A Gossip-Based System for Fast Approximate Score Computation in Multinomial Bayesian Networks

Arun Zachariah, Praveen Rao, Anas Katib, Monica Senapati
University of Missouri-Kansas City
az2z7@mail.umkc.edu, raopr@umkc.edu, anaskatib@mail.umkc.edu, msenapati@mail.umkc.edu

Kobus Barnard
University of Arizona
kobus@cs.arizona.edu

***Abstract*—In this paper, we present a system for fast approximate score computation, a fundamental task for score-based structure learning of multinomial Bayesian networks. Our work is motivated by the fact that exact score computation on large datasets is very time consuming. Our system enables approximate score computation on large datasets in an efficient and scalable manner with probabilistic error bounds on the statistics required for score computation. Our system has several novel features including gossip-based decentralized computation of statistics, lower resource consumption via a probabilistic approach of maintaining statistics, and effective distribution of tasks for score computation using hashing techniques. The demo will provide a real-time and interactive experience to a user on how our system employs the principle of gossiping and hashing techniques in a novel way for fast approximate score computation. The user will be able to control different aspects of our system's execution on a cluster with up to 32 nodes. The approximate scores output by our system can be then used by existing score-based structure learning algorithms.**

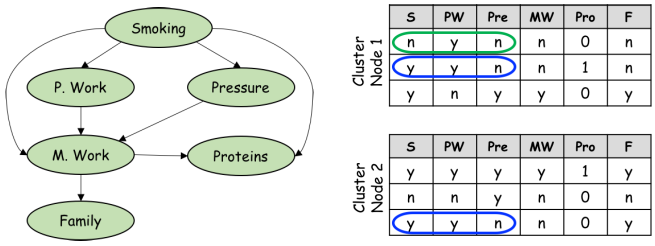
I. Introduction

Scalable machine learning (ML) systems are becoming indispensable for data-driven decision making on large datasets and are key drivers of business for companies like Amazon, Baidu, Google, Facebook, Microsoft, and others. Several new frameworks have emerged for scalable ML tasks (e.g., GraphLab [11], MLlib [12], Parameter Server [10], Petuum [16], SystemML [3]). Although statistical models provide an elegant framework to gain knowledge from data [5], the volume and variety of big data demands a paradigm shift due to the prevalence of massive heterogeneous datasets, which are popularly stored and analyzed using large-scale commodity clusters.

In this work, we focus on Bayesian networks (BNs) [14], which provide a natural way for knowledge representation and reasoning over heterogeneous data under uncertainty. BNs have been successfully used in many areas including medical/fault diagnosis, bioinformatics and computational biology, automated reasoning systems, and data clustering. A BN compactly encodes the joint probability distribution of a set of random variables/features of a dataset using a set of conditional probabilities of these variables given their parents in a directed acyclic graph (DAG). To learn a BN from a dataset, we need to learn its structure and the parameters of the conditional probability distributions that best fit the observed data. Then probabilistic inference queries can be posed on the BN to gain meaningful insights from the dataset.

Score-based structure learning [9] is one class of approximate structure learning algorithms wherein a search space of possible structures is searched by applying a scoring function. However, when a dataset is large and stored in a cluster, *it is essential to first compute the scoring function on the dataset in a scalable and efficient manner* for efficient structure learning. Our system called DiSC (**Di**stributed **S**core **C**omputation) is precisely designed for this purpose and enables fast approximate score computation over large-scale distributed data. The novel features of DiSC include: (a) gossip-based, decentralized computation of statistics required for score computation; (b) lower resource consumption via a probabilistic approach of maintaining statistics for score computation; and (c) effective distribution of tasks for score computation (on large datasets) by synergistically combining consistent hashing [15] and locality sensitive hashing (LSH) [7]. The approximate scores computed by DiSC can then be used for score-based structure learning in tools like R. Complete details about the design, implementation, and evaluation of DiSC can be found in a recent publication [8].

II. Background and Motivation



(a) (b)

Fig. 1. (a) Example of a BN (b) Data records distributed across 2 nodes

Consider a coronary heart disease dataset[1] consisting of 6 binary random variables, namely, `Smoking` (*S*), `Physical Work` (*PW*), `Mental Work` (*MW*), `Pressure` (*Pre*), `Proteins` (*Pro*), and `Family` (*F*). Let Figure 1(a) denote the BN learned on these variables using a score-based structure learning algorithm. The algorithm uses a scoring function (e.g., Bayesian Dirichlet equivalence (BDe) score) to compute the posterior probability distribution of a possible BN given the data. At each step in the search, it attempts to improve the overall score of the BN by modifying the DAG structure via local steps such as edge deletion, addition, reversal, etc., and

[1] https://rdrr.io/cran/bnlearn/man/coronary.html

compute a score difference of the affected variables. Different search strategies can be used by the algorithm (e.g., greedy hill-climbing), and when the network score does not improve further, it terminates.

