structure Search Using Gaussian Processes

Blake Anderson and Terran Lane

University of New Mexico, Albuquerque NM 87131, USA

Abstract. In this paper we introduce two novel methods for performing Bayesian network structure search that make use of Gaussian Process regression. Using a relatively small number of samples from the posterior distribution of Bayesian networks, we are able to find an accurate function approximator based on Gaussian Processes. This allows us to remove our dependency on the data during the search and leads to massive speed improvements without sacrificing performance. We use our function approximator in the context of Hill Climbing, a local-score based search algorithm, and in the context of a global optimization technique based on response surfaces. We applied our methods to both synthetic and real data. Results show that we converge to networks of equal score to those found by traditional Hill Climbing, but at a fraction of the total time.

1 Introduction

Bayesian networks compactly represent a joint probability distribution by exploiting a set of independence assumptions between the set of random variables. A Bayesian network is a probabilistic graphical model where the random variables are represented as nodes and the conditional dependencies of the random variables are represented as edges. They are frequently used in bioinformatics, image processing, medical applications, and many other problem domains [1].

An important problem associated with Bayesian networks is *model selection*: given a dataset, find a structure that maximizes the posterior probability with respect to the data. Unfortunately, the space of all structures is super-exponential in the number of random variables and finding the optimal structure is known to be NP-complete [2].

Many methods have been devised to overcome the obstacle of searching through such a large space. These methods usually employ greedy methods to cut down on the size of the space [3] and rely on the decomposability of local scoring metrics such as the BDeu score [4]. With decomposable metrics, the chosen modification can be evaluated separately from the rest of the network. In traditional score-based search algorithms, the evaluation of the modification still requires looking at the entire dataset. Even with clever caching techniques, as the size of the dataset increases, the overhead incurred by scanning the data becomes very expensive.

We present two novel solutions to this problem. The first dramatically speeds up current score-based search algorithms without sacrificing performance by using a Gaussian Process to approximate the search landscape. With Gaussian Process regression, we aim to approximate a function, $f : \mathcal{G} \to \mathbb{R}$, capable of computing reasonable BDeu scores given the structure of the Bayesian network, $\mathcal{G}$. We begin constructing our function by sampling a constant number of structures from the super-exponential structure space. We calculate the exact BDeu score for each structure of this sample, giving us a set of structure/value pairs. We then use a Gaussian Process regressor with a graph kernel to learn a model of the function from structures to scores. Given this model, we can discard the training data during the iterations of the search algorithm. Our Gaussian Process based function approximator can easily be used in a wide variety of search algorithms to speed up the process of determining which step in the space to take. We illustrate this point by implementing the Hill Climbing algorithm [5] using our method and showing the corresponding speed-ups.

Our second approach is related to the response surface framework from optimization theory [6], in which one draws a sample of points from the objective function and then fits a regression model to that sample. Then one optimizes the response surface, which is typically more tractable to work with than the objective function. Using the mean-squared error of the response surface, we can locate the point on the surface that has the highest probability of having a better score than the current minimum Bayesian network score. We then find the true score of this point, update the response surface, and iterate. Response surfaces allow us to quickly find a global optimum using a sparse number of samples.

The use of a function approximator to model a Bayesian network scoring metric has been recently explored [7]. There, the authors construct a metagraph of Bayesian networks, and then use the eigenvectors of the metagraph to interpolate the scores of unknown networks, given the scores of a small number of sampled points. However, Yackley et al. show only that they can accurately approximate the true scoring function; they do not actually apply their function approximator to search. We go beyond that work by demonstrating the use of such approximators to dramatically speed up structure search.

There has been little work done to exploit Gaussian Processes to improve the different aspects of Bayesian networks. Gaussian Process networks is the only true prior instance of combining these two fields that we are aware of [8]. Most Bayesian networks based on continuous variables use Gaussian distributions to model continuous random variables. Gaussian Process networks were developed to cope with the problem that these networks can only learn linear dependencies in the data. A Gaussian Process prior was developed for the random variables which had the advantage of being able to learn functional dependencies between the data. The prior also allowed the marginal probability of the data to be computed in closed form.

2 Background

2.1 Bayesian Network Structure Search

Let $\mathbf{U} = \{X_1, \ldots, X_n\}$ be a set of random variables. A Bayesian network is a probabilistic graphical model that compactly encodes the joint probability distribution among these random variables. Let $B = \langle G, \Theta \rangle$ be a Bayesian network described by graph, G, and conditional probability distributions, Θ. G is a directed acyclic graph where the nodes represent the random variables of $\mathbf{U}$ and the edges represent statistical dependencies among the nodes. We can compactly represent the joint probability distribution because each node, X_i, is statistically independent of all other nodes in the graph given its parent set, $\mathbf{Pa}(X_i)$, with respect to G. Θ_i is the conditional probability distribution for X_i, which encodes the node's dependence on its parents. Given the independence assumption, we can write the joint probability distribution as

$$P_B(X_1, \ldots, X_n|\Theta_i) = \prod_{i=1}^{n} P_B(X_i|\mathbf{Pa}(X_i), \Theta_i) \tag{1}$$

We are interested in learning a $B = \langle G, \Theta \rangle$ that maximizes some score of the structure, given a dataset $\mathbf{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where each $\mathbf{x}$ is a vector of the values for X_i in the dataset. Typical scoring functions include BDeu [4], BIC [9], and MDL [10]. For simplicity, in this paper we use BDeu, though the basic framework here is independent of the scoring function.

