Scalable Score Computation for Learning Multinomial Bayesian Networks over Distributed Data[*]

Praveen Rao[†], Anas Katib[†], Kobus Barnard[⋆], Charles Kamhoua[‡], Kevin Kwiat[‡], Laurent Njilla[‡]
[†] Department of Computer Science & Electrical Engineering, University of Missouri-Kansas City
[⋆] Department of Computer Science, University of Arizona
[‡] Air Force Research Lab, Cyber Assurance Branch
raopr@umkc.edu, anaskatib@mail.umkc.edu, kobus@cs.arizona.edu, {charles.kamhoua.1,kevin.kwiat,laurent.njilla}@us.af.mil

Abstract

In this paper, we focus on the problem of learning a Bayesian network over distributed data stored in a commodity cluster. Specifically, we address the challenge of computing the scoring function over distributed data in a scalable manner, which is a fundamental task during learning. We propose a novel approach designed to achieve: (a) scalable score computation using the principle of gossiping; (b) lower resource consumption via a probabilistic approach for maintaining scores using the properties of a Markov chain; and (c) effective distribution of tasks during score computation (on large datasets) by synergistically combining well-known hashing techniques. Through theoretical analysis, we show that our approach is superior to a MapReduce-style computation in terms of communication bandwidth. Further, it is superior to the batch-style processing of MapReduce for recomputing scores when new data are available.

1 Introduction

Today, there is tremendous interest in designing new methodologies for gaining insights over big data to enable timely and effective decision making. It is reported that big data analytics can strengthen national security and provide resilience to cyber attacks.[1] While statistical models provide an elegant framework to gain knowledge from data, the volume and variety of big data demands a paradigm shift–datasets are heterogeneous, massive, and distributed in nature. Massive datasets are being stored and processed in large-scale commodity clusters, and several new frameworks have emerged for scalable machine learning (Low et al. 2012; Li et al. 2014; MLlib 2015).

Among the different statistical models, Bayesian networks (BNs) provide a natural way for knowledge representation and reasoning over heterogeneous data under uncertainty (Pearl 2000). BNs have been successfully used in many areas including medical/fault diagnosis, bioinformatics and computational biology, and others. They play a key role in automated reasoning systems and in data clustering (Grossman and Domingos 2004; Heller and Ghahramani 2005). To learn a BN from the data, we need to learn its structure and the parameters of the conditional probability distributions that best fit the observed data. As exact structure learning of BNs is NP-complete (Chickering 1996), approximate structure learning techniques have been developed over the years. We are particularly interested in score-based learning algorithms, which use heuristic search for approximate structure learning, wherein a search space of possible structures is searched by applying a scoring function. However, for efficient structure learning on large-scale distributed data, *it is essential to first compute the scoring function on the data in a scalable manner*, which is the focus of this work.

We propose a novel approach called DiSC (**Di**stributed **S**core **C**omputation) designed to achieve: (a) scalable score computation using the principle of gossiping; (b) lower resource consumption via a probabilistic approach for maintaining scores using the properties of a Markov chain; and (c) effective distribution of tasks during score computation (on large datasets) by synergistically combining consistent hashing and locality sensitive hashing (LSH). Through theoretical analysis, we show that DiSC is superior to MapReduce-style score computation in terms of communication bandwidth. We also show that DiSC is superior to the batch-style processing of MapReduce for recomputing scores when new data are available.

2 Background & Motivation

2.1 Score-Based Learning of BNs

A lot of advances have been made over the last few decades in score-based learning algorithms (Koller and Friedman 2009). At each step in the search, the algorithm 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 computing a score difference of the affected variables. Different search strategies (*e.g.*, greedy hill-climbing, simulated annealing) can be used, and when the network score does not improve further, the algorithm terminates. If the structure is known, parameter estimation

[*]Approved for Public Release; Distribution Unlimited: 88ABW-2016-5881, dated 18 Nov 2016. The first author would like to acknowledge the support of the U.S. Air Force Summer Faculty Fellowship Program, National Research Council Research Associateship Program, and University of Missouri Research Board.

[1]www.whitehouse.gov/sites/default/files/docs/big_data_privacy_report_5.1.14_final_print.pdf

is done by computing sufficient statistics over the data in one pass (*e.g.*, parameters of a Dirichlet distribution for a multinomial random variable).

