Fast Bayesian Variable Selection Algorithms for High Dimensional Genomics Data

Wellington Cabrera¹, Carlos Ordonez¹*, David Sergio Matusevich¹ and Veerabhadran Baladandayuthapani²

Abstract

Large-scale molecular genomics assays such as gene and microRNA expression produce high-dimensional data where the number of features (variables) greatly exceed the sample size. One of the fundamental scientific questions of interest is to select important markers that affect relevant clinical outcomes – thus providing insights into the development and prognosis of many diseases especially cancer. However, the solution requires exploring a high-dimensional combinatorial search space especially when considering multiple platforms and their interactions. In this work we study the solution of variable selection with well-known Bayesian methods, which often require thousands of MCMC iterations. We present several algorithmic optimizations to accelerate the MCMC method to make it work efficiently inside a database system. Our optimizations include sufficient statistics, variable preselection, hash tables and calling a linear algebra library. We motivate and apply our methods to integrative analysis of multi-platform genomics data in brain cancer, where-in we integrate gene and microRNA expression datasets to predict survival times of patients. Our methods produce sparse sets of genes and microRNAs that could be potentially related to the development of the disease. We also show our DBMS-based algorithm is orders of magnitude faster than the R statistical package. Our work shows a DBMS is a promising platform to solve difficult statistics problems especially in the genomics arena.

Keywords: algorithms; MCMC; variable selection; DBMS; Microarray

Introduction

Most diseases (especially cancer) consists of changes at multiple molecular levels that drive the disease development and progression. Advances in biomedical technology has now enabled researchers to simultaneously measure the molecular activity from diverse platforms such as gene (mRNA) and microRNA(miRNA) assays. Each of these platforms provide markers that contain complimentary information about the development of the disease. The prediction of clinical outcomes or phenotypes based on such markers is one of most pressing problems in cancer genomics today– since these markers could aid development of future targeted therapies based on the genomic footprint.

Towards this end, researchers are interested in finding small (sparse) sets of variables (genes/miRNA) that affect an outcome of interest (e.g survival times of patients) with enough accuracy. Although complex models could have better in-sample error measurements than smaller models, complex models are have a proclivity to over-fitting and are prone to a large generalization errors [1, 2]. In contrast, smaller
Typical data mining problems, such as those found in marketing or web data, generally consist of a very large number of samples, \( n \), and relatively fewer dimensions, \( d \), when compared to \( n \). Genomics data, on the other hand, typically consist of few hundreds samples (patients) and thousands or tens of thousands of variables (genes). This means that genomics data analysis is an instance of what is commonly known as the “small \( n \) large \( d \)” problem. Several data mining techniques can be applied to genomics data, such as Regression \([4]\), Classification \([5]\) and Clustering \([6]\), however the data analysis of a small sample size in a high dimensional space remains a difficult problem \([1]\).

Several technical and computational issues arise when dealing with a high-\( d \) variable selection problem. Tens of thousands of dimensions could result in very large data structures, such as matrices, that use a very large portion of the available memory, possibly requiring more space than what is available to the user. For example, a typical covariance matrix could have more than 140 million elements. Furthermore, MCMC methods, such as the one we present in this work, require the computation of \( 2^d \) posterior probabilities, each of them requiring a different covariance matrix to be inverted, as well as other linear algebra operations. \([c]\)

Database Management Systems (DBMS) are a common repository of high dimensional data sets in hospitals. The analysis of such data sets inside a DBMS provides key benefits such as querying, more efficient processing, security and data integrity. However, DBMSs are architected for query and transaction processing, rather than matrix computations, and numerical algorithms, which presents another challenge to high-\( d \) Bayesian variable selection, apart from the increase in time complexity.

In this work we propose, analyze and evaluate a variable selection algorithm for linear regression on high dimensional genomics data sets that aims to produce parsimonious models (few variables selected), using Gibbs Sampling and introducing a probability prior on the size of the model, fully integrated to the DBMS. We analyze a high dimensional data set of Glioblastoma Multiforme (brain tumors) patients, consisting of gene (mRNA) and microRNA expression to predict a patient’s survival time after diagnosis.

We organized this article in the following manner. We first provide some background information to the reader regarding the methods used as well as the notation used throughout the paper. Then we describe the algorithm and the optimizations used to implement it as part of a database. We describe the experiments performed and we compare the results to the literature. We conclude this paper with a brief survey of related work and our conclusions.

---

[a] needs more justification why this is a data mining problem
[b] Replace microarray with genomics all throughout to make it more general. microarray is specific to expression but in this paper you have both mRNA and miRNA
[c] I think we need a few sentences here about the basic Bayesian variable selection technique here to just to lay the context. xAlso related work can come here rather than at the end. See later footnote.
Background

In this section we will define several terms related to the problem as well as the main steps of the algorithm.

