RESEARCH

Distributed Bayesian Networks Reconstruction on the Whole Genome Scale

Alina Frolova[1*] and Bartek Wilczynski[2]

*Correspondence:
a.o.frolova@imbg.org.ua
[1]Institute of Molecular Biology and Genetics, Zabolotnogo 150, 03680 Kyiv, Ukraine
Full list of author information is available at the end of the article

Abstract

Background: Bayesian networks are directed acyclic graphical models widely used to represent the probabilistic relationships between random variables. Recently, they have been applied in various biological contexts, including gene regulatory networks and protein-protein interactions inference. Generally, learning Bayesian networks from experimental data is NP-hard, leading to widespread use of heuristic search methods giving suboptimal results. However, in cases when the acyclicity of the graph can be ensured, it is possible to find the optimal network in polynomial time. While our previously developed tool BNFinder implements polynomial time algorithm, reconstructing networks with the large amount of experimental data still leads to numerous days of the computations given single CPU.

Results: In the present paper we propose parallelized algorithm designed for multi-core and distributed systems and its implementation in the improved version of BNFinder - our tool for learning optimal Bayesian networks. The new algorithm has been tested on simulated datasets as well as different experimental data showing that it has much better efficiency of parallelization than the previous version. When tested on the DREAM datasets in comparison with other methods, BNFinder gives consistently the best results in terms of the area under the ROC curve as well as in the number of positive predictions at the top of the prediction ranking. The latter is especially important for the purposes of the future experimental validation of the predictions.

Conclusions: We show that the new method can be used to reconstruct networks in the size range of thousands of genes making it practically applicable to whole genome datasets of prokaryotic systems and large components of eukaryotic genomes. Our benchmarking results on realistic datasets indicate that the tool should be useful to wide audience of researchers interested in discovering dependencies in their large-scale transcriptomic datasets.

Keywords: Bayesian networks learning; gene regulatory networks inference; distributed computing; DREAM challenge

Background

Bayesian networks (BNs) are graphical representations of multivariate joint probability distributions by factorization consistent with the dependency structure among variables. In practice, this often gives concise structures that are easy to interpret even for non-specialists. A BN is a directed acyclic graph with nodes representing random variables and edges representing conditional dependencies between the nodes. Nodes that are not connected represent variables that are conditionally independent of each other [1]. In general, inferring BN structure is NP-hard [2], however

it was shown by Dojer [3] that it is possible to find the optimal network structure in polynomial time when datasets are fixed in size and the acyclicity of the graph is pre-determined by external constraints. The latter is true when dealing with dynamic BNs or when user defines the regulation hierarchy restricting the set of possible edges in case of static BNs. This algorithm was implemented in BNFinder - a tool for BNs reconstruction from experimental data [4].

One of the common use of BNs in bioinformatics is inference of interactions between genes [5] and proteins [6]. Even though it was originally developed for this purpose, BNFinder is a generic tool and can be used not only for the reconstruction of gene regulatory networks from expression profiles. Since its original publication, it was successfully applied to linking expression data with sequence motif information [7], identifying histone modifications connected to enhancer activity [8] and to predicting gene expression profiles of tissue-specific genes [9]. Even though it can be applied to many different datasets, the practical usage of the algorithm is limited by its running times that can be relatively long. The bottleneck is the fact that the running time are defined by the most complex variable (which in case of genetic networks is a gene with the biggest number of parents/regulators). Since the algorithm published by Dojer [3] was relatively easily parallelizable, we have developed a new version of BNFinder that takes advantage of multiple cores via the multiprocessing python module.

Implementation

The general scheme of the learning algorithm is following: for each of the random variables find the best possible set of parent variables by considering them in a carefully chosen order of increasing cost function. Current BNFinder 2 version [10] can be considered **variable-wise** as it includes a simple parallelization based on distributing the work done on each variable between the different threads. However, such approach has natural limitations. Firstly, the number of parallelized tasks cannot exceed the number of random variables in the problem, meaning that in the cases where only a few variables are considered (e.g. in classification by BNs) we get a very limited performance boost. Secondly, **variable-wise** parallelization is vulnerable (in terms of performance) to the datasets with highly heterogeneous variables, i.e. variables whose true dependency graph has a wide range of connections. As the time spent on computing parents sets for different variables varies - it leads to randomly uneven load of threads. In biology we usually observe networks with scale-free topology consisting of a few hub nodes with many parents and a large number of nodes that have one or small number of connections [11]. If one applies **variable-wise** algorithm to such networks the potential gain in the algorithm performance is not greater than in the case where all the nodes have as many parents as the most biggest hub node.