Computing the scoring function is a fundamental task during approximate structure learning. Let d denote the data instances/records. Suppose X_i denotes a multinomial random variable and $Val(X_i)$ denotes the set of possible values of X_i. Let $x_i^j \in Val(X_i)$ denote a possible value of X_i. Let $Pa_{X_i}^G$ denote the parents of X_i in a DAG G. Note that $X_i|Pa_{X_i}^G$ is also called a family. Suppose $Val(Pa_{X_i}^G)$ denotes all possible configurations of $Pa_{X_i}^G$ (*i.e.*, assignment of values to the parents). Let $u_i \in Val(Pa_{X_i}^G)$ denote a particular configuration of X_i's parents. For each configuration, let $M[u_i] = \sum_{x_i^j \in Val(X_i)} M[x_i^j, u_i]$, and let $\alpha_{X_i|u_i} = \sum_{x_i^j \in Val(X_i)} \alpha_{x_i^j|u_i}$ denote the prior parameters of the Dirichlet distribution. The tuple containing all $M[x_i^j, u_i]$ is referred to as the sufficient statistics (*i.e.*, the number of data instances where $X_i = x_i^j$ with parent configuration u_i). Assuming the Bayesian Dirichlet equivalence (BDe) scoring function (Koller and Friedman 2009),

$$score(X_i|Pa_{X_i}, d) = \prod_{u_i \in Val(Pa_{X_i}^G)} \frac{\Gamma(\alpha_{X_i|u_i})}{\Gamma(\alpha_{X_i|u_i} + M[u_i])} \times \left(\prod_{x_i^j \in Val(X_i)} \frac{\Gamma(\alpha_{x_i^j|u_i} + M[x_i^j, u_i])}{\Gamma(\alpha_{x_i^j|u_i})} \right). \quad (1)$$

The total score of a DAG G for $X_1, \ldots, X_n$ on d is the product of the family scores, *i.e.*, $score(G, d) = \prod_{i=1}^n score(X_i|Pa_{X_i}, d)$. (We can compute the logarithm of the total score to replace products by sums.) During learning, we only need to compute the change in the score due to the DAG operations. When data instances are distributed, it is a serious challenge to compute the required sufficient statistics for the family scores–the focus of our work.

2.2 Learning a BN in a Commodity Cluster

Recently, parallel methods for scalable BN learning and reasoning using MapReduce were proposed for a shared-nothing cluster (Basak et al. 2012; Chen et al. 2013; Fang et al. 2013; Zhao, Xu, and Gao 2013; SMILE-WIDE 2014). One may wonder if we can simply develop a parallel algorithm to compute the family scores using the map and reduce operations in Apache Spark (Zaharia et al. 2010). This can be done by identifying all possible families that may be needed during structure learning and partial counts on individual data blocks (in the map phase) and computing the required sufficient statistics for each family (in the reduce phase). However, as shown later in Section 3.4, the communication cost of this algorithm grows linearly with the number of cluster nodes. Furthermore, the batch-oriented nature of MapReduce requires complete re-execution when new data instances are added.

2.3 Gossip Algorithms

Gossip algorithms were used by companies like Amazon and Facebook to build global-scale computing platforms (DeCandia et al. 2007; Lakshman and Malik 2009). They are attractive in large-scale distributed systems due to their simplicity, decentralized nature, high scalability, ability to tolerate failures, and ability to provide probabilistic guarantees. The essence of these algorithms lies in the exchange of information or aggregates between a pair of nodes, using a probability transition matrix for the given network topology. It has been shown that after a provably finite number of rounds/time intervals and a provably finite number of message exchanges, the information has reached all the nodes or the aggregates have converged to the true value (Kempe, Dobra, and Gehrke 2003; Boyd et al. 2005).

In this work, we draw inspiration from a state-of-the-art gossip algorithm proposed by Mosk-Aoyama *et al.* (Mosk-Aoyama and Shah 2008) to compute the sum of values stored on n nodes. We call this algorithm `SUM`. Let $P = [P_{ij}]$ denote a (doubly stochastic) probability transition matrix, where P_{ij} is the probability that node i contacts node j during gossip. Each node has a local clock that ticks at the times of rate 1 Poisson process. Let x_i denote the value at node i. Each node i maintains r independent exponential random variables with rate x_i, say E_{li} where $l = 1$ to r. A node becomes active when its local clock ticks, selects a neighbor with probability P_{ij}, and then they exchange their current *state*. It computes for $l = 1$ to r, $m_l = \min_{i=1}^n E_{li}$. Finally, it uses $\frac{r}{\sum_{l=1}^r m_l}$ as the estimate of $\sum_{i=1}^n x_i$. Suppose $T_{SUM}(\epsilon, \delta, P)$ is the smallest time at which each node has computed the sum such that it is within δ of the true average with probability at least $1 - \epsilon$. (This is called the convergence speed.) Then $T_{SUM}(\epsilon, \delta, P) = O\left(\frac{log\ n + log\ \epsilon^{-1} + log\ \delta^{-1}}{\delta^2 \Phi(P)}\right)$, where $\Phi(P)$ denotes the conductance of the communication topology.