Linear Regression

Linear regression is a statistical model that replicates the relationship between a scalar dependent variable and a set of explanatory variables (or independent variables). Let \( X = \{x_1, \ldots, x_n\} \) be a set of \( n \) data points with \( d \) explanatory variables and \( Y = \{y_1, \ldots, y_n\} \) be a set of numbers such that each \( x_i \) is associated with \( y_i \). If we represent the data set by matrices \( Y \ (1 \times n) \) and \( X \ (d \times n) \), the linear regression model can be expressed as:

\[
Y = \beta^T X + \epsilon
\]

(1)

where \( \beta = [\beta_0, \ldots, \beta_d] \) is the vector of regression coefficients; \( \epsilon \) represents the error and has a Gaussian distribution and \( X \) is \( X \) augmented with a row of \( n \) 1s stored in an extra dimension \( X_0 \). The vector \( \beta \) is usually estimated using the ordinary least squares method. Citation

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Dimensions</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Y )</td>
<td>1 ( \times ) ( n )</td>
<td>Dependant Variable</td>
</tr>
<tr>
<td>( X )</td>
<td>( d \ \times \ n )</td>
<td>Independent Variable</td>
</tr>
<tr>
<td>( \beta )</td>
<td>( (d + 1) \ \times \ n )</td>
<td>Regression Coefficients</td>
</tr>
<tr>
<td>( \beta )</td>
<td>( (d + 1) \ \times \ 1 )</td>
<td></td>
</tr>
</tbody>
</table>

Variable Selection

Variable selection is the search for the best subsets of variables that are good predictors of \( Y \) (citations??). The assumption of this search is that the data contains features that are redundant and can be safely excluded from the model, since they provide little relevant information. There are many reasons to undertake this search:

- To express the relationship between the dependent variables and the explanatory variables in the simplest way possible [7]
- To identify the which variables are important and which are negligible predictors [7]
- To facilitate data visualization and interpretation [8]

Finding which subsets are the most appropriate is not computationally trivial, since there are \( 2^d \) potential combinations of variables. Evidently, this problem cannot be solved using a brute force method when the number of dimensions is high. Traditional greedy methods (stepwise selection, for instance (citations??)) focus on finding the best subset of variables, but the probabilistic Bayesian approach aims to identify several promising subsets of variables. The set of selected variables can be represented by a \( d \)-dimensional vector \( \gamma \in \{0, 1\}^d \), such that \( \gamma_j = 1 \) if the variable \( j \) is selected and \( \gamma_j = 0 \) otherwise. We denote by \( M_\gamma \) the model that selects \( k \) of the \( d \) variables, corresponding to the vector \( \gamma \). Given \( \gamma \) we can easily find how many variables were selected by performing the dot product \( k = \gamma^T \cdot \gamma \). Throughout this paper we will use \( \gamma \) as an index on selected variables such as \( \beta_\gamma \) and \( X_\gamma \).
As mentioned before, an exhaustive search on the $2^d$ subsets of variables is impractical for even a moderately large $d$. While many techniques have been developed to deal with such a problem, a Bayesian approach provides information about the prior probabilities as an added bonus. Since it is very difficult to produce a solution to such a problem, several authors [9, 7], exploit Markov chain Monte Carlo techniques, such as the Metropolis-Hastings Method and the Gibbs sampler, to obtain accurate results and to sample from the posterior distributions.

The Gibbs sampler, the approach taken in this work, uses the posterior probability $\pi(\gamma|X,Y)$ as a criterion for the selection of promising sets of variables. The Gibbs sampler is a Markov chain Monte Carlo method to obtain a sequence of observations approximated from the posterior probability. This sequence of observations is characterized by a vector $\gamma[i]$, that describes the variables selected at iteration $i$ of the sequence.

$\gamma[i]$ is obtained from the previous vector, $\gamma[i-1]$, as follows: For every variable $x_j$ the normalized probabilities of $p(\gamma[i]_j = 0)$ and $p(\gamma[i]_j = 1)$ are calculated. Based on these probabilities either $\gamma[i]_j = 0$ or $\gamma[i]_j = 1$ is chosen by sampling and the $j$ position of vector $\gamma[i]$ is updated. After $N$ iterations we obtain the Markov chain sequence $\gamma[0], \ldots, \gamma[N]$. To avoid introducing a bias into the prior computation due to initial instabilities, the first $B$ iterations of the Markov chain are usually discarded. This is called the burn in period. After the burn in, it is assumed that we have reached a stable distribution (the process becomes ergodic) and we can safely sample the priors.

We base the computation of $\pi(\gamma|X,Y)$ on the Zellner G-prior. Zellner’s G-prior relies on a conditional Gaussian prior for $\beta$ and an improper (Jeffreys) prior for $\sigma^2$ [10], with parameters $c$ and $\tilde{\beta}$.

$$
\beta|\sigma^2, X \sim N_{k+1}(\tilde{\beta}, c\sigma^2 (XX^T)^{-1}) \\
\sigma^2 \sim \pi(\sigma^2|X) \propto \sigma^{-2}
$$

In the literature the most common prior probability used for $\gamma$ is the uniform prior $\pi(\gamma|X) = 2^{-d}$ [9, 10]. In contrast, since we aim to find parsimonious models (fewer variables selected is better) we use a prior that gives less weight to models with many variables: $\pi(\gamma|X) \propto \exp(-\alpha k)$, where $k$ is the number of variables selected by the model, and calculated from $\gamma$ as previously explained. We can use the parameter $\alpha \geq 0$ to fine tune the influence of the size of the model in the posterior probability. It is important to mention that the case $\alpha = 0$ reverts $\pi$ to the uniform prior.

