(Accepted for publication Nov. 9, 2018)

X

Fast Approximate Score Computation on Large-Scale Distributed Data for Learning Multinomial Bayesian Networks

ANAS KATIB, University of Missouri-Kansas City, USA
PRAVEEN RAO, University of Missouri-Kansas City, USA
KOBUS BARNARD, University of Arizona, USA
CHARLES KAMHOUA, Army Research Lab, USA

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 an efficient and scalable manner, which is a fundamental task during learning. While exact score computation can be done using the MapReduce-style computation, our goal is to compute approximate scores much faster with probabilistic error bounds and in a scalable manner. We propose a novel approach which is designed to achieve: (a) decentralized 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. We conduct theoretical analysis of our approach in terms of convergence speed of the statistics required for score computation, and memory and network bandwidth consumption. We also discuss how our approach is capable of efficiently recomputing scores when new data are available. We conducted a comprehensive evaluation of our approach and compared with the MapReduce-style computation using datasets of different characteristics on a 16-node cluster. When the MapReduce-style computation provided exact statistics for score computation, it was nearly 10 times slower than our approach. Although it ran faster on randomly sampled datasets than on the entire datasets, it performed worse than our approach in terms of accuracy. Our approach achieved high accuracy (below 6% average relative error) in estimating the statistics for approximate score computation on all the tested datasets. In conclusion, it provides a feasible tradeoff between computation time and accuracy for fast approximate score computation on large-scale distributed data.

ACM Reference Format:
Anas Katib, Praveen Rao, Kobus Barnard, and Charles Kamhoua. 2018. Fast Approximate Score Computation on Large-Scale Distributed Data for Learning Multinomial Bayesian Networks. *ACM Trans. Knowl. Discov. Data.* X, X, Article X (December 2018), 41 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Today, there is tremendous interest in designing new methodologies for gaining insights over big data to enable timely and effective decision making. Big data technologies will be transformative in many domains and will enable scientific and technological advances in national security, healthcare

Authors' addresses: Anas Katib, University of Missouri-Kansas City, Kansas City, MO, 64110, USA, anaskatib@mail.umkc.edu; Praveen Rao, University of Missouri-Kansas City, Kansas City, MO, 64110, USA, raopr@umkc.edu; Kobus Barnard, University of Arizona, Tucson, AZ, 85719, USA, kobus@cs.arizona.edu; Charles Kamhoua, Army Research Lab, Adelphi, MD, 20783, USA, charles.a.kamhoua.civ@mail.mil.

1556-4681/2018/12-ARTX $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

delivery, science and engineering, retail, education, and others [51]. While statistical models provide an elegant framework to gain knowledge from data [18], 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 using frameworks like Apache Hadoop [62] and Apache Spark [64]. Several new frameworks have emerged for scalable machine learning problems (*e.g.*, GraphLab [42, 43], MLlib [46], Parameter Server [40], Petuum [63], SystemML [12, 25]).

Among the different statistical models, Bayesian networks (BNs) provide a natural way for knowledge representation and reasoning over heterogeneous data under uncertainty [50]. 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 [26, 28]. More recently, researchers are employing BNs for causal discovery of biomedical knowledge from big data [17]. A BN can model causal relationships among features/attributes of the data. It provides a way to assert the conditional independencies between different features of the data, modeled as random variables. It compactly encodes the joint probability distribution of the random variables by a set of conditional probabilities of these variables given their parents in a directed acyclic graph (DAG).

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. Because exact structure learning of BNs is NP-complete [16], 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 and efficient manner*, which is the focus of this work.

To motivate the problem at hand, let us consider tweets posted by users of Twitter. Tweets exemplify massive, heterogeneous, loosely structured data on the Web. Twitter has more than 500 million users, and every day more than 400 million tweets are posted by users. Tweets are publicly available, have 100+ attributes, and attribute values can be missing and noisy. New attributes may appear in tweets, and not all attributes may be present in all of them. Hashtags (*e.g.*, #baseball, #uselection, #fashionpolice) are used frequently by users in tweets to indicate specific topics or categories. There are thousands of hashtags in use today. A Bayesian approach to modeling tweets [41, 61] has several use cases including automatic topic labeling, clustering, identifying causality among tweets, predicting the popularity of tweets/hashtags, detecting latent events, and so on. A BN can be learned on hashtags and other attributes such as users mentioned in a tweet, timezone, geo-location, language, retweet status of a tweet, etc. Probabilistic reasoning queries can also be posed on tweets using BNs for the above use cases.

Example 1.1. Consider a large dataset of tweets. Let us model the tweets using binary random variables. Let $t_1, \ldots, t_n$ denote n hashtags of interest. We define n binary random variables, one for each hashtag. For each tweet, if hashtag t_i is present, then $T_i = 1$, and $T_i = 0$ otherwise. Now we build a BN on $T_1, \ldots, T_n$. Let Figure 1(a) denote the learned structure of the BN. (Only $T_1, \ldots, T_9$ are shown for simplicity.) Each node/variable in the BN has a conditional probability distribution. An example is shown in Figure 1(a) for T_1.

We can perform probabilistic reasoning queries on the BN. Suppose we wish to predict the probability that a tweet has hashtag t_i given that hashtags t_j, t_k, and t_l are present, absent, and present, respectively. We can pose a query $\Pr(T_i = 1 | T_j = 1, T_k = 0, T_l = 1)$ on the BN. Let us extend our model by including a multinomial random variable R for attribute retweeted_count in each tweet.

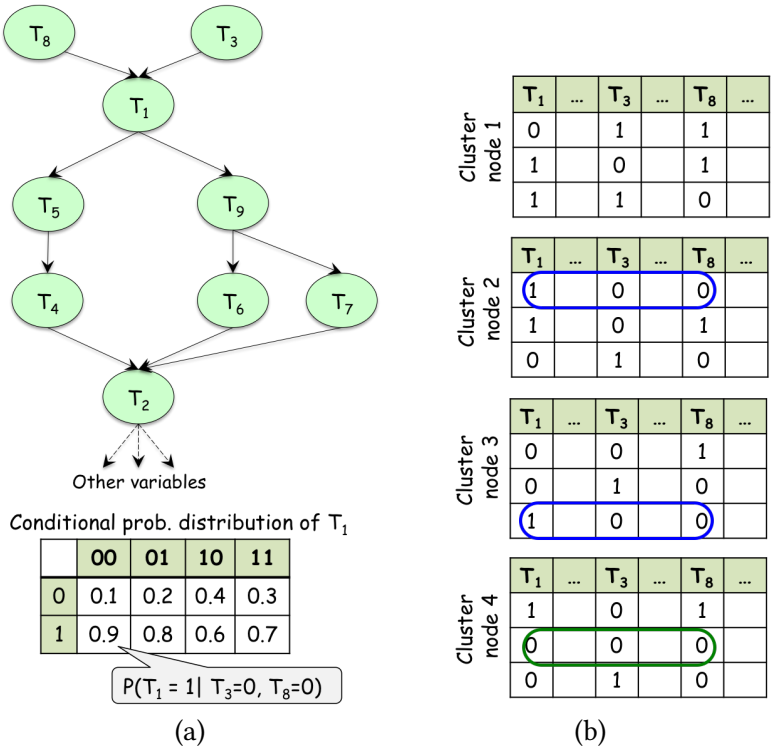


Fig. 1. (a) An example of a BN on variables $T_1, \ldots, T_n$. (b) Data instances for the variables $T_1, \ldots, T_n$ distributed across four cluster nodes.

After learning a new BN on the random variables, suppose we wish to predict the popularity of a tweet based on retweeted_count given the presence, absence, and presence of hashtags t_i, t_j, and t_k, respectively. We can pose a maximum a posteriori query $\mathbf{argmax_r}\ \Pr(R = r|T_i = 1, T_j = 0, T_k = 1)$, whose output can be used to estimate the most likely value of retweeted_count, and hence the tweet's popularity.

We formulate the task of score computation on large-scale distributed data as a *scalable data aggregation problem*. This is because score computation, which will be formally introduced in Section 2, requires computing the frequency counts of different values of a random variable and its parents (a.k.a. sufficient statistics) on the entire dataset. The key contributions of this work are as follows.

- We propose a novel approach called DiSC (**Di**stributed **S**core **C**omputation) for fast approximate score computation over large-scale distributed data. The key features of DiSC are: (a) decentralized 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).
- We conduct theoretical analysis of DiSC in terms of convergence speed (for a given accuracy and confidence bound) of the sufficient statistics required for score computation, and memory and network bandwidth consumption. We also discuss how DiSC is capable of efficiently recomputing scores when new data are available.

- Finally, we conducted a comprehensive performance evaluation of DiSC on a 16-node cluster setup on CloudLab [3]. We compared DiSC with the MapReduce-style computation (implemented using Apache Spark) to compute the frequency counts for score computation. We used datasets with different characteristics containing up to 200M data instances/records each for the evaluation. When the MapReduce-style computation provided exact values of frequency counts needed to compute scores, it was nearly 10 times slower than DiSC. Although it ran faster on randomly sampled datasets than on the entire datasets, it performed worse than DiSC in terms of accuracy. DiSC computed approximate values of frequency counts but achieved high accuracy (below 6% average relative error) in estimating them on all the tested datasets. Thus, DiSC provides a feasible tradeoff between computation time and accuracy for fast approximate score computation on large-scale distributed data.

The rest of this paper is organized as follows: Section 2 provides background and motivation for this work; Section 3 introduces DiSC and discusses a basic approach and an improved approach to lower resource consumption; Section 3.5 presents theoretical analysis of DiSC and comparison with the MapReduce-style computation; Section 4 describes the performance evaluation and comparison between DiSC and the MapReduce-style computation; and finally, we provide our concluding remarks in Section 5.

A preliminary version of this work appeared in the AAAI 2017 Workshop on Distributed Machine Learning [52].

2 BACKGROUND AND MOTIVATION

2.1 Score-Based Learning of BNs

Over the last few decades, several advances have been made in score-based learning algorithms [36]. 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 is done by computing sufficient statistics over the data in one pass (*e.g.*, parameters of a Dirichlet distribution for a multinomial random variable).

We provide a brief discussion on scoring functions and refer the reader to the work by Carvalho [14], who conducted a comprehensive analysis and comparison of scoring functions for learning Bayesian networks. Computing a scoring function is a fundamental task during approximate structure learning. The goal is to find the best Bayesian network that fits the data. Let d denote the data instances/records. Given a scoring function ϕ, one aims to maximize the value of $\phi(G, d)$, where G is a Bayesian network. A scoring function is designed to compute the posterior probability distribution of G conditioned on d, *i.e.*, $P(G|d)$. The best Bayesian network is the one that maximizes the posterior probability. As $P(d)$ is the same for all possible networks, it is sufficient to compute $P(G, d)$. Popular scoring functions are of two types: information-theoretic scoring functions (based on information theory) and Bayesian scoring functions [14]. These scoring functions are decomposable in the sense that they can be computed by first computing the individual score of a variable given its parent. Our work in this paper applies to a broad class of decomposable scoring functions proposed in the literature that require computing the sufficient statistics over the data, which is essentially a set of frequency counts of how many data instances have a particular assignment of values for a variable and its parents in a Bayesian network.

As a motivating example, let us consider the Bayesian Dirichlet equivalence (BDe) scoring function [14, 36]. 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. We will use $M[\cdot]$ to denote the frequency counts computed over the data instances d. For each configuration u_i, let $M[u_i] = \sum_{x_i^j \in Val(X_i)} M[x_i^j, u_i]$, where 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). 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 BDe scoring function is stated as follows:

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

Note $\Gamma(n) = (n-1)!$. The total score of a DAG G for $X_1, \ldots, X_n$ on d is the product of the family scores, *i.e.*, $score(G, d) = \prod_{i=1}^{n} score(X_i|Pa_{X_i}, d)$. (The logarithm of the total score is usually computed to replace all the products and divisions by sums and differences. This makes it easier to compute the scoring function during learning.) During learning, we only need to compute the change in the score due to the DAG operations. When data instances are distributed, computing the required sufficient statistics for the family scores is challenging; this challenge is the motivation for our work.

Example 2.1. Consider the BN shown in Figure 1(a). Let the data instances for the variables $T_1, \ldots, T_n$ be distributed on four cluster nodes as shown in Figure 1(b). Consider the family $T_1|T_3, T_8$. The sufficient statistics for $T_1|T_3 = 0, T_8 = 0$ is (1,2), because there is 1 data instance with $T_1 = 0$, $T_3 = 0$, and $T_8 = 0$ (*i.e.*, on node 4) and 2 instances with $T_1 = 1$, $T_3 = 0$, and $T_8 = 0$ (*i.e.*, on nodes 2 and 3). Similarly, the sufficient statistics for $T_1|T_3 = 1, T_8 = 0$ is (3,1). Once the required sufficient statistics are available, the family score of $T_1|T_3, T_8$ can be computed using Equation 1.

If the structure of a BN is given/known, then the parameters of the conditional probability distributions that best fit the observed data need to be learned. This also requires computing the sufficient statistics of families efficiently and becomes challenging on massive datasets when a large number of variables are present in the BN.

2.2 Parallel BN Learning

Due to the computational complexity of BNs, parallel algorithms were proposed for structure learning of BNs on high-performance computing platforms and shared-memory architectures [37, 47, 49]. Recently, parallel methods for scalable BN learning and reasoning using the MapReduce paradigm were proposed for a shared-nothing cluster [11, 15, 22, 57, 65]. More recently, Arias *et al.* [9] developed parallel versions of Bayesian network classifiers (*e.g.*, Tree Augmented Naive Bayes, k-Dependence Bayesian classifier) by computing multidimensional contingency tables using the MapReduce paradigm on Apache Spark [64]. One may wonder whether we can simply develop a parallel algorithm to compute the family scores using the map and reduce operations in Apache Spark. 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). As shown by the results reported later in Section 4, the MapReduce-style of computing sufficient statistics is very slow and time consuming. Furthermore, the batch-oriented nature of MapReduce requires complete re-execution when new data instances are available.

2.3 Distributed Machine Learning Frameworks

In recent years, there has been much interest in developing scalable distributed machine learning frameworks given the growing number of use cases in the industry. GraphLab [42] exploited common patterns in machine learning algorithms such as sparse computational dependencies and asynchronous iterative computation. Its efficacy was demonstrated on parameter learning and inference in Markov Random Fields, Gibbs sampling, and other machine learning tasks. GraphLab was later extended to operate in a distributed environment with reduced network congestion and latency, and support for fault-tolerance [43]. MLlib [46], a library of Apache Spark, provides scalable implementations of common machine learning algorithms using Apache Spark and its primitives. Another recent framework is the Parameter Server [40], which was developed to scale distributed machine learning algorithms. It proposed an efficient way to aggregate and synchronize model parameters in a distributed setting using asynchronous communication and flexible consistency models. Its efficacy was demonstrated on Sparse Logistic Regression, Latent Dirichlet Allocation, and Distributed Sketching. Recently, Petuum [63] was developed for scaling both data-parallel and model-parallel machine learning algorithms by considering properties such as error tolerance, dynamic structure, and nonuniform convergence. Its benefit was demonstrated on tasks such as topic modeling, deep learning, and Lasso regression.

SystemML [12, 25] proposed a declarative, high-level language for writing machine learning algorithms. Efficient execution plans were generated for these algorithms using SystemML's cost-based optimizer. The algorithms were executed on top of data parallel frameworks such as Apache Hadoop's MapReduce and Apache Spark. A few systems have been proposed to integrate statistical machine learning with a DBMS for improved performance and efficiency (*e.g.*, MADlib [29], UDA-GIST [39]). Recently, Edward [60] was developed for probabilistic modeling on large datasets using TensorFlow [8]. Edward enables a user to build a model of a phenomena (*e.g.*, using directed graphical models and neural networks), reason about the model, and criticize how well the model fits of the data. Edward can exploit GPUs for parallelism. AMIDST [44] is a Java toolbox for scalable probabilistic machine learning and allows a user to build probabilistic graphical models and perform scalable inference. To process large data streams and large-scale datasets, AMIDST employs Apache Flink [23] and Apache Spark [58]. Using a Bayesian approach of updating a model as new data arrive, AMIDST avoids relearning a model from scratch when new data arrive.

Whereas prior efforts focused on scaling a broad class of machine learning algorithms, our goal is centered around fast approximate score computation, a fundamental task during BN structure learning, on large-scale distributed data. Like others, we also aim for a scalable and fault-tolerant solution, which is highlighted next.

2.4 Gossip Algorithms

Gossip algorithms are used by companies such as Amazon and Facebook to build global-scale computing systems like Dynamo [20] and Cassandra [38]. They are also being used in the blockchain technology for scalable data dissemination among peers [30]. 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. Prior work on gossip algorithms have mainly focused on information exchange (or rumor spreading) [21, 24, 33] and computing aggregates (and separable functions) [13, 32, 34, 35, 48]. 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. Previous studies have shown that after a provably finite number of rounds/time intervals and a provably finite number of message exchanges, the information reached all the nodes or the aggregates converged to the true value.