2.4 Challenges and Motivation

There are several technical challenges that must be addressed to develop a scalable score computation approach over large-scale distributed data. First, data blocks are distributed across nodes in a cluster. Therefore, it is pragmatic to move computations to data (Dean and Ghemawat 2004). Second, the score computation should be efficient, scalable, tolerate failures and changes to the cluster topology, and provide provable guarantees on the quality of the estimated scores. This requires fast aggregate computation (*e.g.*, sum) over distributed data, effective load balancing of tasks, and redundancy to cope with failures. While a straightforward application of `SUM` sounds promising, unfortunately, it does not yield a scalable solution for score computation of families. (We provide more details in Section 3.2.) Therefore, we must design a new algorithm by adapting `SUM`. Third, when new data are produced, efficient recomputation of family scores over a large dataset is necessary for faster relearning than a batch-style approach.

3 Our Approach

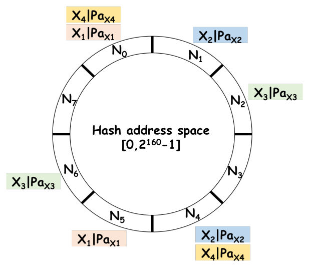
In this section, we present DiSC and explain the key ideas that underpin its design. We also present the theoretical analysis of DiSC w.r.t. its performance and scalability. DiSC addresses two key issues during score computation: (a) distribution of families across cluster nodes for load balancing, and (b) scalable score computation of families in a fault-tolerant manner. DiSC can be viewed as a black box (by different score-based BN learning algorithms) to provide an estimate of a family score over large-scale distributed data. DiSC can also be used to learn the parameters of a BN when the structure is already known. (See Table 1 for frequently used notations in the remainder of the paper.)

Notation	Description
$f = X\|Pa_X$	A family f where X is a random variable and Pa_X is the set of parents
N_i	A node in the cluster
$[s_{N_i}, e_{N_i}]$	The interval assigned to N_i in consistent hashing address space
$\mathbb{F}_{N_i}$	The family list of the cluster node N_i
$\mathbb{L}$	The hash function that combines LSH and consistent hashing
k	Number of hash values output by $\mathbb{L}$
h_f^j	The j^{th} hash value output by $\mathbb{L}$
Φ	Conductance of a network of cluster nodes
$\mathbf{O}, \mathbf{P}^f, \mathbf{Q}^f$	Doubly stochastic transition matrices
$\boldsymbol{\pi}_f = [\pi_f^1 \dots \pi_f^n]$	A row matrix denoting the stationary distribution of a Markov chain with n states for family f
$\mathbb{D}$	Number of distinct families in the network
SSA_f	Sufficient statistics array of the family f
$1-\epsilon$	Desired confidence of an estimate via gossip
$1-\delta$	Desired accuracy of an estimate via gossip
T_{SUM}	Convergence speed of SUM
T_{DiSC}	Convergence speed of DiSC

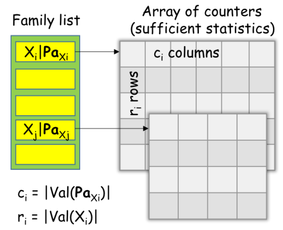
Table 1: Table of notations

3.1 Distribution of Families

Given a cluster with n nodes, we assume they are connected by an overlay network, where any two nodes can communicate with each other in a finite number of hops (*e.g.*, using a Distributed Hash Table (DHT) (Stoica et al. 2001)). The decomposability property of the Bayesian scoring function (*e.g.*, Equation 1) enables us to achieve distributed score computation. There are two issues that arise. First, we must distribute the task of computing the scores of families across the cluster nodes in a scalable, load-balanced, and fault-tolerant manner. This implies that when the learning algorithm is running on a cluster node, the score of a family may not be available locally and requires communication with another cluster node. Thus, the second issue is to allow a cluster node to manage similar families so that we can minimize the number of network lookups during BN learning.