To compute the score of a BN using a decomposable scoring function [4], we must first compute the score of each variable given its parents, a fundamental task during structure learning. Given a variable X with a set of parent variables denoted by Pa_X, $X|Pa_X$ is called a family. To compute a family's score, *sufficient statistics* of the family are needed [9]. Suppose the dataset is distributed across two cluster nodes as shown in Figure 1(b). The sufficient statistics of $S|PW = y, Pre = n$ is denoted by a pair of frequency counts (1, 2). This is because there is *one* data record where $S = n$, $PW = y$, and $Pre = n$ (green box) and *two* data records where $S = y$, $PW = y$, and $Pre = n$ (blue boxes). The sufficient statistics of the family $S|PW, Pre$ is a 2D array with 8 frequency counts, considering all possible assignments of PW and Pre. This example shows that to efficiently compute the scoring function, the sufficient statistics of every possible family required during score-based learning must be computed in a scalable and efficient manner. Therefore, we formulate this task on large-scale distributed data as a *scalable data aggregation problem*.

Several technical challenges must be addressed to develop an effective solution on large datasets. First, data blocks are distributed across nodes in a cluster, and it is infeasible to move all the blocks to one node for computing the sufficient statistics. Second, the score computation should be efficient and scalable, tolerate node failures/packet losses and changes to the cluster topology, and provide provable guarantees on the accuracy of the estimated sufficient statistics. This requires fast computation of *sums* over distributed data, effective load balancing of tasks, and redundancy to cope with failures. Although a straightforward application of a gossip algorithm for computing sums [13], hereinafter referred to as SUM, sounds promising, it unfortunately does not yield an efficient solution as all the families will be stored and processed by every cluster node leading to high network bandwidth consumption especially when number of families is very large [8]. This can arise when an input dataset has large number of random variables. A good load balancing approach is needed. Thus, one must rethink how this gossip algorithm can be adapted for sufficient statistics computation. Third, when new data are produced, efficient recomputation of family scores is required for faster relearning on large datasets.

Note that every data record in a dataset/table contributes to the sufficient statistics of a family. Therefore, every node will be involved in computing the sufficient statistics of a family. Furthermore, horizontally partitioning the dataset/table will avoid a family from being split across different partitions and therefore, prevent data shuffling as an entire data record will be on a single machine.

III. Design of DiSC

We present the design of DiSC and its novel aspects to enable fast approximate score computation on large datasets. The key ideas include effective load balancing by distributing families across cluster nodes using LSH and consistent hashing, decentralized computation of sufficient statistics of families using the principle of gossiping, and probabilistic dropping of families at nodes to lower network bandwidth consumption. Figure 2 shows the overall system architecture of DiSC and its three core components: *the Router*, *the Initializer*, and *the Manager*. Next, we describe each component and its benefit.