In this work, we draw inspiration from a state-of-the-art gossip algorithm proposed by Mosk-Aoyama and Shah [2, 48] to compute the sum of values stored on n nodes. We call this algorithm SUM. Let $P = [P_{ij}]$ denote a (doubly stochastic and symmetric) 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, $m_l = \min_{i=1}^{n} E_{li}$. Note that the minimum of a set of exponential random variables is an exponential random variable with rate equal to the sum of the rates of the exponential random variables in the set. Finally, SUM 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 all nodes have computed the sum such that the estimate is within δ of the true sum with probability at least $1 - \epsilon$. (This is called the convergence speed.) By choosing $r = \Theta(\delta^{-2}(1 + \epsilon^{-1}))$, it was shown that $T_{SUM}(\epsilon, \delta, P) = O\left(\frac{log\ n + log\ \epsilon^{-1} + log\ \delta^{-1}}{\delta^2 \Phi(P)}\right)$, where $\Phi(P)$ denotes the conductance of the communication topology. Thus, if higher accuracy or confidence is desired by SUM, then a higher value of r must be chosen. When r is increased, the number of exponential random variables maintained at each node also increases along with the size of messages exchanged during gossip.

2.5 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 [19]. Second, the score computation should be efficient and scalable, tolerate failures and changes to the cluster topology, and provide provable guarantees on the accuracy of the estimated sufficient statistics. This requires fast aggregate computation (*e.g.*, sum) over distributed data, effective load balancing of tasks, and redundancy to cope with failures. Although a straightforward application of SUM sounds promising, it unfortunately 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 needed for faster relearning compared to a batch-style approach.

Every data instance/record in the dataset/table will contribute to the sufficient statistics of a family either as a zero or larger value. Hence, every node in the network is involved in computing the sufficient statistics to avoid moving the data to a central location. One may wonder if we can partition the dataset vertically. However, this will introduce additional complexity when a family spans variables across different partitions. Shuffling of data will be required. Hence, it is better to horizontally partition the dataset, where an entire data instance is on a single machine.

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. convergence speed, memory and network bandwidth consumption, and score recomputation when new data are available. DiSC addresses two key issues to achieve fast approximate score computation: (1) distribution of families across cluster nodes for load balancing and (2) approximate score computation of families in an efficient, scalable, and fault-tolerant manner. DiSC can be viewed as a black box (by different score-based BN learning algorithms) to provide an estimate of family scores over large-scale distributed data. Table 1 lists the 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
$\|Val(X)\|$	Number of possible values of X
$\|Val(Pa_X)\|$	Number of possible configurations of Pa_X
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
$E_f^i[\cdot]$	An array of exponential random variables for a counter in SSA_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) [59]). 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.

We address the above issues by synergistically combining consistent hashing [59] and LSH [31]. In consistent hashing, only a finite fraction of the keys needs 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 [27]. 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 $\{h^1_{S_1}, \ldots, h^k_{S_1}\}$ and $\{h^1_{S_2}, \ldots, h^k_{S_2}\}$ denote the outputs of LSH on sets S_1 and S_2, respectively. Prior work has shown 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.*, $h^i_{S_1} = h^i_{S_2}$ $(1 \leq i \leq k)$.

Similar to a DHT, let N_0,..., N_{n-1} denote the n cluster nodes mapped to a m-bit hash address space. We partition the address space $[0, 2^m - 1]$ equally among the cluster nodes. Let $[s_{N_i}, e_{N_i}]$ denote the interval assigned to N_i. (A similar method of assigning ranges is used by Cassandra [38] and Dynamo [20].) Let $\mathbb{L}$ denote LSH on a set that produces k hash values in the range $[0, 2^m - 1]$ (*e.g.*, using SHA-1 or MD5). Given a family $f = X|Pa_X$, we first represent it as a set of random variables $\{X\} \cup Pa_X$. Let $\{h^1_f, \ldots, h^k_f\}$ 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^j_f, where $1 \leq j \leq k$. 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.

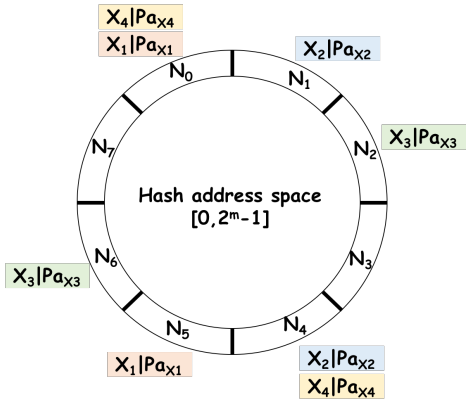
Example 3.1. An example of assignment of families is shown in Figure 2(a). Cluster nodes $N_0, \ldots, N_7$ are assigned intervals in the hash address space. Suppose there are four families $f_1 = X_1|Pa_{X_1}$, $f_2 = X_2|Pa_{X_2}$, $f_3 = X_3|Pa_{X_3}$, and $f_4 = X_4|Pa_{X_4}$. Let $\mathbb{L}$ produce $k = 2$ 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.

Once the families are assigned to cluster nodes, it is possible to apply a gossip algorithm such as SUM to compute the sufficient statistics of the families. For this, we must maintain an array of counters for each family and perform gossiping. However, this will lead to an undesirable scenario in which every node ends up tracking every family, as shown in Figure 2(b). This has shortcomings for the following reasons: each node will have to spend more resources maintaining the families and exchange large messages during gossip. Similar observations were reported in XGossip [54–56], albeit for a different problem and gossip algorithm. So, it is desirable to have families distributed in the manner shown in Figure 2(c), which is the ultimate goal of DiSC.

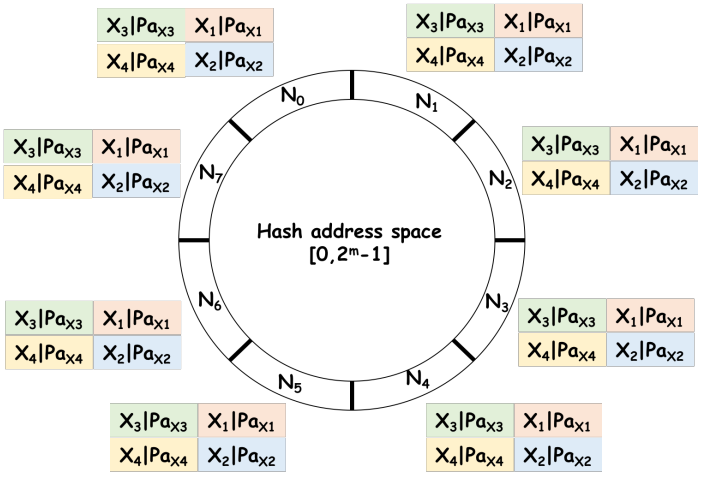
Next, we show a basic approach to compute scores in DiSC using SUM and point out the aforementioned shortcomings. Then, we improve DiSC using our idea of probabilistically dropping families during gossip.

3.2 A Basic Approach for Gossip-Based Score Computation

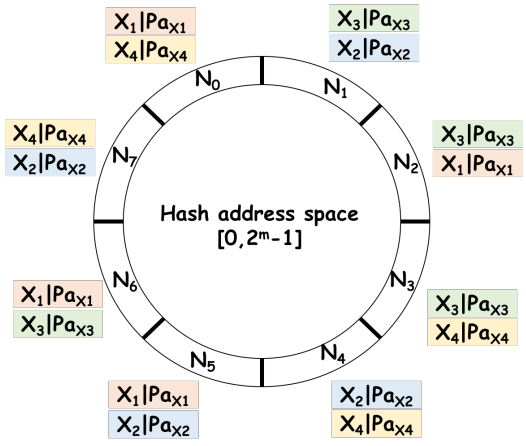
The next challenge is to compute the scores of families in a scalable manner on large distributed data. We need to compute the sufficient statistics of each family. Once we have the sufficient statistics of a family, Equation 1 can be used to compute the score of the family. We first present a basic approach for computing sufficient statistics by applying SUM (Section 2.4) over all the families of interest. Suppose there are n cluster nodes. Each node N_i stores the families assigned to it in its family list $\mathbb{F}_{N_i}$. In the family list, for each family f, a 2D array of size $r' \times c'$ is maintained, where $r' = |Val(X)|$ and $c' = |Val(Pa_X)|$. Each element in this 2D array contains a list of r exponential random variables needed to estimate a value in the sufficient statistics of a family. We call this 2D array the sufficient statistics array (SSA) of f denoted by SSA_f. Figure 3 shows an example of a family list.



(a) Initial assignment of families to the cluster nodes



(b) An undesirable scenario where all the nodes track all the families



(c) A desirable scenario

Fig. 2. Assignment of families during gossip

Next, we present the basic approach to compute scores in DiSC. First, each family is assigned to cluster nodes by invoking Algorithm 1. Depending on the value of k, hash values are constructed

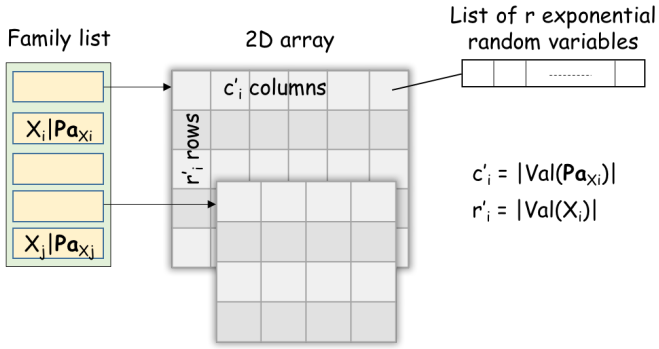


Fig. 3. An example of a family list at a node

for the family (Line 2). For each hash value, the family is routed to the node for that hash value (Line 4). That node adds the family to its family list and initializes the SSA (Line 5).

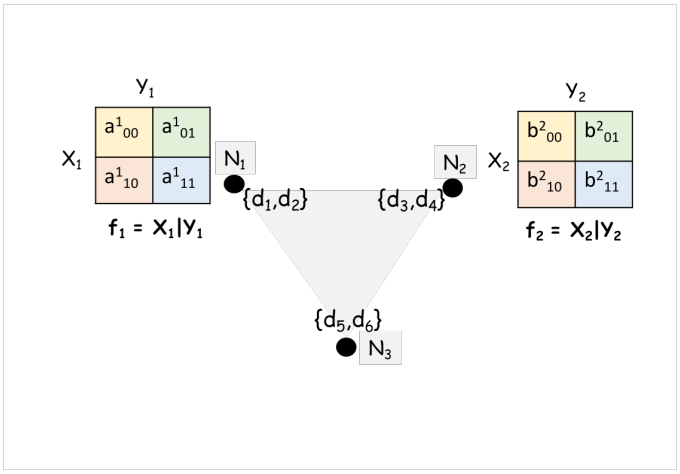
Algorithm 1 AssignFamily(f)

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 the cluster node N_l that is responsible for h_f^j, i.e., $h_f^j \in [s_{N_l}, e_{N_l}]$
5: Add f to the family list $\mathbb{F}_{N_l}$ of N_l and set SSA_f to NULL

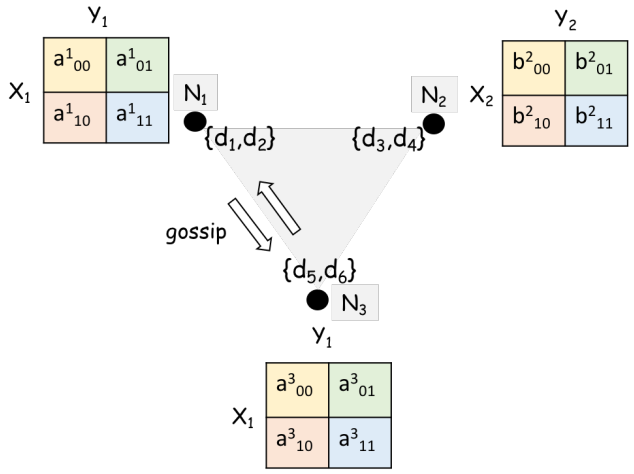
Algorithm 2 shows the actions performed by every cluster node. Consider node N_i. It first initializes the SSAs for every family in $\mathbb{F}_{N_i}$ by using local data instances and generating the exponential random variables for the partial sufficient statistics of the family by invoking InitLocalState(·) (Lines 2-4). The local clock is initialized as a rate 1 Poisson process (Line 6). (The specific implementation of the local clock is shown later in Section 4.) When its local clock ticks, it becomes active during gossiping and does the following steps: Pick a neighbor N_j with probability O_{ij} (Line 8). Exchange between N_i and N_j the SSAs of the families in their family lists (Lines 9-10). (Note all the families in their family lists are exchanged.) For each family in the family list of N_i, the minimum is computed for the exponential random variables for each element of the SSA of the family in $\mathbb{F}_{N_i}$ (Lines 11-17). Finally, those families in $\mathbb{F}_{N_j}$ that are not in $\mathbb{F}_{N_i}$ are added to $\mathbb{F}_{N_i}$ along with their SSAs (Lines 18-19).

Algorithm 4 lists the steps performed by a cluster node when it is receives messages from other nodes during gossiping. If the family in a received message is not in the family list of the receiving node, then the node initializes the SSA of the family (using any local data blocks) (Lines 4-6). The node responds to the sender with the SSAs of the families in its family list (Line 7). Next, for the families in the family list, the minimum of the exponential random variables for each element in an SSA is computed (Lines 8-13).

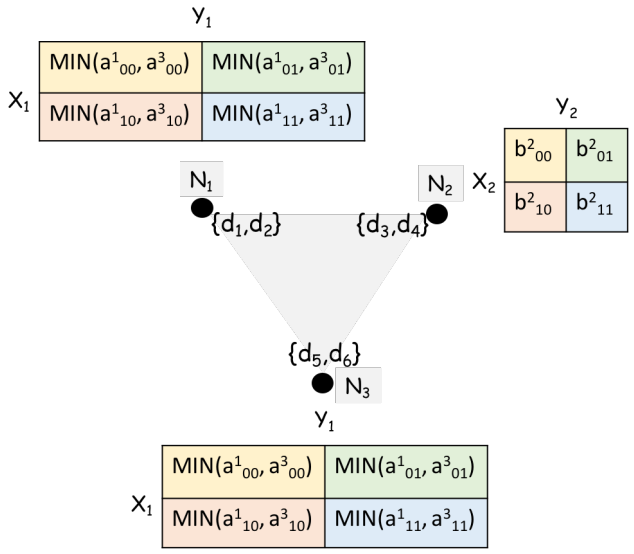
Example 3.2. Consider a cluster with 3 nodes, N_1, N_2, and N_3 as shown in Figure 4(a). Let $\{d_1, d_2\}$, $\{d_3, d_4\}$, and $\{d_5, d_6\}$ denote the data instances stored on N_1, N_2, and N_3, respectively. Let $f_1 = X_1|Y_1$ and $f_2 = X_2|Y_2$ denote two families on binary random variables. Suppose we need to compute the sufficient statistics of f_1 and f_2 using DiSC. As shown in the figure, N_1 is responsible for f_1; N_2 is responsible for f_2. The exponential random variables in the SSAs for these families is also shown. Let $r = 1$ (i.e., the number of exponential random variables per counter). Let a_{jk}^i denote the value



(a) Initial values of exponential random variables in the SSAs



(b) Gossip between N_1 and N_3



(c) Updated exponential random variables after gossip

Fig. 4. Steps during gossip

Algorithm 2 DiSC-Basic-Send()

Let N_i denote the cluster node executing this procedure
for all $f \in \mathbb{F}_{N_i}$ **do**
  $SSA_f \leftarrow$ InitLocalState(f)
  Store SSA_f for f in $\mathbb{F}_{N_i}$
Let $\mathbf{O}$ denote the doubly stochastic symmetric transition matrix for the n cluster nodes, s.t. $O_{ij} = \frac{1}{n}$
Initialize rate 1 Poisson process at node N_i as the local clock for gossiping
for each local clock tick **do**
  Pick a neighbor N_j with probability O_{ij}
  Send exponential random variables in SSA for families in F_{N_i} to N_j
  Receive exponential random variables in SSA for families in F_{N_j} from N_j
  for each family $f \in \mathbb{F}_{N_i}$ **do**
    for each element $e \in SSA_f$ **do**
      Let E_f^i denote the list of exponential random variables for e in SSA_f
      Let E_f^j denote the list of exponential random variables for e received from N_j
      if $E_f^j \neq$ null **then**
        for $q = 1$ to r **do**
          $E_f^i[q] \leftarrow min(E_f^i[q], E_f^j[q])$
  for all family $f' \in \mathbb{F}_{N_j}$ s.t. $f' \notin \mathbb{F}_{N_i}$ **do**
    Add f' and its SSA to $\mathbb{F}_{N_i}$

Algorithm 3 InitLocalState(f)

Read local data and compute the counts for f and store in a 2D array
for each counter in 2D the array **do**
  Let v denote the value of the counter
  Generate the list of r independent exponential random variables with rate v and store in SSA_f
return SSA_f

of the exponential random variable for the frequency count of $X_1 = j|Y_1 = k$ on the data instances stored at N_i. Similarly, b_{jk}^i is for $X_2 = j|Y_2 = k$ on N_i.