Under Zellner’s prior the parameter $c$ also influences the number of variables selected by the model: large $c$ values lead to parsimonious models, whereas small values of $c$ promote saturated models [11]. Since we aim to solve problems with very large dimensionality ($d \gg n$) we will use a high $c$ value, in order to select a smaller set of variables. We calculate the posterior probability $\pi(\gamma|X,Y)$ using the marginalized posterior distribution [10]:

$$
\pi(\gamma|X,Y) \propto (c + 1)^{\frac{-n+\frac{k}{2}}{2}} \left(YY^T - \frac{c}{c+1} YX_{\gamma}(X_{\gamma}X_{\gamma}^T)^{-1}X_{\gamma}^T Y^T - \frac{1}{c+1} \tilde{\beta}^T X_{\gamma}X_{\gamma}^T \tilde{\beta}\right)^{-n/2} \exp(-\alpha k)
$$
High-D Bayesian Variable Selection

In this section we review our algorithm’s main steps: Variable Pre-selection and Gibbs Sampling.

Variable Pre-Selection
As previously explained, very high dimensional data sets results on extremely large number of floating point operations and big matrices that don’t fit in memory. Currently we can deal with high-\(d\) data sets more easily by using variable pre-selection. This is an effective strategy to deal with high dimensionality and reduces the computational burden significantly [12]. Several authors have proposed the pre-selection of variables based on the marginal correlation rankings of the features [12, 13, 8].

Our algorithm makes an initial pre-selection of variables from the original very high dimensional model \(M\) to a high dimensional model \(M'\) with dimensions \(d < p\), more suitable for computation. In order to ease the computational complexity of the problem, our experiments reduce the original dimensionality between four and ten times.

The initial pre-selection is performed by Correlation Ranking. This consists of calculating the correlation between each dependent variable \(X_i\) and the independent variable \(Y\). After all the correlations are calculated, they are sorted in descending order (ranking) and the top \(d\) ranked variables are chosen for the new, reduced, model \(M\). Explain preselection by beta

Gibbs Sampler for \(d \gg n\)
The Gibbs sampler step requires presetting the following parameters:

\[
N : \text{Number of iterations} \\
B : \text{Number of burn in iterations} \\
c : \text{Parameter of Zellner’s prior} \\
\beta : \text{Something}
\]

We reuse the correlation ranking we calculated for the preselection to initialize the vector \(\gamma^0\). The variables chosen as the initial set are those having a correlation greater than a threshold defined by the user. These variables are the starting point for the Gibbs sampler. All variables are initialized with \(\tilde{\beta} = 0\).

As in any Monte Carlo algorithm, the first few iterations while the model is stabilizing are not useful from a statistical point of view. Therefore we set the parameter \(B\), to specify how many iterations are discarded (burn-in), as previously stated. It is important to mention that if the Markov Chain was infinite \((N \rightarrow \infty)\) the effects of the burn in period would be completely washed away. However we cannot discount the effect of the burn in period since we are dealing with finite chains (even if they are very long). The parameters \(N\) and \(B\) refer to the total number of iterations the process will run and the number of iterations that will be discarded as part of the burn-in period.

As mentioned above, the number of variables selected \(k\) can be calculated by performing the dot product \(k = \gamma^T \cdot \gamma\). If \(n\) is the number of data points in the data
set, when we run the Gibbs sampler in a sub-model, at every iteration the algorithm requires that \( k < n \), to avoid the possibility of the matrix \( XX^T \) being singular. This can be explained by realizing that we are dealing with an under-determined system of linear equations. In such a system, if the number of variables is equal or larger than the number of equations, the linear regression will give unrealistic results. Furthermore, for performance reasons it is better to keep \( k < 100 \).

When we calculate the posterior probability for \( \gamma_j = 0 \) and \( \gamma_j = 1 \) we need to perform matrix operations, the most time consuming of which is the matrix inversion \( (XX^T)^{-1} \). In order to optimize the execution time, these operations are performed using LAPACK.

**Optimizing the Gibbs Sampler for \( d \gg n \) in a DBMS**

In this work we will assume that the data set is stored in a table. The dimensions (features) are stored as rows and each column is a distinct data point. This is the natural storage layout for the case \( d \gg n \). As an example, data sets containing genetic expressions can contain more than ten thousand dimensions, but only a few hundred observations; this is due to the expense of collecting data from human beings.