While **variable-wise** algorithm seems to be the most straightforward one, it is also possible to consider different possible parents sets in parallel denoting **set-wise** algorithm. It means that in the first step we compute singleton parents sets using all available threads, in the second step we compute two-element parents sets in parallel and so on, until we reach parents sets size limit or score function limit. However, **set-wise** algorithm requires more synchronizations between the threads

[12] in comparison with **variable-wise**. As it is difficult to tell, which problem might be more important in practice, we have implemented and tested two approaches: **set-wise** only and **hybrid** one - a combination of **variable-wise** and **set-wise**. Obviously that **hybrid** algorithm gives exactly the same speedup as current version of BNFinder, which implements **variable-wise** algorithm, when the number of cores is fewer or equal to the number of random variables. In the opposite case (when the number of cores is greater than the number of random variables) the excessive number of cores is allocated to **set-wise** type of parallelism to get additional boost in the performance.

Figures 1,2 and 3 show python pseudocode for **variable-wise**, **set-wise** and **hybrid** algorithms accordingly, which was simplified in comparison to the original implementation for better illustration. As was stated above **set-wise** algorithm uses all given cores to compute parents sets for one gene and after finding parents it proceeds with the next gene. On the contrary **hybrid** algorithm uniformly distributes cores between genes, for example, if user has 3 genes in the network and 6 cores available, each gene will have 2 cores for computing its parents set. So, the complexity of **set-wise** (left side of inequality) and **hybrid** (right side of inequality) algorithms can be described in following way:

$$\frac{\sum_{i=1}^{n} t_i}{k} = \frac{\text{avg}_{i=1}^{n}\, t_i n}{k} \leq \frac{\max_{i=1}^{n} t_i n}{k}$$

where k is the core number, n is the number of random variables, and t is the time one needs to compute optimal parents set for one variable.

Thus, the time to reconstruct the whole network in case of **set-wise** approach is the sum of time needed for each random variable, which is in fact average time one spends on finding the parents set for one variable, while inferring BN with **hybrid** approach is bounded by the maximum time one spends on one variable.

Results

Performance testing

Algorithms comparison. We compared implementations of three different algorithms: **variable-wise**, **set-wise** and **hybrid**. The original implementation (**variable-wise**) serves as a baseline for computing the speedup and efficiency of the parallelization. For testing we used synthetic benchmark data as well as real datasets concerning protein phosphorylation network published by Sachs et al. [13]. **Set-wise** and **hybrid** algorithms performance on 20 genes synthetic network were almost identical, while the speedup of the **hybrid** algorithm was better - 34x versus 29x (See Figure 4). Obviously, for reasons described in the implementation section the **variable-wise** algorithm is reasonable to use only with 20 cores or less. Efficiency comparison showed that **hybrid** algorithm has more unstable behaviour, performing better when the number of cores correlates with the number of genes (see Figure 4).

Real experimental dataset showed significant difference between two algorithms. Clearly, the **set-wise** algorithm outperforms the **hybrid** one: with the efficiency of 0.5 it showed 8x speedup on phosphorylation network with 11 genes, while the **hybrid** algorithm showed only 1.5x speedup (See Figure 5). Again, **variable-wise**

algorithm cannot give any speedup with more than 11 cores. Tests on Sachs data revealed **hybrid** algorithm sensitivity to highly heterogeneous variables in the input data, because out of the 11 genes in the network only one gene has 6 parents, while others have 1-2 parents. Importantly, the better performing algorithm is also the one showing more consistent behaviour.

Even though both new algorithms are better than current BNFinder 2 version we consider **set-wise** algorithm better one, cause it gives more predictable outcome.

Tests on benchmark and Sachs data were performed on the same server with AMD Opteron (TM) Processor 6272 (4 CPUs with total of 64 cores) and 512GB RAM. During the tests server was loaded only by regular system processes, but to ensure statistical significance we performed each test several times, so Figures 4 and 5 represent average results.

Stress testing. After establishing that the best algorithm is the **set-wise**, we decided to test it on large datasets to see if we are able to use it in genome-wide analyses. Previously we compared BNFinder with Banjo software on the data provided with it [4, 14]. But for the new algorithm testing we chose Challenge 5 (Genome-Scale Network Inference) from DREAM2 competition (Dialogue for Reverse Engineering Assessments and Methods) [15, 16]. Challenge data is a log-normalized compendium of Escherichia coli expression profiles, which was provided by Tim Gardner to DREAM initiative [17]. The participants were not informed about the data origin and were provided only with 3456 genes x 300 experiments dataset and the list of transcription factors.