Next, we show how the gossip phase works. Suppose the local clock of N_1 ticks first. Let N_1 pick N_3 to exchange the state. As shown in Figure 4(b), N_1 and N_3 exchange the exponential random variables. N_3 has to compute a_{00}^3, a_{01}^3, a_{10}^3, and a_{11}^3 over $\{d_5, d_6\}$. N_3 learns about f_1 from N_1 and updates its family list. As shown in Figure 4(c), N_1 and N_3 compute the minimum of a_{ij}^1 and a_{ij}^3 after exchanging state.

After several clock ticks, the nodes reach a state as shown in Figure 5(a). The final values of $\mathbf{a}_{ij}$ and $\mathbf{b}_{ij}$ are shown in Figure 5(b). Each node maintains the SSAs for f_1 and f_2. The estimates of the sufficient statistics of f_1 (on any node) are $(\frac{1}{a_{00}}, \frac{1}{a_{01}}, \frac{1}{a_{10}}, \frac{1}{a_{11}})$ because $r = 1$. In addition, the estimates of the sufficient statistics of f_2 (on any node) are $(\frac{1}{b_{00}}, \frac{1}{b_{01}}, \frac{1}{b_{10}}, \frac{1}{b_{11}})$. □

Next, we state results on the convergence of DiSC and the size of the family list at each node.

Algorithm 4 DiSC-Basic-Receive()

```
1: Let N_j denote the cluster node executing this procedure
2: while new message is received do
3:   for each family f in the message do
4:     if f ∉ F_{N_j} then
5:       SSA_f ← InitLocalState(f)
6:       Store f and SSA_f in F_{N_j}
7:   Respond to sender with exponential random variables for the families in F_{N_j}
8:   for each family f in the message do
9:     for each element e ∈ SSA_f in F_{N_j} do
10:      Let E_f^j denote the list of exponential random variables for e in SSA_f
11:      Let E_f^i denote the list of exponential random variables for e received from sender N_i
12:      for q = 1 to r do
13:        E_f^j[q] ← min(E_f^i[q], E_f^j[q])
```

THEOREM 3.3. *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 δ with a probability of at least $1 - \epsilon$. Then $T_{DiSC}(f, \epsilon, \delta) = T_{SUM}(\epsilon, \delta, \mathbf{O})$.*

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. DiSC is based on SUM with the probability transition matrix $\mathbf{O}$ to estimate SSA_f. Thus, the convergence speed of DiSC is $T_{SUM}(\epsilon, \delta, \mathbf{O})$. □

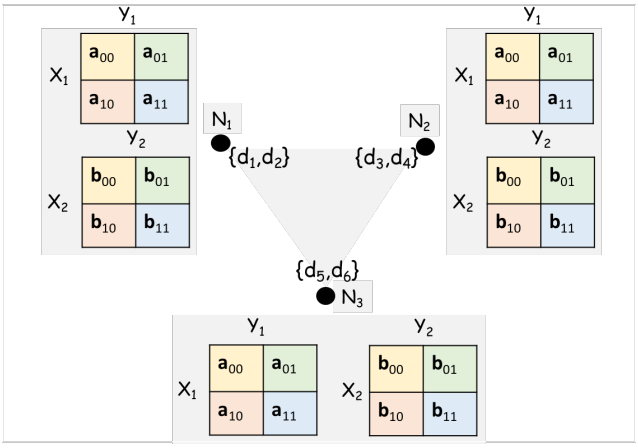
THEOREM 3.4. *Let $\mathbb{D}$ denote the number of distinct families and k denote the number of hash values output by $\mathbb{L}$. The expected value of the size of family list at a cluster node is $O(\mathbb{D})$.*

Proof. During gossip, each time a node communicates with another node, its learns any new families that the other node has. Ultimately, every node learns every distinct family in the network. Thus, the expected size of the family list at each node is upper bound by the number of distinct families. □

Unfortunately, a major drawback of the basic approach of score computation in DiSC 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 compute the sufficient statistics on during learning. As a result, each node will send large messages through the network during gossip leading to increased network bandwidth consumption.

3.3 An Improved Approach for Gossip-Based Score Computation

To overcome the above limitations, we develop an improved algorithm by using a probabilistic approach for guaranteeing a bound on the number of families managed by each node. As shown in Figure 2(c), we would like each node to manage only a finite fraction of the families with high probability. 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



(a) Final state on all the nodes

(b) Final values of the exponential random variables on all the nodes

Fig. 5. Final state of gossiping

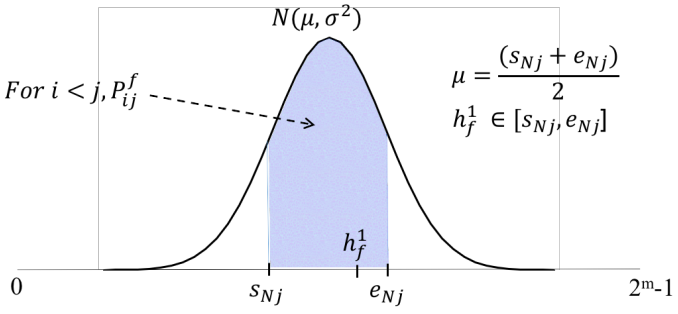


Fig. 6. The function P_{ij}^{f}

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. Let us define a few transition matrices. Let us define $\mathbf{O}$ to be a doubly stochastic transition matrix where $O_{ij} = \frac{1}{n}$. For each family f, let us define $\mathbf{P}^f$, another doubly stochastic transition matrix, s.t. $P^f_{ij} = \int_{s_{N_j}}^{e_{N_j}} N(\mu, \sigma^2)\, dh$ for $i < j$, where $\mu = \frac{(s_{N_j}+e_{N_j})}{2}$. (Note that $\mathbf{P}^f$ is defined to be a symmetric matrix.) That is, for f, we can select a normal distribution with mean $\frac{(s_{N_j}+e_{N_j})}{2}$ and some standard deviation σ. An illustration of P^f_{ij} is shown in Figure 6. An interesting observation is that P_{ij} peaks when N_j is responsible for f. In addition, the total area under the curve of a normal distribution is 1. Finally, let us define $\mathbf{Q}^f$ to be a doubly stochastic transition matrix s.t. $Q^f_{ij} = O_{ij} \times P^f_{ij}$ for $i \neq j$.

Algorithms 5 and 6 show the steps performed by the improved approach for gossip-based score computation in DiSC. Algorithm 5 lists the actions performed by every cluster node. Consider node N_i. Similar to the basic approach, it first initializes the SSAs for every family in $\mathbb{F}_{N_i}$ by using local data instances and generating the exponential random variables for the partial sufficient statistics of the family by invoking InitLocalState(·) (Lines 2-4). The local clock is initialized as a rate 1 Poisson process. When its local clock ticks, it becomes active during gossiping and does the following steps: Pick a neighbor N_j with probability O_{ij}. N_i sends the SSAs of all the families in its family list to N_j. N_i receives the SSAs for the families in its family list from N_j (Lines 9-10). (Note that this is different from what is done in the basic approach.) The SSAs for every family in $\mathbb{F}_{N_i}$ is updated based on the SSAs received from N_j by computing the minimum of the exponential random variables (Lines 11-17). The next steps involve dropping families probabilistically (Lines 18-26), which is a major difference from the basic approach. A list $\mathbb{G}$ is maintained to keep track of families (and their SSAs) that N_j should be informed to add to its family list after N_i drops them. For each family f in $\mathbb{F}_{N_i}$, if N_i is responsible for f, add f and SSA_f to $\mathbb{G}$ with probability P^f_{jk}. Otherwise, with probability P^f_{jk}, remove f from the family list and add f and SSA_f to $\mathbb{G}$. Once all the families are processed, inform N_j to store the families in $\mathbb{G}$ in its family list.

This key idea of probabilistically removing a family from the family list of a cluster node during score computation prevents the family list from growing very large. (See Section 3.5 for a bound on the size of the family list.)

Algorithm 6 lists the steps performed by every cluster node when it is receives messages from other nodes during gossiping. If the message received contains exponential random variables of families, and if a family under consideration is not in the family list of the receiving node, then the receiving node initializes the SSA of the family (using any local data blocks) by calling InitLocalState(·) (Lines 4-8). A temporary family list is maintained to keep track of SSAs of families that are seen for the first time by the receiving node. The receiving node responds to the sender with the SSAs of all the families it knows that are also in the sender's family list (Line 9). The receiving node then updates its family list by computing the minimum of exponential random variables in the SSAs (Lines 10-16). If the message from the sender indicates adding a set of families, then the receiving node stores them in its family list (Lines 17-20).

3.4 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 efficiently precomputed the sufficient statistics of large number of families required during learning. When the learning algorithm executes on a cluster node and needs the sufficient statistics of a family, it will invoke Algorithm 7. The algorithm first computes the hash values for the family by applying $\mathbb{L}$ (Line 2). For each hash value, the algorithm contacts the cluster node

Algorithm 5 DiSC-Improved-Send()

1: Let N_i denote the cluster node executing this procedure
2: **for all** each $f \in \mathbb{F}_{N_i}$ **do**
3: $SSA_f \leftarrow$ InitLocalState(f)
4: Store SSA_f for f in $\mathbb{F}_{N_i}$
5: Let $\mathbf{O}$ denote the doubly stochastic transition matrix of a Markov chain representing the n cluster nodes, s.t. $O_{ij} = \frac{1}{n}$
6: Initialize rate 1 Poisson process at node N_i as the local clock for gossiping
7: **for** each local clock tick **do**
8: Pick a neighbor N_j with probability O_{ij}
9: Send exponential random variables in SSA for families in F_{N_i} to N_j
10: Receive exponential random variables in SSA for families in F_{N_i} from N_j
11: **for** each family $f \in \mathbb{F}_{N_i}$ **do**
12: **for** each element $e \in SSA_f$ **do**
13: Let E^i_f denote the list of exponential random variables for e in SSA_f
14: Let E^j_f denote the list of exponential random variables for e received from N_j
15: **if** $E^j_f \neq$ null **then**
16: **for** $q = 1$ to r **do**
17: $E^i_f[q] \leftarrow min(E^i_f[q], E^j_f[q])$
18: $\mathbb{G} \leftarrow \emptyset$
19: **for** each family $f \in \mathbb{F}_{N_i}$ **do**
20: Let $\mathbf{P}^f$ denote a doubly stochastic transition matrix for f as described in Section 3.3
21: **if** $\exists j$, $1 \leq j \leq k$, s.t. $h^j_f \in [s_{N_i}, e_{N_i}]$ **then**
22: With probability P^f_{ij}, add f and SSA_f to $\mathbb{G}$
23: **else**
24: With probability P^f_{ij}, remove f and SSA_f from $\mathbb{F}_{N_i}$ and add f and SSA_f to $\mathbb{G}$
25: **if** $\mathbb{G} \neq \emptyset$ **then**
26: Inform N_j to store families in $\mathbb{G}$

responsible for that hash value to obtain the SSA for the family (Line 5).[1] For each element in SSA_f received from a node, the algorithm computes the estimate over r exponential random variables and stores it in a 2D array denoted by est_j (Lines 7-13). Finally, the element-wise median of estimate arrays serves as the final estimate of the sufficient statistics of the family. Because of LSH, it is more likely for the learning algorithm to retrieve the SSAs of similar families from the same node, potentially reducing the network latency during learning.

One may wonder how DiSC is beneficial to a structure learning algorithm and why parallelism is needed for score computation of families. Suppose we wish to learn the structure of a BN with large number of variables, namely, $X_1, \ldots, X_n$ over large number of data instances distributed across nodes in a cluster. During structure learning, we need to know the scores of certain families as the DAG operations are executed. One solution is the learn the score of a family of interest during the execution of the structure learning algorithm. This would require reading all the data instances each time a family score is needed. But in a distributed setup and on large-scale data, this could make the structure learning algorithm wait on computing the sufficient statistics over distributed data.

[1] In a system like Voldemort, the lookup cost is O(1).

Algorithm 6 DiSC-Improved-Receive()

Let N_j denote the cluster node executing this procedure
while new message is received **do**
  if the message contains exponential random variables of families **then**
    $\mathbb{F}_{tmp} \leftarrow \emptyset$
    for each family f in the message **do**
      if $f \notin \mathbb{F}_{N_j}$ and f is seen first time **then**
        $SSA_f \leftarrow$ InitLocalState(f)
        Store f and SSA_f in $\mathbb{F}_{tmp}$
    Respond to the sender with exponential random variables for the families in the message that N_j has in either $\mathbb{F}_{tmp}$ or $\mathbb{F}_{N_j}$
    for each family f in the message **do**
      if $f \in \mathbb{F}_{N_j}$ **then**
        for each element $e \in SSA_f$ **do**
          Let E_f^j denote the list of exponential random variables for e in SSA_f
          Let E_f^i denote the list of exponential random variables for e received from sender N_i
          for $q = 1$ to r **do**
            $E_f^j[q] \leftarrow min(E_f^i[q], E_f^j[q])$
  else if the message indicates storing families **then**
    for each f in the message **do**
      if $f \notin \mathbb{F}_{N_j}$ **then**
        Add f and SSA_f to $\mathbb{F}_{N_j}$

Algorithm 7 GetSufficientStatistics(f)

Let $f = X|Pa_X$
$\{h_f^1, \ldots, h_f^k\} \leftarrow \mathbb{L}(\{X\} \cup Pa_X)$
for $j = 1$ to k **do**
  Route f to the cluster node N_l that is responsible for h_f^j, i.e., $h_f^j \in [s_{N_l}, e_{N_l}]$
  Receive SSA_f for f stored in $\mathbb{F}_{N_l}$ from N_l
  Initialize est_j to denote a 2D array (with $r' \times c'$ counters) to store the estimates of the sufficient statistics of f (from N_l)
  for each element $e \in SSA_f$ **do**
    Let E_f^l denote the list of exponential random variables for e
    $temp \leftarrow 0$
    for $q = 1$ to r **do**
      $temp \leftarrow temp + E_f^l[q]$
    Let u, v denote the array indices for e in est_j
    $est_j[u][v] \leftarrow \frac{r}{temp}$
return element-wise median for the arrays est_1, ..., est_k

Another solution is to precompute the scores of possible families needed during structure learning[2] on large-scale distributed data. It is true that there will be many families that do not get considered

[2]For instance, we can consider for every variable, a family with up to a certain number of parents.

during structure learning but are still precomputed. However, if the precomputation of sufficient statistics can be done efficiently, it is well worth the effort. Therefore, for efficiency, parallelism and cluster computing should be exploited. This way the score of a family is readily available to the structure learning algorithm when updating the overall score due to DAG operations. We therefore pursue the latter solution of precomputing the sufficient statistics of families for structure learning.

3.5 Theoretical Analysis

3.5.1 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 3.5. *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 δ with a probability of at least $1 - \epsilon$. 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 SSAs of families with probability O_{ij} in Algorithm 5 (Lines 9-10). However, we move a family from one node to another only with probability $Q^f_{ij} = O_{ij} \times P^f_{ij}$ in Algorithm 5 (Lines 18-26). (Note that $Q^f_{ij} \leq O_{ij}$ 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 SSAs 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 3.6. *For a family f, let $\boldsymbol{\pi}_f = [\pi^1_f \ldots \pi^n_f]$ 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 $E(|\mathbb{F}_{N_i}|) = \sum_{f \in \mathbb{D}} \pi^i_f + \frac{k\mathbb{D}}{n}$.*

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), $E(Z) = \sum_{f \in \mathbb{D}} E(Y_f) + E(U) = \sum_{f \in \mathbb{D}} \pi^i_f + \frac{k\mathbb{D}}{n}$. □

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 $X_i|Pa_{X_i}$ is a 2D array of size $r'_i \times c'_i$, 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 3.6. According to Theorem 3.5, the number of time steps required by DiSC for convergence of the sufficient statistics of a family is

given by $T_{DiSC}(f, \epsilon, \delta)$. Each time step has on an average n clock ticks, one per node [13]. Suppose each node maintains r exponential random variables per element in the 2D array for 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.

3.5.2 Comparison With MapReduce-Style Computation. Hereinafter, we use MR-SS (**M**ap**R**educe-based **S**ufficient **S**tatistics) to denote the MapReduce-style of computing sufficient statistics. We provide insight on the difference between DiSC and MR-SS, in terms of the network bandwidth consumption. For MR-SS, we assume a simple model: The map phase is run on all the cluster nodes to process all the data blocks to produce intermediate key-value pairs. The reducer phase, which may run on a subset of cluster nodes, needs to receive the intermediate key-value pairs. Thus, in the worst case, all the intermediate key-value pairs produced during the map phase must be transmitted across the network. We will assume compression is not used by both approaches.