(a) Initial assignment



(b) Family list

Figure 1: Assignment of families to cluster nodes

We address the above issues by synergistically combining consistent hashing (Stoica et al. 2001) and LSH (Indyk and Motwani 1998). In consistent hashing, only a finite fraction of the keys need to be redistributed when there is a change in the size of the hash table (or cluster) allowing DHTs to scale. Using LSH, data items that are more similar are more likely to produce collisions. We can design LSH for sets using $k \times l$ random linear hash functions as follows (Haveliwala et al. 2002): For each linear hash function, apply it on each item in a set and compute the minimum of the hash values. Create k groups each with l minimum hash values; concatenate l minimum values in each group and apply another hash function (*e.g.*, SHA-1) to produces a value in the integer range $[0, m]$. Finally, produce a total of k values for a set. Let $\{h_{S_1}^1, \dots, h_{S_1}^k\}$ and $\{h_{S_2}^1, \dots, h_{S_2}^k\}$ denote the outputs of LSH on sets S_1 and S_2, respectively. It is known that if the similarity (*i.e.*, Jaccard index) between S_1 and S_2 is p, the probability that there exists at least one pair of identical hash values is $1-(1-p^l)^k$, *i.e.*, $h_{S_1}^i = h_{S_2}^i$ $(1 \le i \le k)$.

Like in a DHT, let N_0,..., N_{n-1} denote the n cluster nodes mapped to a 160-bit hash address space. We partition the address space $[0, 2^{160}-1]$ equally among the cluster nodes. Let $[s_{N_i}, e_{N_i}]$ denote the interval assigned to N_i. (A similar way of assigning ranges is employed by Cassandra (Lakshman and Malik 2009) and Dynamo (DeCandia et al. 2007).) Let $\mathbb{L}$ denote LSH on a set that produces k hash values in the range $[0, 2^{160}-1]$ using SHA-1. Given a family $f = X|Pa_X$, we first represent it as a set of random variables $\{X\} \cup Pa_X$. Let $\{h_f^1, \dots, h_f^k\}$ denote the k hash values output by $\mathbb{L}(\{X\} \cup Pa_X)$. We assign f to every cluster node whose assigned interval contains any h_f^j, where $1 \le j \le k$. Essentially, through consistent hashing, we distribute the families almost evenly across nodes in a cluster. Through LSH, we can ensure that two similar sets/families are assigned to the same node with high probability. This will be useful to a score-based learning algorithm when retrieving the scores of similar families. Due to k values output by LSH, multiple cluster nodes will be assigned a family and are responsible for computing the score of that family. Thus, DiSC can cope with node failures for high availability.

Example 1 *An example of assignment of families is shown in Figure 1(a). Cluster nodes* $N_0, \dots, N_7$ *are assigned intervals in the hash address space. Suppose there are four families* $f_1 = X_1|Pa_{X_1}$, $f_2 = X_2|Pa_{X_2}$, $f_3 = X_3|Pa_{X_3}$, *and $f_4 = X_4|Pa_{X_4}$. Let $\mathbb{L}$ produce $k = 2$ hash values. Therefore, each family is assigned to two nodes in the cluster. Suppose the set representations of $\{X_1\} \cup Pa_{X_1}$ and $\{X_4\} \cup Pa_{X_4}$ have high similarity. As shown in the figure, N_0 is assigned both f_1 and f_4 due to the property of LSH.*

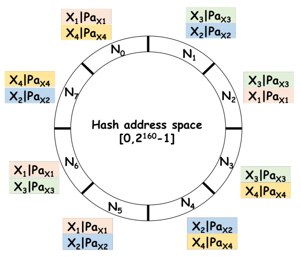
Each node N_l stores the families assigned to it in its family list $\mathbb{F}_{N_l}$. For each family f, an array of $r \times c$ counters is maintained, where $r = |Val(X)|$ and $c = |Val(Pa_X)|$. We call this the sufficient statistics array (SSA) of f denoted by SSA_f. Algorithm 1 shows the overall steps. Figure 1(b) shows an example of a family list.