We tested BNFinder on Challenge 5 data with different parents sets limit parameter (i.e. number of potential parents) to see whether it has any effect on the accuracy of resulting network. Also it is important to know, that parents sets limit increases the computation time dramatically in non-linear way, especially in case of dataset with many variables. We compared **set-wise** algorithm performance with context likelihood of relatedness (CLR) algorithm - an extension of the relevance networks approach, that utilizes the concept of mutual information [17]. We chose CLR, because it is very fast and easy to use tool, which provides good results. In addition, CLR-based algorithm - synergy augmented CLR (SA-CLR) was best performed algorithm on Challenge 5 [18].

Table 1 DREAM2 Challenge 5 data testing. CLR with cutoff means limiting output results to 100000 genes interactions. l stands for BNF parents sets limit.

	CLR	CLR with cutoff	BNF, l=1	BNF, l=2	BNF, l=3
CPU time, hours	0.7879	0.1999	2.0021	383.7149	109200
Actual time, hours	0.2626	0.0666	0.0667	12.7904	336
CPU number	3	3	30	30	∼325

The CLR tests were performed on the GP-DREAM platform, designed for the application and development of network inference and consensus methods [19]. BNFinder tests for parents sets limit 1 and 2 were performed on the same server as previous BNFinder tests in this paper, however, with the limit 3 we had to use more powerful environment. In this case it was Ukrainian Grid Infrastructure [20], which was accessed through nordugrid-arc middleware only (arc client version 4.1.0), so the tasks submitting process was automated and unified [21]. The results in Table

1 are not precisely comparable due to differences in used hardware, especially when using such heterogeneous environment as Grid. Also we couldn't obtain stable number of cores over time with the Grid, as the clusters were loaded with other tasks. So, during the experiment we were utilizing from 250 to 400 cores at the same time.

Even though computing with parents sets limit 3 takes significant amount of time and resources, it is clear that BNFinder is able to reconstruct genome-scale datasets, significantly broadening its application range. Moreover, BNFinder is well adapted to parallel and distributed computing.

Accuracy testing

Previously we compared accuracy of BNFinder algorithm with Banjo [4] on data provided with it and separately on Sachs data [10], which we used in this work to test the performance. However, as we already had results from DREAM testing, it was reasonable to test both BNFinder and CLR with Challenge 5 gold standard network.

As we've stated before, BNFinder requires external constraints ensuring network acyclicity for inferring static Bayesian networks (with dynamic Bayesian networks you might have the loops). In the case of Challenge 5 data such constrains are encoded in the transcription factors list. Since allowing TF-TF interactions would lead to the cycles in the network, BNFinder is unable to infer interactions between TFs. In order to make a more realistic comparison of the results we compared BNFinder result with a slightly modified gold standard network, where TF-TF interactions were excluded. Overall, it didn't influence the total accuracy much as the gold standard contains very few of such interactions. As CLR is unable to infer the sign of interaction (inhibition or activation) we used unsigned gold standard. The results are presented in Table 2.

Table 2 Algorithms accuracy test on DREAM2 Challenge 5 data. BNFinder and CLR are compared with the best scored method. Area Under the PR and ROC Curves are considered as well as precision at n-th correct prediction. BNFinder is used with parents sets limit 1 and suboptimal parents sets 100, CLR is used with default parameters.

	CLR	BNF	BNF, gold standard without TF-TF	Winner: Team 48
AUPR	0.051398	0.028769	0.030584	0.059499
AUROC	0.617187	0.606326	0.629420	0.610643
n=1	1	1	1	1
n=2	1	1	1	1
n=5	0.5	0.714286	0.714286	0.714286
n=20	0.588235	0.8	0.8	0.689655
n=100	0.591716	0.164745	0.164745	0.675676
n=500	0.025204	0.014771	0.014771	0.036044

It is important to note that the overall AUROC is comparable between the methods with BNFinder leading in the gold standard without TF-TF interactions. Even more relevant is the fact that the fraction of positive predictions at the very top 20 of the rank is by far the highest for BNFinder.

Exploring BNFinder parameter space. The main advantage of BNFinder in comparison with heuristic search Bayesian tools such as Banjo is that BNFinder reconstructs optimal networks, which also means that with the same parameters one will always get the same result. However, with BNFinder one can use number of parameters such as scoring functions (Bayesian-Dirichlet equivalence, Minimal Description

Length or Mutual information test), static or dynamic (also with self-regulatory loops) BNs, perturbation data, or even prior information on the network structure. All of these may alter results significantly, so, naturally we are interested in choosing best parameters for a particular dataset. Here we studied the impact of two very important parameters: parents sets limit and number of suboptimal parents sets (gives alternative sets of regulators with lower scores). The results are shown in Table 3.