**Sufficient Statistics**

The efficient computation of statistical models relies on using sufficient statistics, as described in [14]. Sufficient statistics are defined as follows:

\[
\begin{align*}
    n &= |X| \quad (4) \\
    L &= \sum_{i=1}^{n} x_i \quad (5) \\
    Q &= XX^T = \sum_{i=1}^{n} x_i x_i^T \quad (6)
\end{align*}
\]

For computing linear models we consider an extension of sufficient statistics. Let \( X = [1, X] \) be the augmented \( X \) matrix; since \( X \) is a \( d \times n \) matrix, \( X \) is a \( (d+1) \times n \) matrix such that:

\[
X_{i,j} = \begin{cases} 
1 & \text{if } i = 0 \\
X_{i,j} & \text{if } i > 0 
\end{cases} \quad (7)
\]

and let \( Z = [X, Y] \) be the \( (d + 2) \times n \) matrix where we store the augmented \( X \) matrix and \( Y \):

\[
Z_{i,j} = \begin{cases} 
X_{i,j} & \text{if } i < d + 1 \\
Y_j & \text{if } i = d + 1 
\end{cases} \quad (8)
\]
Γ = ZZ^T contains a summary of the whole data set, sufficient to compute the linear model.

\[
\Gamma = \begin{bmatrix}
    \sum x_i^2 & \sum x_i y_i \\
    \sum x_i y_i & \sum y_i^2
\end{bmatrix}
\]

\[
= \begin{bmatrix}
    \sum x_i^2 & \sum x_i y_i \\
    \sum x_i y_i & \sum y_i^2
\end{bmatrix}
\]

Notice that \( XX^T, YX^T \) and \( YY^T \) can be calculated once, in one pass, at the beginning of the algorithm. Furthermore from equation 9 we can see that \( \Gamma \) is symmetric: \( \Gamma = \Gamma^T \). This has important implications both on the issue of storage as well as the simplification of some of our computations.

We will use the sufficient statistics by Equations 5, 6 and 4 in order to avoid reading the data set on each iteration. The data set \( X \) is loaded in RAM and then our algorithm computes \( n, L \) and \( YX^T \) in one pass. These matrices are then stored in memory and available at any time during the process. However since \( d \gg n \), \( Q \) is a very large matrix, and storing it in memory could lead to memory overflows. In each iteration, a projection of \( Q \) is required: We need a projection of \( Q \) formed of just the variables currently selected in the vector \( \gamma \). Therefore, we calculate \( Q = XX^T \) on demand: if \( Q_{i,j} \) is required in the computation, it is calculated at such a time.

We can derive the posterior of \( \pi(\gamma|X,Y) \) as an expression dependent on \( n, L, Q \) and \( YX^T \):

\[
\pi(\gamma|X,Y) \propto (c + 1)^{-\frac{n+1}{2}} \left( YY^T - \frac{c}{c+1} (YX^T)(Q)(YX^T)^{-1}(XX^T)\right) - (c + 1)^{-1} \beta^T (Q)^{-1} \beta \exp(-\alpha k)
\]

Hashing

Each iteration of the algorithm requires the computation of \( 2d \) posterior probabilities, each involving a matrix inversion. However, since the posterior depends on the vector \( \gamma \), it is unnecessary to perform the calculation for equivalent sets of variables. Therefore, to avoid redundant calculations we can store each \( \gamma \) and its corresponding posterior probability in a hash table. We use the boolean representation of \( \gamma \) as the key and the probability as the data stored. Since a few hundred iterations could insert hundreds of thousands of entries, the loading factor of the hash table must be controlled to avoid running out of memory.

Pruning

Another way of optimizing the algorithm is to discard low-frequency variables, based on a frequency lower limit, \( f \), set by the user. Good results are achieved when variables with frequency \( f \leq 1\% \) are discarded after the burn-in. If the frequency of a variable is very low during the burn-in, it is unlikely that it is useful. After the burn-in
Algorithm 1 The Variable Selection Algorithm.

\textbf{Input:} \( X = \{x_1, x_2, \ldots, x_p\}, Y, d, B, N, c \) (Anything else???)

\textbf{Output:} Markov Chain

\textbf{Pre-Selection:}
- Calculate marginal correlation for all variables in the data set
- Sort variables by correlation in descending order and pick top \( d \)
- Calculate sufficient statistics

\textbf{Iterations:}
- for \( I = 1 \) to \( N \) do
  - for \( i = 1 \) to \( d \) do
    - Choose a variable \( x_j \) at random
    - if Variable \( x_j \) is blocked then
      - Continue to next iteration
    - end if
    - Look for probability of \( \gamma_{ij}^I \) in the hash table
    - if \( \gamma_{ij}^I \) is not found then
      - Calculate \( \text{Prob}(\gamma_{ij}^I) \)
    - end if
    - Randomly choose between the new and old models.
    - Append \( \gamma_{ij}^I \) to the Markov chain
  - end for
  - if \( I = B \) then
    - Eliminate variables below the threshold (block variables).
  - end if
- end for

Period, the cumulative frequency with which each variable has appeared as selected by the sampler is calculated. Variables that fall below \( f \) are not considered again. This reduces the time that the algorithm takes per iteration since the reduction of \( d \) favorably impacts the time complexity. Furthermore, since there are less dimensions, the efficiency of the hash table increases: less variables to consider implies less collisions in the table.