Algorithm 1 AssignFamily(f)

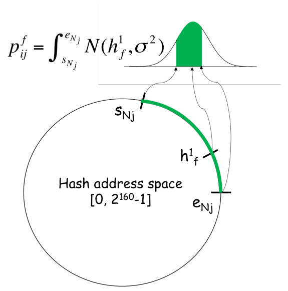
```
1: Let $f = X|Pa_X$
2: $\{h_f^1, \ldots, h_f^k\} \leftarrow \mathbb{L}(\{X\} \cup Pa_X)$
3: for j=1 to $k$ do
4:     Route $f$ to a cluster node $N_l$ such that $h_f^j \in [s_{N_l}, e_{N_l}]$
5:     Add $f$ to the family list $\mathbb{F}_{N_l}$ of $N_l$ and initialize the
       counters in $SSA_f$ to 0
```

3.2 Gossip-based Score Computation

The next challenge is to compute the scores of families in a scalable manner on large distributed data. We need to compute the sufficient statistics of each family. By straightforward application of SUM (Section 2.3), the counters can be updated to obtain the desired sufficient statistics. Unfortunately, a major drawback of this approach is that each node will learn about more families each time it gossips and eventually track the sufficient statistics of all the families known to the cluster nodes. This will defeat the purpose of gossiping because of potentially very large number of unique families (*e.g.*, when a dataset has large number of variables) to consider during learning. As a result, each node may run out of main memory due to a very large family list and consume high network bandwidth during each message exchange. (Similar rationale was used in XGossip[2] albeit for a different problem and gossip algorithm.)



(a) A good assignment



(b) The function P_{ij}^f

Figure 2: DiSC

To overcome the above limitations, we develop a gossip algorithm (inspired by SUM) to scalably compute the sufficient statistics of families. Our algorithm employs a probabilistic approach for guaranteeing a bound on the number of families managed by each node. As shown in Figure 2(a), we would like each node to manage only a finite fraction of the families. This is achieved using a Markov chain and its attractive properties. A Markov chain is modeled by t states, $s_1, \ldots, s_t$, where the probability of transitioning from one state to another is given by a transition matrix $\mathbf{T}$. The stationary distribution of the Markov chain is denoted by a row matrix $\boldsymbol{\pi} = [\pi^1 \ldots \pi^t]$ s.t. $\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{T}$. Over a long run, the probability of being at a particular state s_i converges to the stationary distribution π_i independent of the starting state.

We model the n cluster nodes by a Markov chain with n states. We define a few transition matrices. We define a doubly stochastic transition matrix $\mathbf{O}$ where $O_{ij} = \frac{1}{n}$. For each family f, we define another doubly stochastic transition matrix $\mathbf{P}^f$ s.t. $P_{ij}^f = \int_{s_{N_j}}^{e_{N_j}} N(h_f^1, \sigma^2)$ for $i \neq j$. That is, for f, we define a normal distribution with mean h_f^1 and standard deviation σ. An illustration of P_{ij}^f is shown in Figure 2(b). Finally, we define a doubly stochastic transition matrix $\mathbf{Q}^f$ s.t. $Q_{ij}^f = O_{ij} \times P_{ij}^f$ for $i \neq j$.

The steps involved during gossiping are listed in Algorithms 2 and 3. Algorithm 2 shows the actions performed by every cluster node. Consider node N_i. When its local clock ticks, it becomes active during gossiping and does the following for each family $f \in \mathbb{F}_{N_i}$: Pick a neighbor N_j with probability O_{ij}. Exchange the state with N_j for updating sufficient statistics of f. Compute P_{ij}^f using h_f^1 as the mean and a preselected σ. With probability P_{ij}^f, do the following: Inform N_j to add f to its family list, and if N_j is not responsible for f, remove f from $\mathbb{F}_{N_i}$. This key idea of probabilistically removing a family from the family list of a cluster node, prevents the list from growing very large. (See Section 3.4 for a bound on the size of the family list.)

Algorithm 3 lists the steps performed by every cluster node when it is receives messages from other nodes during gossiping. If the family under consideration is not in the family list of the receiving node, then it initializes the SSA of the family (using any local data blocks). The node uses the sender's state and updates the counters with new estimates. It adds the family to its family list if instructed.

3.3 Retrieving Scores During Learning

DiSC can be viewed as a black box by (a serial or parallel version of) a score-based learning algorithm, wherein it has precomputed the sufficient statistics of families required during learning. When the learning algorithm executes on a cluster node and needs the sufficient statistics of a family, the node's family list is checked. If the family is present, then its SSA is fetched without any network communication. Otherwise, a cluster node storing the family should be contacted (by applying $\mathbb{L}$ on the family) to fetch the SSA. Because of LSH, it is more likely for the learning algorithm to retrieve the SSAs of similar families from the same node, thereby reducing the network latency during learning.

