Distributed Identification of Top-l Inner Product Elements and its Application in a Peer-to-Peer Network

Kamalika Das, Kanishka Bhaduri, Kun Liu, and Hillol Kargupta *Senior Member, IEEE*

***Abstract*— Inner product measures how closely two feature vectors are related. It is an important primitive for many popular data mining tasks, e.g., clustering, classification, correlation computation, and decision tree construction. If the entire data set is available at a single site, then computing the inner product matrix and identifying the top (in terms of magnitude) entries is trivial. However, in many real-world scenarios, data is distributed across many locations and transmitting the data to a central server would be quite communication-intensive and not scalable. This paper presents an approximate *local* algorithm for identifying top-l inner products among pairs of feature vectors in a large asynchronous distributed environment such as a peer-to-peer (P2P) network. We develop a probabilistic algorithm for this purpose using order statistics and Hoeffding bound. We present experimental results to show the effectiveness and scalability of the algorithm. Finally, we demonstrate an application of this technique for interest-based community formation in a P2P environment.**

***Index Terms*— distributed data mining, inner product, peer-to-peer network**

I. Introduction

THE inner product between two vectors measures how similar or close they are to each other. It is a very important primitive for many data mining tasks such as clustering, classification, correlation computation and decision tree construction [1][2]. In many application scenarios, it is often desirable to know only the top inner products. For example, consider the formation of interest-based online communities in a peer-to-peer (P2P) environment [3]. P2P networks are large, dynamic, asynchronous, and with little central control. It is very difficult, if not impossible, to transfer all the data to a single peer to do the computation since no one would have such extensive storage and computational capabilities, let alone the enormous communication overhead. In the online community formation example, each peer may be associated with a feature vector describing its web surfing patterns and the goal is to find peers having similar interest (browsing patterns). This helps in routing queries to peers with relevant interests resulting in better network-search results. In most cases, each peer may be interested in finding only a few peers with similar interest and not all of them. Many other applications such as network intrusion detection over data streams [4], query routing in sensor networks, efficient decision tree construction in distributed environment demonstrate the same needs. If the entire data can be conveniently accessed, it is easy to compute the inner product matrix and determine the top ones. However, much of the world's data is distributed over a multitude of systems connected by communications channels of varying capacity. This calls for new techniques to perform data mining in a distributed environment.

In this paper, we consider the problem of identifying the global top-l inner products (attribute-wise) from distributed data. We assume that data is scattered among a large number of peers such that each peer has exactly the same set of attributes (or features). In the data mining literature, this is often referred to as horizontally partitioned (homogeneously distributed) data scenario. We propose an order statistics-based approximate *local* algorithm for solving the problem. Here the local algorithm is one where a peer communicates only with its neighbors (formal definition given later). At the heart of our algorithm are the ordinal approximation based on theories from order statistics [5] and the cardinal approximation using Hoeffding bound [6]. To the best of our knowledge, there does not exist any algorithm in the literature that can do a global ranking in a distributed setting without global communication of all the data. Our experimental results demonstrate that the algorithm achieves very high accuracy with only a small fraction of the communication required for data centralization.

The rest of the paper is organized as follows. Section II overviews some related work. Section III introduces the notations, problem definition and a brief overview of the algorithm. Section IV and V present the details of the algorithm. Section VI gives the definition of the *local* algorithm and proves that our algorithm is indeed *local*. Section VII studies the accuracy and communication complexity of the algorithm, while Section VIII presents the experimental results. Section IX demonstrates an application of our technique, i.e., client-side web mining for community formation in a P2P setting. Section X compares this work to other existing distributed inner product computation algorithms. Finally, Section XI concludes this paper.

II. Related Work

Distributed data mining deals with the problem of data analysis in environments with distributed data, computing

Manuscript received ...; revised ....

Kamalika Das, Kanishka Bhaduri and Hillol Kargupta are with the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, Baltimore, MD 21250. E-mail:{kdas1, kanishk1, hillol}@cs.umbc.edu. Kun Liu is with the IBM Almaden Research Center, San Jose, CA 95120. This work is partially done when Kun is at IBM Almaden. Email: kun@us.ibm.com. Hillol Kargupta is also affiliated with AGNIK LLC, Columbia, MD 21045.

nodes, and users. This area has seen considerable amount of research during the last decade. For a formal introduction to the area, interested readers are referred to the books by Kargupta et al. [7] and [8].