\textbf{LAPACK Integration}

A major bottleneck in the computation of Gibbs Sampling is the calculation of the inverse of \( Q_\gamma \). We optimized this operation by integrating with the well known numerical library LAPACK (Linear Algebra Package). Intel MKL, a highly optimized library for Intel processors, was integrated into the DBMS due to its remarkable performance.

\textbf{Time Complexity}

In order to calculate the time complexity of the function, we will break it down to its component subroutines. The preprocessing of the data set involves the variable pre-selection and the one pass calculation of \( n, L \) and \( YX^T \). The pre-selection requires the calculation of the all the correlations and sorting them: \( O(np + p \log p) \). The calculation of the sufficient statistics can be done in time \( O(nd) \), which is by definition less than \( O(np) \). Therefore the total time spent in the preprocessing of the table is dominated by the pre-selection. Since this is performed only once, it is negligible compared to the time complexity of the main steps of the process.

For every dimension we need the probabilities \( \gamma_j = 0 \) and \( \gamma_j = 1 \). These probabilities could be already stored on the hash table. Since the key of the hash table is stored as a sparse array of integers, we need an operation with time complexity
$O(d)$ to transform the binary representation of the $\gamma$ vector into an sparse array of integers. Then, this array is compared against the key, which also takes $O(d)$ time per iteration. In the worst case we do not find these probabilities in the hash table and we need to calculate them. This computation involve calculating $Q_\gamma$ on the fly. This is performed in time $O(nk^2)$. After the calculation, the matrix $Q_\gamma$ must be inverted. Therefore the total time for the probability calculation is $O(nk^2 + k^3)$. Since this calculation is performed $2d$ times (in the worst case of no hits in the hash table), the total time consumed by computation of the probabilities in one iteration is $O(d + ndk^2 + dk^3)$.

Therefore since $k < n < d$ (recall that $k$ is the number of variables selected by the Gibbs Sampler in the current iteration and it is always less than $n$, since we want a parsimonious model), we can simplify the time complexity to

$$T = O(dnk^2) \text{ per iteration}$$

Evidently, this is a worst case scenario calculation and the execution time is significantly improved by the use of hashing and low-frequency variable pruning.

**Database Implementation**

Most DBMSs allow User Defined Functions (UDF) and Table Valued Functions (TVF), written in a high level language, such as C++ or C#. UDFs and TVFs exploit the flexibility and speed of the C programming language. They can be used in a query like a regular SQL function or table. Moreover, UDFs are automatically executed in parallel. The optimization of the calculation of the sufficient statistics using UDFs is explained in detail in [14].

Correlation Raking is implemented as a UDF and is performed directly in the database. The calculation of the correlations themselves is also calculated as a UDF that we called `correlationXY`. This UDF takes two SQL arrays of equal length and returns a scalar number, the correlation between the arrays. This UDF is called as part of a SELECT statement.

`correlationXY(sqlArray::Load(x1,x2...xn), sqlArray::Load(y1,y2...xn))`

Using `correlationXY` it is simple to write an SQL query to derive the reduced model from the original model. Varying the number of selected variables is done trivially in the same SQL statement.

**Experimental Evaluation**

In previous sections we described the basic Gibbs sampler, as well as the improvements we devised in order to implement this algorithm as part of a DBMS. Here we present the results of our experiments as well as the evaluation of their accuracy and time complexity. We used our algorithm to search for parsimonious models of the survival time for patients suffering Glioblastoma Multiforme. The data sets are presented in Table 2. The variable of interest $Y$ is the patient’s survival time. We will apply our algorithm to data set $X_1$ (gene expression), $X_2$ (microRNA expression) and $X_1 \cup X_2$ (joint analysis of gene and microRNA expression). Gene/mRNA
and microRNA expression provide differently molecular information about the progression of the disease and the main scientific questions are not only to model the effects each platform independently but joint effects as well, on patients’ prognosis.

Table 2: data sets used in this work.

<table>
<thead>
<tr>
<th>data set</th>
<th>Dimensions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>11972</td>
<td>Gene Expression</td>
</tr>
<tr>
<td>X2</td>
<td>534</td>
<td>microRNA Expression</td>
</tr>
<tr>
<td>X1 ∪ X2</td>
<td>12506</td>
<td>Combined Gene/microRNA Expression</td>
</tr>
</tbody>
</table>

We run several experiments for this high dimensional data set \( (p \approx 12000) \), using the system described in Table 3. We preselect from 1000 to 3000 variables for covariate sets \( X1 \) and \( X1 \cup X2 \). We show in Table 6 that there is consistence in the top frequency variables among our experiments. In addition, we analyze the accuracy of the results running experiments with different parameters and measuring the average and maximum values of the regression coefficient in each experiment.

Table 3: DBMS and server characteristics.