[2] https://github.com/UMKC-BigDataLab/XGossip

Algorithm 2 DiSC-Gossip()

```
1: Let $N_i$ denote the cluster node executing this procedure
2: Let $\mathbf{O}$ denote the doubly stochastic transition matrix of a Markov chain representing the $n$ cluster nodes, s.t. $O_{ij} = \frac{1}{n}$
3: Initialize rate 1 Poisson process at node $N_i$ for gossiping
4: for each local clock tick do
5:    Pick a neighbor $N_j$ with probability $\mathbf{O}_{ij}$
6:    for each family $f \in \mathbb{F}_{N_i}$ do
7:       Send state to $N_j$ and receive state from $N_j$ to update the sufficient statistics {We compute the minimum of exponential random variables like SUM}
8:       Let $\mathbf{P}^f$ denote a doubly stochastic transition matrix for $f$, where $P_{ij}^f = \int_{s_{N_j}}^{e_{N_j}} N(h_f^1, \sigma^2)$ for $i \neq j$
9:       Compute $P_{ij}$ given $h_f^1$ and $\sigma$
10:      if $\exists j, 1 \leq j \leq k$, s.t. $h_f^j \in [s_{N_i}, e_{N_i}]$ then
11:         With prob. $P_{ij}^f$, send message to $N_j$ to store $f$
12:      else
13:         With prob. $P_{ij}^f$, remove $f$ from $\mathbb{F}_{N_i}$ and send message to $N_j$ to store $f$
```

Algorithm 3 DiSC-Recv()

```
1: Let $N_j$ denote the cluster node executing this procedure
2: while new message is received do
3:    if the message contains state of family $f$ then
4:       if $f \notin \mathbb{F}_{N_j}$ then
5:          Initialize the sufficient statistics of $f$ using the local data blocks if any
6:       Update the sufficient statistics for $f$ using the sender's state {We compute the minimum of exponential random variables like SUM}
7:       Send local state of $f$ to the sender
8:    else if the message indicates adding $f$ to $\mathbb{F}_{N_j}$ and $f \notin \mathbb{F}_{N_j}$ then
9:       Add $f$ and its sufficient statistics to $\mathbb{F}_{N_j}$
```

3.4 Theoretical Analysis of DiSC

We present the theoretical analysis of DiSC by considering the following metrics: (a) accuracy and confidence of the estimated sufficient statistics of a family, (b) convergence speed of the gossip algorithm, and (c) memory and network bandwidth requirement during gossip. We state a theorem on the convergence speed of DiSC to estimate the sufficient statistics of a family.

Theorem 1 *Suppose node N_i is responsible for computing the score of a family f. Let $T_{DiSC}(f, \epsilon, \delta)$ denote the smallest time at which N_i can estimate the sufficient statistics for f within a relative error of ϵ with a probability of at least $1 - \delta$. Then $T_{SUM}(\epsilon, \delta, \mathbf{O}) \leq T_{DiSC}(f, \epsilon, \delta) \leq T_{SUM}(\epsilon, \delta, \mathbf{Q}^f)$.*

Proof. The dissemination speed of a gossip algorithm to compute SSA_f will depend on how fast the state of the nodes are exchanged through the network. Suppose we use SUM with the probability transition matrix $\mathbf{O}$ to estimate SSA_f. Then the convergence speed is $T_{sum}(\epsilon, \delta, \mathbf{O})$. In DiSC, we exchange the node states with probability O_{ij} (Line 7 in Algorithm 2). But we exchange a family only with probability $Q_{ij}^f = O_{ij} \times P_{ij}^f$ (Line 10 in Algorithm 2). (Note that $Q_{ij}^f \leq O_{ij}$ for $i \neq j$.) Therefore, the dissemination speed of DiSC cannot be faster than SUM with transition matrix $\mathbf{O}$. Therefore, $T_{SUM}(\epsilon, \delta, \mathbf{O}) \leq T_{DiSC}(f, \epsilon, \delta)$. However, DiSC is at least as fast as SUM with transition matrix $\mathbf{Q}^f$, because the node states are exchanged each time a node i contacts j with probability O_{ij}. Therefore, $T_{DiSC}(f, \epsilon, \delta) \leq T_{SUM}(\epsilon, \delta, \mathbf{Q}^f)$. □

The next theorem states the expected value of the number of families tracked by each node. This key property enables DiSC to scale with increasing number of families to consider when learning a BN.