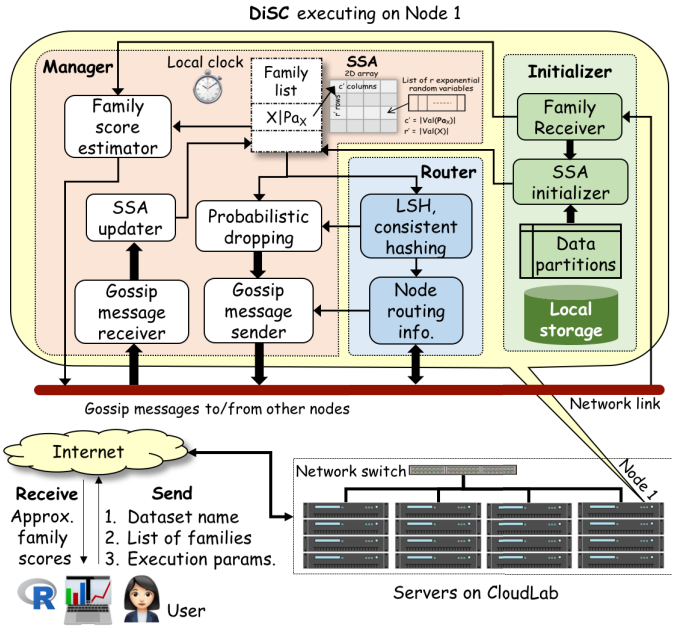


Fig. 2. System overview of DiSC

The Router is key to achieving load balancing by distributing families across cluster nodes. Given a family $X|Pa_X$, it synergistically combines LSH (for sets) and consistent hashing to produce K hash IDs of 160 bits each [8]. [2] The 160-bit hash address space is equally divided across cluster nodes similar to systems like Dynamo [6]. Each family is assigned to K cluster nodes, which provides redundancy to tolerate node failures. We say that K nodes will be *responsible for* one family. Due to LSH, similar families are highly likely to be assigned to the same node, which can reduce the network lookup cost when computing scores of similar families.

The Initializer at a node receives families that the node is responsible for (based on hashing). The sufficient statistics of the assigned families are computed using the local data partitions available at that node. The Initializer exploits multiple cores to speed up the initialization process. For each family $X|Pa_X$, a 2D array of size $r' \times c'$ is maintained, where r' denotes the number of values that can be assigned to X and c' denotes the number of possible assignments of values to variables in Pa_X. This 2D array is called the sufficient statistics array (SSA) of the family. Each element in this 2D array has a list of r independent exponential random variables initialized based on the frequency counts computed on the local data partitions. (The value of r controls the accuracy of

[2] A total of $L \times K$ random linear hash functions are used by LSH.

the estimates computed by DiSC.) That is, if the frequency count corresponding to an element of the SSA (computed on the local partitions) is some value α, then each of the r independent exponential random variables is defined with rate $\lambda = \alpha$. The Initializer initializes the family list on the node containing all the families that the node is responsible for along with their respective SSAs. During gossiping, wherein a node picks another node at random and exchanges the SSAs, the exponential random variables are updated. Over time, each node's SSA will become more accurate in estimating the sufficient statistics.

The Manager is responsible for gossip-based decentralized computation of sufficient statistics while ensuring that the average family list size is a limited fraction of the total number of families. We draw inspiration from SUM [13], a state-of-the-art gossip algorithm to compute the sum of values stored on a set of nodes. The Manager maintains a local clock that ticks at the times of rate 1 Poisson process. At each clock tick, the Manager say on node n_i) picks another node n_j at random and exchanges the SSAs with n_j. Suppose n_i and n_j have a family f in their family list. They essentially exchange the SSAs of f. Let $[E_{i1}, \ldots, E_{ir}]$ denote the exponential random variables maintained by n_i for an element in the SSA of f. Let $[E_{j1}, \ldots, E_{jr}]$ denote the exponential random variables maintained by n_j for the corresponding element in the SSA of f. After gossiping, n_i and n_j will update each element in their SSAs for f. For example, the list of exponential random variables for the aforementioned element in the SSA is updated to $[\min(E_{i1}, E_{j1}), \ldots, \min(E_{ir}, E_{jr})]$ on both nodes. This is because SUM is based on a key property: the minimum of q exponential random variables with rate $x_1, \ldots, x_q$, respectively, is an exponential random variable with rate $x_1 + \ldots + x_q$. When a family is received during gossiping but not in the family list of a node, then it is added to the family list along with its SSA.

Straightforward application of SUM causes the family list of all nodes to maintain all the families, thereby increasing the size of messages sent during gossiping and the total network bandwidth consumption [8]. This also defeats the goal of achieving load balancing. Therefore, the Manager employs a probabilistic approach to drop families in the family list that the node is not responsible for. The intuition is that if n_i picks n_j to exchange SSAs where n_j is responsible for a family, then that family is dropped with a higher probability after exchanging the SSAs. If a normal distribution is used to define the probability of dropping a family at different nodes, by using the property of Markov chains, probabilistic bounds can be computed on the convergence speed of DiSC and the expected family list size [8]. The exchange of gossip messages is optimized: When a family list and the SSAs cannot fit in a single UDP packet, they are broken into multiple packets. In addition, the messages are compressed to reduce the network bandwidth consumption. To compute the sufficient statistics of a family f, the Manager must estimate the frequency counts corresponding to the elements in the SSA of f. Suppose $[E'_{i1}, \ldots, E'_{ir}]$ denote the r exponential random variables maintained by n_i for an element in the SSA of f after DiSC completes gossiping. Then the estimated frequency count for that element in the SSA is given by $\frac{r}{\sum_{q=1}^{r} E'iq}$ [13]. Different decomposable scoring functions [4] can be computed once the sufficient statistics of families is available.