<table>
<thead>
<tr>
<th>DBMS</th>
<th>SQL Server 2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating Sys.</td>
<td>Windows XP</td>
</tr>
<tr>
<td>Processor</td>
<td>Intel Xeon X3210 Quad-core CPU, 2.153 GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>4 GB</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>650 Gb; 7200 RPM</td>
</tr>
</tbody>
</table>

Experimental Set-up and Parameters Used

Since \( d \gg n \), the problem is under-determined and shows a proclivity to over-fitting. Therefore, if the parameter \( c \) is not large enough, the Gibbs sampler has a tendency to choose as many variables as there are data points. We studied the effect of this parameter on the number of variables chosen as can be seen in Table 4.

Table 4: Influence of the \( c \) parameter on the size of the final models in data set \( X1 \) \( (N = 30000 \) and \( d = 2000) \).

<table>
<thead>
<tr>
<th>( c )</th>
<th>Time [h:min]</th>
<th>( R^2 )</th>
<th>( k )</th>
<th>( R_{max}^2 )</th>
<th>( k_{max} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>3:35</td>
<td>0.646</td>
<td>28</td>
<td>0.888</td>
<td>50</td>
</tr>
<tr>
<td>120000</td>
<td>2:18</td>
<td>0.516</td>
<td>20</td>
<td>0.736</td>
<td>34</td>
</tr>
<tr>
<td>150000</td>
<td>1:31</td>
<td>0.490</td>
<td>18</td>
<td>0.677</td>
<td>28</td>
</tr>
</tbody>
</table>

All experiments were run using a \( B \) of 4000 or 10000 iterations and 30000 or 100000 total iterations (See Table 5). The threshold, \( f \), for pruning was set so that variables that appeared less than twice during the burn-in were discarded. The results are presented in the form of frequencies of the most often chosen variables by the algorithm as well as \( k \), the final size of the output, that is the number of non-zero elements of the vector \( \gamma_i \). It is important to note that while the frequency of the variables is very stable from experiment to experiment, \( k \) depends on the input parameters.

We evaluated the efficiency of the hash table for a model with \( d = 3000 \) and a frequency threshold \( f = 0.05\% \) (low pruning). We found that the table has a 94\% of hits. In other words, the use of a hash table has a very high impact in the reduction of the time spent calculating probabilities.
Table 5: $\bar{R}^2$ for several sub-model sizes.

<table>
<thead>
<tr>
<th>data set</th>
<th>$p$</th>
<th>$d$</th>
<th>$c$</th>
<th>$N$</th>
<th>Time [h:min]</th>
<th>$\bar{R}^2$</th>
<th>$k$</th>
<th>$R^2_{\text{max}}$</th>
<th>$k_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X1$</td>
<td>11972</td>
<td>1000</td>
<td>20000</td>
<td>30000</td>
<td>2:14</td>
<td>0.602</td>
<td>30</td>
<td>0.828</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>11972</td>
<td>2000</td>
<td>120000</td>
<td>30000</td>
<td>2:18</td>
<td>0.516</td>
<td>20</td>
<td>0.736</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>11972</td>
<td>3000</td>
<td>500000</td>
<td>30000</td>
<td>3:52</td>
<td>0.526</td>
<td>17</td>
<td>0.712</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>11972</td>
<td>1000</td>
<td>500000</td>
<td>100000</td>
<td>2:44</td>
<td>0.461</td>
<td>21</td>
<td>0.694</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>11972</td>
<td>2000</td>
<td>200000</td>
<td>100000</td>
<td>6:46</td>
<td>0.457</td>
<td>16</td>
<td>0.753</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>11972</td>
<td>3000</td>
<td>600000</td>
<td>100000</td>
<td>11:21</td>
<td>0.507</td>
<td>15</td>
<td>0.722</td>
<td>27</td>
</tr>
<tr>
<td>$X2$</td>
<td>534</td>
<td>534</td>
<td>2000</td>
<td>30000</td>
<td>3:14</td>
<td>0.355</td>
<td>37</td>
<td>0.624</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>534</td>
<td>534</td>
<td>4000</td>
<td>30000</td>
<td>1:44</td>
<td>0.278</td>
<td>29</td>
<td>0.444</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>534</td>
<td>534</td>
<td>4000</td>
<td>100000</td>
<td>5:26</td>
<td>0.265</td>
<td>28</td>
<td>0.471</td>
<td>61</td>
</tr>
<tr>
<td>$X1 \cup X2$</td>
<td>12506</td>
<td>2000</td>
<td>200000</td>
<td>30000</td>
<td>2:34</td>
<td>0.504</td>
<td>16</td>
<td>0.649</td>
<td>24</td>
</tr>
</tbody>
</table>

Results

Figure 1 shows the evolution of the marginal posterior probabilities of different gene expressions as a function of the number of iterations. We ordered the variables by their final frequency. This is a valid representation due to the fact that variables have no particular order (in the DBMS they are ordered alphabetically, but any ordering is acceptable). This particular organization of the variables shows how the variables go from being mostly distributed across the $d$ variables to just a few variables having the highest frequencies, while the others slowly tend to zero.