Let us analyze the process of computing the scores of $\mathbb{D}$ distinct families using MR-SS. 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 and combined to produce the final sufficient statistics for the $\mathbb{D}$ families. On a cluster of n nodes and b data blocks, the map phase will produce intermediate key-value data of size proportional to $n \times b \times \sum_{f \in \mathbb{D}} (r_f \times c_f)$ 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 $O(nb\mathbb{D}\mathbb{S})$, 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 $O(\frac{log(n)}{\Phi(\mathbf{Q}^f)})$, given a user-specified accuracy, LSH parameters, and other user-defined parameters. Each time step has on an average n clock ticks, one per node [13]. Each clock tick results in communication. For simplification of analysis, suppose we assume DiSC does not drop families. Then $\Phi(\mathbf{Q}^f) = \Phi(\mathbf{O}) \approx 0.5$ [2]. For $\mathbb{D}$ families, the total communication cost is $O(nlog(n)\mathbb{D}\mathbb{S})$. As b grows faster than $log(n)$ asymptotically, DiSC has lower communication cost compared to MR-SS.

3.6 Recomputing Family Scores on New Data

Because gossiping can be stopped on the cluster nodes after a period of time and started again, DiSC can efficiently recompute the family scores as new data are produced. Unlike AMIDST [44] that is designed for a streaming scenario where new data arrive continuously, we focus on stored datasets that may be updated over time but not in real-time. Suppose a new data block with some number of data instances is added to a cluster node. This node will compute the SSAs for all families under consideration over the data block using InitLocalState(·). By applying $\mathbb{L}$, a node responsible for each family can be identified, for example, based on the first hash value output by $\mathbb{L}$. Next, the families can be grouped by the node responsible for them. The SSAs for each group of families can be sent to the respective node responsible for those families. The receiving node can update its family list with the SSAs by computing the minimum of exponential random variables similar to the regular gossip phase of DiSC. Recall that a node responsible for a family never drops the family during gossip. Once the SSAs are updated by the nodes, all the cluster nodes can continue to execute DiSC, thereby disseminating the sufficient statistics computed over the new data to other nodes, resulting in efficient score recomputation.

In contrast to DiSC, the batch-style processing of MapReduce must process the entire dataset (with new data) to obtain the new sufficient statistics of the families. As a result, DiSC is better choice than MR-SS for score recomputation when new data are available.

4 PERFORMANCE EVALUATION

In this section, we report the performance evaluation of DiSC and comparison with MR-SS. We demonstrate that DiSC provides a feasible tradeoff between computation time and accuracy for fast approximate score computation on datasets with different characteristics.

4.1 Implementation and Environment

For implementation, we used the Java package for gossip algorithms available online [1, 53]. The code was compiled using Java 1.8. For MR-SS, we implemented the code in Scala using Scala 2.11.8. The code was executed using Apache Spark 2.0.2 and Apache Hadoop 2.7.3. We conducted all our experiments on CloudLab [3], a *testbed* for cloud computing research and new applications. We ran all the experiments for DiSC and MR-SS on 16 nodes in the Utah data center of CloudLab. (Spark was run in the standalone mode on the cluster.) These nodes were configured with OpenStack Mitaka on Ubuntu 16.04. Each node had eight 64-bit ARMv8 cores, 120 GB of flash storage, and 64 GB RAM. Each node was configured with a network link speed of 1Gbps. One node was run as the controller and the remaining were configured as compute nodes. All 16 nodes were used to run the experiments.

4.2 Local Clock and Exponential Random Variables

For DiSC, we needed a way to generate Poisson processes and exponential random variables. We implemented local clocks (rate 1 Poisson process) using Algorithm 8 and generated exponential random variables using Algorithm 9. Note that these algorithms are based on the work done by McQuighan [45].

Algorithm 8 LocalClock(λ, c)

1: Let λ denote the rate of Poisson process
2: Let c denote a positive integer (a.k.a. delay constant)
3: Pick a random number x uniformly between 0 and 1
4: $y \leftarrow -\frac{log(1-x)}{\lambda}$
5: Sleep for $c \times y$ seconds
6: **return**

Algorithm 9 ExpRand(v)

1: Let v denote a positive integer and the rate of the exponential random variable
2: Pick a random number x uniformly between 0 and 1
3: $y \leftarrow -\frac{log(1-x)}{v}$
4: **return** y

4.3 MR-SS

Algorithm 10 lists the steps performed by MR-SS to compute the sufficient statistics of families. The input file is a CSV (comma-separated values) file (stored in HDFS) containing the values of the n binary random variables. Note that $flatMap$ applies *myMapFunc* on a block of lines to produce a collection of key-value pairs. The key is a family. The *reduceByKey* applies *myReduceFunc* to all key-value pairs with the same key. That is, the partial counts for each family is added to produce the true count across all the lines in the input file. The code for MR-SS was written in Scala.

Algorithm 10 MR-SS

Let $\mathbb{D}$ denote the list of families
$record\_blks \leftarrow read\_record\_blocks(\text{“}/hdfs\_file\text{”})$
$partialSSA \leftarrow record\_blks.flatMap(mapFunc)$
$finalSSA \leftarrow partialSSA.reduceByKey(reduceFunc)$
Write $finalSSA$ to a file
return

function mapFunc(blk)
Let blk denote a block of records
for all $f \in \mathbb{D}$ **do**
    Initialize the counter array CA_f of size $r \times c$
for all f in $\mathbb{D}$ **do**
    Compute sufficient statistics of f over b records in rec_blk and store in CA_f
    Output/return key-value pair (f, CA_f)
end function

function reduceFunc(CA_f^a, CA_f^b)
Let CA_f^a and CA_f^b denote two counter arrays for the same family
Compute CA by performing element-wise addition of CA_f^a and CA_f^b
Output/return CA
end function

Note that DiSC and MR-SS both stop at computing the sufficient statistics of families. The scores can be computed by applying Equation 1.

4.4 Datasets

We conducted the experiments using three synthetic datasets, the HIGGS dataset [10] from the UC Irvine (UCI) Machine Learning repository [7], and a dataset based on tweets collected from Twitter.

The three synthetic datasets, S_1, S_2, and S_3, each had 100 binary random variables (or features) and 200 million rows. We generated the data instances for these datasets as follows: For each dataset, we assumed 5 multinomial random variables, each of which could take 20 distinct values. We used a binomial distribution $B(n, p)$ to generate the values for each variable, where n is the number of trials and p is the probability of success in each trial. We set $n = 19$ for the three datasets to assign 20 distinct values to each multinomial random variable. A variable was assigned a value k with probability $\binom{n}{k}p^k(1-p)^{n-k}$. We used $p = 0.25$ for S_1, $p = 0.5$ for S_2, and $p = 0.75$ for S_3 so that the datasets have different distributions. After generating the data instances for the multinomial random variables for a particular n and p, we used one-hot encoding to map them into binary random variables. The sufficient statistics for a family of multinomial random variables can be computed by examining the counters computed for corresponding binary random variables.

The HIGGS dataset was based on Monte Carlo simulations and contained 11 million data instances, 1 class label (0 or 1), and 28 features. The 21 features were based on properties measured by particle detectors; 7 features were functions of the 21 features. The 28 features were assigned real numbers in the dataset. For each feature, we only considered the integer part of a real number assigned to it and mapped it to a multinomial random variable based on the number of unique integer values for that feature. For example, if a feature was assigned to five values in the dataset, namely, 1.1, -2.2,

3.3, -4.4, and -4.2, then this feature was mapped to a multinomial random variable with 4 levels. We then represented each multinomial random variable as a set of binary random variables using one-hot encoding. So in total, we had 240 binary random variables for HIGGS. In order to have a larger dataset for the experiments, we replicated the 11 million instances 16 times.

The Tweets dataset was based on 200 million tweets collected during 2016. We extracted the language attribute in each tweet along with other Boolean attributes such as *isVerified*, *isPossiblySensitive*, *isRetweeted*, *isGeoEnabled*, and so on. Because the language feature/attribute was a categorical variable, we used one-hot encoding to convert it to a set of binary features based on the language value like *en*, *es*, *jp*, *fr*, *ru*, and so on. Essentially, the features of this dataset were a set of binary random variables.

Table 2 summarizes the characteristics of the four datasets we used for the experiments. On each dataset, the sufficient statistics of 10,000 families were computed. The average number of binary random variables per family was 3.96 for the synthetic datasets, 5 for the HIGGS dataset, and 3.97 for the Tweets dataset.

Dataset	No. of instances	No. of binary random variables	No. of families	File size
S_1	200M	100	10,000	37.2 GB
S_2	200M	100	10,000	37.2 GB
S_3	200M	100	10,000	37.2 GB
HIGGS	176M	240	10,000	79.0 GB
Tweets	200M	136	10,000	50.7 GB

Table 2. Datasets and their characteristics

4.5 Setup and Evaluation Metrics

DiSC was executed by specifying a time budget after which the gossiping was terminated. One process was started on each cluster node. After the time budget expired, all the processes were gracefully terminated. The processes were programmed to output the estimates of the counters for each family into a log file. On the other hand, MR-SS was run as a Spark job using the *spark-submit* command to use all the 128 cores in the cluster. Both the executor memory and driver memory were set to 50 GB. We ran MR-SS with LZ4 compression [5], a lossless data compression algorithm, and Java serializer as well as Snappy compression [6] and Kyro serializer [4]. One executor was run on each cluster node; each executor used 8 cores on the node.

We compared DiSC and MR-SS by measuring the total wall-clock time to compute the sufficient statistics of a given set of families on the aforementioned datasets. By design, MR-SS computes exact sufficient statistics. Nonetheless, we investigated how random sampling of the data instances in a dataset could speed up MR-SS albeit obtaining approximate sufficient statistics. Suppose we randomly select $t\%$ of the data instances in a dataset. Then we will first compute the sufficient statistics on the samples using MR-SS. To estimate the true value of sufficient statistics, we will multiply the estimated counts by $\frac{100}{t}$. In the experiments, we computed the accuracy of MR-SS as follows: Compute the relative error of each counter in the counter array maintained for a family (Algorithm 10). Compute the average relative error over all the families. When the entire dataset was processed by MR-SS (*i.e.*, no sampling), 100% accuracy was achieved.

By design, DiSC computes approximate sufficient statistics. We computed the accuracy of the estimates of sufficient statistics of the families. We did the following at each cluster node: Compute

the relative error of the estimated count for each element of the SSA of a family that the cluster node is responsible for. Compute the average over all the families that the node is responsible for. Only the families that a node is responsible for are considered during relative error calculation at a node, because given a family only the nodes responsible for the family should be contacted to compute the sufficient statistics estimates. For the time budget, we ran DiSC long enough on a dataset so that the average relative error computed on each node reached and stayed below 10%. For DiSC, we also measured the total number of messages sent by the cluster nodes during execution, average message size, percentage of messages lost, and average size of the family lists, which provided insight into the reduction in the network bandwidth consumption due to dropping families probabilistically during gossiping.

4.6 Impact of r on the Relative Error of the Estimates in DiSC

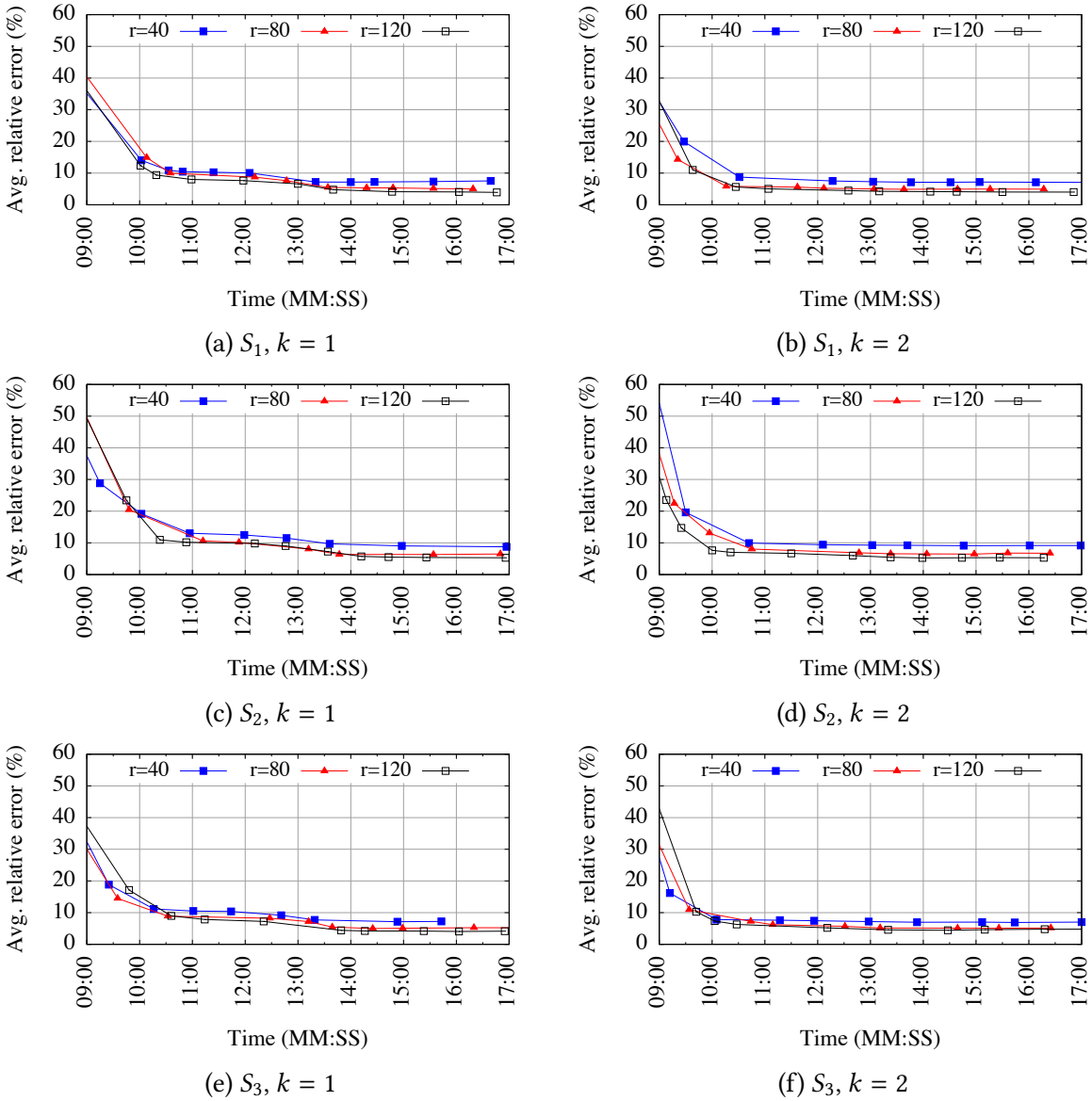
As discussed in Section 2.4, the parameter r should be chosen based on the desired accuracy and confidence. The value of r will increase when higher level of accuracy/confidence is desired. However, by increasing r, we also increase the size of messages exchanged during gossip, and hence the network communication cost. In DiSC, we maintain r exponential random variables per element of the 2D array that serves as the SSA for a family. To understand the impact of r on the accuracy of the estimates of sufficient statistics, we empirically studied the accuracy achieved by DiSC for different values of r: 40, 80, and 120. To understand the robustness of DiSC, we tested with different values of k. Note that k controls the level of redundancy by assigning a family to k cluster nodes during the execution of DiSC. Thus, a family will always be stored/maintained in the family lists of k cluster nodes and will never be dropped by these nodes. Note that when a node gossips with another node it may learn new families that are not in its family list. Hence, a family could be in the family list of $\geq k$ nodes. If a node fails during execution of DiSC, having $k > 1$ is beneficial so that we do not lose a family permanently. (See Algorithms 1 and 7 where k is mentioned.) Although in our experiments, we did not have any node failures, we still varied k from 1 to 3 to understand how k impacts DiSC's performance and resource consumption.

As gossiping progresses in DiSC, the estimate of the sufficient statistics of a family *tends to converge closer to the true value.* Hence, we expect the average relative error to decrease and eventually stabilize at a positive value on each cluster node. The total time budget to execute DiSC was chosen empirically by observing how soon the relative errors stabilized for our cluster setup and a delay constant of $c = 10$ for the local clock (Algorithm 8).

In the interest of space, we show the results for one of the cluster nodes. Similar results were obtained on the remaining nodes. Let us first discuss the results for the synthetic datasets. Figures 7(a)-7(b) show how the average relative error *decreased* and stabilized on one of the cluster nodes over time for S_1 given different values of k. Similarly, Figures 7(c)-7(d) and Figures 7(e)-7(f) show how the average relative error decreased on a cluster node over time for S_2 and S_3, respectively. (Similar trends were observed for $k = 3$.) In each plot, the X-axis denotes the wall-clock time (MM:SS) and the results are shown from the 9 minute mark, and the Y-axis denotes the % average relative error computed for the families that the cluster node is responsible for. In general, as r was increased, the final accuracy of DiSC improved and the average relative error of DiSC stabilized to a lower value by the completion of the time budget. For example, in Figure 7(a), the final average relative error was 7.5%, 5.0%, and 3.9% for $r = 40$, $r = 80$, and $r = 120$, respectively.[3]

For HIGGS and Tweets, similar trends were observed for the average relative error when r was increased from 40 to 120. Figures 8(a)-8(b) and Figures 8(c)-8(d) show how the average relative

[3]Towards the end of the time budget, the red line ($r = 80$) is usually sandwiched between the blue line ($r = 40$) and the black line ($r = 120$).