Due to the independence assumptions of Bayesian networks, certain scoring functions are *decomposable*, meaning that the score of one family can be computed independently from the scores of the rest of the network. This property is used extensively by search algorithms, as they are able to compute the score of each change they consider efficiently. Structure learning is generally accomplished using a decomposable scoring function, such as BDeu, and local search strategies.

Perhaps the simplest local search strategy is Hill Climbing. This method starts with an empty graph, and then tries to find the one-step change, adding, reversing, or deleting an edge, that increases the BDeu score of the Bayesian network the most. There are several related approaches that improve upon this simple idea. These include Repeated Hill Climbing, which just has random restarts, and Look-Ahead Hill Climbing, which chooses a set of the best changes and tries to avoid local minima by looking ahead a certain number of steps to determine which is the best choice.

2.2 Gaussian Process Regression

A Gaussian Process represents a collection of random variables, in which any finite subset has a joint Gaussian distribution [11]. The entire Gaussian Process can be thought of as an infinite dimensional Gaussian

$$\mathcal{N}(\mu, \mathbf{K}) \tag{2}$$

where μ is the mean function and $\mathbf{K}$ is the covariance function.

The mean function is typically taken to be zero without loss of generality. The squared exponential is a typical covariance function for real-valued vectors

$$K(\mathbf{x}, \mathbf{x}') = \sigma_f^2 exp\left(-\frac{1}{2\mathbf{l}^2}|\mathbf{x} - \mathbf{x}'|^2\right) + \sigma_n^2 \delta_{xx'} \tag{3}$$

where σ_n^2, σ_f^2, and $\mathbf{l}$ are hyperparameters and δ is the Kronecker delta. The hyperparameters are usually set by maximizing the marginal likelihood through optimization techniques such as gradient descent.

Regression begins by building up the covariance matrix with all of the training data, $\mathbf{D} = \{\mathbf{X}, \mathbf{f}\}$. $\mathbf{X}$ consists of all the instances, $\mathbf{x}$, in the dataset, and $\mathbf{f}$ is the vector of labels for each $\mathbf{x}$. The covariance matrix is built such that each entry of the matrix is $K(\mathbf{x}, \mathbf{x}')$. Using this information, we can construct the joint probability distribution between our training data

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} = \mathcal{N}\left(\begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix}\right) \tag{4}$$

where $\mathbf{K}$ is the covariance matrix between datapoints in the training data, $\mathbf{K}_{**}$ is the covariance matrix between datapoints in the testing data, $\mathbf{K}_*$ is the covariance matrix between the training and testing data, and $\mathbf{f}_*$ is the vector of predicted values.

If we condition the joint probability distribution on the observed training data and assume noise-free observations, we have

$$\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{f} \sim \mathcal{N}\left(\mathbf{K}_*\mathbf{K}^{-1}\mathbf{f}, \mathbf{K}_{**} - \mathbf{K}_*\mathbf{K}^{-1}\mathbf{K}_*^T\right) \tag{5}$$

Therefore, we use the Gaussian Process to arrive at an estimate of our true function. The mean estimated values for our testing data, $\mathbf{X}_*$, are $\mathbf{K}_*\mathbf{K}^{-1}\mathbf{f}$.

3 Gaussian Process Structure Search

We illustrate two methods that take advantage of Gaussian Processes to decrease the time taken to find an optimal Bayesian network. The first method, Section 3.1, embeds a Gaussian Process with a fixed set of samples within the Hill Climbing structure search algorithm. The second method, Section 3.2, takes advantage of the Gaussian Process's error bounds to iteratively sample the structure space, where each new sample maximizes the probability to improve the current best score.

3.1 Greedy Search Based on Gaussian Process Regressor

To use Gaussian Processes in the context of a score-based search algorithms, we need a sampling strategy and a suitable kernel. Sections 3.1.1 and 3.1.2 present our solutions to these problems. Section 3.1.3 details how to incorporate Gaussian Processes within conventional Hill Climbing structure search.

Sampling the Structure Space The first component we need to address is how we choose to sample the space of Bayesian networks, which acts as our training data. The initial covariance matrix, $\mathbf{K}$, is calculated using these samples. We began by sampling from the structure space uniformly at random. Upon trying different sampling strategies, we found that by focusing on sparse Bayesian networks, we were able to more accurately capture the posterior probability. Currently, we focus on Bayesian networks that consist of a few family level dependencies per each training sample. In these networks, 3-10 nodes (depending on the number of random variables) are chosen at random. The parent sets of these nodes are composed of 3-6 randomly selected nodes that obey the acyclic constraint.