In Figure 2 we show the frequency across models of the dimensions. For the top 50 models that were found for the $X1 \cup X2$ data set after 100000 iterations we counted the repeated variables. We found that of the 396 different dimensions that were selected by the different models, 147 variables were selected more than once, 30 were selected more in more than 10% of the models and one was selected in 90% of the models. This indicates that there is consistency on the variable selection across models.

Figure 3 shows the frequencies of the top ten variables across 4 different experiments on the $X1 \cup X2$ data set. The experiments were run using the same parameters, $d = 2000$, $c = 200000$ and $N = 30000$; in order to investigate the repeatability of the results. We can see that even across experiments, the frequencies are remarkably similar. Table 6 shows the data as well as the rankings for the variables. Apart from some aberrant results, the rankings for the variables are similar and this confirms the stability of our results.

Table 6: Rankings and posterior probabilities for the top 10 variables.

<table>
<thead>
<tr>
<th>Name</th>
<th>Experiment 1</th>
<th>Experiment 2</th>
<th>Experiment 3</th>
<th>Experiment 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsa-mir-223</td>
<td>1</td>
<td>0.815</td>
<td>1</td>
<td>0.816</td>
</tr>
<tr>
<td>55711_at</td>
<td>2</td>
<td>0.394</td>
<td>2</td>
<td>0.390</td>
</tr>
<tr>
<td>79647_at</td>
<td>3</td>
<td>0.320</td>
<td>3</td>
<td>0.336</td>
</tr>
<tr>
<td>51164_at</td>
<td>4</td>
<td>0.260</td>
<td>4</td>
<td>0.276</td>
</tr>
<tr>
<td>hsa-mir-222</td>
<td>5</td>
<td>0.246</td>
<td>5</td>
<td>0.242</td>
</tr>
<tr>
<td>80117_at</td>
<td>6</td>
<td>0.220</td>
<td>6</td>
<td>0.230</td>
</tr>
<tr>
<td>6251_at</td>
<td>7</td>
<td>0.220</td>
<td>7</td>
<td>0.219</td>
</tr>
<tr>
<td>6716_at</td>
<td>8</td>
<td>0.217</td>
<td>8</td>
<td>0.217</td>
</tr>
<tr>
<td>2581_at</td>
<td>9</td>
<td>0.178</td>
<td>9</td>
<td>0.197</td>
</tr>
<tr>
<td>9337_at</td>
<td>10</td>
<td>0.175</td>
<td>10</td>
<td>0.157</td>
</tr>
</tbody>
</table>
Figure 1: The evolution of marginal posterior probabilities.
Figure 2: Dimension posterior probability across the top 50 models.

Figure 3: Variable posterior probabilities across experiments.
Biological interpretations

We ranked the genes based on their marginal posterior probabilities. The top 10 genes/miRNA signatures for $X_1 \cup X_2$ are presented in Table 8. Among these markers hsa-mir-222 and hsa-mir-223 had been identified before by [15] and [16] as part of the GBM prediction signature. This would indicate that our method is in agreement with previously reported experimental results. While our experiments find some top markers that have been previously implicated in the literature we also find a list of new markers that could merit further functional validation for development of future therapeutic strategies.

Table 7: Top Ten Variables for the $X_2$ data set.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Posterior Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsa-mir-148a</td>
<td>0.859</td>
</tr>
<tr>
<td>hsa-mir-197</td>
<td>0.512</td>
</tr>
<tr>
<td>kshv-mir-k12-12</td>
<td>0.434</td>
</tr>
<tr>
<td>hsa-mir-222</td>
<td>0.415</td>
</tr>
<tr>
<td>hsa-mir-216</td>
<td>0.301</td>
</tr>
<tr>
<td>hsa-mir-340</td>
<td>0.268</td>
</tr>
<tr>
<td>hsa-mir-623</td>
<td>0.206</td>
</tr>
<tr>
<td>hsa-mir-223</td>
<td>0.202</td>
</tr>
<tr>
<td>hsa-mir-217</td>
<td>0.185</td>
</tr>
</tbody>
</table>

Table 8: Top Ten Variables for the combined data set $X_1 \cup X_2$.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Gene Name</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsa-mir-223</td>
<td>microRNA</td>
<td>0.819</td>
</tr>
<tr>
<td>55711_at</td>
<td>AKIRIN1</td>
<td>0.400</td>
</tr>
<tr>
<td>79647_at</td>
<td></td>
<td>0.331</td>
</tr>
<tr>
<td>51164_at</td>
<td></td>
<td>0.275</td>
</tr>
<tr>
<td>hsa-mir-222</td>
<td>microRNA</td>
<td>0.251</td>
</tr>
<tr>
<td>80117_at</td>
<td>ARL14</td>
<td>0.229</td>
</tr>
<tr>
<td>6251_at</td>
<td>RSU1</td>
<td>0.217</td>
</tr>
<tr>
<td>6716_at</td>
<td>SRDA5</td>
<td>0.216</td>
</tr>
<tr>
<td>2581_at</td>
<td>GALC</td>
<td>0.179</td>
</tr>
<tr>
<td>9337_at</td>
<td>CNOT8</td>
<td>0.168</td>
</tr>
</tbody>
</table>

Performance Evaluation

In time comparisons with R, our algorithm shows a 30 to 100 fold time improvement, depending on the number of dimensions of the experiment. For instance in the case of $d = 534$ (data set $X_2$) R performs 1000 iterations in 8378 seconds, while we perform 30000 iterations in 8620 seconds. In short, similar runs in R would take almost 3 days to complete. A comparison between the time required by an R program and our algorithm is shown in Table 9. We attribute the substantial speed increase to the use of hashing to reduce the number of operations, to low variable frequency pruning and to the use of optimized subroutines for linear algebra operations.
Table 9: Time in seconds per 1000 iterations.