(a) S_1, $k = 1$ (b) S_1, $k = 2$

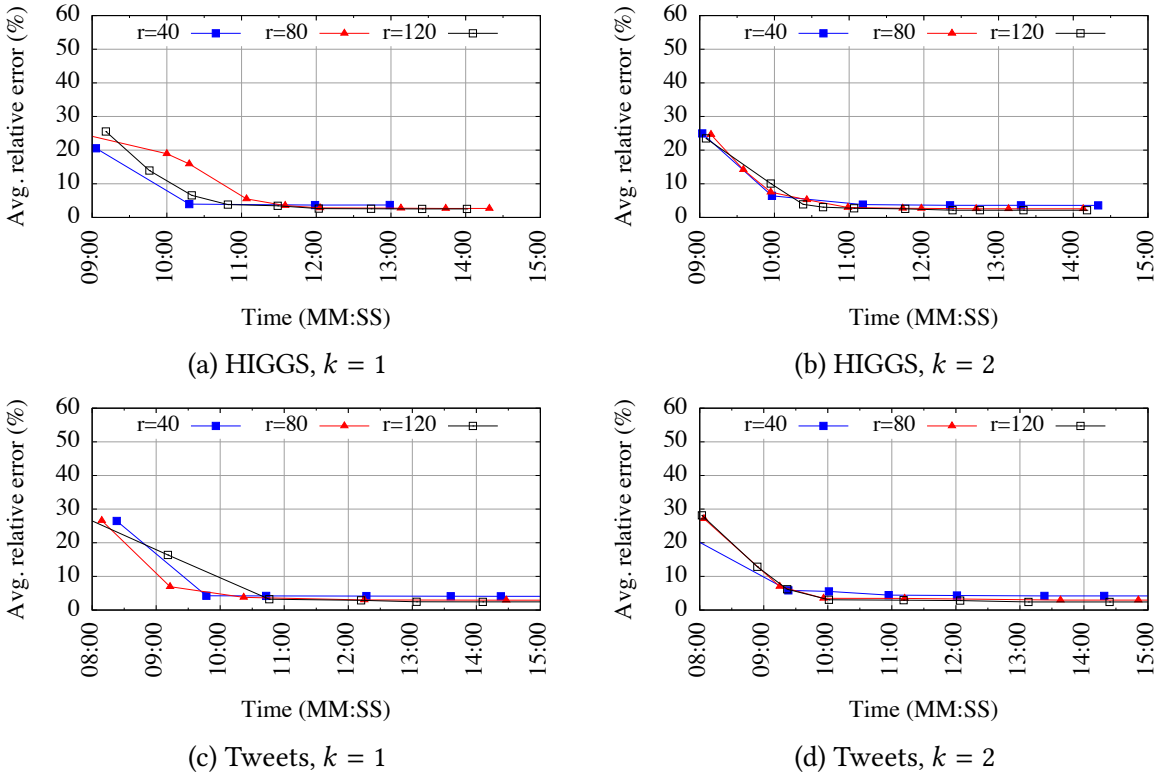
(c) S_2, $k = 1$ (d) S_2, $k = 2$

(e) S_3, $k = 1$ (f) S_3, $k = 2$

Fig. 7. Synthetic datasets: impact of r on the average relative error of the estimates computed by DiSC on a cluster node

error decreased and stabilized at a cluster node for different values of k for HIGGS and Tweets, respectively. (Similar results were obtained for $k = 3$.)

In our evaluation, we empirically studied the impact of r on the average relative error of DiSC. For our cluster setup and all the tested datasets, the average relative error was under 6% by the end of the time budget for $r = 120$. So for the remainder of this section, unless necessary, we present the results achieved by DiSC for $r = 120$ as this setting yielded the best accuracy. Note that in a different cluster setup, one would need to choose r based on the desired accuracy/confidence and network bandwidth budget as increasing r increases the size of gossip messages in DiSC.



(a) HIGGS, $k = 1$ (b) HIGGS, $k = 2$

(c) Tweets, $k = 1$ (d) Tweets, $k = 2$

Fig. 8. HIGGS and Tweets: impact of r on the average relative error of the estimates computed by DiSC on a cluster node

Dataset	DiSC (hh:mm:ss)							MR-SS (no sampling)	
	Total time	$k = 1, r = 120$		$k = 2, r = 120$		$k = 3, r = 120$		(LZ4/Java)	(Snap./Kyro)
		First node	Last node	First node	Last node	First node	Last node	Total time (hh:mm:ss)	Total time (hh:mm:ss)
S_1	**0:17:00**	0:10:19	0:13:20	0:09:31	0:13:36	0:09:26	0:13:08	02:50:53	02:49:54
S_2	**0:17:00**	0:11:31	0:13:49	0:09:45	0:13:32	0:09:41	0:13:58	02:50:00	02:50:30
S_3	**0:17:00**	0:10:23	0:13:55	0:09:48	0:13:09	0:09:39	0:13:27	02:50:24	02:49:07
HIGGS	**0:15:00**	0:09:31	0:12:53	0:09:41	0:11:19	0:09:37	0:11:42	02:33:53	02:29:35
Tweets	**0:17:00**	0:09:14	0:12:50	0:08:49	0:12:00	0:08:44	0:12:50	02:50:38	02:47:31

Table 3. Total wall-clock time (hh:mm:ss) taken by DiSC and MR-SS (no sampling). The times when (in hh:mm:ss) the first one and the last one among the cluster nodes achieved below 10% average relative error are also reported.

4.7 DiSC vs MR-SS

Next, we compare DiSC and MR-SS in terms of computation time and accuracy to compute the sufficient statistics of a given set of families. For every dataset, 10,000 families were considered during the sufficient statistics computation. As discussed earlier, DiSC was executed by specifying a time budget after which gossiping was terminated. When MR-SS was executed on an entire dataset, it provided *exact sufficient statistics* – 100% accuracy. However, when it was executed on a sampled

dataset, it computed approximate sufficient statistics. DiSC provided only approximate sufficient statistics.

We begin with the case where MR-SS operated on the entire datasets (*i.e.*, no sampling). Table 3 shows the comparison of DiSC and MR-SS in terms of wall-clock time to compute the sufficient statistics of the families. For S_1, S_2, and S_3, DiSC took 17 minutes for the setting $r = 120$. However, MR-SS required 2 hrs 50 mins and was up to 10 times slower than DiSC. We tested MR-SS with LZ4 compression/Java serializer and Snappy compression/Kyro serializer. As shown, both configurations gave similar results. For HIGGS, DiSC required only 15 minutes for the setting $r = 120$. MR-SS finished in nearly 150 minutes and was 10 times slower than DiSC. Note that the HIGGS dataset had smaller number of data instances as compared to the synthetic datasets. For Tweets, DiSC required 17 minutes for the setting $r = 120$. MR-SS was nearly 10 times slower than DiSC. Clearly, DiSC was significantly faster than MR-SS for approximately computing sufficient statistics of families on all the five datasets. We also observed when the cluster nodes achieved below 10% average relative error (for the families they are responsible for) when executing DiSC. In Table 3, we report the time when the first one among the cluster nodes achieved below 10% average relative error. We also report the time when the last one among the cluster nodes achieved below 10% average relative error. This shows that different cluster nodes see improvement in the accuracy of their estimates of sufficient statistics over a period of time in a distributed setting. *It is fair to conclude that computing sufficient statistics approximately using DiSC is nearly 10 times faster (in terms of computation time) than MR-SS for learning a BN on large-scale distributed data.*

Let us closely analyze the accuracy of the sufficient statistics computed by DiSC. Table 4 shows the average relative error (%) achieved by each cluster node for the different datasets for varying values of k. Note that all the reported numbers in this table are for $r = 120$. We highlight two observations. First, for a given dataset and a particular value of k, every cluster node achieved similar average relative error on different subsets of families, and, therefore, similar accuracy in estimating the sufficient statistics. Second, the average relative errors were within 5.9% for the synthetic datasets, within 2.59% for HIGGS, and within 3.19% for Tweets. DiSC achieved very good accuracy in estimating the sufficient statistics for all the five datasets. *The above results demonstrate the robustness of our approach in estimating the sufficient statistics of families for datasets with different characteristics.*

Next, we report how MR-SS performed on random samples of the datasets. We chose 10%, 8%, and 4% of the data instances randomly in each dataset to create different sample sizes. Table 5 shows the wall-clock time taken and % average relative error achieved by MR-SS (computed over all the 10,000 families) to estimate the sufficient statistics on different sample sizes. The table also shows the results for MR-SS without any sampling. While the time taken by MR-SS significantly reduced when samples were processed, the accuracy degraded as smaller sample sizes were tested with. For comparison, we show a representative case of DiSC for $k = 1$ and $r = 120$. DiSC achieved the best accuracy (*i.e.*, lowest relative error) compared to MR-SS executed on the different samples sizes. For instance, DiSC achieved an average relative error of 2.62% for HIGGS. However, MR-SS achieved poorer accuracy than DiSC with a much higher average relative error of 9.51% for 10% sample size. Note that DiSC's average relative error reported in the table was also computed over all the 10,000 families for fair comparison. *Our evaluation indicates that although MR-SS ran faster on random samples of the tested datasets than on the entire datasets, it was still slower than DiSC and performed worse than DiSC in terms of accuracy.*

4.8 Convergence Speed of DiSC

We analyzed the convergence speed of DiSC empirically by computing the % average relative error on each cluster node over time for the tested datasets. (The average relative error was computed

Dataset	k	N_1 (%)	N_2 (%)	N_3 (%)	N_4 (%)	N_5 (%)	N_6 (%)	N_7 (%)	N_8 (%)	N_9 (%)	N_{10} (%)	N_{11} (%)	N_{12} (%)	N_{13} (%)	N_{14} (%)	N_{15} (%)	N_{16} (%)
S_1	1	4.25	4.01	4.35	4.20	4.14	3.76	**4.48**	4.04	4.42	3.98	4.20	4.07	4.10	3.91	3.97	4.20
	2	4.25	4.11	4.14	4.24	4.26	4.12	3.94	3.98	4.19	3.95	**4.31**	3.97	4.10	4.01	4.11	3.89
	3	4.12	4.12	4.21	3.94	4.00	3.87	4.03	4.19	4.09	3.97	**4.23**	4.05	3.99	3.88	4.14	3.98
S_2	1	5.12	**5.90**	4.94	5.18	5.30	5.56	4.89	5.17	5.43	5.32	5.73	5.44	5.28	5.40	5.54	5.29
	2	5.52	5.16	5.23	5.57	5.30	5.56	**5.76**	5.57	5.52	5.22	5.25	5.16	5.34	5.56	5.59	5.50
	3	5.50	5.60	5.51	5.59	5.27	5.01	**5.64**	5.11	5.51	5.23	5.39	5.27	5.27	5.56	5.28	5.31
S_3	1	4.15	3.92	4.28	4.27	4.14	3.82	4.40	4.08	4.38	4.54	**4.55**	4.43	4.01	4.13	4.20	4.21
	2	4.28	4.77	4.74	4.56	4.80	4.40	4.48	4.23	4.98	4.31	4.87	4.46	**5.04**	4.27	4.52	4.12
	3	4.04	3.97	4.29	4.04	4.34	4.12	4.10	4.26	4.20	4.26	4.19	4.10	4.05	4.09	**4.44**	4.36
HIGGS	1	2.39	2.21	2.17	2.20	2.19	2.53	2.24	2.39	2.44	2.10	2.10	**2.59**	2.14	2.42	2.33	2.17
	2	2.14	2.10	2.11	2.12	2.09	2.08	2.20	2.14	2.08	**2.27**	2.17	2.09	2.09	2.13	2.12	2.04
	3	**2.20**	2.12	2.12	2.08	2.12	2.13	2.16	2.08	2.12	2.12	2.10	2.11	2.12	2.13	2.05	2.11
Tweets	1	2.35	2.54	2.53	2.49	2.50	2.73	**3.19**	2.51	2.69	2.74	2.43	2.37	2.49	2.44	2.43	2.53
	2	2.39	2.32	2.39	2.35	2.31	2.34	2.39	2.37	2.36	2.36	2.4	2.42	2.34	**2.44**	2.38	2.34
	3	2.49	2.68	2.55	2.55	2.38	2.49	2.45	2.33	2.48	2.51	2.59	2.50	2.45	2.49	2.47	**2.69**

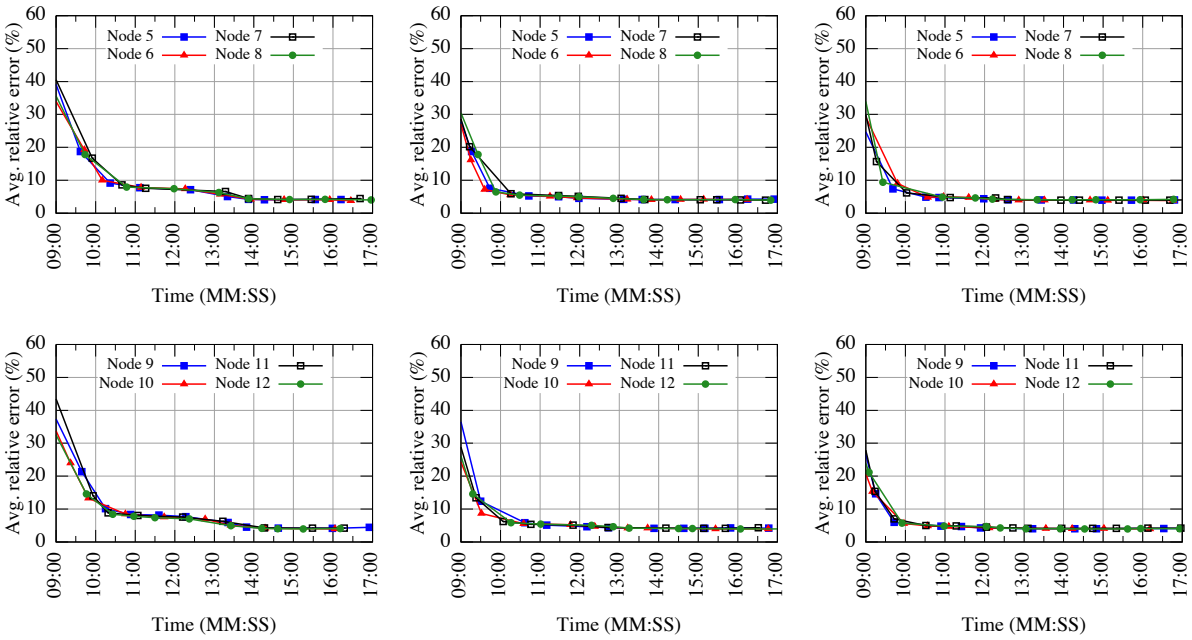
Table 4. Average relative error achieved (%) by DiSC on each cluster node N_1-N_{16} for varying values of k and $r = 120$. The maximum value is shown in bold.

Dataset	MR-SS								DiSC ($k = 1, r = 120$)	
	No sampling		10% sampling		8% sampling		4% sampling			
	Total time	Avg. rel. error	Total time	Avg. rel. error	Total time	Avg. rel. error	Total time	Avg. rel. error	Total time	Avg. rel. error
S_1	2:50:53	0%	0:51:43	9.78%	0:58:48	9.31%	0:38:46	11.80%	0:17:00	4.13%
S_2	2:50:00	0%	0:51:03	10.36%	0:50:54	10.65%	0:49:06	13.88%	0:17:00	5.34%
S_3	2:50:24	0%	0:50:41	9.79%	0:51:20	11.25%	0:49:53	13.02%	0:17:00	4.22%
HIGGS	2:33:53	0%	0:28:00	9.51%	0:27:52	9.52%	0:27:40	11.88%	0:15:00	2.62%
Tweets	2:50:38	0%	1:50:59	11.07%	1:56:15	10.58%	1:51:43	13.39%	0:17:00	2.56%

Table 5. Performance of MR-SS by on sampled datasets. The time taken (in hh:mm:ss) is shown along with % average relative error across all families.