Table 3 Testing BNFinder with different parameters on DREAM2 Challenge 5 data.

	Subparents: 25			Subparents: 50		Subparents: 100	
Parents sets limit	1	2	3	1	2	1	2
AUPR	0.028645	0.030669	0.023191	0.028686	0.031859	0.028769	0.031372
AUROC	0.573914	0.574862	0.570626	0.587847	0.587919	0.606326	0.597899
n=1	1	0.333333	0.125	1	0.5	1	0.333333
n=2	1	0.4	0.222222	1	0.5	1	0.5
n=5	0.714286	0.625	0.102041	0.714286	0.714286	0.714286	0.555556
n=20	0.8	0.714286	0.162602	0.8	0.714286	0.8	0.714286
n=100	0.167224	0.241546	0.205761	0.164745	0.251256	0.164745	0.249377
n=500	0.018126	0.024327	0.024021	0.015903	0.023938	0.014771	0.023157

For parents sets limit 3 we have used only 25 subparents (suboptimal parents sets), because we did not see significant improvement when it was increased and it would have been too extensive to complete the computations in realistic time for larger parents limit sizes.

The gold standard consists of TFs and sigma-factors regulations taken from RegulonDB version 4 [15]. As stated in [15] RegulonDB is 85% complete, so some false-positives may be incorrectly assigned. In addition, only 771 out 1095 genes in the gold standard have from 1 to 3 parents (regulators), while the maximum number is 10 (see Table 4), which means that parents sets limit 3 is not nearly enough to infer true interactions.

In Table 5 we have summarized the total number of interactions returned by BNFinder with different maximal parents per gene and different number of suboptimal parent sets. The results indicate that increasing the size of the allowed parent set leads to the decrease in the total returned edges in the network. This may seem surprising at first, but it is consistent with highly overlapping suboptimal parents sets. In such case, when we allow parent sets to be larger, it leads to actually removing at least some spurious interactions.

Taken together with the results shown in Table 3, it leads to the conclusion, that if the user is interested in the very top of the strongest interactions in the network, he or she should use small numbers of sub-optimal parent sets and small limit on the parent-set size. However, if one is interested in discovering the more global picture of the true regulatory network, one should focus on the higher number of sub-optimal parent sets with limit on the set size as high as it is computationally feasible. In the particular example of the DREAM network studied in our work, it seems that looking at 50 or 100 subparents gives best accuracy when looking at the top 100 predicted edges.

Table 4 The number of genes parents in DREAM2 Challenge5 gold standard. The total number of genes in gold standard is 1095.

Parents number	1	2	3	4	5	6	7	8	9	10
Genes number	269	317	185	125	111	42	37	5	2	2

Table 5 The number of the interactions in the output network based on different BNFinder parameters.

	Subparents=25	Subparents=50	Subparents=100
Parent limit=1	78366	156685	313061
Parent limit=2	65108	116424	213389

Given the results, we advise that before applying BNFinder to any kind of data one should study the data origin and consider the vertices degree in the network graph in order to choose optimal parents sets limit and number of suboptimal parents.

Discussion

DREAM results interpretation. As we stated above gold standard provided by DREAM2 Challenge 5 cannot be considered as a complete one. While it is hard to estimate the magnitude of the problem exactly, we can assume there is a substantial number of results incorrectly classified as false-positives. Experimentally validating all inferred interactions does not seem to be possible as it would take too much time to be practical. Therefore researches usually verify only a selected few of them [17]. It means that results in Table 2 are not absolute and tested tools could behave differently on the different datasets, not even mentioning that AUPR characteristic is way too low for all the tools.

To be more precise only 1146 from 3456 genes are present in gold standard (1095 genes and 152 transcription factors, some of them counted among the 1095 [15]). Therefore DREAM evaluation scripts exclude interactions, which are not in gold standard, from predicted network before computing AUROC and AUPR. Assuming the dataset covers all possible states of the biological system the inferring of each interaction is equally difficult/simple. However, in the reality we have to deal with groups of genes, some of them are differentially expressed and some are not. It is possible to imagine the situation that some tool perfectly predicted interactions not present in gold standard and poorly predicted other interactions, which happened to be not differentially expressed. Basically, using DREAM dataset and evaluation scripts we compare how the tools predicted gold standard only interactions, while not comparing at all how the tools inferred others. As we do not know the design of the experiments we cannot find differentially expressed genes, but we can make more accurate comparison between the tools by excluding non gold standard genes from the dataset before reconstructing the networks.