Through empirical investigation, we found that the number of samples needed to get a sufficiently conditioned covariance matrix seems to be quite small, although more work needs to be done to quantify this. For example, we observe high posterior probabilities for the networks that we find on synthetic datasets with only 100 samples (Section 4.3), which is much smaller than all possible structures (a super-exponential space).

The String Kernel We chose a variant of the string kernel for both its simplicity and reasonable initial results [12]. To formulate a Bayesian network in the form of a string, we let each edge in the network represent a character in our alphabet. Therefore, our kernel is

$$K(\mathbf{x}, \mathbf{x}') = \sum_{e \in E} w_e \phi_e(\mathbf{x}) \phi_e(\mathbf{x}') \tag{6}$$

where $\mathbf{x}$ and $\mathbf{x}'$ are the Bayesian networks we are trying to find the covariance of, E is the set of all possible edges, ϕ_e is a feature detector that returns 1 if edge e exists in the network and 0 otherwise, and $\mathbf{w}$ is the vector of hyperparameters for the covariance matrix.

To find $\mathbf{w}$, we maximize the log-likelihood of the training data with respect to $\mathbf{w}$. We first initialize the hyperparameters to reasonable random values. Then we use an optimization technique (here, gradient descent) to find a locally optimal set of hyperparameters. The details of finding a globally optimal set of hyperparameters is not important for our discussion, but has been explored extensively [11].

There has also been a great deal of work relating to graph kernels based on the Laplacian [13]. These Diffusion, or Heat, kernels are usually characterized as $K(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{|V|} (1 - \lambda_i)^k \phi_i(\mathbf{x}) \phi_i(\mathbf{x}')$ where $|V|$ is the size of the vertex set, λ_i is the ith eigenvalue, and $\phi_i(\mathbf{x})$ is the eigenvector associated with the ith eigenvalue. This approach tends to focus on the global smoothness constraints of the graph whereas the string kernel focuses on more localized graph features. Although preliminary results favored the simple string kernel, further investigation is needed to determine the efficacy of more general graph kernels in our approach.

```
**function:** Gaussian Process Structure Search
**input:** number of samples $n$, dataset $\mathbf{D}$,
        number of Random Variables $r$
**output:** high-scoring Bayesian network, *net*
1: $samples \leftarrow$ findSamples($n$, $r$)
2: $\mathbf{f} \leftarrow$ computeBDeuScores($samples$)
3: $\mathbf{K} \leftarrow$ findCovariance($samples$)
4: $\mathbf{w} \leftarrow$ optimizeParams($\mathbf{K}$, $\mathbf{f}$)
5: calculate $\mathbf{K}^{-1}\mathbf{f}$
6: $score \leftarrow$ initialize $net$
7: while not converged do
8:     $oSCs \leftarrow$ oneStepChanges
9:     $\mathbf{K}_* \leftarrow$ findCovariance($samples$, $oSCs$)
10:    $\mathbf{f}_* \leftarrow \mathbf{K}_*\mathbf{K}^{-1}\mathbf{f}$
11:    if max($\mathbf{f}_*$) $> score$
12:        add change to $net$
13:        update $score$
14: return $net$
```

Fig. 1. Pseudocode for Gaussian Process Structure Search using the Hill Climbing approach

The Gaussian Process Hill Climbing Algorithm We have currently implemented the Hill Climbing algorithm [5] using a Gaussian Process to approximate the local scores (Figure 1). We begin with a random, sparse Bayesian network. At each iteration of the algorithm, we compute all eligible one step changes, edge additions, edge reversals, and edge deletions. Then we use the function appromixator we learned from our Gaussian Process framework to select the change that will lead us to the greatest increase in the network's BDeu score.

From Equation (5), we see that the mean of the Gaussian Process is

$$\mathbf{f}_* \sim \mathbf{K}_*\mathbf{K}^{-1}\mathbf{f} \tag{7}$$

To compute $\mathbf{K}$, we first need to sample the space of Bayesian networks as described in 3.1.1. With these n samples, we create our $n \times n$ covariance matrix, $\mathbf{K}$, using Equation (6). To compute $\mathbf{f}$, we calculate the BDeu score of each Bayesian network from our pool of samples. As $\mathbf{K}$ and $\mathbf{f}$ are only based on our n samples, we can precompute $\mathbf{K}^{-1}\mathbf{f}$ for use in each iteration.

$\mathbf{K}_*$ is the covariance matrix between all of our one step changes and our n samples as computed by (6). Therefore, to get the approximate BDeu score of each one step change, we simply multiply $\mathbf{K}_*$ by the precomputed matrix $\mathbf{K}^{-1}\mathbf{f}$. Then we search $\mathbf{f}_*$ to find the highest score and add the corresponding change to the current Bayesian network. The algorithm stops once we are no longer able to find a change to the Bayesian network that increases the BDeu score.