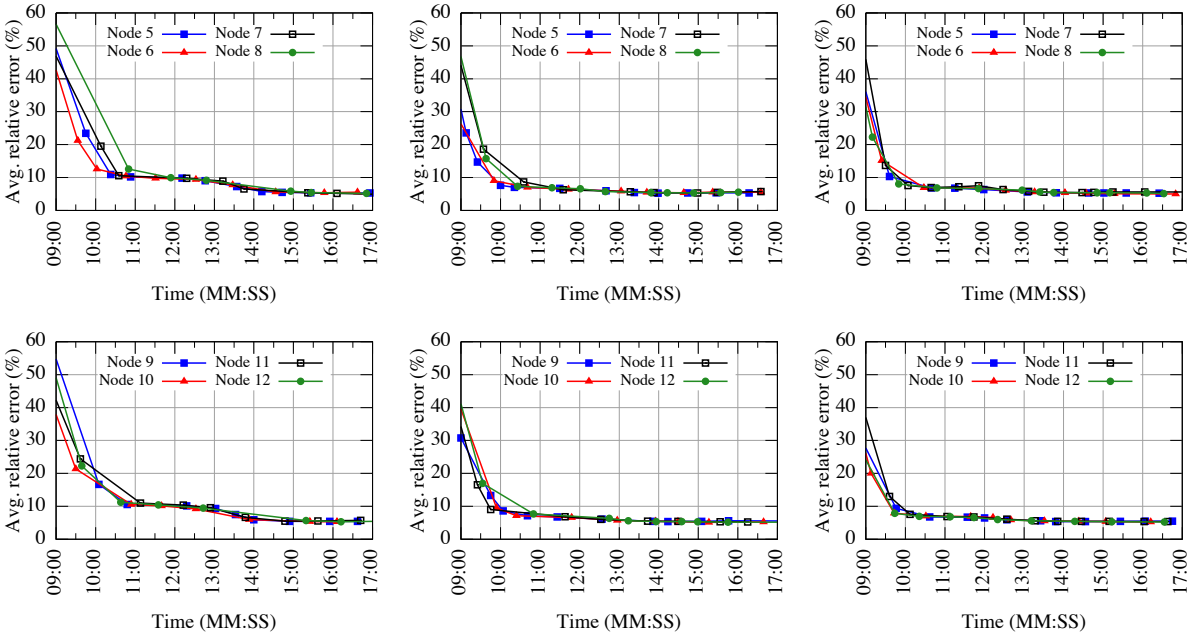
over the families that a cluster node was responsible for.) As gossiping progresses, the estimate of the sufficient statistics of a family tends to converge to a value that is close to but not exactly the true value. Hence, we expect the average relative error to decrease and eventually stabilize at a positive value by the end of the time budget on each cluster node. The accuracy will depend on DiSC's parameters such as r, the size of the cluster, time budget, and others. We show the results of 8 nodes, namely, N_5-N_{12} and refer the reader to the Appendix for the results of the remaining nodes. Figure 9 shows how the average relative error drops over time and stabilizes on nodes N_5-N_{12} for varying values of k. The average relative error is shown for different wall-clock times starting at the 9-minute mark. Overall, all nodes N_1-N_{16} produced accurate estimates of the sufficient statistics of families with low relative error by the end of the time budget, which was 17 minutes for S_1. Similar trends were observed for S_2 and S_3 as shown in Figures 10 and 11, respectively. The average relative errors were under 6% in all cases. In our implementation of DiSC, the cluster nodes initialized their local state (*i.e.*, SSAs of families) based on the local data blocks during the initial phase of execution. A node started exchanging messages once it finished computing the local state.

Figure 12 shows how the average relative error of the sufficient statistics estimates drops over time and stabilizes on a set of cluster nodes for HIGGS. The average relative error was within 2.59% in all cases by the end of the time budget. Finally, Figure 13 shows how the average relative error of



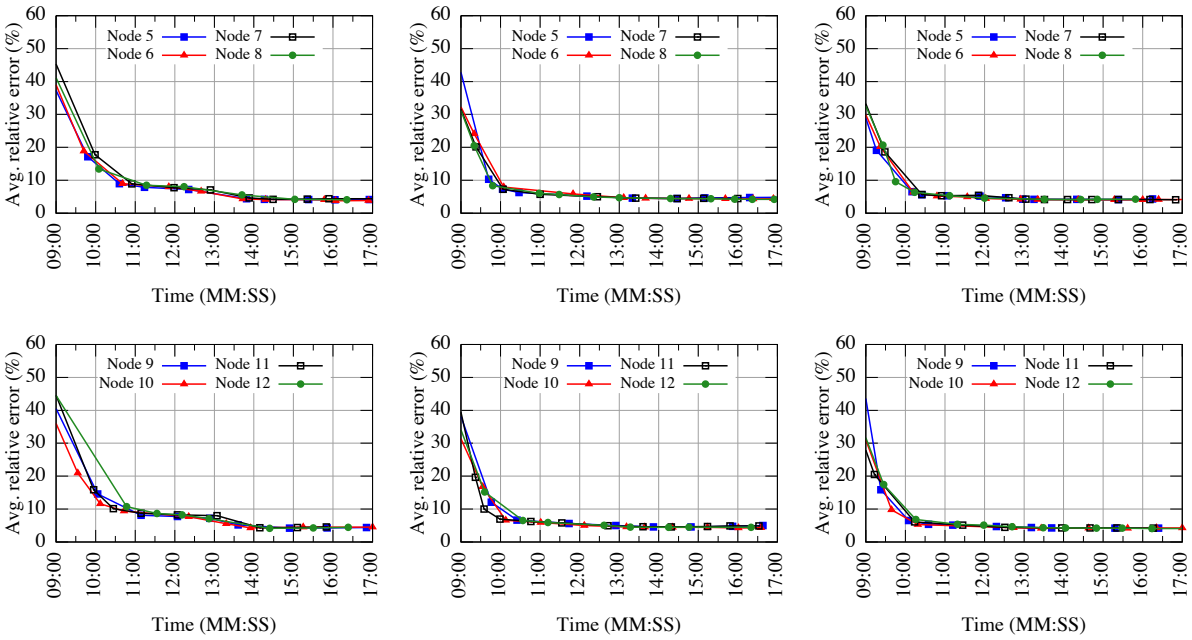
(a) $k = 1, r = 120$ (b) $k = 2, r = 120$ (c) $k = 3, r = 120$

Fig. 9. Dataset S_1: convergence speed of DiSC on cluster nodes N_5-N_{12} (200M data instances)



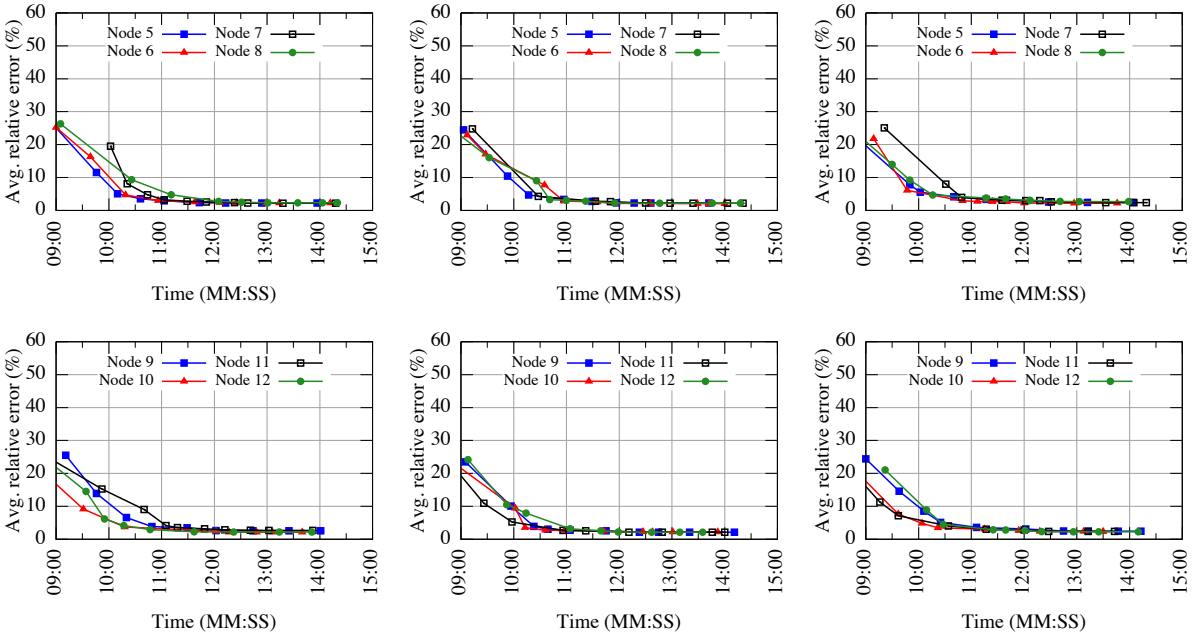
(a) $k = 1, r = 120$ (b) $k = 2, r = 120$ (c) $k = 3, r = 120$

Fig. 10. Dataset S_2: convergence speed of DiSC on cluster nodes N_5-N_{12} (200M data instances)



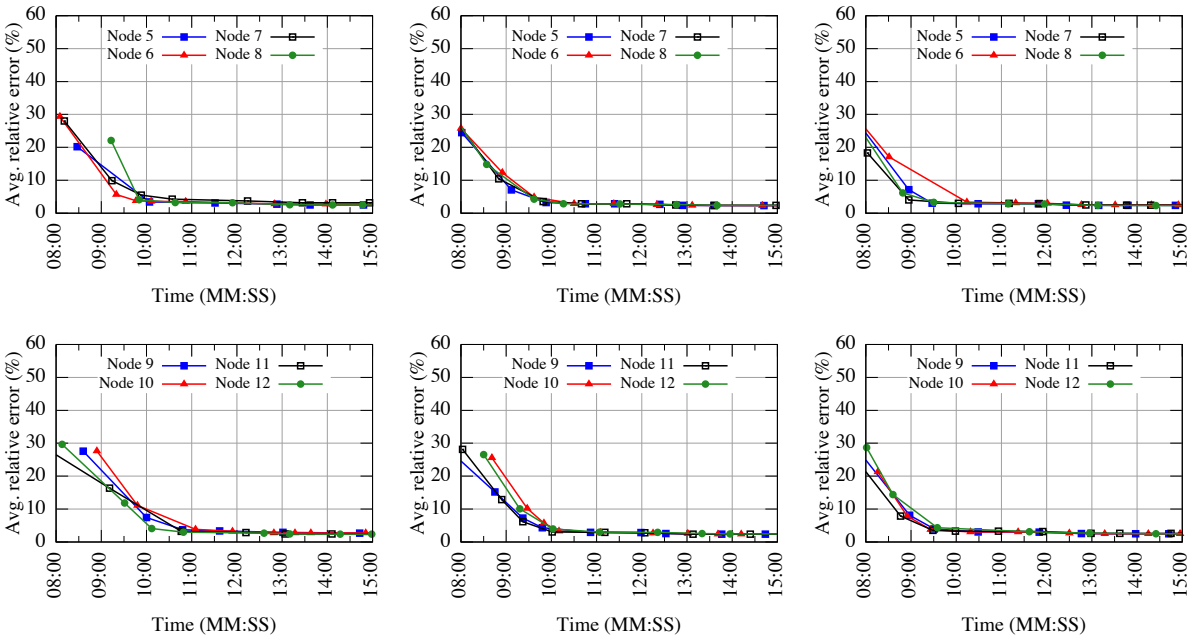
(a) $k = 1, r = 120$ (b) $k = 2, r = 120$ (c) $k = 3, r = 120$

Fig. 11. Dataset S_3: convergence speed of DiSC on cluster nodes N_5-N_{12} (200M data instances)



(a) $k = 1, r = 120$ (b) $k = 2, r = 120$ (c) $k = 3, r = 120$

Fig. 12. Dataset HIGGS: convergence speed of DiSC on cluster nodes N_5-N_{12} (176M data instances)



(a) $k = 1$, $r = 120$ (b) $k = 2$, $r = 120$ (c) $k = 3$, $r = 120$

Fig. 13. Dataset Tweets: convergence speed of DiSC on cluster nodes N_5-N_{12} (200M data instances)

estimates drops over time and stabilizes for Tweets. The average relative error was within 3.19% in all cases by the end of the time budget. *Overall, DiSC's convergence speed was fast, and it estimated the sufficient statistics of families with high accuracy.*

4.9 Communication Cost of DiSC

We report the communication cost of DiSC for the five datasets by varying the value of k and r. Table 6 reports the total number of messages sent by the cluster nodes during the execution of DiSC, % of messages lost during execution, and average message size. Note that these values were computed over the entire time budget for which DiSC was executed. That is, even though the average relative error on the nodes stabilized before the end of the time budget, the nodes still continued to gossip with other nodes till the end of the time budget. Each message sent by a node was compressed using Snappy compression and delivered through the network as a UDP (User Datagram Protocol) packet.

Let us first analyze the results for the synthetic datasets. As expected, the number of messages increased as r was increased from 40 through 120 for every dataset due to increase in the total amount of data to transmit the exponential random variables. Also, as k was increased, the total number of messages exchanged also increased. Similar to the synthetic datasets, for HIGGS and Tweets, the total number of messages sent by the cluster nodes increased as r was increased from 40 to 120. Also, the number of messages increased with increase in k. This is a cost to pay for achieving increased redundancy for fault-tolerance. One may wonder if $k > 1$ is too much redundancy for DiSC in the given setup on CloudLab where there were no node failures during execution. While this is true, our goal was to test the robustness of DiSC as k was increased and gain insights on how k affected the performance of DiSC.

Snappy compression provided significant benefit to DiSC in all cases. The average compression ratio for the messages was 40.6%, 27.5%, and 40.5% for S_1, S_2, and S_3, respectively. For HIGGS and

Dataset	k	r = 40			r = 80			r = 120		
		# of messages sent	% of messages lost	Avg. message size (bytes)	# of messages sent	% of messages lost	Avg. message size (bytes)	# of messages sent	% of messages lost	Avg. message size (bytes)
S_1	1	297,311	1.81%	36,974.02	589,576	3.44%	36,749.17	888,316	4.04%	36,005.52
	2	315,747	2.47%	37,106.94	678,801	2.85%	36,841.08	996,140	3.98%	36,224.52
	3	332,108	2.01%	37,224.08	763,833	3.45%	36,891.64	1,059,339	4.13%	36,401.34
S_2	1	278,438	2.30%	44,836.29	549,797	3.47%	44,713.73	816,135	3.83%	43,795.38
	2	312,434	2.82%	44,979.14	632,803	4.20%	44,782.18	954,651	4.17%	44,139.51
	3	364,044	2.48%	45,082.57	674,835	4.32%	44,839.84	1,107,472	4.84%	44,240.76
S_3	1	276,018	2.60%	36,997.95	585,358	3.15%	36,787.08	823,924	3.47%	35,972.28
	2	317,732	2.29%	37,182.32	659,178	2.74%	36,904.41	988,215	4.36%	36,295.86
	3	344,609	2.38%	37,274.98	692,584	2.99%	36,974.82	1,031,404	4.23%	36,497.82
HIGGS	1	191,336	1.93%	39,254.09	366,314	4.06%	38,936.50	610,607	5.20%	38,645.70
	2	203,013	2.16%	39,523.69	435,332	4.19%	39,295.28	675,949	4.99%	38,499.67
	3	219,964	1.72%	39,632.80	434,236	4.22%	39,478.74	736,433	5.60%	38,468.32
Tweets	1	134,178	1.61%	48,163.67	284,473	1.38%	48,331.03	420,951	3.06%	47,707.04
	2	147,556	1.77%	48,423.50	309,644	2.78%	48,543.28	468,153	2.67%	47,950.37
	3	148,774	1.33%	48,548.52	308,988	2.93%	48,733.34	481,049	2.69%	48,050.40

Table 6. Total number of messages sent during the execution of DiSC. Each message was compressed and sent through the network as a UDP packet.

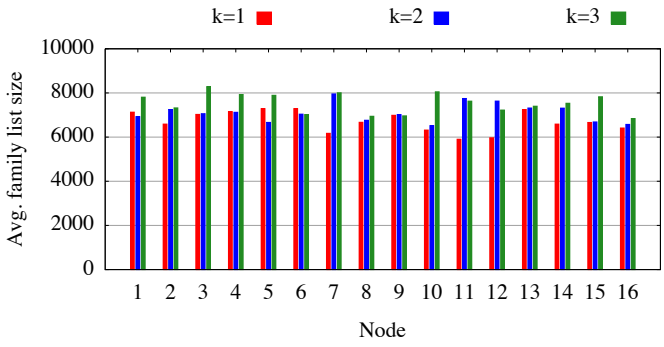


Fig. 14. Dataset S_1: reduction in the size of the family list at the nodes due to dropping of families in DiSC (r = 120)

Tweets, the average compression ratio was 64.8% and 61.9%, respectively. We attribute a better compression ratio for HIGGS and Tweets due to the difference in the distribution of sufficient statistics values for the families between them and the synthetic datasets. Overall, compression provided significant reduction in the communication cost for DiSC. In hindsight, we could have modified DiSC's implementation to pack as many SSAs as possible (after compression) in a UDP packet. This would have reduced the number of messages exchanged.

Table 6 also reports the percentage of messages lost during the execution of DiSC. DiSC was able to cope with lost messages while achieving high accuracy due to the inherent ability of gossip algorithms to tolerate failures.

4.10 Impact of Probabilistically Dropping Families

Next, we study the impact of dropping families probabilistically in DiSC. As stated in Section 3.2, the basic approach will result in all the cluster nodes learning all the families under consideration. Thus, the size of the family list on each node will reach 10,000 for the synthetic datasets, HIGGS, and Tweets.

