GigaScience, 6, 2017, 1–10

doi: 10.1093/gigascience/gix078
Advance Access Publication Date: 22 August 2017
Technical Note

TECHNICAL NOTE

fastBMA: scalable network inference and transitive reduction

Ling-Hong Hung[1], Kaiyuan Shi[1], Migao Wu[1], William Chad Young[2], Adrian E. Raftery[2] and Ka Yee Yeung[1,*]

[1]Institute of Technology, University of Washington, Tacoma Campus, Box 358426, 1900 Commerce Street, Tacoma, WA 98402-3100, U.S.A. and [2]Department of Statistics, University of Washington, Box 354322, Seattle, WA 98195-4322, U.S.A.

*Correspondence address. Ka Yee Yeung, Institute of Technology, University of Washington, Tacoma Campus, Box 358426, 1900 Commerce Street, Tacoma, WA 98402-3100, U.S.A.; Tel: 253-692-4924; Fax: 253-692-5862; E-mail: kayee@uw.edu

Abstract

Inferring genetic networks from genome-wide expression data is extremely demanding computationally. We have developed fastBMA, a distributed, parallel, and scalable implementation of Bayesian model averaging (BMA) for this purpose. fastBMA also includes a computationally efficient module for eliminating redundant indirect edges in the network by mapping the transitive reduction to an easily solved shortest-path problem. We evaluated the performance of fastBMA on synthetic data and experimental genome-wide time series yeast and human datasets. When using a single CPU core, fastBMA is up to 100 times faster than the next fastest method, LASSO, with increased accuracy. It is a memory-efficient, parallel, and distributed application that scales to human genome-wide expression data. A 10 000-gene regulation network can be obtained in a matter of hours using a 32-core cloud cluster (2 nodes of 16 cores). fastBMA is a significant improvement over its predecessor ScanBMA. It is more accurate and orders of magnitude faster than other fast network inference methods such as the 1 based on LASSO. The improved scalability allows it to calculate networks from genome scale data in a reasonable time frame. The transitive reduction method can improve accuracy in denser networks. fastBMA is available as code (M.I.T. license) from GitHub (https://github.com/lhhunghimself/fastBMA), as part of the updated networkBMA Bioconductor package (https://www.bioconductor.org/packages/release/bioc/html/networkBMA.html) and as ready-to-deploy Docker images (https://hub.docker.com/r/biodepot/fastbma/).

Keywords: network inference; Bayesian models; time series; distributed computing; gene regulation; Cholesky decomposition; Dijkstra's algorithm; Bloom filter; Docker container; optimized software findings

Background

Genetic regulatory networks capture the complex relationships between biological entities, which help us to identify putative driver and passenger genes in various diseases [1, 2]. Many approaches have been proposed to infer genetic networks using gene expression data, for example, co-expression networks [3], mutual information-based methods [4, 5] Bayesian networks [6–8], ordinary differential equations [9, 10], regression-based methods [11–15], and ensemble methods [16]. In addition, methods have been proposed to infer gene networks using multiple data sources (e.g., [17–20]). We have previously described

Received: 12 January 2017; **Revised:** 10 June 2017; **Accepted:** 10 August 2017

ScanBMA [14], an implementation of Bayesian model averaging (BMA) [21] for inferring regulatory networks. ScanBMA is available from the "networkBMA" Bioconductor package [22], written in R and C++. It has been shown that ScanBMA generates compact accurate networks that can incorporate prior knowledge.

In this paper, we present fastBMA, which is completely written in C++ and uses more efficient and scalable regression and hashing methods. The algorithmic improvements increase the speed by a factor of 30 on smaller sets (Fig. 4A), with greater increases observed on larger sets due to improved scalability (Fig. 4B). fastBMA is parallelized using both OpenMP and MPI, allowing for further increases in speed when using multiple cores and processors. Although fastBMA uses the same core methodology as ScanBMA, the increased scalability allows for more thorough sampling of the search space to increase accuracy. The new probabilistic hashing procedure used by fastBMA is faster and utilizes 100 000 times less memory when analyzing large numbers of variables (see the fastBMA Methodology section). This allows fastBMA to operate on genome-scale datasets without limiting the possible regulators of a given gene to a smaller subset.

A final feature of fastBMA is the implementation of a new method for eliminating redundant indirect edges in the network. The post-processing method can also be used separately to eliminate redundant edges from networks inferred by other methods. The code is open source (M.I.T. license). fastBMA is available from GitHub [23], in R as part of the networkBMA package [24], and as Docker images [25]. The Docker containers include all the supporting dependencies necessary for MPI and make it much easier to run fastBMA on a local or cloud cluster.

Bayesian model averaging

We can formulate gene network inference as a variable selection problem where the dependent variable (target gene expression) is modeled as a function of a set of predictor variables (regulatory gene expression). A regression model can be formed by fitting equation (1).

$$X_i = \beta_0 + \sum_{h \in H} \beta_{h,i} X_h + \epsilon_{i,}, \quad (1)$$

where X_i, is the expression level of gene i, H is the set of regulators for gene i in a candidate model, $\beta's$ are the regression coefficients, and $\epsilon_i N \overset{iid}{\sim} (0, \sigma_\epsilon^2)$ is the error term for gene $i = 1 \ldots n$.

Time series data can also be modeled by using the expression at the previous time point to predict the next time point.

$$X_{i,t} = \beta_{0,i} + \sum_{h \in H} \beta_{h,i} X_{h,t-1} + \epsilon_{i,t}, \quad (2)$$

where $X_{i,t}$ is the expression level of gene i at time t, H is the set of regulators for gene i in a candidate model, $\beta's$ are the regression coefficients, and $\epsilon_{i,t} N \overset{iid}{\sim} (0, \sigma_\epsilon^2)$ is the error term for gene $i = 1 \ldots n$ and time $t = 2, \ldots T$.

Different candidate models can be constructed from different sets of regulator genes. Models can be evaluated based upon a measure of their goodness of fit, such as the sum of residuals. However, in genetic analyses, the number of genes often exceeds the number of samples, and many different models can fit the data reasonably well. The core idea behind the BMA methods is that, given a set of starting prior model probabilities, we can find the posterior probability of each model and make a consensus prediction giving proportionately more weight to the more probable models. In terms of gene regulation, the posterior probability that gene *j* is a regulator of gene *i* is the sum of the posterior probabilities of all candidate models that include gene *j* in the set of regulators of *i*. This posterior probability becomes the weight of the edge drawn from gene *j* to gene *i* in the gene network. Estimates of the weights from prior knowledge can be used to seed the calculation of models to increase accuracy. Alternatively, a set of uniform starting weights based on the average number of edges observed in biological networks can be used when there is no additional information [26].