As stated in challenge synopsis the data was log-normalized [16]. So, first of all we performed quality check with ArrayQualityMetrics R packet [22] and quantile normalizing [23], which is shown in Figure 6. After that we excluded all the genes, which are not present in the gold standard interactions list, and inferred the networks on modified data. The results in Table 6 show that accuracy of both tools increased significantly, which is caused by data dimensionality reduction and by the fact, that we do not exclude highly ranked interactions (which are not in the gold standard). We also used BNFinder with increased number of parents and suboptimal parents sets, which gave us even better result. Table 6 represents more precise tools comparison, however, we still believe that such evaluation of inference methods are far from giving us full-scale and optimal picture of accuracy performance, so experimental validation is still needed.

Table 6 Testing BNFinder on modified DREAM data. Full dataset is unmodified DREAM2 Challenge 5 dataset with 3456 genes, while gold standard genes only dataset contains 1146 genes. CLR is used with default parameters. BNF: l stands for parents limit parameter, i - for number of suboptimal parents.

	Full dataset		Gold standard genes only dataset		
	CLR	BNF	CLR	BNF, l=1, i=100	BNF, l=2, i=300
Precision-Recall	0.051398	0.028769	0.069420	0.048650	0.060894
ROC curve	0.617187	0.606326	0.710807	0.733716	0.808610
n=1	1	1	1	1	1
n=2	1	1	1	1	1
n=5	0.5	0.714286	1	1	1
n=20	0.588235	0.8	0.740741	1	1
n=100	0.591716	0.164745	0.628931	0.373134	0.442478
n=500	0.0252	0.014771	0.054295	0.038285	0.053787

Scalability and optimization. Even though **set-wise** algorithm with only one level of parallelization (between different parents sets) are proved to be faster and more efficient on the single computational unit (e.g. one physical server) one might argue that such algorithm is not scalable when using distributed computing. Nevertheless, we successfully used BNFinder on 4 different clusters incorporated in Ukrainian Grid. This task was achieved by introducing new parameter called "subset", which allows user to specify the subset of variables for which BNFinder will compute the parents sets. The variables in the subset are considered sequentially, i.e. **set-wise** algorithm is used. Such approach allowed us to distribute genes subsets among all the clusters, so we could use computational resources more efficiently. Moreover, new parameter could be useful for those researches who are interested only in particular variables, but not the whole network.

It is important to note, that using parents set limit might remove the problem of highly heterogeneous variables in the input data, especially if one uses very low limit. Consider taking benchmark data we used before with parents sets limit 2 and 6, which reduces the number of tasks for **set-wise** algorithm. Therefore time spent on each random variable in case of parents set limit 2 should be almost equal, making usage of **hybrid** algorithm more preferable. Figure 7 shows the difference between two algorithms in accordance with different parents set limit. As limit grows the complexity of finding parents sets grows in non-linear way, thus **set-wise** algorithm predictably shows better results, so we can compare it with Fig. 4, where limit are not used. Taking into account this important note we optimized BNFinder to use **hybrid** algorithm, when limit is less or equal to 3, and **set-wise** algorithm in the opposite case. We chose parents sets limit 3, cause in most cases using bigger limit significantly increases computational complexity.

Conclusions

Here we presented new improved version of BNFinder algorithm for Bayesian networks reconstruction. Our method is highly parallel and gives the research community the ability to obtain the results much faster using the publicly available grids and clusters. Along with the parallelization we showed that BNFinder implementation is scalable enough to run it on any kinds of distributed heterogeneous systems (clusters, Grid). Such improvement allowed us to significantly extend the application range of the tool, which was confirmed by reconstructing genome-scale dataset from DREAM2 Challenge 5 competition. We additionally tested BNFinder performance and accuracy on synthetic and real biological data, which showed that

BNFinder is competitive with the best-performing inference methods. We provide the new BNFinder implementation freely for all interested researchers under a GNU GPL license.

Availability and requirements
Project name: BNFinder
Project home page: https://github.com/sysbio-vo/bnfinder
Supplemental material: https://github.com/sysbio-vo/bnfinder/releases/tag/v2.2
Operating system(s): Platform independent
Programming language: Python
Other requirements: Python 2.4 or higher. Python 3 is not supported
License: GNU GPL Library version 2
Any restrictions to use by non-academics: None

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

Author's contributions
All the computational experiments, the implementation and majority of text writing were done by AF. BW contributed the initial idea of parallelizing BNFinder, supervised the study and helped with finishing the manuscript.