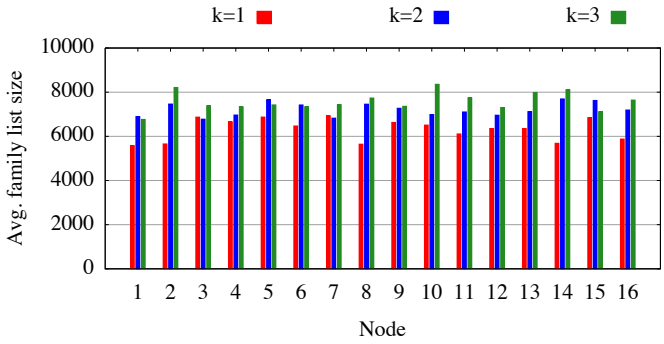


Fig. 15. Dataset S_2: reduction in the size of the family list at the nodes due to dropping of families in DiSC ($r = 120$)

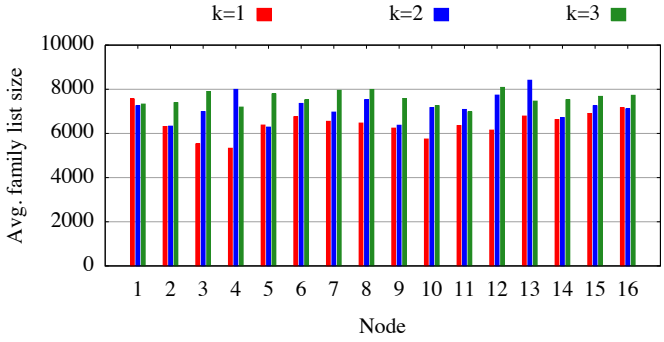


Fig. 16. Dataset S_3: reduction in the size of the family list at the nodes due to dropping of families in DiSC ($r = 120$)

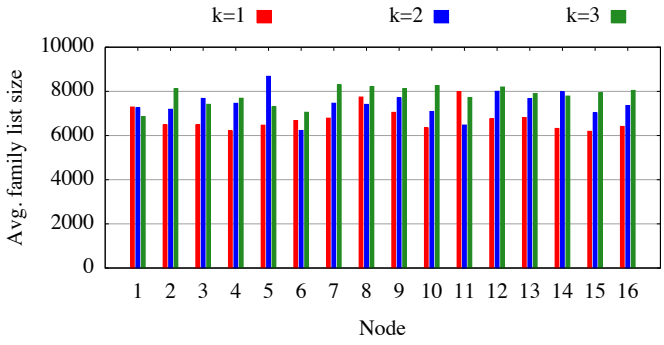


Fig. 17. Dataset HIGGS: reduction in the size of the family list at the nodes due to dropping of families in DiSC ($r = 120$)

The improved approach in DiSC, however, drops families probabilistically to control the size of the family lists on each node. This is because the size of the family list dictates the amount of data exchanged during gossip, and DiSC aims to lower the network bandwidth consumption. In our implementation, during gossip, we dropped a family f from the family list of a node N_i (assuming that N_i is not responsible for f) with probability 0.8 if the neighbor N_j selected to send a gossip message is responsible for f. Otherwise, we dropped with a lower probability of 0.4.

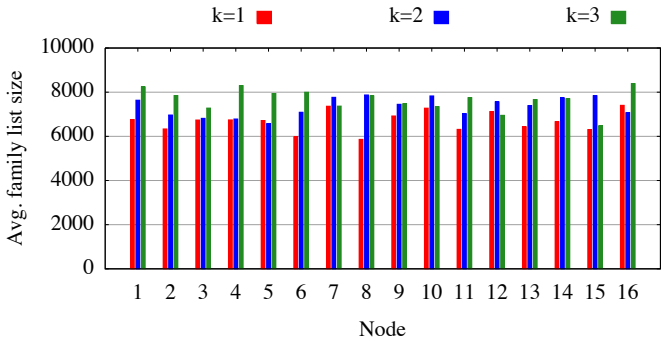


Fig. 18. Dataset Tweets: reduction in the size of the family list at the nodes due to dropping of families in DiSC ($r = 120$).

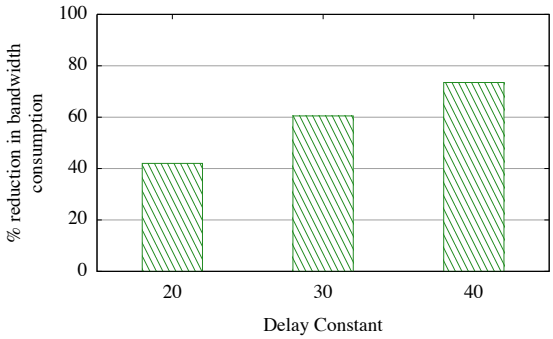


Fig. 19. Reduction in bandwidth due to increase in the delay constant. The results are for the dataset S_2 with $r = 120$ and $k = 2$.

Figures 14, 15,and 16 show the average family size on each cluster node achieved during the execution of DiSC for different values of k on the datasets S_1, S_2, and S_3, respectively. Figures 17 and 18 show the average family size results on each cluster node for different values of k on HIGGS and Tweets, respectively. As expected, with least redundancy, *i.e.*, $k = 1$, the size of the family list at each cluster node tends to be the lowest. Clearly, DiSC's ability to drop families probabilistically significantly reduced the size of the family lists on all the cluster nodes compared to what a basic approach would have achieved.

For all the experiments reported so far, we chose a delay constant $c = 10$ for the local clock as shown in Algorithm 8. We decided to slow down the local clock and increase the time interval between clock ticks by increasing c. As expected, this resulted in lower number of messages sent during the execution of DiSC, thereby leading to reduction in network bandwidth consumption. Figure 19 shows the % reduction in bandwidth consumption of DiSC with increase in the delay constant to 20, 30, and 40 for a representative case. (The results are for the dataset S_2 with $r = 120$ and $k = 2$.) For $c = 20$ and $c = 30$, all the cluster nodes converged to under 10% average relative error in the given time budget of 18 minutes. However, for $c = 40$, this was not the case for the same time budget. Thus, tuning the delay constant is another way to lower the network bandwidth consumption of DiSC.

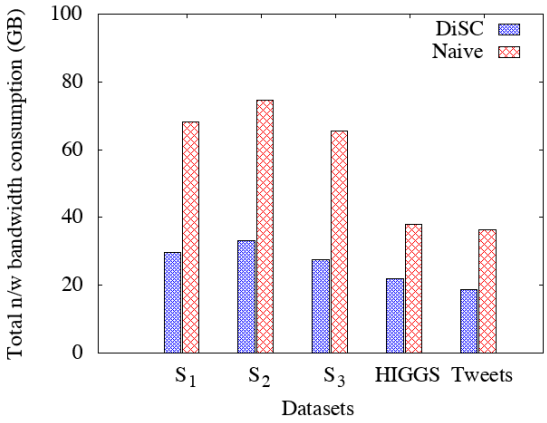


Fig. 20. Comparison of a naive approach and DiSC based on total network bandwidth consumption. We report the results on the five datasets for $k = 1$ and $r = 120$.

4.11 Benefit of Distributing Families Across Cluster Nodes in DiSC

One may wonder how DiSC benefits by distributing the families across cluster nodes and dividing the work among cluster nodes to compute the sufficient statistics of only a subset of families. To understand this, we compared DiSC with a *naive approach* of gossiping to compute the sufficient statistics of families. In the naive approach, we assigned all the 10,000 families to each cluster node at the beginning. At each gossip round, two nodes exchanged the exponential random variables for the SSA of every family. No families were dropped probabilistically. Hence, the size of the family list at each node remained at 10,000. Note that the naive approach also uses gossiping and is decentralized.

We expect the naive approach to increase the communication cost significantly. This was precisely observed in our experiments as shown in Figure 20. The communication cost of the naive approach was between 1.72 to 2.37 times higher than that of DiSC. Thus, DiSC's approach of distributing the families among cluster nodes for load balancing and distributed processing of sufficient statistics of families was superior than the naive approach. In fact, as the total number of families increases, the naive approach must be run with a much larger delay constant to allow the exchange of SSAs of all the families during a gossip round leading to slower convergence of the estimates.

4.12 Summary of Performance Evaluation

Below we summarize the key observations of our performance evaluation.

- DiSC provides a feasible tradeoff between computation time and accuracy for fast score computation on large-scale distributed data. Although it computes approximate sufficient statistics of families, it was nearly 10 times faster than MR-SS, which is based on MapReduce-style computation and computed exact sufficient statistics. Although random sampling of the tested datasets enabled MR-SS to run faster than on the entire datasets, it was still slower than DiSC and performed worse than DiSC in terms of accuracy.
- DiSC's decentralized, gossip-based computation of sufficient statistics provides a robust approach for computing sufficient statistics of families and achieves very high accuracy (less than 6% average relative error) on datasets with different characteristics. DiSC can gracefully tolerate loss of messages during execution.
- DiSC's approach of probabilistically dropping families provides significant benefit in reducing the size of the family lists during execution, thereby reducing the communication cost of DiSC.

In addition, a naive approach of maintaining all families at the cluster nodes and gossiping them without carefully dividing the work among these nodes significantly increases the communication cost, and thereby validates DiSC's effective design.

5 CONCLUSIONS

Score computation is a fundamental task during structure learning of a multinomial BN. In this paper, we presented an efficient approach called DiSC for fast approximate score computation on large-scale distributed datasets stored in a cluster. DiSC's novelty is based on the following: (a) a decentralized algorithm for scalable score computation using the principle of gossiping, (b) properties of Markov chains and a probabilistic approach to lower resource consumption, and (c) consistent hashing and LSH for effective distribution of tasks for score computation on large datasets. We presented the theoretical analysis of DiSC in terms of convergence speed (for a given accuracy and confidence bound) of the sufficient statistics, and memory and network bandwidth consumption. We also discussed how DiSC is capable of efficiently recomputing scores when new data are available. We conducted comprehensive evaluation of DiSC and MR-SS on datasets with different characteristics using a 16-node cluster. When MR-SS provided exact sufficient statistics of families, it was nearly 10 times slower than DiSC. Although it ran faster on randomly sampled datasets than on the entire datasets, it performed worse than DiSC in terms of accuracy. DiSC achieved high accuracy (below 6% average relative error) in estimating the sufficient statistics of families on all the tested datasets. Thus, DiSC provides a feasible tradeoff between computation time and accuracy for fast approximate score computation on large-scale distributed data. The code and datasets are available at https://github.com/UMKC-BigDataLab/DiSC.

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their insightful comments. Part of this work was performed while the second author (P. R.) held an NRC Research Associateship award at Air Force Research Lab, Rome, New York. He would like to acknowledge the support of the U.S. Air Force Summer Faculty Fellowship Program and the University of Missouri Research Board, and the partial support of the National Science Foundation Grant No. 1747751. The first author (A. K.) would like to acknowledge the support of King Abdullah Scholarship Program (Saudi Arabia).

REFERENCES

[1] Java-gossip. https://code.google.com/archive/p/java-gossip/.
[2] Gossip Algorithms. *Foundations and Trends in Networking*, 3(1):1–125, 2008.
[3] CloudLab. https://www.cloudlab.us/, 2017.
[4] Kyro. https://github.com/EsotericSoftware/kryo, 2017.
[5] LZ4 - Extremely Fast Compression. https://github.com/lz4/lz4, 2017.
[6] Snappy, a Fast Compressor/Decompressor. https://github.com/google/snappy, 2017.
[7] UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets.html, 2017.
[8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, 2016.
[9] J. Arias, J. A. Gamez, and J. M. Puerta. Learning distributed discrete Bayesian Network Classifiers under MapReduce with Apache Spark. *Knowledge-Based Systems*, 117:16 – 26, 2017.
[10] P. Baldi, P. Sadowski, and D. Whiteson. Searching for Exotic Particles in High-Energy Physics with Deep Learning. *Nature Commun.*, 5:4308, 2014.
[11] A. Basak, I. Brinster, X. Ma, and O. Mengshoel. Accelerating Bayesian Network Parameter Learning using Hadoop and MapReduce. In *Proc. of 2012 BigMine Workshop*, pages 1–8, 2012.
[12] 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,

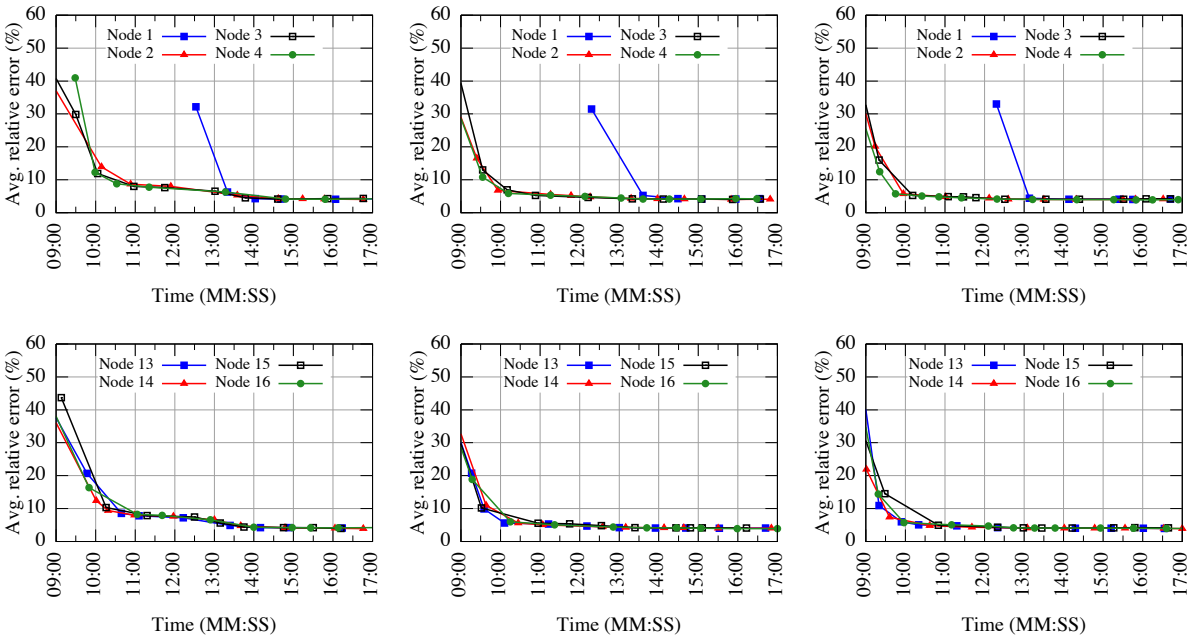
Sept. 2016.
[13] S. P. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip Algorithms: Design, Analysis and Applications. In *Proc. of INFOCOM 2005*, pages 1653–1664, 2005.
[14] A. M. Carvalho. Scoring Functions for Learning Bayesian Networks. Technical report, IST, TULisbon/INESC-ID Tech. Report 54/2009, Apr. 2009.
[15] W. Chen, T. Wang, D. Yang, K. Lei, and Y. Liu. Massively Parallel Learning of Bayesian Networks with MapReduce for Factor Relationship Analysis. In *Proc. of Intl. Joint Conf. on Neural Networks*, pages 1–5, 2013.
[16] D. Chickering. Learning from Data: Artificial Intelligence and Statistics V. chapter Learning Bayesian Networks is NP-Complete, pages 121–130. 1996.
[17] G. F. Cooper, I. Bahar, M. J. Becich, P. V. Benos, J. Berg, J. U. Espino, C. Glymour, R. C. Jacobson, M. Kienholz, A. V. Lee, X. Lu, and R. Scheines. The Center for Causal Discovery of Biomedical Knowledge from Big Data. *Journal of the American Medical Informatics Association*, 22(6):1132–1136, 2015.
[18] N. R. Council. *Frontiers in Massive Data Analysis*. The National Academies Press, Washington, DC, 2013.
[19] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. of the 6th OSDI Conference*, pages 137–150, 2004.
[20] 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.
[21] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. In *Proc. of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12, 1987.
[22] Q. Fang, K. Yue, X. Fu, H. Wu, and W. Liu. A MapReduce-based Method for Learning Bayesian Network from Massive Data. In *Proc. of 2013 APWeb Conference*, pages 697–708, 2013.
[23] A. Flink. https://flink.apache.org, 2017.
[24] C. Georgiou, S. Gilbert, R. Guerraoui, and D. Kowalski. On the Complexity of Asynchronous Gossip. In *Proc. of the 27th ACM Symposium on Principles of Distributed Computing*, pages 135–144, Toronto, Canada, 2008.
[25] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan. SystemML: Declarative Machine Learning on MapReduce. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 231–242, 2011.
[26] D. Grossman and P. Domingos. Learning Bayesian Network Classifiers by Maximizing Conditional Likelihood. In *Proc. of the 21st International Conference on Machine Learning*, pages 46–54, 2004.
[27] 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.
[28] K. A. Heller and Z. Ghahramani. Bayesian Hierarchical Clustering. In *Proc. of the 22nd International Conference on Machine Learning*, pages 297–304, Bonn, Germany, 2005.
[29] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar. The MADlib Analytics Library: Or MAD Skills, the SQL. *Proc. VLDB Endow.*, 5(12):1700–1711, Aug. 2012.
[30] Hyperledger. http://hyperledger-fabric.readthedocs.io/en/release-1.0/gossip.html, 2017.
[31] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the 13th ACM Symposium on Theory of Computing*, pages 604–613, 1998.
[32] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-Based Aggregation in Large Dynamic Networks. *ACM Transactions on Computer Systems*, 23:219–252, August 2005.
[33] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized Rumor Spreading. In *IEEE Symposium on Foundations of Computer Science*, pages 565–574, 2000.
[34] S. Kashyap, S. Deb, K. Naidu, R. Rastogi, and A. Srinivasan. Efficient Gossip-Based Aggregate Computation. In *Proc. of the 35th ACM Principles of Database Systems*, Chicago, IL, 2006.
[35] D. Kempe, A. Dobra, and J. Gehrke. Gossip-Based Computation of Aggregate Information. In *Proc. of the 44th IEEE Symposium on Foundations of Computer Science*, pages 482–491, Oct 2003.
[36] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.
[37] E. J. Kontoghiorghes. *Handbook of Parallel Computing and Statistics*. Chapman & Hall/CRC, 2005.
[38] A. Lakshman and P. Malik. Cassandra: A Structured Storage System on a P2P network. In *Proc. of the 21st Symposium on Parallelism in Algorithms and Architectures*, page 47, Alberta, Canada, 2009.
[39] K. Li, D. Z. Wang, A. Dobra, and C. Dudley. UDA-GIST: An In-database Framework to Unify Data-parallel and State-parallel Analytics. *Proc. VLDB Endow.*, 8(5):557–568, Jan. 2015.
[40] 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.