The Manager can incrementally update the sufficient statistics of families given new data records without processing the entire dataset again. To do so, the sufficient statistics are computed for all the families using the new data records. The exponential random variables are generated as before based on the frequency counts obtained for the new records. For each family, one of the nodes is chosen so that the minimum of exponential random variables is computed with the existing SSA for the family. The gossiping phase is restarted thereby updating the sufficient statistics of the families.

IV. Demonstration Scenarios

DiSC was developed using Java, Scala, Python, and a publicly available gossip package.[3] The user interface (UI) was developed using Django and CanvasJS. (The code is available at https://github.com/UMKC-BigDataLab/DiSC.) DiSC will run on CloudLab [1], a testbed for cloud computing, using up to 32 physical nodes in CloudLab's Utah data center. Each node is installed with Ubuntu 16.04.10 and has a 10-core Intel processor, 480 GB SSD, and 64 GB RAM. As DiSC outperformed the MapReduce-style computation of sufficient statistics both in terms of speed and accuracy (when sampling was used) [8], we will not execute the MapReduce-style computation during the demo. The reader is referred to a recent publication for detailed performance evaluation results [8].

Figures 3(a) and 3(b) show two screenshots of DiSC. Below we present four interactive and engaging scenarios for a user.

• *Scenario 1:* The first scenario will enable the user to configure DiSC to run on CloudLab. The user will select the number of nodes to use, dataset name, list of families for score computation, and execution parameters such as the LSH parameters for load balancing and redundancy during execution. In addition, he/she will choose r, the accuracy tuning parameter for sufficient statistics estimates, based on how much network bandwidth should be consumed during gossip. He/she can select a family $X|Pa_X$ to analyze in real-time during the execution of DiSC along with the score (e.g., BDeu, K2) to be estimated. Five datasets will be available: HIGGS [2] with 176 million data instances, a dataset based on 200 million Twitter tweets, and three synthetic datasets generated using binomial distribution with 200 million data instances. (For each dataset, 10,000 families will be used for score computation.) The user will experience how DiSC is robust to different underlying data distributions.

• *Scenario 2:* The second scenario enables the user to experience in real-time how DiSC estimates the sufficient statistics of families with high accuracy. He/she will observe in real-time the convergence speed of DiSC and how the sufficient statistics of the selected family improves in accuracy over