Acknowledgements
This work was partially supported by the National Center for Science grant (decision number DEC-2012/05/B/N22/0567) and Foundation for Polish Science within the SKILLS programme. Also this work was supported by National program of Grid technologies implementation and usage in Ukraine (grant number 69-53/13 2013).
For calculations authors used computational clusters of Taras Shevchenko National University of Kyiv, Institute of Molecular Biology and Genetics of NASU, Institute of Food Biotechnology and Genomics of NASU, joint cluster of SSI "Institute for Single Crystals" and Institute for Scintillation Materials of NASU.
We would like to express out gratitude to Prof. Maria Obolenska from Institute of Molecular Biology and Genetics for the proofreading and critique.

Author details
[1]Institute of Molecular Biology and Genetics, Zabolotnogo 150, 03680 Kyiv, Ukraine. [2] Institute of Informatics, University of Warsaw, Banacha 2, 02-089 Warsaw, Poland.

References

1. Friedman, N., Koller, D.: Being bayesian about network structure. a bayesian approach to structure discovery in bayesian networks. Machine learning **50**(1-2), 95–125 (2003)
2. Chickering, D.M., Heckerman, D., Meek, C.: Large-sample learning of bayesian networks is np-hard. The Journal of Machine Learning Research **5**, 1287–1330 (2004)
3. Dojer, N.: Learning bayesian networks does not have to be np-hard. In: Královic, R., Urzyczyn, P. (eds.) Mathematical Foundations of Computer Science 2006: 31st International Symposium, MFCS 2006, Stará Lesná, Slovakia, August 28-September 1, 2006, Proceedings. LNCS sublibrary: Theoretical computer science and general issues, pp. 305–314. Springer, Berlin/Heidelberg (2006)
4. Wilczyński, B., Dojer, N.: Bnfinder: exact and efficient method for learning bayesian networks. Bioinformatics **25**(2), 286–287 (2009)
5. Zou, M., Conzen, S.D.: A new dynamic bayesian network (dbn) approach for identifying gene regulatory networks from time course microarray data. Bioinformatics **21**(1), 71–79 (2005)
6. Jansen, R., Yu, H., Greenbaum, D., Kluger, Y., Krogan, N.J., Chung, S., Emili, A., Snyder, M., Greenblatt, J.F., Gerstein, M.: A bayesian networks approach for predicting protein-protein interactions from genomic data. Science **302**(5644), 449–453 (2003)
7. Dabrowski, M., Dojer, N., Zawadzka, M., Mieczkowski, J., Kaminska, B.: Comparative analysis of cis-regulation following stroke and seizures in subspaces of conserved eigensystems. BMC systems biology **4**(1), 86 (2010)
8. Bonn, S., Zinzen, R.P., Girardot, C., Gustafson, E.H., Perez-Gonzalez, A., Delhomme, N., Ghavi-Helm, Y., Wilczyński, B., Riddell, A., Furlong, E.E.: Tissue-specific analysis of chromatin state identifies temporal signatures of enhancer activity during embryonic development. Nature genetics **44**(2), 148–156 (2012)
9. Wilczynski, B., Liu, Y.-H., Yeo, Z.X., Furlong, E.E.: Predicting spatial and temporal gene expression using an integrative model of transcription factor occupancy and chromatin state. PLoS computational biology **8**(12), 1002798 (2012)
10. Dojer, N., Bednarz, P., Podsiadło, A., Wilczyński, B.: Bnfinder2: Faster bayesian network learning and bayesian classification. Bioinformatics, 323 (2013)
11. Barabasi, A.-L., Oltvai, Z.N.: Network biology: understanding the cell's functional organization. Nature Reviews Genetics **5**(2), 101–113 (2004)
12. McCool, M., Reinders, J., Robison, A.: Structured Parallel Programming: Patterns for Efficient Computation. Elsevier, Waltham, MA (2012)
13. Sachs, K., Perez, O., Pe'er, D., Lauffenburger, D.A., Nolan, G.P.: Causal protein-signaling networks derived from multiparameter single-cell data. Science **308**(5721), 523–529 (2005)
14. Smith, V.A., Yu, J., Smulders, T.V., Hartemink, A.J., Jarvis, E.D.: Computational inference of neural information flow networks. PLoS computational biology **2**(11), 161 (2006)