Estimating model posterior probabilities

Estimation of the posterior probabilities of the models can be accomplished by a variety of methods, some of which are very computationally intensive [12]. The original BMA [21] and iterative BMA (iBMA) methods [27] use the Bayesian Information Criterion (BIC) [28], which is simple to calculate and penalizes larger models that are easier to fit. However, BIC is an asymptotic approximation that is most accurate for large sample sizes. As an alternative, ScanBMA provided the option of using Zellner's *g* prior [29] to compute the posterior probabilities. The *g* prior parameter can be estimated as the value that maximizes the total posterior probability of the models. Adjusting the range of possible values for the *g* prior allows us to tune the method for smaller sample sizes and produce better networks. fastBMA exclusively uses the *g* prior to estimate the posterior probabilities and replaces ScanBMA's R code with a faster C++ implementation for the expectation maximization (EM) optimization of the *g* parameter.

Sampling candidate models

The number of possible candidate models grows exponentially with the number of possible regulators, necessitating an efficient methodology to find a subset of reasonable models. In the original implementation of BMA for genetic regulatory network inference, the leaps and bounds algorithm [30] was used to identify the *n* best models for a given number of variables. Occam's window [31] was then used to discard models with much lower posterior probabilities than the best model. The leaps and bounds algorithm scales poorly and is limited in practice to fewer than 50 variables. Iterative BMA (iBMA) uses a preprocessing step to rank all variables (genes), iteratively applies the original BMA to the top *w* variables ($w = 30$ by default), and discards predictor variables with low posterior inclusion probabilities [13]. In the iterative step, new variables from the ranked list are added to replace the discarded variables. This procedure of repeatedly applying BMA and variable swaps is continued until the *w* top-ranked variables have been processed. In contrast to iBMA, ScanBMA removes the restriction of the search space to an initial list of variables [14]. ScanBMA keeps a list of the best current linear regression models found so far and adds or removes a variable from these models to search for better models. The process is repeated until no new models are added or removed from the best set of models. ScanBMA's greedy approach and the implementation of its core routines in C++ enable it to typically run faster than iBMA. In this paper, we present fastBMA that uses the ScanBMA approach but exploits the fact that new models are based upon existing models. In particular, new

Table 1: Differences between BMA implementations

	BMA	iBMA	ScanBMA	fastBMA
Max variables	50	100	3500	10 000$^+$
Parallel	No	No	No	MPI/OpenMP
Post-processing	No	No	No	Transitive reduction
Prior specification	BIC	BIC	BIC/g prior	g prior
Implementation	R/Fortran	R/Fortran	R/C^{++}	C^{++}

```
foreach gene $X_i$ in $\{X_1 .. X_N\}$
    MAXIMIZE $\sum_{i,j} Prob(X_i \text{ depends on } X_j)$ for different g prior values
        FIT the data to linear models  $(X_{i,t} = \beta_{0,i} + \sum_{h \in H} \beta_{h,i} X_{h,t-1} + \epsilon_{i,t})$
        EVALUATE models using g prior
        AVERAGE the best models to calculate posterior probabilities
        that the expression of gene $X_i$ depends on gene $X_j$
    end
    DRAW an edge from node $X_j$ to $X_i$  if the posterior probability
    that $X_i$ depends on $X_j$ > desired confidence
end
POSTPROCESS network by eliminating edge from $X_i$ to $X_j$ if a better
indirect path exists that connects $X_i$ to $X_j$
```

Figure 1: Outline of fastBMA algorithm.

models are fitted using the results from the existing models, which increases the speed and scalability of the search.

Post-processing graphs by transitive reduction

BMA and other methods for reconstructing biological networks can generate edges between genes that are the result of indirect regulation through 1 or more intermediate genes. While having edges that represent either direct or indirect interactions is perfectly acceptable in a graph, biological networks are usually represented by edges that represent direct interactions. Such networks allow for more straightforward identification of potential driver genes. For genetic networks, it is therefore desirable to remove edges between nodes where the regulation is indirect (transitive reduction). This can be done through post-processing of the inferred network. One intuitive approach is based on eliminating direct edges between 2 nodes when there is a better indirect path [32]. For example, *Bosnacki* recently proposed comparing P-values of the best edge in an indirect path with that of the direct path [33]. fastBMA introduces a similar approach that reduces transitive reduction to a shortest-path problem, which can be solved more efficiently for the sparse graphs typically found in gene regulatory networks.

Table 1 summarizes the key differences between the different BMA implementations.

fastBMA Methodology

Fig. 1 shows an outline of fastBMA. In this section, we report our algorithmic and implementation contributions in fastBMA and our evaluation procedure. The pseudocode for the entire implementation is provided in the supplementary materials.

Algorithmic outline of fastBMA

The core approach for fastBMA is similar to that used by ScanBMA. The best models are found using ScanBMA's search strategy with a starting value of g in the interval [1... NumberOfSamples]. Brent minimization [34] is then used to find the value g in the interval that gives rise to the set of models with the highest total marginal probability. A graph is constructed by drawing edges between genes with an edge weight equal to the average posterior probability of the regulator over the set of reasonable models. Transitive reduction is applied to this graph to remove edges that can be adequately explained by a better indirect path. A final graph is constructed by retaining edges with weights greater than a given cutoff.

There are 4 major algorithmic improvements that increase the speed, scalability, and accuracy of fastBMA:

1. Parallel and distributed implementation
2. Faster regression by updating previous solutions
3. Probabilistic hashing
4. Post-processing with transitive reduction

Parallel and distributed implementation

Parallelization can be accomplished by using a shared memory system, such as OpenMP [35], which is designed for assigning work to different threads in a single CPU with multiple cores. In contrast, Message Passing Interface (MPI) [36] launches multiple processes on 1 or more CPUs and passes messages between processes to coordinate the distribution of work. Both of these approaches have their respective advantages and disadvantages. OpenMP is applicable only to CPUs on a single machine and is a bit slower for fastBMA. MPI is usable on a single machine or a cluster but requires some work to set up. fastBMA implements both approaches, allowing the user to choose the preferred methodology based on their requirements.

Inferring the entire regulatory network involves finding the regulators for every gene in the set. Since each of these determinations is carried out separately, each thread or process can be assigned the task of finding the regulator for a subset of genes in the set. When OpenMP is used, it provides a scheduler that dynamically assigns the regression calculations for a given gene to each thread. Threads work simultaneously on their tasks and receive a new task when they finish the previous task. All threads share access to memory, and the same input data for the regression is available to all the threads. The parallel code only extends to the regression loop; the final transitive reduction post-processing and output is done by a single thread.