Theorem 2 *For a family f, let $\boldsymbol{\pi}_f = [\pi_f^1 \ldots \pi_f^n]$ denote the stationary distribution of the Markov chain with the transition matrix $\mathbf{Q}^f$ containing n states. Let $\mathbb{D}$ denote the number of distinct families and k denote the number of hash values output by $\mathbb{L}$. Then $E(|\mathbb{F}_{N_i}|) = \sum_{f \in \mathbb{D}} \pi_f^i + \frac{k\mathbb{D}}{n}$.*

Proof. Let us define a binary random variable Y_f to indicate the presence or absence of f in $\mathbb{F}_{N_i}$. Suppose $Y_f = 1$ when $f \in \mathbb{F}_{N_i}$ and $Y_f = 0$ otherwise. Let U denote a random variable that denotes the number of families N_i is responsible for via $\mathbb{L}$. We define a random variable $Z = \sum_{f \in \mathbb{D}} Y_f$ + U, an unbiased estimator of $|\mathbb{F}_{N_i}|$. Consistent hashing in $\mathbb{L}$ ensures that the families are distributed evenly across the nodes with high probability. Furthermore, $\mathbb{L}$ produces k hash values per family. Thus, over a long run (*i.e.*, clock ticks), $E(Z) = \sum_{f \in \mathbb{D}} E(Y_f) + E(U) = \sum_{f \in \mathbb{D}} \pi_f^i + \frac{k\mathbb{D}}{n}$. □

The intuition for the above theorem is that the probability of a family being stored in the family list of a node will converge to the stationary distribution of the underlying Markov chain. In addition, a node is also responsible for storing a fraction of all the distinct families due to $\mathbb{L}$.

Next, we discuss the memory and network bandwidth requirement. The SSA of each family $X_i|Pa_{X_i}$ contains $r_i \times c_i$ counters, where $r_i = |Val(X_i)|$ and $c_i = |Val(Pa_{X_i})|$. Over a long run, the expected number of families stored by a node is given by Theorem 2. According to Theorem 1, the number of clock ticks required by DiSC for convergence of the sufficient statistics of a family is given by $T_{DiSC}(f, \epsilon, \delta)$. Suppose each node maintains r exponential random variables per counter in a family's SSA. During each clock tick, for a family $X_i|Pa_{X_i}$, two nodes exchange $r \times r_i \times c_i$ exponential random variables to compute their minimum.

3.5 DiSC vs MapReduce

Suppose we develop a MapReduce program for computing the scores of $\mathbb{D}$ distinct families. In the map phase, the partial counts for each family $f \in \mathbb{D}$ on each block of data are computed. During the reduce phase, the sufficient statistics across all the data blocks for each family is obtained. On a cluster of n nodes, the map phase will produce intermediate key-value data of size proportional to $n \times \sum_{f \in \mathbb{D}} (r_f \times c_f)$ words, assuming maximum parallelism. During the reduce phase, the intermediate key-value data must be moved to the reducers through the network. Hence, the communication cost is $O(n\mathbb{D}\mathbb{S})$, where $\mathbb{S}$ is the size of the largest SSA in $\mathbb{D}$. In DiSC, the number of time steps (involving communication) for estimating the sufficient statistics of a family is $O(log(n))$, given a user-specified accuracy, LSH parameters, communication topology, and other user-defined parameters. For $\mathbb{D}$ families, the total communication cost is $O(log(n)\mathbb{D}\mathbb{S})$. Hence, DiSC is superior to MapReduce in terms of communication bandwidth.

3.6 Computing Family Scores on New Data

Because gossiping can be done continuously in the background, DiSC can efficiently compute the family scores as new data are produced. As each node computes the minimum of the exponential random variables (like in SUM), we make the following observations: At time t_0, let X_i^0 denote the exponential random variable with rate x_i^0 maintained by a node for a family. At time t_1, let the sufficient statistics of the family increase to x_i^1 due to new data instances. Let X_i^1 denote the new exponential random variable with rate x_i^1. It is known that $\Pr(X_i^0 < X_i^1) = \frac{x_i^0}{x_i^0 + x_i^1}$. Therefore, with probability $\frac{x_i^1}{x_i^0 + x_i^1}$, the other cluster nodes learn about x_i^1 by updating their minimum as they continue to gossip. As such, only the nodes receiving new data instances must reinitialize the exponential random variables for the affected sufficient statistics. On the other hand, the batch-style processing of MapReduce must process the entire dataset (with new data) to obtain the new scores of families. As a result, DiSC is superior to MapReduce for fast score recomputation.