15. Stolovitzky, G., Prill, R.J., Califano, A.: Lessons from the dream2 challenges. Annals of the New York Academy of Sciences **1158**(1), 159–195 (2009)
16. DREAM2, Challenge 5 Synopsis. http://wiki.c2b2.columbia.edu/dream/index.php?title=D2c5
17. Faith, J.J., Hayete, B., Thaden, J.T., Mogno, I., Wierzbowski, J., Cottarel, G., Kasif, S., Collins, J.J., Gardner, T.S.: Large-scale mapping and validation of escherichia coli transcriptional regulation from a compendium of expression profiles. PLoS biology **5**(1), 8 (2007)
18. Watkinson, J., Liang, K.-c., Wang, X., Zheng, T., Anastassiou, D.: Inference of regulatory gene interactions from expression data using three-way mutual information. Annals of the New York Academy of Sciences **1158**(1), 302–313 (2009)
19. Reich, M., Liefeld, T., Gould, J., Lerner, J., Tamayo, P., Mesirov, J.P.: Genepattern 2.0. Nature genetics **38**(5), 500–501 (2006)
20. Ukrainian National Grid Infrastructure. http://infrastructure.kiev.ua/en/
21. Ellert, M., Grønager, M., Konstantinov, A., Kónya, B., Lindemann, J., Livenson, I., Nielsen, J.L., Niinimäki, M., Smirnova, O., Wäänänen, A.: Advanced resource connector middleware for lightweight computational grids. Future Generation computer systems **23**(2), 219–240 (2007)
22. Kauffmann, A., Gentleman, R., Huber, W.: arrayqualitymetrics—a bioconductor package for quality assessment of microarray data. Bioinformatics **25**(3), 415–416 (2009)
23. Bolstad, B.M., Irizarry, R.A., Åstrand, M., Speed, T.P.: A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. Bioinformatics **19**(2), 185–193 (2003)

Figures

```
# Learns optimal parents set of size <= limit
def learn_parents(gene, data, parents, limit):
    parents_set_size = 1
    while acceptable_score(graph_score + data_score) and
            (parents_set_size <= limit):
        # Compute scores for all possible parents sets of parents_set_size
        for genes_subset in subsets(parents, parents_set_size):
            data_score = score_function(select_data_subset(genes_subset), gene)
            graph_score = score_function(genes_subset, gene)
        parents_set_size += 1

# Pool function argument denotes the number of processes to create
pool = multiprocessing.Pool(cores)
# Applying learn_parents() function to all the genes
results = pool.map(learn_parents, [(gene, data, regulators, limit)
    for gene in genes])
```

Figure 1 Variable-wise algorithm python pseudocode.

Additional Files

Additional file 1 — BNFinder source code

BNFinder is written on python, so you will need python 2.4 or higher in order to run it.

```
# Learns optimal parents set of size <= limit
def learn_parents(gene, data, parents, limit):
    parents_set_size = 1
    # Pool function argument denotes the number of processes to create
    inner_pool = multiprocessing.Pool(cores)
    while acceptable_score(graph_score + data_score) and
            (parents_set_size <= limit):
        # Compute scores for all possible parents sets of parents_set_size
        # in parallel
        data_score = inner_pool.map(score_function,
            [select_data_subset(data, genes_subset), gene]
            for genes_subset in subsets(parents, parents_set_size))
        graph_score = inner_pool.map(score_function, [genes_subset, gene]
            for genes_subset in subsets(parents, parents_set_size))
        parents_set_size += 1

# Applying learn_parents() function to all the genes
results = map(learn_parents, [(gene, data, regulators, limit)
    for gene in genes])
```

Figure 2 Set-wise algorithm python pseudocode.

```
# Learns optimal parents set of size <= limit
def learn_parents(gene, data, parents, limit):
    parents_set_size = 1
    # Pool function argument denotes the number of processes to create
    inner_pool = multiprocessing.Pool(cores/genes)
    while acceptable_score(graph_score + data_score) and
            (parents_set_size <= limit):
        # Compute scores for all possible parents sets of parents_set_size
        # in parallel
        data_score = inner_pool.map(score_function,
            [select_data_subset(data, genes_subset), gene]
            for genes_subset in subsets(parents, parents_set_size))
        graph_score = inner_pool.map(score_function, [genes_subset, gene]
            for genes_subset in subsets(parents, parents_set_size))
        parents_set_size += 1

pool = multiprocessing.Pool(cores)
# Applying learn_parents() function to all the genes in parallel
results = pool.map(learn_parents, [(gene, data, regulators, limit)
    for gene in genes])
```

Figure 3 Hybrid algorithm python pseudocode.