When MPI is used, we initially split the tasks evenly among the available CPUs. In the case of MPI processes, memory is not shared. Instead the input data is read by a master process and distributed to all the participating processes using MPI's broadcast command. All processes then work on their tasks simultaneously in parallel and send messages to all the other processes so that all processes know which tasks are being worked upon. The length of time required for each calculation varies considerably, and, as a result, some processes will finish before others. A process that finishes early then works on tasks initially assigned to other processes that have not yet been started. When all the regulators for all the genes have been found, a master process gathers the predictions, performs transitive reduction post-processing, and outputs the final complete network. OpenMP can also be used in conjunction with MPI to further subdivide the tasks among threads available to a CPU.

Faster regression by updating previous solutions

Even with the above parallel implementation, each individual calculation of regulators is still accomplished by a single process. If the regression procedure is too slow, this step can be rate-limiting for large numbers of genes regardless of the number of processors available. ScanBMA uses Cholesky decomposition to triangularize the regression matrix and obtain the regression coefficients through back substitution. These calculations have a time complexity of $O(n^3)$, where n is the number of variables in the model. However, in the case of fastBMA, new regression models are based upon the previous models and involve the addition or removal of a single variable. It is possible to use the triangular matrix of the previous model to calculate the triangular matrix and regression coefficients for the new model. fastBMA's new C++ implementation of this update algorithm is based on the Fortran code from the qrupdate library [37].

The time required for Cholesky decomposition becomes $O(n^2)$ when updating the previous solution. Average sampled model sizes for typical applications range between 5 and 20, and this would be the expected speedup when using a single thread. However, fastBMA further optimizes the implementation by pre-calculating matrix multiplications and using lower-level linear algebra routines from OpenBLAS [38] for further speed increases. OpenBLAS is an optimized open source implementation of the Basic Linear Algebra Subprograms (BLAS) routines. Custom wrappers were added to allow the use of the OpenBLAS Fortran libraries. Our initial prototyping indicated that the improvements in the regression procedure account for the majority of the 30-fold increase in speed observed for smaller search spaces on a single thread.

Replacing the hash table with a constant time and constant space probabilistic filter

In order to understand the necessity and efficacy of the new probabilistic filter used by fastBMA, we must first understand the limitations of the simple hash table used by ScanBMA. Before evaluating a newly generated model, ScanBMA checks to see if that model has been previously evaluated. This is done by using a hash table to store a string representing the indices of the variables in the model. For smaller sets, the time and space required for this operation are negligible compared to the time and space required to calculate the regression coefficients. However, when the number of variables is in the thousands, this operation becomes the bottleneck. A regular hash table uses a hash function to map the model to a bucket. When the number of models is small relative to the number of buckets (small load factor), it is unlikely that 2 models will be put in the same bucket, and the time taken to look up a model is just the time to map the model to a bucket. For lexicographical strings, the hash function is applied to small substrings and the values are combined. The time required for hashing the whole string is proportional the length of the string. In the case of ScanBMA, the length of the strings formed from the concatenated variable indexes is proportional to the number of variables n. Thus for small numbers of models, the time complexity of the lookup operation will also be $O(n)$.

However, when the load factor is large, it is likely that multiple models map to the same bucket. The resulting collisions must be resolved by searching through the models in the bucket. For the C++ *unordered set* container used by ScanBMA, this has worse-case $O(m)$ time complexity where m is the number of models giving a total time complexity of $O(nm)$ for the lookup procedure when m is large. In addition, the memory required to store the hash table will be $O(m)$. Unfortunately, when a large number of mostly uninformative variables are coupled with a large Occam's window, m grows very rapidly. In these cases, we observed that the memory and time requirements of the hashing procedure soon become limiting. For example, even though it only runs a single thread, ScanBMA will run of out memory on a 56 GB machine when there are large numbers of variables and no informative priors.

It is vital that the ScanBMA algorithm does not sample a model more than once to ensure that the method will converge and terminate. However, the methodology is quite tolerant of falsely excluding models that have not been sampled. ScanBMA only explores a small sample of the possible models—the vast majority of models are normally excluded. Furthermore, in the BMA approach, many models are averaged to obtain the final edges. Variables that are important appear in many models. In the rare case where a good model is falsely excluded, the impact is minimized because the key regulators in the falsely excluded model will be found in other models. When such false negatives are tolerated, an alternative to using a hash table is to ignore the collisions. This saves both time and space by removing the dependence on m for both time and space complexity. An example of a noisy or probabilistic hashing approach is the Bloom filter [39], which has been used for bioinformatics applications [40] due to fast computation and low memory requirements.

fastBMA includes an optimized implementation of a probabilistic hash (see Fig. 2) that has constant time and constant memory complexity. The dependence of the computation time on m is eliminated by ignoring collisions, and the dependence

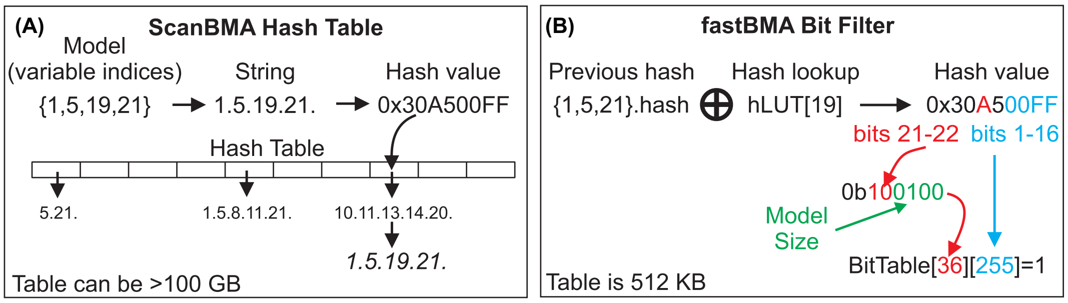

Figure 2: ScanBMA hash table vs fastBMA bit filter. The differences between the hashing methods used by ScanBMA (A) and fastBMA (B) are shown. ScanBMA concatenates the indices of the regulator variables in the model to form a unique string. The string is then mapped to 1 of a set of buckets. Strings mapping to the same bucket are kept in a second data structure, which must be navigated to look up the string. In contrast, fastBMA pre-calculates the hashes for all the possible variables. New regression models are based upon the previous models and involve the addition or removal of a single variable. The hash value for the new model is obtained by XORing the hash value for the variable to be added or deleted with the hash value of the previous model. The hash value is used to map the model to a position in a 512-kB bit table, with the row dependent on the number of variables. Mapping different sized models to different rows prevents the large number of collisions that would otherwise arise when using the XOR operator to combine hash values. A bit is set in the bit table to indicate that the model has been observed. Collisions are ignored; it is possible to falsely conclude that a novel model has been evaluated when it has not. As discussed in the Methods section, this type of error is well tolerated by the fastBMA protocol.