3.2 Global Optimization of Score Function using Gaussian Processes

There has been a great deal of work looking at how response surfaces can model hard functions [6, 14]. The most straightforward method using these techniques is a two-step process. The first step involves fitting a response surface to the training data and then optimizing the parameters of the model. In the second step, we use the model constructed in step one to find the search point that contributes the most to the variance of the model, and then add that point to our training data.

Fit the Response Surface We begin by building our Gaussian Process with the data we have already observed. The log-likelihood of this data is

$$-\frac{n}{2}\log(\sigma^2) - \frac{1}{2}\log|\mathbf{K}| - \frac{(\mathbf{y}-\mathbf{1}\mu)^{\mathbf{T}}\mathbf{K}^{\mathbf{-1}}(\mathbf{y}-\mathbf{1}\mu)}{2\sigma^2} \tag{8}$$

We take the derivative of the log-likelihood with respect to the parameters of the covariance matrix. By setting this equal to 0 and solving, we can find the optimal values for our parameters. We find the optimal values for σ^2 and μ by taking the derivative of (8) with respect to σ^2 and μ and setting it equal to 0

$$\hat{\mu} = \frac{\mathbf{1}^{\mathbf{T}}\mathbf{K}^{\mathbf{-1}}\mathbf{y}}{\mathbf{1}^{\mathbf{T}}\mathbf{K}^{\mathbf{-1}}\mathbf{1}} \tag{9}$$

$$\hat{\sigma}^2 = \frac{(\mathbf{y}-\mathbf{1}\hat{\mu})^{\mathbf{T}}\mathbf{K}^{\mathbf{-1}}(\mathbf{y}-\mathbf{1}\hat{\mu})}{n} \tag{10}$$

The mean-squared error of our Gaussian Process at point $\mathbf{x}$, derived in [6] and omitted here for space constraints, is

$$e^2(\mathbf{x}) = \hat{\sigma}^{\mathbf{2}}\left[\mathbf{1} - \mathbf{K}_*^{\mathbf{T}}\mathbf{K}^{\mathbf{-1}}\mathbf{K}_* + \frac{(\mathbf{1}-\mathbf{K}_*^{\mathbf{T}}\mathbf{K}^{\mathbf{-1}}\mathbf{K}_*)}{\mathbf{1}^{\mathbf{T}}\mathbf{K}^{\mathbf{-1}}\mathbf{1}}\right] \tag{11}$$

The expected mean of any new point, $\mathbf{x}^*$, given by our Gaussian Process with mean μ is

$$\hat{y}(\mathbf{x}^*) = \hat{\mu} + \mathbf{K}_*^{\mathbf{T}}\mathbf{K}^{\mathbf{-1}}\mathbf{K}_* \tag{12}$$

Find New Minimum Having found all the parameters of our model, we can begin to search for new points that will give us the most information, thereby leading us to a new function minimum. We begin by defining T, the desired new local minimum

$$T = f_{min} - \kappa|f_{min}| \tag{13}$$

where f_{min} is the current minimum of sampled points, and κ is our improvement parameter. For each new point analyzed, we calculate the probability of improvement for that point, $\mathbf{x}$, as

$$\mathcal{P}_{improve}(\mathbf{x}) = \mathbf{\Phi}\left(\frac{T-\hat{y}(\mathbf{x})}{e(\mathbf{x})}\right) \tag{14}$$

Dataset	Number of Attributes	Number of Datapoints
colic	23	368
credit-g	21	1,000
hypothyroid	30	3,772
kr-vs-kp	37	3,196
letter	16	20,000
lymph	19	148
mushroom	22	8,124
segment	19	2,310
sick	30	3,772
splice	62	3,190

Table 1. UCI dataset statistics.

where Φ is defined as the normal cumulative distribution function.

We now maximize the probability of improvement, Equation (14), which allows us to avoid making numerous evaluations of our Bayesian network scoring function that is heavily data dependent. To maximize this function, we use a greedy strategy, looking at local changes to the network, evolving from the current minimum. The value of T plays an important role in this procedure. If T is set too low, predominantly local changes will be found that slightly improve the score. If T is set too high, the search will be excessively global and will become unstable.

4 Evaluation

In this section we empirically evaluate the ability of Gaussian Processes to aid in Bayesian network structure search. We show that using Gaussian Process function approximators provide a suitable alternative to current data-based scoring metrics. Our evaluation focuses on two axes of performance: the time until convergence and the BDeu score of the final network.

4.1 Environment

All experiments were run on an Intel Core 2 Quad CPU 2.40 GHz computer with 4 GB of memory under Ubuntu 8.04. We implemented all code within the Weka framework, version 3-4-13 [15]. When comparing the standard algorithm to the algorithm using Gaussian Processes, we changed as little code as possible.