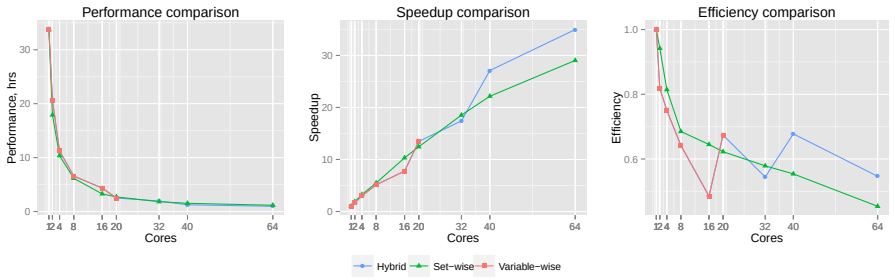

Figure 4 Synthetic data testing. Comparing performance, speedup and efficiency of algorithms on synthetic benchmark data: 20 variables x 2000 observations.

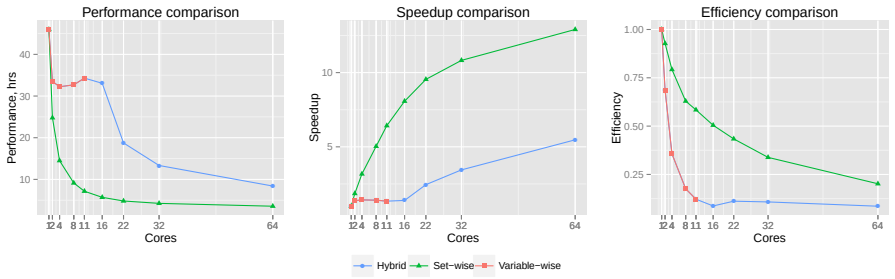

Figure 5 Biological data testing. Comparing performance, speedup and efficiency of algorithms on protein phosphorylation data: 11 variables x 1023 observations [13].

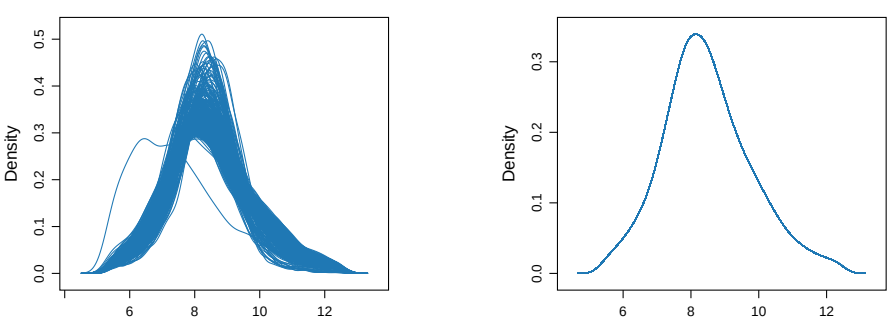

Figure 6 Density plots comparison. Quality control of DREAM2 Challenge 5 data before quantile normalization (left) and after (right).

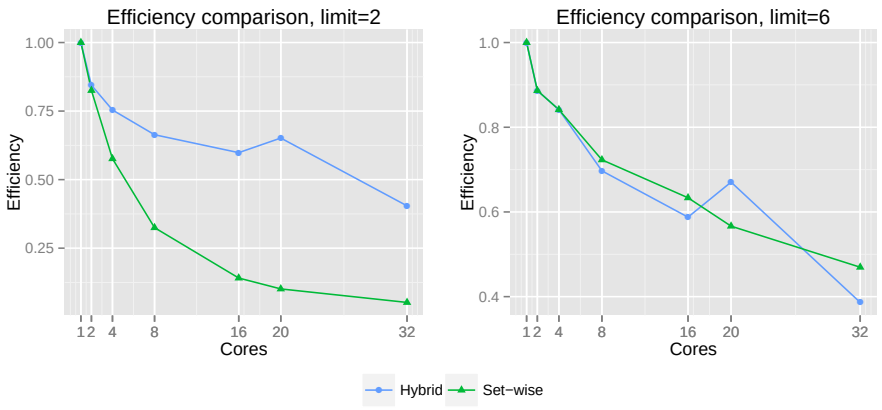

Figure 7 Synthetic data testing with different parents sets limit. Comparing efficiency of set-wise and hybrid algorithms with parents sets limit 2 and 6.