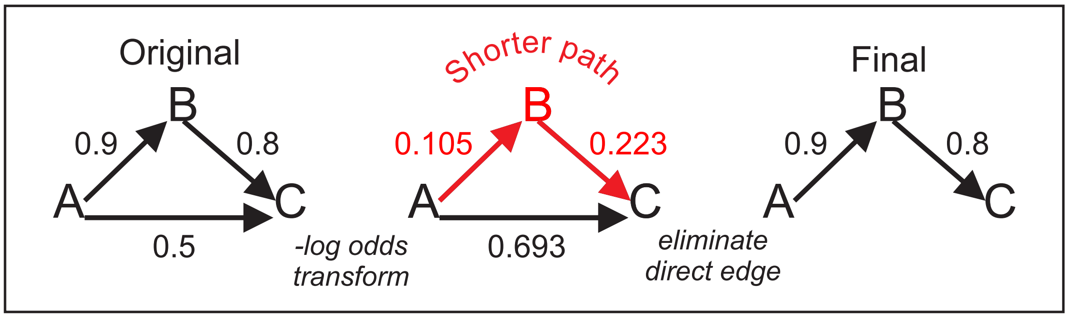

Figure 3: Transitive reduction post-processing. A simple example of the transitive reduction procedure is illustrated. The 3 edge weights in the mini-graph are the posterior probabilities that A regulates B, B regulates C, and A regulates C. The probability of A regulating C through an indirect path through edges A→B→C is the product of the edge weights for A→B and B→C. We take the negative log of the probabilities (middle panel) to transform the multiplication into distances. The indirect path A→B→C is shorter than the direct path A→C, which is equivalent to the probability of A regulating C through B being greater than the probability of A directly regulating C. As a result, the edge between A and C is removed.

on *n* is eliminated by using an updatable hash function (MurmurHash3 [41]) that calculates the hash value of a model based on the hash value of the previous model. fastBMA uses the hash value of the model to map it to a location in a 2-dimensional bit table. The bit at that location is then set to 1. Any model that hashes to a table location with a set bit will not be processed. The error rate for the filter is initially very low, and errors are more likely near the end of the search when more bits in the table have been set. This meshes well with the search process used by fastBMA: errors at the end of the search have even less impact because almost all changes to good models are rejected at that point.

Our benchmarking confirms that ignoring collisions does not degrade the accuracy of fastBMA. Using a bit table of just 512 *kilo*bytes gives identical results for a smaller synthetic dataset and almost identical results for the larger genome-wide experimental dataset. This is reflected in Fig. 4A, where the accuracy of fastBMA is the essentially the same (actually slightly higher) than ScanBMA when using the same search window. However, ScanBMA can use hundreds of *giga*bytes of memory to maintain a string hash table during wide searches over the yeast dataset.

The implementation of the methodology is also further optimized for speed. New hash values are derived from old ones by looking up the value of the pre-calculated hash for the variable to be added or deleted and using XOR to combine it with the previous hash. This procedure is very fast and invertible but normally would cause severe collision problems, with the same hash being associated with different sets of variables. This is solved by mapping hashes from models of different sizes to different rows of the bit table. fastBMA uses a bit table of 64 rows by 65 326 columns. fastBMA maps the lower 16 bits of the hash value to obtain the column *c* and uses bits 21 and 22 combined with the last 4 bits of the model size to obtain the row *r* (see Fig. 2). The value of the bit table at row *r* and column *c* is set to indicate that the hash value has been seen. Thus the hashing/insert/lookup procedure is constant time, using a very small number of fast bit operations. The tiny size of the bit table (512 kB) also makes the lookup operation very cache friendly. During our prototyping of different versions of fastBMA, we found that the optimized bit filter was much faster than using a full hash table even for small datasets where the load factor is small and there are few collisions.

Transitive reduction: eliminating edges when there is a better indirect path

fastBMA's transitive reduction methodology is based on eliminating direct edges between 2 nodes when there is a better alternative indirect path. This approach was first described by Wagner [32]. *Bosnacki* recently proposed comparing *P*-values of the best edge in an indirect path with that of the direct path [33]. fastBMA uses the stronger criterion of comparing the

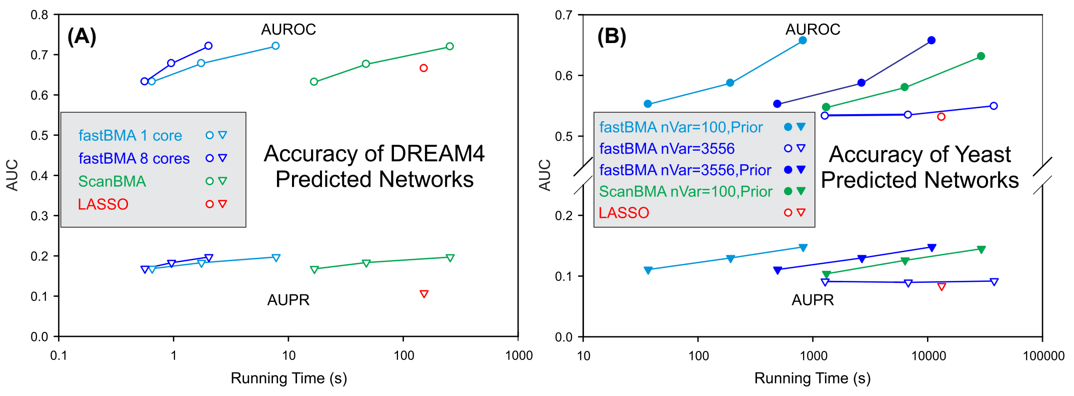

Figure 4: Graphs of the overall accuracy of networks as a function of running time on the DREAM4 simulated (A) and yeast time series data (B). The area under the receiver operating character curve (AUROC) and area under the precision recall curve (AUPR) of networks inferred from the DREAM4 dataset using fastBMA (no post-processing), ScanBMA, and LASSO are plotted against the running times. The different points within a line segment represent fastBMA and ScanBMA with increasingly wider searches as determined by the odds ratio (OR) parameter (OR = 100, 1000, 10 000)—the leftmost point representing the smallest OR, which is the fastest and least accurate. LASSO does not have an equivalent parameter and was run with the default settings. For the yeast datasets, prior probabilities of regulatory relationships (informative priors) were obtained using external data sources as described in Lo et al. For all methods not using informative priors (including LASSO), variables were ordered by their absolute correlation to the response variable. For the ScanBMA on the yeast dataset, the search space was restricted to the 100 variables with the highest prior probabilities. fastBMA was run with a search space of 100 variables using 1 core and all 3556 variables using 8 cores, with and without the Lo et al. prior probabilities. All tests were conducted using Ubuntu 14.04 on an A10 Microsoft Azure cloud instance, which is an Intel Xeon CPU with 8 cores and 56 GB of RAM and is the average of 5 runs. Docker images were not used during benchmarking. Error bars are not shown as the variation between runs is too small to appear on the graphs.