[41] K. W. Lim, C. Chen, and W. Buntine. Twitter-Network Topic Model: A Full Bayesian Treatment for Social Network and Text Modeling. In *NIPS 2013 Topic Model Workshop*, pages 1–5, Australia, 2013.
[42] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. Hellerstein. GraphLab: A New Framework for Parallel Machine Learning. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, UAI'10, pages 340–349, Catalina Island, CA, 2010.
[43] 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.
[44] A. R. Masegosa, A. M. Martínez, D. Ramos-López, R. Cabañas, A. Salmerón, T. D. Nielsen, H. Langseth, and A. L. Madsen. AMIDST: a Java Toolbox for Scalable Probabilistic Machine Learning. *CoRR*, abs/1704.01427, 2017.
[45] P. McQuighan. Simulating the Poisson Process. http://www.math.uchicago.edu/~may/VIGRE/VIGRE2010/REUPapers/Mcquighan.pdf, 2010.
[46] 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. *Journal of Machine Learning Research*, 17(34):1–7, 2016.
[47] S. Misra, V. Md., K. Pamnany, S. P. Chockalingam, Y. Dong, M. Xie, M. R. Aluru, and S. Aluru. Parallel bayesian network structure learning for genome-scale gene networks. In *Proc. of the Intl. Conference for High Performance Computing, Networking, Storage and Analysis*, pages 461–472, 2014.
[48] D. Mosk-Aoyama and D. Shah. Fast Distributed Algorithms for Computing Separable Functions. *IEEE Transactions on Information Theory*, 54(7):2997–3007, 2008.
[49] O. Nikolova and S. Aluru. Parallel Bayesian Network Structure Learning with Application to Gene Networks. In *Proc. of Intl. Conf. for High Performance Computing, Networking, Storage and Analysis*, pages 1–9, 2012.
[50] J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.
[51] J. Podesta, P. Pritzker, E. Moniz, J. Holdren, and J. Zients. Big Data: Seizing Opportunities, Preserving Values. http://www.whitehouse.gov/sites/default/files/docs/big_data_privacy_report_5.1.14_final_print.pdf, 2014.
[52] P. Rao, A. Katib, K. Barnard, C. Kamhoua, K. Kwiat, and L. Njilla. Scalable Score Computation for Learning Multinomial Bayesian Networks over Distributed Data. In *Proc. of the 2017 AAAI Workshop on Distributed Machine Learning (DML)*, pages 498–504, San Francisco, CA, 2017.
[53] S. Serbu, E. Rivière, and P. Felber. Network-Friendly Gossiping. In *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, SSS '09, pages 655–669, Lyon, France, 2009.
[54] V. Slavov, A. Katib, and P. Rao. A Tool for Internet-Scale Cardinality Estimation of XPath Queries Over Distributed Semistructured Data. In *Proc. of the 30th IEEE International Conference on Data Engineering*, pages 1270–1273, Chicago, USA, 2014.
[55] V. Slavov and P. Rao. Towards Internet-Scale Cardinality Estimation of XPath Queries Over Distributed XML Data. In *Proc. of the 6th International Workshop on Networking Meets Databases*, pages 1–8, Athens, Greece, 2011.
[56] V. Slavov and P. R. Rao. A gossip-based approach for Internet-Scale cardinality estimation of XPath queries over distributed semistructured data. *The VLDB Journal*, 23(1):51–76, 2014.
[57] SMILE-WIDE. http://smilewide.github.io/main, 2014.
[58] A. Spark. https://spark.apache.org, 2017.
[59] 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.
[60] D. Tran, A. Kucukelbir, A. B. Dieng, M. Rudolph, D. Liang, and D. M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.
[61] W. Wei, K. Joseph, W. Lo, and K. Carley. A Bayesian Graphical Model to Discover Latent Events from Twitter. In *Proc. of the 9th International AAAI Conference on Web and Social Media*, 2015.
[62] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.
[63] 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 *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1335–1344, Sydney, Australia, 2015.
[64] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In *Proc. of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, pages 10–10, Boston, MA, 2010.
[65] Y. Zhao, J. Xu, and Y. Gao. A Parallel Algorithm for Bayesian Network Parameter Learning Based on Factor Graph. In *Proc. of IEEE Intl. Conf. on Tools with Artificial Intelligence*, pages 506–511, 2013.

APPENDIX

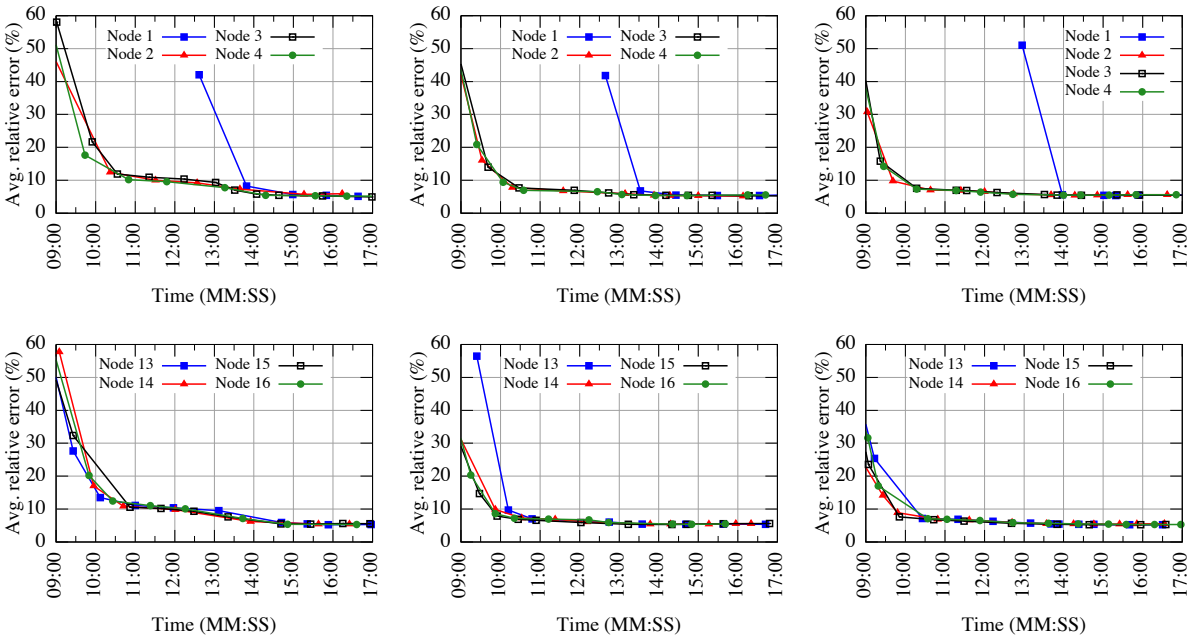
In this section, we show the convergence speed of DiSC on nodes N_1-N_4 and N_{13}-N_{16}. Figure 21 shows the convergence speed of DiSC on S_1. Figure 22 shows the convergence speed of DiSC on S_2. Figure 23 shows the convergence speed of DiSC on S_3. Figure 24 shows the convergence speed of DiSC on HIGGS. Finally, Figure 25 shows the convergence speed of DiSC on Tweets.

Note that N_1 was configured as the controller node in the cluster and ran slower than the others. Hence, we notice that the convergence on N_1 (blue line) starts later than the other nodes. Recall that the nodes initially compute the local state (*i.e.*, SSAs of families) based on local data blocks before beginning the gossip phase.



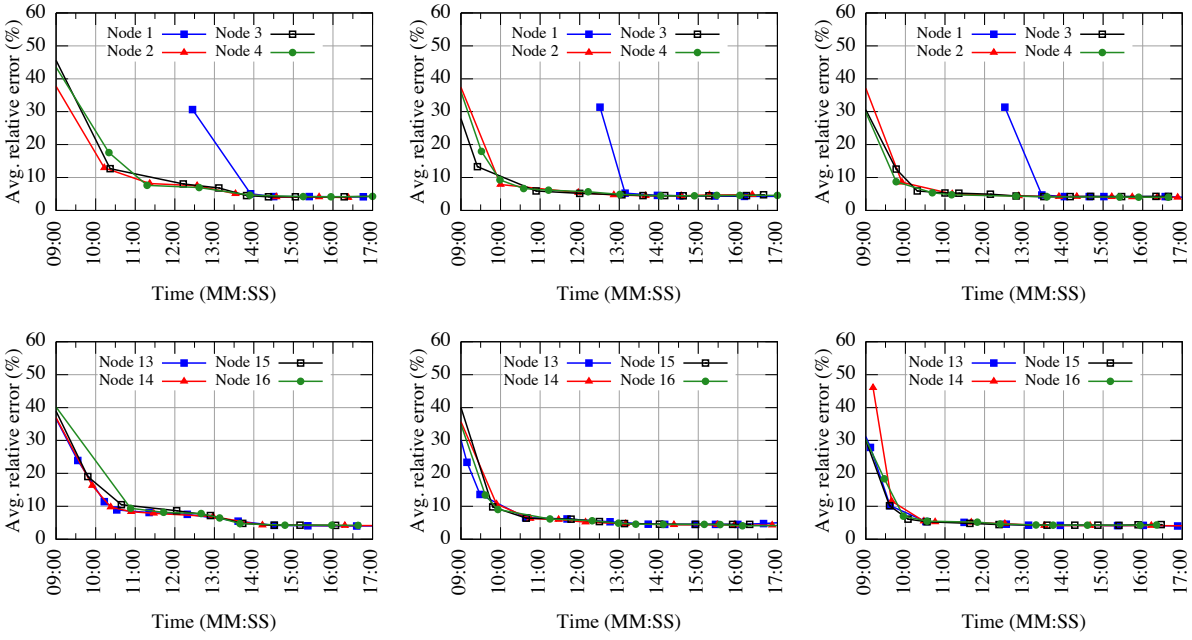
(a) $k = 1$, $r = 120$ (b) $k = 2$, $r = 120$ (c) $k = 3$, $r = 120$

Fig. 21. Dataset S_1: convergence speed of DiSC on cluster nodes N_1-N_4 and N_{13}-N_{16} (200M data instances)



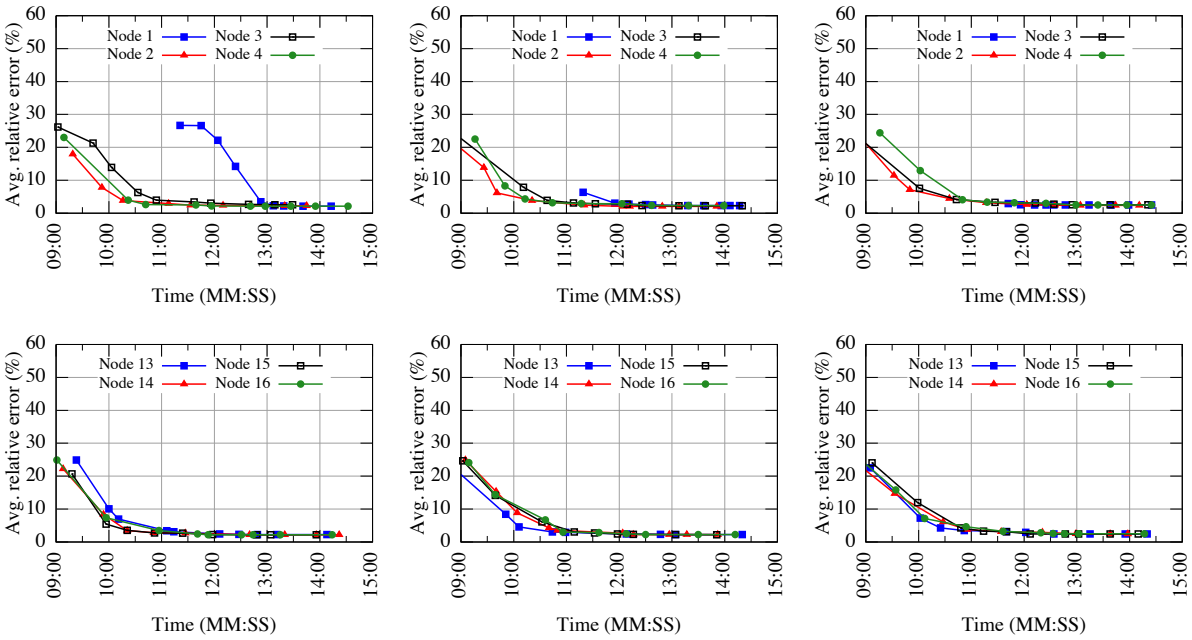
(a) $k = 1$, $r = 120$ (b) $k = 2$, $r = 120$ (c) $k = 3$, $r = 120$

Fig. 22. Dataset S_2: convergence speed of DiSC on cluster nodes N_1-N_4 and N_{13}-N_{16} (200M data instances)



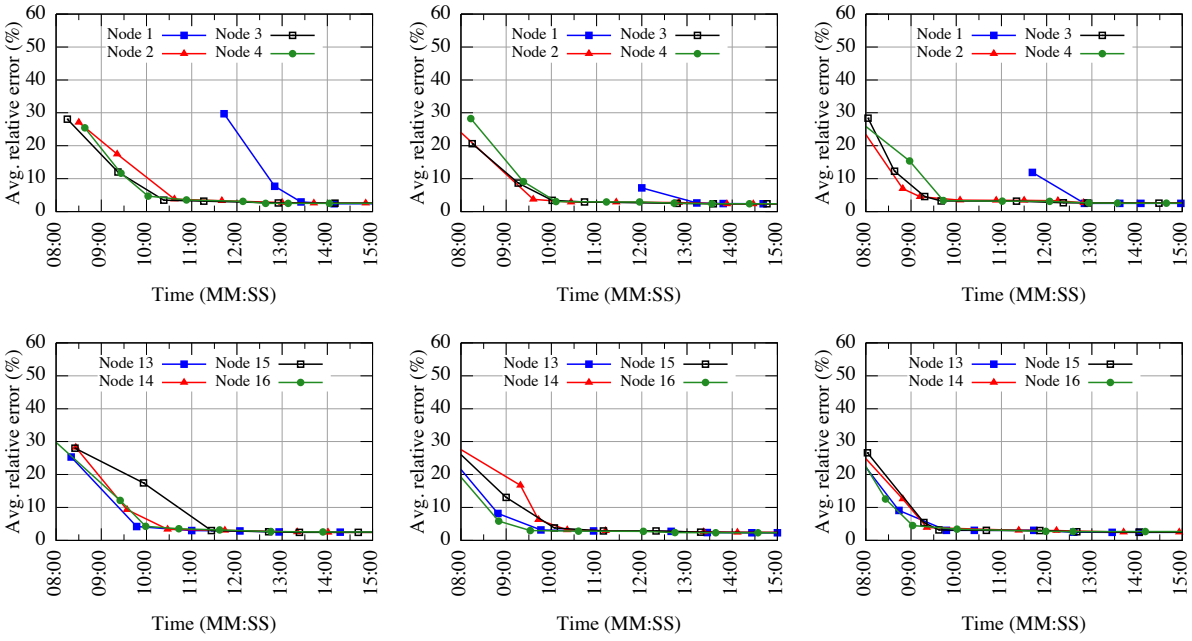
(a) $k = 1$, $r = 120$ (b) $k = 2$, $r = 120$ (c) $k = 3$, $r = 120$

Fig. 23. Dataset S_3: convergence speed of DiSC on cluster nodes N_1-N_4 and N_{13}-N_{16} (200M data instances)



(a) $k = 1$, $r = 120$ (b) $k = 2$, $r = 120$ (c) $k = 3$, $r = 120$

Fig. 24. Dataset HIGGS: convergence speed of DiSC on cluster nodes N_1-N_4 and N_{13}-N_{16} (176M data instances)



(a) $k = 1$, $r = 200$ (b) $k = 2$, $r = 200$ (c) $k = 3$, $r = 200$

Fig. 25. Dataset Tweets: convergence speed of DiSC on cluster nodes N_1-N_4 and N_{13}-N_{16} (200M data instances)