4.2 Datasets

In our evaluation we used both synthetic datasets and standard datasets obtained from the UCI machine learning data repository [16]. The datasets from the UCI data repository include letters, lymph, and mushroom among others. These

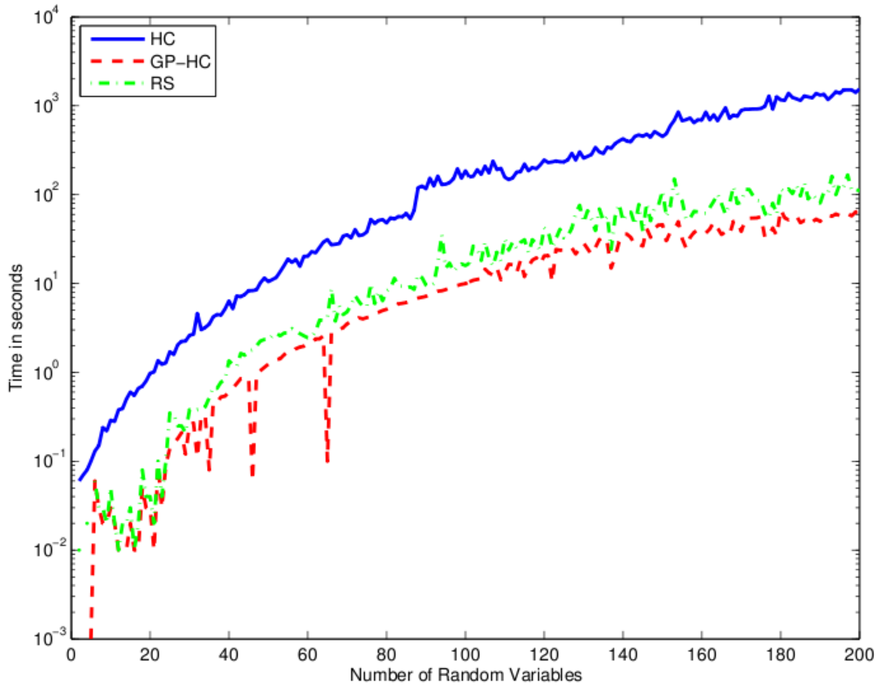

Fig. 2. Speed comparison between the native Hill Climbing algorithm (HC), Hill Climbing using Gaussian Processes (GPHC), and the response surface global optimization technique (RS). This is a semilog plot where the time is the seconds taken to build the model.

datasets range from 148 to 20,000 datapoints and 16 to 62 attributes and are detailed in Table 1. To discretize these datasets we used the Weka discretization filter which is based on Fayyad & Irani's MDL method [17].

The synthetic data is based on Bayesian networks ranging from 2 to 200 random variables generated by Weka. The structure of the networks is based on a randomly generated connected graph which obeys the acyclic constraint. Discrete conditional probability tables are then computed for the network based on a Dirichlet prior. The synthetic data we use is composed of 10,000 samples from these instantiated networks.

4.3 Results

We ran the response surface algorithm (RS), the Hill Climbing algorithm (HC) and its Gaussian Process counterpart (GPHC) on the synthetic datasets. The results of these experiments are illustrated in Figure 2 (time is presented using a log scale for convenience). We see that making use of Gaussian Processes to approximate the local scoring function and in the framework of global optimization based on response surfaces dramatically decreases the time of the search.

Number of Random Variables	HC Score	GP-HC Score	RS Score
120	-1356301 ($\pm$34187)	-1409523 ($\pm$28617)	**-1320017 ($\pm$32864)**
160	-1813199 ($\pm$43761)	-1847623 ($\pm$39149)	**-1752383 ($\pm$37488)**
200	-2249932 ($\pm$50558)	-2261139 ($\pm$46373)	**-2205866 ($\pm$48227)**

Table 2. Synthetic data results for the Hill Climbing and response surface algorithms on datasets that contain 120, 160, and 200 random variables. The log of the BDeu score with 95% confidence intervals are given. Winners are bolded.

The average time until convergence on a network with 10 random variables decreases from 0.29 seconds (HC) to 0.03 seconds (GPHC) and 0.08 seconds (RS). The effects are even more pronounced with networks of 200 random variables: 1683.81, 64.66, and 110.39 seconds for HC, GPHC, and RS, respectively. On these datasets, making use of Gaussian Processes dramatically increases the search speed.

To evaluate the performance of our methods in terms of the BDeu scores on the synthetic data, we took a closer look at the datasets that contained 120, 160, and 200 random variables. This allowed us to see how the performance of these methods scaled to datasets with a large number of random variables. We ran each algorithm on 10 different bootstrap samples of the original datasets. The results of these experiments are given in Table 2. The response surface method consistently wins. Unfortunately, GP-HC performs slightly worse than traditional hill climbing. These experiments support our hypothesis that our methods are competitive with traditional approaches on datasets with large numbers of random variables, and in the case of RS, actually outperform the traditional methods.