overall posterior probability of the entire path. The linear regression model underlying BMA does not distinguish between direct and indirect paths. However, BMA is usually seeded with the prior probabilities of a direct interaction between genes, and the posterior probabilities that constitute the edge weights in a fastBMA network are intended to be estimates of the confidence that there is a direct interaction. The overall probability of any path can be estimated (assuming independence) by multiplying the edge weights together. Equivalently, we can transform the edge weights by taking the negative log, and the highest probability path becomes the path with the lowest sum of negative log edge weights (see Fig. 3). The question of whether a better indirect regulatory chain exists is thus mapped to the question of whether a shorter indirect path exists between the 2 nodes. This is the a shortest path problem that can be solved by Dijkstra's method with time complexity of $O(N\,E\,\log N + N^2 \log N)$, where E is the number of edges and N is the number of nodes. By comparison, the GPU methodology of *Bosnacki* is $O(N^3)$ using a less selective criterion of comparing the best edge in the path. The search is also bounded: once a path's distance exceeds the direct distance, there is no need to further explore that path. In addition, fastBMA produces graphs with few high-weight edges, and, in practice, the algorithm is much faster than the worst case as most searches are quickly terminated.

Datasets used for testing

We have previously benchmarked ScanBMA [14] against other network inference methods (MRNET [5], CLR [42], ARACNE [4], DBN [8], and LASSO [11, 43]) on smaller test sets. In this study, we focus on comparing fastBMA only to ScanBMA and LASSO, which were the 2 most accurate methods in these benchmarks and are the only 2 methods that could infer networks from the larger datasets in a reasonable time. We also compare fastBMA to other methodologies, GENIE3 [44] and Jump3 [45], which have demonstrated high accuracy on the simulated DREAM datasets.

We used the following 3 datasets for testing.

1. *Simulated* 10-gene and 100-gene time series data (5 sets of each) and the corresponding reference networks from DREAM4 [46–50]. As these datasets are simulated; the true regulatory relationships are known and are used to evaluate the accuracy of the predicted networks. For Fig. 4A, all the 10-gene and 100-gene datasets were used and treated as 1 big dataset. Individual 100-gene networks were used to generate Table 2.
2. Yeast time series expression data (ArrayExpress E-MTAB-412) consisting of 3556 genes over 6 time points and 97 replicates [51]. Being actual data, there is no absolute ground truth. Instead, we compared the regulatory predictions with the literature-curated regulatory relationships from the YEASTRACT database [52].
3. Human single-cell time series RNA-Seq data GSE52529 (9776 genes) from GEO [53]. As no satisfactory gold standard was available, we only used this to demonstrate that fastBMA could scale to noisy human genome-wide expression data.

Assessment metrics and testing methodology

We define a true positive (TP) as an edge in the inferred network that is also present in the ground truth or gold standard set. False positives (FP) are edges in the inferred network that are missing in the gold standard. False negatives (FN) are missing edges in the inferred network that are present in the gold standard, and true negatives (TN) are missing edges that are also missing in the gold standard. Precision (TP/(TP+FP)) and recall (TP)/(TP+FN) are useful measures of the positive predictive value and sensitivity of the methodology. However, precision and recall are dependent on the threshold used for the edge weights. Plots of precision vs recall over different values for the threshold give a more complete picture of the accuracy of the network inference. Similarly, receiver operating characteristic plots of TP/(TP+FN) vs FP/(FP+TN) for different thresholds are also useful, though less so than precision-recall plots because we are more interested in TP in sparse biological networks. We distill the overall information of these plots into a single number by estimating the area under the curve (AUC), i.e., area under precision recall curve (AUPR) and area under receiver operating curve (AUROC) for all

Table 2: Comparison of AUPR on Dream-4100 gene networks for Jump3, GENIE3, and fastBMA

	Network 1	Network 2	Network 3	Network 4	Network 5	Running time per network
Jump3	**0.270**[3]	0.110	0.200	0.180	**0.174**	48 hours[1]
GENIE3	0.228	0.096	**0.230**	0.157	0.168	N/A
fastBMA	0.232	**0.127**	0.189	**0.227**	0.158	3 seconds (using 1 thread)[2]

[1]Jump3 timings were from an Intel i7 processor @1.7 GHz.
[2]fastBMA timings were from an Intel Xeon E5–2670 processor @ 2.6 GHz.
[3]The highest AUPR for each network are in bold type.

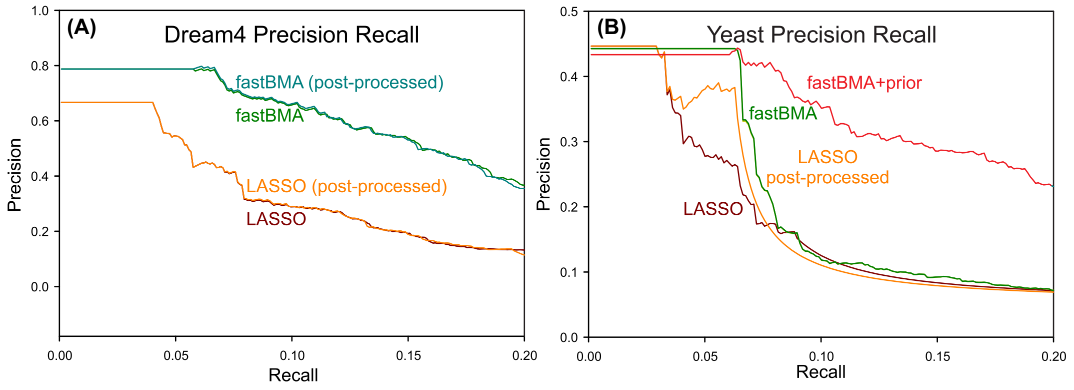

Figure 5: Precision-recall curves. In (A) the precision-recall curves were plotted for the networks inferred from the DREAM4 data using LASSO, LASSO+post-processing, fastBMA+post-processing (odds ratio = 10 000). No informative priors were available for this synthetic dataset. In (B), curves were plotted for the networks inferred from the yeast time series data using LASSO, LASSO+post-processing, fastBMA, and fastBMA with informative prior. For the yeast dataset, curves for post-processed networks for fastBMA are not shown as they are essentially identical to the curves for networks inferred without post-processing.

possible threshold values. Due to the size of the larger yeast networks, all AUC calculations were done using custom software fastROCPRC [54], written in C++. We primarily use AUPR and AUROC for the assessment as these metrics measure the overall performance of the methods. In practice, however, predicting *some* edges accurately, even if only for the most confident predictions, is still valuable for narrowing down a set of potential interactions to be further explored. Hence, we also plot the precision-recall graph to assess where the differences in accuracy are occurring. Timings for ScanBMA, fastBMA, and LASSO were the average of 5 runs on the same 8-core 56-GB Microsoft Azure A10 instance. fastBMA and ScanBMA were compiled on the instance and the binaries used. For the Jump3/GENIE3 comparison, we did not run the software ourselves but relied upon the published running times.