4 Conclusions

In this paper, we presented DiSC, a novel approach for scalable score computation during learning of a multinomial BN over big data stored in a cluster. DiSC is based on the principle of gossipping, properties of Markov chains, and leverages well-known hashing techniques. Through theoretical analysis, we showed that DiSC is superior to a MapReduce-style computation in terms of communication bandwidth. In addition, DiSC is superior to batch-oriented MapReduce for recomputation of scores when new data are available.

References

Basak, A.; Brinster, I.; Ma, X.; and Mengshoel, O. 2012. Accelerating Bayesian Network Parameter Learning using Hadoop and MapReduce. In *Proc. of 2012 BigMine Workshop*, 1–8.

Boyd, S. P.; Ghosh, A.; Prabhakar, B.; and Shah, D. 2005. Gossip Algorithms: Design, Analysis and Applications. In *Proc. of INFOCOM 2005*, 1653–1664.

Chen, W.; Wang, T.; Yang, D.; Lei, K.; and Liu, Y. 2013. Massively Parallel Learning of Bayesian Networks with MapReduce for Factor Relationship Analysis. In *Proc. of Intl. Joint Conf. on Neural Networks*, 1–5.

Chickering, D. 1996. Learning from Data: Artificial Intelligence and Statistics V. chapter Learning Bayesian Networks is NP-Complete, 121–130.

Dean, J., and Ghemawat, S. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. of the 6th OSDI Conference*, 137–150.

DeCandia, G.; Hastorun, D.; Jampani, M.; Kakulapati, G.; Lakshman, A.; Pilchin, A.; Sivasubramanian, S.; Vosshall, P.; and Vogels, W. 2007. Dynamo: Amazon's Highly Available Key-Value Store. In *Proc. of 21st Symp. on Operating Systems Principles*, 205–220.

Fang, Q.; Yue, K.; Fu, X.; Wu, H.; and Liu, W. 2013. A MapReduce-based Method for Learning Bayesian Network from Massive Data. In *Proc. of 2013 APWeb Conference*, 697–708.

Grossman, D., and Domingos, P. 2004. Learning Bayesian Network Classifiers by Maximizing Conditional Likelihood. In *Proc. of the 21st International Conference on Machine Learning*, 46–54.

Haveliwala, T. H.; Gionis, A.; Klein, D.; and Indyk, P. 2002. Evaluating Strategies for Similarity Search on the Web. In *Proc. of the 11th WWW Conference*, 432–442.

Heller, K. A., and Ghahramani, Z. 2005. Bayesian Hierarchical Clustering. In *Proc. of the 22nd International Conference on Machine Learning*, 297–304.

Indyk, P., and Motwani, R. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the 13th ACM Symposium on Theory of Computing*, 604–613.

Kempe, D.; Dobra, A.; and Gehrke, J. 2003. Gossip-Based Computation of Aggregate Information. In *Proc. of the 44th IEEE Symposium on Foundations of Computer Science*, 482–491.

Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press.

Lakshman, A., and Malik, P. 2009. Cassandra: A Structured Storage System on a P2P network. In *Proc. of the 21st Symposium on Parallelism in Algorithms and Architectures*, 47.

Li, M.; Andersen, D. G.; Park, J. W.; Smola, A. J.; Ahmed, A.; Josifovski, V.; Long, J.; Shekita, E. J.; and Su, B.-Y. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *Proc. of the 11th OSDI Conference*, 583–598.

Low, Y.; Gonzalez, J.; Kyrola, A.; Bickson, D.; Guestrin, C.; and Hellerstein, J. M. 2012. Distributed GraphLab: A framework for machine learning in the cloud. In *Proc. of PVLDB Conference*, 716–727.

MLlib. 2015. http://spark.apache.org/mllib.

Mosk-Aoyama, D., and Shah, D. 2008. Fast Distributed Algorithms for Computing Separable Functions. *IEEE Transactions on Information Theory* 54(7):2997–3007.

Pearl, J. 2000. *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

SMILE-WIDE. 2014. http://smilewide.github.io/main.

Stoica, I.; Morris, R.; Karger, D.; Kaashoek, M. F.; and Balakrishnan, H. 2001. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. of the 2001 ACM-SIGCOMM Conference*, 149–160.

Zaharia, M.; Chowdhury, M.; Franklin, M. J.; Shenker, S.; and Stoica, I. 2010. Spark: Cluster Computing with Working Sets. In *Proc. of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, 10–10.

Zhao, Y.; Xu, J.; and Gao, Y. 2013. A Parallel Algorithm for Bayesian Network Parameter Learning Based on Factor Graph. In *Proc. of IEEE Intl. Conf. on Tools with Artificial Intelligence*, 506–511.