Next we looked at how the size of the dataset effects the speed of the algorithms. We used Weka to create a synthetic Bayesian network of 25 random variables and sampled datasets consisting of 50 to 10,000 datapoints in increments of 50 from this network. The results of these experiments are presented in Figure 3. The time it takes for the conventional Hill Climbing search algorithm to run scales roughly linearly with the data. Figure 3 shows that the time it takes for the Gaussian Process methods to run is relatively independent of the size of the dataset. It is interesting to note that on small datasets our Gaussian Process methods do run slower than the conventional version of Hill Climbing. We save time by avoiding looking at the data once we initialize our covariance matrix, but if the dataset is small, directly using the data is not as expensive. In this case, setting up our matrices and performing the matrix operations is more time consuming than traditional ways of computing local scores.

To evaluate how well our methods can search the structure space of Bayesian networks and find a high-scoring network, we used a subset of datasets from the UCI machine learning data repository. All the algorithms ran until convergence and the BDeu scores of the Bayesian networks found were then compared. The averaged results over 10 runs for the response surface and Hill Climbing approaches are listed in Table 3. For most of the datasets, the Gaussian Process

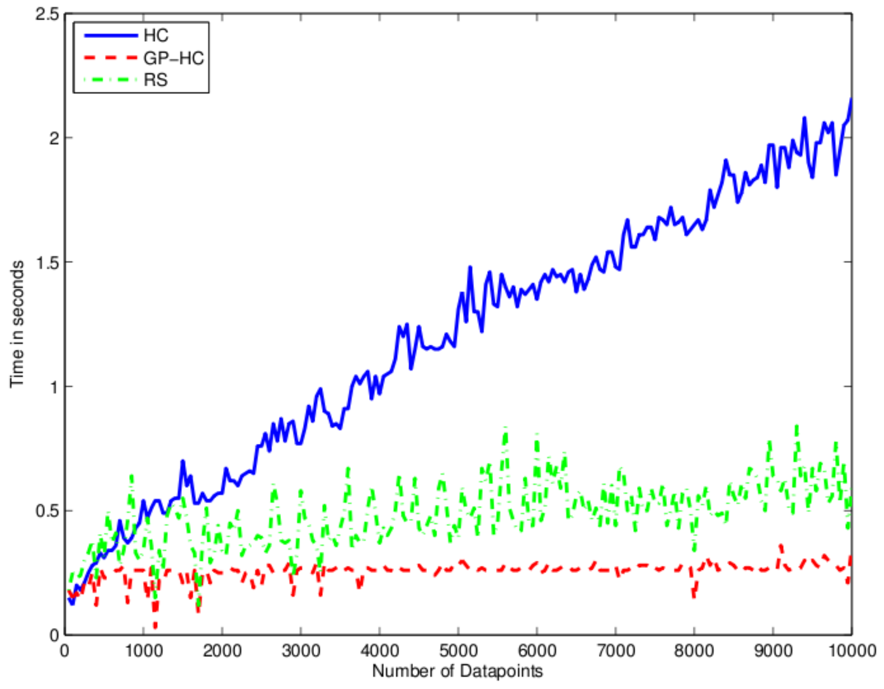

Fig. 3. Speed comparison between the native Hill Climbing algorithm (HC), Hill Climbing using Gaussian Processes (GPHC), and the response surface global optimization technique (RS) on datasets drawn from a Bayesian Network consisting of 25 random variables.

Hill Climbing and response surface algorithms found a better structure than the Hill Climbing algorithm which is quite promising. These results also reinforce the previous results regarding the speed of our method. In all datasets except for two, our methods are significantly faster than the Hill Climbing algorithm. The colic and lymph datasets, where we perform poorly with respect to time, are both extremely small datasets consisting of only 148 and 368 datapoints, respectively. Again, in these instances, the time it takes to perform the matrix operations dominates the time taken to directly compute the scores from the data.

We finally take an empirical look at the number of samples required to sufficiently initialize the covariance matrix of the Gaussian Process for GPHC. We use a dataset of 10,000 datapoints from a synthetic Bayesian network generated by Weka composed of 25 random variables. The Gaussian Process version of Hill Climbing is then used to evaluate the effect of the number of sampled networks. It is important to stress that only GP-HC uses a fixed number of samples and not the response surface method. RS is in some ways a smarter algorithm, as it searches the space and determines which samples and how many samples it