Results

We applied our fastBMA algorithm to both simulated and real time series gene expression data. We had previously tested several methods on these datasets [14] and found that ScanBMA and LASSO were the fastest and most accurate methods. Therefore, we focused on comparing the fastBMA results to ScanBMA and LASSO [43, 55] in Figs 4 and 5. LASSO is a non-Bayesian linear regression method that uses a penalty term to prevent overfitting to models with many variables. It is written in Fortran and is the basis for one of the fastest network inference methods available. Both fastBMA and ScanBMA control the breadth of the search by varying the odds ratio threshold that defines the size of Occam's window. The odds ratio is the confidence in the query model relative to the best model. Models outside of this window are discarded. Hence, a larger odds ratio threshold drives a wider search, which naturally takes longer to complete.

We ran both ScanBMA and fastBMA with increasingly larger windows (odds ratios 100, 1000, 10 000) and the time and accuracy, as measured by AUROC and AUPR, plotted as line segments in Fig. 4. The exception was that in Fig. 4B ScanBMA was restricted to 100 variables and using priors due to the time and memory required to run it using all 3556 variables with the larger odds ratios. All the line segments have a positive slope, indicating that larger windows do increase the accuracy, at the expense of using more computation time. For both the synthetic DREAM4 and experimental yeast datasets, with or without prior information, the line segments for fastBMA in Fig. 4A (where the conditions are identical) are well to the left of the corresponding line segments for ScanBMA. The x-axis is logarithmic, indicating that fastBMA is orders of magnitude faster than ScanBMA when using the same parameters. Alternatively, one can use a larger odds ratio with fastBMA and obtain a more accurate result in the same time it would take to run ScanBMA with a smaller odds ratio. This is especially important for larger datasets such as the yeast dataset with non-informative priors, where it is impractical to run ScanBMA. On the same datasets, fastBMA is also more accurate and faster than LASSO, the degree and nature of improvement depending on whether the user chooses to emphasize speed or accuracy through the choice of the odds ratio parameter.

One of the main advantages of the BMA methods is that they are able to incorporate prior information to improve inference. This was not possible for the DREAM4 dataset as it is a synthetic dataset, for which relevant prior information is not available. In this case, an uninformative uniform prior probability is used. However, for the yeast dataset, we had access to priors from external data sources [12]. Specifically, we applied a supervised learning approach to a training dataset consisting of regulator-gene pairs and various attributes assembled from diverse gene expression data, genome-wide binding data, protein-protein interactions, gene ontology terms, and prior knowledge from the literature. We computed predicted probabilities of regulatory relationships using this supervised learning approach, and these predicted probabilities were used as priors in the regression step. These priors are available in the *lopriors.tsv* file in the supplementary materials. The use of informative priors also allowed us to triage the variables to be explored to the 100 variables with the highest prior probabilities, saving considerable computational resources. In addition, using informative priors often increases computational efficiency by restricting the search to a smaller space. As expected, using informative priors increases the accuracy and decreases the running time of fastBMA relative to LASSO. In addition, we ran fastBMA without informative priors and without restricting the number of variables (i.e., using all 3556). This is beyond the capabilities of ScanBMA when using wider search windows. Even on this computationally demanding task, inferring the yeast network without informative priors, fastBMA is faster than LASSO with increased accuracy as assessed by AUROC and AUPR.

A common use for computational network inference is to identify a small set of potential regulators that could be verified with further experiments. For this use case, an improvement in the precision of the most confident predictions is more important than a small improvement in the overall performance of the method. As some of the differences in AUC for the yeast dataset are relatively small, we plotted the precision recall curves in Fig. 5. We see that the precision of the most confident predictions (i.e., lowest recall) is increased. The advantage of using informative priors when available is very clear. However, even when prior knowledge is not available, the fastBMA algorithm is superior, which is especially evident in the case of the DREAM4 dataset.

The effect of post-processing is more limited. In Fig. 5, the precision-recall curves for the DREAM4 dataset are almost identical for fastBMA and LASSO with and without post-processing. The same result was observed for fastBMA on the yeast dataset, and for clarity, we did not plot the overlapping precision-recall curves for the post-processed networks for fastBMA. However, we do see that post-processing has an effect on LASSO for the yeast dataset.

We also tested fastBMA on a human single cell RNA-Seq dataset with 9776 variables. Using a 32-core cluster on Microsoft Azure (2 nodes of 16 cores), fastBMA was able to obtain a network in 13 hours without using informative priors. Neither ScanBMA nor LASSO is able to return results for this dataset. We do not have a gold standard for this test—the purpose was to demonstrate that fastBMA could handle a very large and noisy genomic-sized dataset and return a network within a reasonable time even in the worst case scenario where the data is noisy and there is no prior information.

One possible drawback of the fastBMA methodology is the narrow search algorithm, which restricts sampling to models similar to the previously optimal models. While this is a prime reason for the speed of the approach, methodologies that sample the space more thoroughly, especially on smaller datasets, may be prove to be more effective. Table 2 compares fastBMA to the methodology of one of the best performers on the non–time series data from DREAM4, GENIE3, and its successor Jump3, which use an ensemble of decision trees for network inference. In Huyny-Thu et al. [45], the AUPR for both GENIE3 and Jump3 were given for the DREAM4 time series data, allowing for a direct comparison with fastBMA. For these tests, fastBMA was run with an odds ratio of 10 000 and the AUPR compared with those listed in Huynh-Thu et al. [45] in Table 2. The best results for each of the 5 networks are highlighted. In particular, Jump3 has higher AUPR than fastBMA for 3 of the networks, and fastBMA has higher AUPR than GENIE3 in 3 of the networks, with the AUPRs being fairly similar for all the 3 of the methods. However, the running time difference is significant, with fastBMA taking 3 seconds per network and Jump3 taking 2 days. Running times for GENIE3 time series data were not given in Huynh-Thu et al. [45].