[3] https://code.google.com/archive/p/java-gossip

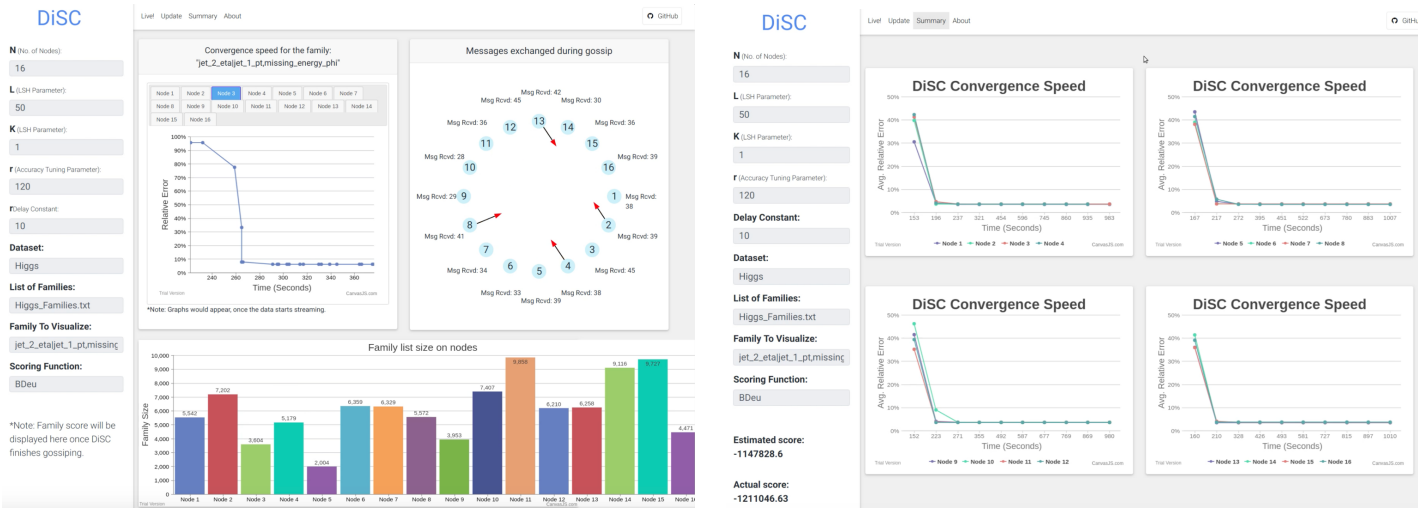


Fig. 3. Two screenshots of DiSC

time on a node that is responsible for the family. The same family can be tracked and visualized on other nodes as well. In addition, the exchange of messages between CloudLab nodes during gossiping can be visualized along with the number of messages received by each node in real-time. Another interesting aspect is how the family list size on each node changes during execution due to the probabilistic dropping approach in DiSC. This information is displayed at the bottom. This real-time experience will unravel the sophistication and merit of decentralized score computation in DiSC.

• *Scenario 3:* The third scenario enables the user to upload new data records to an existing dataset and recompute the family scores efficiently without processing the entire dataset again. He/she will upload a file containing new data records and observe how these records are quickly consumed by DiSC to update the SSA of all the families. The time taken to update the sufficient statistics and the number of messages exchanged during gossiping will be reported.

• *Scenario 4:* The final scenario presents a summary of DiSC's overall performance and provides approximate family scores. This includes the convergence speed of DiSC by plotting the average relative error (%) of the estimated sufficient statistics of the families that a node is responsible for over time, the average family list size during the execution of DiSC, total bandwidth consumption, packet loss rate, benefit of compression, etc., will be displayed. The scores of all the 10,000 input families will be computed. The user will be shown the approximate score of the selected family on the UI. A file containing approximate scores of the input families can be downloaded. These scores can be used for score-based structure learning in tools like R.

The user can go back to *Scenario 1* and change the execution parameters (e.g., no. of nodes, r) to observe the impact on DiSC's performance.

ACKNOWLEDGMENTS

Praveen Rao would like to acknowledge the partial support of NSF Grant No. 1747751.

REFERENCES

[1] CloudLab. https://www.cloudlab.us/, 2017.
[2] UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets.html, 2017.
[3] M. Boehm, M. W. Dusenberry, D. Eriksson, A. V. Evfimievski, F. M. Manshadi, N. Pansare, B. Reinwald, F. R. Reiss, P. Sen, A. C. Surve, and S. Tatikonda. SystemML: Declarative Machine Learning on Spark. *Proc. VLDB Endow.*, 9(13):1425–1436, Sept. 2016.
[4] A. M. Carvalho. Scoring Functions for Learning Bayesian Networks. Technical report, IST, TULisbon/INESC-ID Tech. Report 54/2009, 2009.
[5] N. R. Council. *Frontiers in Massive Data Analysis*. The National Academies Press, Washington, DC, 2013.
[6] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's Highly Available Key-Value Store. In *Proc. of 21st Symp. on Operating Systems Principles*, pages 205–220, 2007.
[7] T. H. Haveliwala, A. Gionis, D. Klein, and P. Indyk. Evaluating Strategies for Similarity Search on the Web. In *Proc. of the 11th WWW Conference*, pages 432–442, 2002.
[8] A. Katib, P. Rao, K. Barnard, and C. Kamhoua. Fast Approximate Score Computation on Large-Scale Distributed Data for Learning Multinomial Bayesian Networks. *ACM Transactions on Knowledge Discovery from Data*, 13(2):1–40, 2019. https://doi.org/10.1145/3301304.
[9] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.
[10] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling Distributed Machine Learning with the Parameter Server. In *Proc. of the 11th OSDI Conference*, pages 583–598, Oct. 2014.
[11] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Distributed GraphLab: A framework for machine learning in the cloud. In *Proc. of PVLDB Conference*, pages 716–727, 2012.
[12] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. MLlib: Machine Learning in Apache Spark. *Jour. of Machine Learning Research*, 17(34):1–7, 2016.
[13] D. Mosk-Aoyama and D. Shah. Fast Distributed Algorithms for Computing Separable Functions. *IEEE Transactions on Information Theory*, 54(7):2997–3007, 2008.
[14] J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.
[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. of the 2001 ACM SIGCOMM Conference*, pages 149–160, San Diego, CA, Aug. 2001.
[16] E. P. Xing, Q. Ho, W. Dai, J.-K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu. Petuum: A New Platform for Distributed Machine Learning on Big Data. In *Proc. of the 21th ACM SIGKDD Conference*, pages 1335–1344, Sydney, Australia, 2015.