Dataset	HC Speed	GP-HC Speed	RS Speed
colic	**.24 (±.04)**	.45 (±.04)	.58 (±.07)
credit-g	.89 (±.26)	**.28 (±.02)**	.37 (±.03)
hypothyroid	1.35 (±.46)	.73 (±.11)	**.70 (±.08)**
kr-vs-kp	4.91 (±.59)	**1.23 (±.19)**	1.61 (±.24)
letter	11.13 (±2.51)	**.33 (±.01)**	.37 (±.02)
lymph	**.15 (±.03)**	.38 (±.13)	.46 (±.08)
mushroom	12.12 (±.96)	**.54 (±.03)**	.92 (±.05)
segment	1.45 (±.18)	**.40 (±.12)**	.52 (±.09)
sick	1.27 (±.21)	.39 (±.06)	**.37 (±.08)**
splice	16.84 (±1.37)	**.50 (±.21)**	1.02 (±.14)
Dataset	**HC Score**	**GP-HC Score**	**RS Score**
colic	-32.03 (±5.64)	-28.40 (±3.59)	**-27.64 (±3.22)**
credit-g	-15.14 (±2.85)	-15.05 (±2.11)	**-14.76 (±1.15)**
hypothyroid	-9.28 (±1.38)	**-7.69 (±1.13)**	-7.81 (±.98)
kr-vs-kp	-23.58 (±4.02)	-22.89 (±1.87)	**-22.59 (±2.57)**
letter	**-145.87 (±9.77)**	-150.59 (±7.65)	-146.13 (±6.34)
lymph	-21.16 (±3.19)	-21.23 (±2.48)	**-20.44 (±2.67)**
mushroom	**-162.18 (±13.58)**	-170.65 (±9.72)	-166.61 (±8.39)
segment	-50.76 (±4.91)	-51.32 (±3.87)	**-50.02 (±3.54)**
sick	-9.51 (±.74)	-8.02 (±1.28)	**-7.67 (±1.09)**
splice	-127.63 (±6.27)	-124.73 (±4.68)	**-123.55 (±3.78)**

Table 3. UCI data results for the Hill Climbing and response surface algorithms. All times are in seconds, the log of the BDeu score is per datapoint, 95% confidence intervals are given, and winners are bolded.

needs. The algorithm was run 10 times for each number of samples with samples ranging from 10 to 500 in intervals of 10. The average score over 10 runs for each number of samples is plotted in Figure 4. The average score stabilizes around 100 samples and does not significantly improve with additional samples. This behavior of needing relatively few samples until the algorithm becomes stable has been observed in a variety of datasets, but more work needs to be done to quantify an optimal number of samples for a given dataset.

5 Conclusions

Finding a Bayesian network that maximizes the posterior probability of the data is a very important problem used in many domains. Many methods rely on decomposable local scoring metrics, such as BDeu, to search through the space of structures. Although these algorithms make structure search tractable, they are still quite slow and do not scale well with the size of the dataset and the number of random variables in the network.

We have presented two methods to speed up score-based search algorithms based on Gaussian Process regression. We have used Gaussian Process regression to approximate the scoring function, $f : \mathcal{G} \rightarrow \mathbb{R}$. To our knowledge, this is the

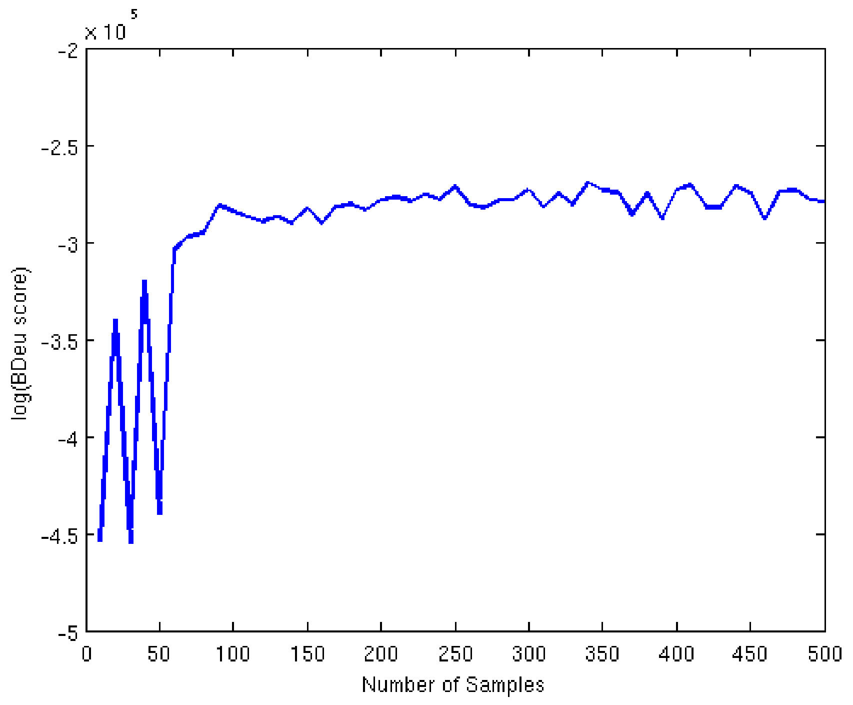

Fig. 4. The effect of the number of sampled networks on the BDeu score of the final network. All experiments were performed on a dataset of 10,000 datapoints drawn from a synthetic Bayesian network consisting of 25 random variables.

first application of using a function approximation technique to approximate local scores within the framework of Bayesian network structure search. The BDeu function approximator has allowed us to remove the dependency on the data during the search and dramatically increase the speed of the search algorithm without sacrificing performance. The second method uses the machinery of response surfaces to iteratively find new Bayesian networks from the structure space based on the error bounds of the Gaussian Process. We have shown results applying these procedures on both synthetic and real data.