Discussion and Conclusions

We have described fastBMA, a parallel, scalable, and accurate method for inferring networks from genome-wide data. We have shown that fastBMA can produce networks of increased accuracy orders of magnitude faster than other fast methods, even when using a single thread. Further speed increases are possible by using more threads or processes. fastBMA is scalable, and we have shown that it can be used to analyze human genomic expression data even in the most computationally demanding situation of noisy data, no informative priors, and considering all genes as possible regulators.

fastBMA includes a new transitive reduction post-processing methodology for removing redundant edges where the predicted regulatory edge can be better explained by indirect paths. Both fastBMA and LASSO already penalize large models and favor the exclusion of redundant variables. This explains why post-processing has minimal impact on the sparse networks predicted by fastBMA and LASSO. In particular, fastBMA produces very sparse networks that are not improved by further processing on any of the datasets tested. LASSO's networks are denser. For the small synthetic DREAM4 set, the post-processing still does not improve the network. However, on the larger experimentally derived yeast dataset, spurious edges do appear in the LASSO networks despite the regularization penalty that discourages larger models. Some of these redundant edges are successfully removed by the transitive reduction post-processing, improving the overall accuracy of the network. Thus the transitive reduction methodology may prove useful as an adjunct to methods and datasets that give rise to denser networks and are more prone to over-predicting edges than fastBMA. With this in mind, and given that this methodology is different from other published methodologies, we have included the ability to run the transitive reduction module of fastBMA on any set of edges, not just those generated by fastBMA.

Although we have focused on biological time series data, fastBMA can be applied to rapidly infer relationships from other high-dimensional analytics data. Also, the fastBMA methodology can be extended for even more demanding applications. For example, multiple bit filters (i.e., a Bloom filter) could be used to hash larger search spaces. fastBMA does have some limitations: the speed relies on sampling a small subset of the search space defined by the initial best set of models. This may not be an optimal strategy when there are many almost equally good dissimilar solutions and no prior knowledge to provide a guide to a set of good starting models. In these cases, especially for smaller networks, there may be better solutions such as Jump3 that can sample the space more thoroughly within a reasonable time frame. However, on the 100-gene DREAM4 datasets in Table 2, the differences in accuracy between the methods were not large, but the speed increase was more than 4 orders of magnitude. We anticipate that the efficiency of fastBMA will be especially useful for very large datasets on the cloud where usage is metered. For this purpose, we have provided Docker images to facilitate deployment on local or cloud clusters.

Availability and requirements

Project name: fastBMA

Project home page: https://github.com/lhhunghimself/fastBMA

Operating system(s): Linux (MacOS and Windows support provided through the Docker container [25] and Bioconductor package [24])

Programming language: C++

Other requirements: gcc version >4.8, OpenBLAS, mpich2 (if MPI desired) to compile code

License: M.I.T.

Any restrictions to use by non-academics: none other than those required by the license

Availability of supporting data

Simulated 10-gene and 100-gene time series data (5 sets of each) and the corresponding reference networks from DREAM4 were obtained from DREAM4 [56]. Yeast time series expression data (ArrayExpress E-MTAB-412) consisting of 3556 genes over 6 time points [51] and literature-curated regulatory relationships from the YEASTRACT database [52].

Human time series RNA-Seq data GSE52529 (9776 genes) were obtained from GEO [53].

Snapshots of the supporting code are also available from the *GigaScience* database, *GigaDB* [23].

List of abbreviations

AUC: area under the curve; AUPR: area under precision recall; AUROC: area under receiver operator curve; BIC: Bayesian information criterion; BMA: Bayesian model averaging; EM: estimation maximization; iBMA: iterative Bayesian model averaging.

Competing interests

The authors declare that they have no competing interests.

Funding

This work was supported by the National Institutes of Health (U54HL127624 to K.Y.Y., R01HD054511 to A.E.R., R01HD070936 to A.E.R.) and Microsoft Azure for Research Awards to K.Y.Y. and L.H.H.

Author contributions

L.H.H. conceived and implemented fastBMA. L.H.H., K.S., M.W., and W.C.Y. tested and benchmarked the software. K.S. and L.H.H. added fastBMA to the networkBMA R package. L.H.H. generated the Docker packages. All authors read and approved the manuscript.

Acknowledgements

We would like to thank Dr. Roger Bumgarner for helpful discussions.

References

1. Barabasi A, Gulbahce N, Loscalzo J. Network medicine: a network-based approach to human disease. Nat Rev Genet 2011;**12**(1):56–68.
2. Karlebach G, Shamir R. Modelling and analysis of gene regulatory networks. Nat Rev Mol Cell Biol 2008;**9**(10):770–80.
3. Zhang B, Horvath S. A general framework for weighted gene co-expression network analysis. Stat Appl Genet Mol Biol 2005;**4**(1):Article17.
4. Margolin AA, Nemenman I, Basso K et al. ARACNE: an algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context. BMC Bioinformatics 2006;**7**(suppl 1):S7.
5. Meyer PE, Kontos K, Lafitte F et al. Information-theoretic inference of large transcriptional regulatory networks. EURASIP J Bioinform Syst Biol 2007;79879.
6. Friedman N, Linial M, Nachman I et al. Using Bayesian networks to analyze expression data. J Computational Biol 2000;**7**(3–4):601–20.
7. Zhu J, Chen Y, Leonardson AS et al. Characterizing dynamic changes in the human blood transcriptional network. PLoS Comput Biol 2010;**6**(2):e1000671.
8. Zou M, Conzen SD. A new dynamic Bayesian network (DBN) approach for identifying gene regulatory networks from time course microarray data. Bioinformatics 2005;**21**(1):71–79.
9. Cao J, Qi X, Zhao H. Modeling gene regulation networks using ordinary differential equations. Methods Mol Biol 2012;**802**:185–97.
10. Wu H, Lu T, Xue H et al. Sparse additive ordinary differential equations for dynamic gene regulatory network modeling. J Am Stat Assoc 2014;**109**(506):700–16.
11. Liu L, Wu F, Zhang W. A group LASSO-based method for robustly inferring gene regulatory networks from multiple time-course datasets. BMC Syst Biol 2014;**8**(suppl 3):S1.
12. Lo K, Raftery AE, Dombek KM et al. Integrating external biological knowledge in the construction of regulatory networks from time-series expression data. BMC Syst Biol 2012;**6**(1):101.
13. Yeung KY, Dombek KM, Lo K et al. Construction of regulatory networks using expression time-series data of a genotyped population. Proc Natl Acad Sci U S A 2011;**108**(48):19436–41.
14. Young W, Raftery AE, Yeung K. Fast Bayesian inference for gene regulatory networks using ScanBMA. BMC Syst Biol 2014;**8**(1):47.
15. Rogers S, Girolami M. A Bayesian regression approach to the inference of regulatory networks from gene expression data. Bioinformatics 2005;**21**(14):3131–7.
16. Marbach D, Costello JC, Kuffner R et al. Wisdom of crowds for robust gene network inference. Nat Methods 2012;**9**(8):796–804.
17. Chen J, Hu Z, Phatak M et al. Genome-wide signatures of transcription factor activity: connecting transcription factors, disease, and small molecules. PLoS Comput Biol 2013;**9**(9):e1003198.
18. Li W, Zhang S, Liu C-C et al. Identifying multi-layer gene regulatory modules from multi-dimensional genomic data. Bioinformatics 2012;**28**(19):2458–66.
19. Zhu J, Zhang B, Smith EN et al. Integrating large-scale functional genomic data to dissect the complexity of yeast regulatory networks. Nat Genet 2008;**40**(7):854–61.
20. Zhu J, Sova P, Xu Q et al. Stitching together multiple data dimensions reveals interacting metabolomic and transcriptomic networks that modulate cell regulation. PLoS Biol 2012;**10**(4):e1001301.
21. Raftery AE. Bayesian model selection in social research. Sociol Methodol 1995;**25**:111–93.
22. Yeung KY, Fraley C, Young WC et al. Bayesian model averaging methods and R package for gene network construction. In: Big Data Analytic Technology For Bioinformatics and Health Informatics (KDDBHI), workshop at the