<table>
<thead>
<tr>
<th>$d$</th>
<th>R Package</th>
<th>Optimized Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>5468</td>
<td>194</td>
</tr>
<tr>
<td>534</td>
<td>8378</td>
<td>196</td>
</tr>
<tr>
<td>1000</td>
<td>22898</td>
<td>272</td>
</tr>
<tr>
<td>2000</td>
<td>$&gt;7.2 \times 10^4$</td>
<td>250</td>
</tr>
</tbody>
</table>

**Related Work**

[4]

In the last few years there has been a surge in the attention given to data analysis of high dimensional microarray data sets. In particular several methodologies have been proposed by the Biostatistics community to tackle the gene selection problem. There has been work done in Random Forests [17], Bayesian Ensemble methods [4], Interactive Bayesian Averaging [18] and Support Vector Machines [7]. There are also a variety of methods penalizing large models that have been investigated in the field of regression models, such as the Bayesian Lasso Method [8], the Bayesian Ridge Method [9] and the Bayesian Elastic Nets [10], among others. Bayesian Variable selection has also been applied to the prediction of survival times in [11], where the authors use a prior probability that enforces the selection of models of an arbitrary, fixed size. Unlike them, we assign a prior on the size that favors the selection of small models, rather than imposing a “hard” restriction. Another method of analyzing survival times was presented in [12] where a special shrinkage prior was introduced. However the authors only considered the cases with medium to moderate values of $d$ (from 20 to 1000 dimensions).

Even though it is natural to store microarray data in a DBMS, little research has gone into using it as a computational platform, rather than just a repository. Since a DBMS offers increased data security and provides the biomedical researcher a flexible query mechanism, we have focused our attention in the problem of microarray data analysis. Several fundamental ideas for the computation of statistical models in a DBMS, including linear regression models, are presented in [19]. Principal component analysis via Singular Value Decomposition (SVD) addressed the microarray data analysis in [20]. Bayesian variable selection in a DBMS was introduced in [21] but the authors focused on the traditional $d < n$ problem, therefore their algorithm cannot be applied to high dimensional microarray data sets.

**Conclusions**

Rewriting Needed:

A Bayesian variable search focused in parsimonious models is key in the analysis of microarray data sets, not only because it brings more practical and interpretable outputs for a biomedical researcher, but it also leads to a more tractable problem by focusing on a few important variables. In the present work, we show that applying a prior probability that favors the emergence of smaller models (models with less variables), and using an appropriately tuned $c$ parameter greatly increases the

[Do we need this section here. Usually in bioinformatics papers you put this in the introduction. I suggest condense this portion into one paragraph and mention it up front in the introduction]
chances of the emergence of parsimonious models. When we applied our algorithm to a microarray data set of cancer patients (TCGA-GBM), we found models of manageable sizes that accurately predict the life expectancy of the subjects.

The tuning of the c parameter must be done in a case by case basis, but we show (Table 4) that increasing its value reduces the number of dimensions chosen by the model. Other consequences of the magnitude of this parameter are the decrease of the running time and the worsening of the correlations $\overline{R}$ and $R_{\text{max}}$. This can be explained by the fact that if more variables are selected by the model we approach the case where the model is over-fitted. The reduction of the running time is a direct consequence of the hashing and the smaller number of variables that remain after the pruning occurs: Less variables are selected and therefore more have a frequency less than the threshold.

The main optimizations presented, hashing of frequently appearing variable combinations, low frequency variable pruning and sufficient statistics were successfully implemented in a DBMS. Our algorithm shows a dramatic improvement in the execution time when compared to the implementation of the Bayesian Search in R, a publicly available program used widely in the literature. Moreover, these optimizations can be implemented in high level programming languages, such as C++ or C#.

Further avenues of research include the analysis of other methods of variable preselection, enabling a flexible Bayesian Hierarchical Modeling and incorporating a more varied set of priors to allow the user more options when setting up an experiment.

Competing interests
The authors declare that they have no competing interests.

Author’s contributions
Text for this section ...

Acknowledgements
This research work was partially supported by National Science Foundation grants IIS 0914861 (to CO) and IIS 0915196 and NIH R01 CA160736 (to VB). The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Cancer Institute or the National Institutes of Health.

Author details
1University of Houston, 77204 Houston, USA. 2MD Anderson Cancer Center, 77030 Houston, USA.

References