Because our approaches do not reference the data during the search, they would be particularly suited for domains with extremely large datasets such as astronomical data [18]. Our methods could also be used to quickly generate sample networks for use in other methods such as MCMC [19].

There are several avenues one can take to extend this research to find more accurate approximations of scoring metrics. The string kernel was mainly used for its simplicity in defining the covariance between graphs. Within the field of spectral graph theory, more sophisticated graph kernels have been explored [13]. We have not pursued this class of kernels because our initial set of experiments with them led to inferior results. We suspect these results to be a by-product of the kernel trying to find global similarities between the graphs. The local nature of the string kernel seems to be better at capturing the different aspects of the Bayesian network with respect to the BDeu score than the general Laplacian-based graph kernel. More work needs to be done to understand why exactly a general graph kernel fails and if these problems can be overcome so that we can make use of these powerful techniques.

It would be helpful to have a more mathematical explanation of the number of samples needed to properly condition the covariance matrix of the Gaussian Process associated with GPHC. In all of our experiments we used 100 samples, this number being derived from empirical evidence over several Bayesian networks of differing size (Figure 4). There is also more work needed to find an optimal sampling strategy for GPHC. We tried numerous methods and settled on the approach described in 3.1.1 based on empirical results. It is very possible that given a superior sampling strategy, we could get away with using far fewer samples.

As the number of samples we take from the Bayesian network structure space increases, we run the risk of slow computation as the cost of inverting $\mathbf{K}$ is $\mathcal{O}(n^3)$. To combat this, it is possible to use the quasi-Newton BFGS method to invert our covariance matrix in $\mathcal{O}(n^2)$ [20]. Making use of this method would increase the speed of our methods as they stand, and allow for the scaling of our methods to higher numbers of samples.

Acknowledgments

This work was supported by the National Science Foundation under NSF grant number IIS-0705681.

References

1. Heckerman, D., Mamdani, A., Wellman, M.P.: Real-world applications of bayesian networks. Communications of the ACM **38**(3) (1995) 24–26
2. Chickering, D.M.: Learning bayesian networks is np-complete. In: Learning from Data: Artificial Intelligence and Statistics V. (1996) 121–130
3. Friedman, N., Goldszmidt, M.: Building classifiers using bayesian networks. In: In Proceedings of the Thirteenth National Conference on Artificial Ontelligence. (1996) 1277–1284
4. Heckerman, D., Geiger, D., Chickering, D.M.: Learning bayesian networks: The combination of knowledge and statistical data. Technical Report MSR-TR-94-09, Microsoft Research (1994)
5. Friedman, N., Nachman, I., Peer, D.: Learning bayesian network structure from massive datasets: The sparse candidate algorithm. In: In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence. (1999) 206–215
6. Jones, D.R.: A taxonomy of global optimization methods based on response surfaces. Journal of Global Optimization **21** (2001) 345–383

7. Yackley, B., Corona, E., Lane, T.: Bayesian network score approximation using a metagraph kernel. In: In Advances in Neural Information Processing Systems. Volume 20. (2008)
8. Friedman, N., Nachman, I.: Gaussian process networks. In: In Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence. (2000) 211–219
9. Chickering, D.M., Heckerman, D.: Efficient approximations for the marginal likelihood of bayesian networks with hidden variables. In: Machine Learning. (1997) 181–212
10. Lam, W., Bacchus, F.: Learning bayesian belief networks: An approach based on the mdl principle. Computational Intelligence **10** (1994) 269–293
11. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. The MIT Press (2006)
12. Lodhi, H., Saunders, C., Cristianini, N., Watkins, C., Scholkopf, B.: Text classification using string kernels. Journal of Machine Learning Research **2** (2002) 563–569
13. Kondor, R.I., Lafferty, J.: Diffusion kernels on graphs and other discrete structures. In: In Proceedings of the Ninteenth International Conference on Machine Learning. (2002) 315–322
14. Osborne, M.A., Garnett, R., Roberts, S.J.: Gaussian processes for global optimization. In: In Learning and Intelligent Optimization Conference. Volume 3. (2009)
15. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd Edition. Morgan Kaufmann, San Francisco (2005)
16. Asuncion, A., Newman, D.: UCI machine learning repository (2007)
17. Fayyad, U., Irani, K.: Multi-interval discretization of continuous-valued attributes for classification learning. In: In Proceedings of the International Joint Conference on Uncertainty in AI. (1993) 1022–1027
18. Tegmark, M., Taylor, A.N., Heavens, A.F., Ring, F., D-Munchen: Karhunen-loeve eigenvalue problems in cosmology: How should we tackle large datasets. The Astrophysical Journal **480** (1997) 2–2
19. Friedman, N., Koller, D.: Being bayesian about network structure. In: Machine Learning. (2000) 201–210
20. Leithead, W.E., Zhang, Y.: O(n2)-operation approximation of covariance matrix inverse in gaussian process regression based on quasi-newton bfgs method. Communications in Statistics-Simulation and Computation **36** (2007) 367–380