20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), New York, NY, 2014. Available at: https://sites.google.com/site/kddbhi/kddbhi2014/.
23. Hung LH, Shi K, Wu M et al. Supporting data for "fastBMA: scalable network inference and transitive reduction." GigaScience Database 2017. http://dx.doi.org/10.5524/100338. Accessed 3 September 2017.
24. networkBMA. https://www.bioconductor.org/packages/release/bioc/html/networkBMA.html. Accessed 3 September 2017.
25. fastBMA docker image. https://hub.docker.com/r/biodepot/fastbma/. Accessed 3 September 2017.
26. Guelzim N, Bottani S, Bourgine P et al. Topological and causal structure of the yeast transcriptional regulatory network. Nat Genet 2002;**31**(1):60–63.
27. Annest A, Bumgarner RE, Raftery AE et al. Iterative Bayesian model averaging: a method for the application of survival analysis to high-dimensional microarray data. BMC Bioinformatics 2009;**10** 1:72.
28. Schwarz G. Estimating the dimension of a model. Ann Stat 1978;**6**(2):461–4.
29. Zellner A. On assessing prior distributions and Bayesian regression analysis with g-prior distributions. In: Goel PK, Zellner A, eds. Bayesian inference and decision techniques: essays in honor of Bruno de Finetti, vol. **6**. Amsterdam: North Holland; 1986:233–43.
30. Furnival GM, Wilson RW. Regressions by leaps and bounds. Technometrics 1974;**16** 4:499–511.
31. Madigan D, Raftery AE. Model selection and accounting for model uncertainty in graphical models using Occam's window. J Am Stat Assoc 1994;**89** 428:1535–46.
32. Wagner A. How to reconstruct a large genetic network from n gene perturbations in fewer than n2 easy steps. Bioinformatics 2001;**17**(12):1183–97.
33. Bosnacki D, Odenbrett MR, Wijs A et al. Efficient reconstruction of biological networks via transitive reduction on general purpose graphics processors. BMC Bioinformatics 2012;**13** 1:281.
34. Brent RP. Algorithms for Minimization Without Derivatives. Englewood Cliffs, NJ: Prentice Hall; 1973.
35. OpenMP. http://www.openmp.org/. Accessed 3 September 2017.
36. MPICH. https://www.mpich.org/. Accessed 3 September 2017.
37. qrupdate. http://sourceforge.net/projects/qrupdate/. Accessed 3 September 2017..
38. OpenBLAS. http://www.openblas.net. Accessed 3 September 2017.
39. Bloom BH. Space/time trade-offs in hash coding with allowable errors. Commun ACM 1970;**13**(7):422–6.
40. Melsted P, Pritchard JK. Efficient counting of k-mers in DNA sequences using a bloom filter. BMC Bioinformatics 2011;**12** 1:333.
41. smhasher. https://github.com/aappleby/smhasher. Accessed 3 September 2017.
42. Faith JJ, Hayete B, Thaden JT et al. Large-scale mapping and validation of *Escherichia coli* transcriptional regulation from a compendium of expression profiles. PLoS Biol 2007;**5**(1):e8.
43. Tibshirani R. Regression shrinkage and selection via the Lasso. J Royal Stat Soc Series B (Methodol) 1996;**58**(1):267–88.
44. Huynh-Thu VA, Irrthum A, Wehenkel L et al. Inferring regulatory networks from expression data using tree-based methods. PLoS One 2010;**5**(9).
45. Huynh-Thu VA, Sanguinetti G. Combining tree-based and dynamical systems for the inference of gene regulatory networks. Bioinformatics 2015;**31**(10):1614–22.
46. Stolovitzky G, Monroe D, Califano A. Dialogue on reverse-engineering assessment and methods: the DREAM of high-throughput pathway inference. Ann N Y Acad Sci 2007;**1115** 1:1–22.
47. Prill RJ, Saez-Rodriguez J, Alexopoulos LG et al. Crowdsourcing network inference: the DREAM predictive signaling network challenge. Sci Signal 2011;**4**(189):mr7.
48. Marbach D, Prill RJ, Schaffter T et al. Revealing strengths and weaknesses of methods for gene network inference. Proc Natl Acad Sci U S A 2010;**107**(14):6286–91.
49. Marbach D, Schaffter T, Mattiussi C et al. Generating realistic in silico gene networks for performance assessment of reverse engineering methods. J Comp Biol 2009;**16**(2):229–39.
50. Prill RJ, Marbach D, Saez-Rodriguez J et al. Towards a rigorous assessment of systems biology models: the DREAM3 challenges. PLoS One 2010;**5**(2):e9202.
51. Yeung KY, Dombek KM, Lo K et al. Construction of regulatory networks using expression time-series data of a genotyped population. Proc Natl Acad Sci U S A 2011;**108**(48): 19436–41.
52. Teixeira MC. The YEASTRACT database: a tool for the analysis of transcription regulatory associations in Saccharomyces cerevisiae. Nucl Acids Res 2006;**34**(90001):D446–51.
53. Trapnell C, Cacchiarelli D, Grimsby J et al. The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. Nat Biotechnol 2014;**32**(4): 381–6.
54. fastROCPRC. https://github.com/lhhunghimself/fastROCPRC. Accessed 3 September 2017.
55. Friedman JH, Hastie T, Tibshirani R. Regularization paths for generalized linear models via coordinate descent. J Stat Softw 2010;**33**(1):1–22.
56. DREAM4. https://www.synapse.org/#!Synapse:syn2825304/wiki/71131. Accessed 3 September 2017.