In this section, we present a brief overview of the work related to this area of research.

A. Distributed inner product computation

Fourier and wavelet transforms can be used for efficiently computing inner product when feature vectors are distributed between two parties. These transformations project the data to a new low-dimensional space where the inner product is preserved. The dominant Fourier and/or wavelet coefficients are transmitted to other parties and the inner product can still be computed from those coefficients with high accuracy. Random projection [9] is another communication-efficient approach for inner product computation in a two-party scenario. This technique has been used by Giannella et al. [1] for decision tree construction over distributed data. Interested readers are referred to [1] for details. These techniques work well for two parties, but do not scale well to large asynchronous network. More discussions are given in Section X.

B. Identifying top-k items

Several techniques exist in the literature for ranking items of a dataset. Wolff et al. [10] present a local algorithm that can be used for monitoring the entries in a certain percentile of the population. In their paper, the authors describe a majority voting algorithm, where each peer, P_i, has a real number b_i, and a threshold $\tau > 0$ (the same threshold at all peers). The goal is for the peers to collectively determine whether $\sum_i b_i$ is above $n\tau$ where n is the number of peers in the network. This technique can be potentially used to find all the entries of the inner product matrix that belong to the p^{th} percentile of the population. However, the major disadvantage is the communication complexity – a separate majority voting problem needs to be invoked for every inner product entry and thus the system will not scale well for large number of features. In the worst case, the communication complexity of the majority voting algorithm may become equal to the order of the size of the network.

Distributed top-k monitoring by Babcock et al. [4] presents a way of monitoring the answers to continuous queries over data streams produced at physically distributed locations. In their paper, the authors assume a central node and the top-k set is always determined by the central node. The coordinator node finds the answers to the top-k queries and distributes it to all the monitor agents. Along with it, the central node also distributes a set of constraints. These constraints allow a monitor node to validate if the current top-k set matches with what it finds from the local stream. If the validation results are true, nothing needs to be done. Otherwise, the monitor agent sends an alert to the coordinator node. The coordinator node re-computes the top-k set based on the current data distribution and sends out both the new top-k and new set of constraints to be validated by each monitor agent. Since the paper assumes that there is a central node, this technique is not directly applicable to many asynchronous large-scale networks such as Mobile ah-hoc networks, vehicular ad hoc networks and P2P networks which is the focus of this work.

Fagin [11] presents a way of combining query results derived from multiple systems. Often disparate databases and type of the query run on them return different types of results; Fagin's paper talks about combining them. It also proposes techniques to retrieve top-k elements from distributed databases. Our algorithm is applicable when there are a large number of nodes. Fagin's solution, when applied to our system, would require every peer to communicate resulting in a highly communication-intensive algorithm.

In the area of information retrieval, several techniques exist for top-k object identification. Balke et al. [12] propose a super-peer approach for finding the top objects. The top queries are handled by the super peers and any other peer in the network can contact these super peers to get the answers to these queries. They also discuss ways to select these super-peers so that any peer can find its closest super peer efficiently. There are also techniques which explore the retrieval algorithms taking into account the relative rankings of objects. Many of these algorithms depend on gossip-based techniques for spreading the ranks of its objects [13]. The major problems with gossip protocols are that they are slow (convergence can take a long time) and not very scalable due to global communication.

C. Peer-to-Peer data mining

P2P data mining is a relatively new research area. It pays careful attention to the distributed resources of data, computing, communication, and human factors in order to use them in a near optimal fashion. Clustering in P2P networks [14], association rule mining [10], monitoring L2 norm [15] are some of the recent work in this area. Interested readers are referred to an overview paper by Datta et al. [16].

In the next section, we present a high-level overview of our algorithm to identify the top inner product entries from the inner product matrices constructed out of horizontally partitioned data.

III. Notations, Problem Definition and Overview of the Algorithm

A. Notations

We assume that there are S nodes $P_1, P_2, ..., P_S$ in the network. Since we are dealing with horizontally partitioned data, let there be c global features, common to all peers. The local data set for peer P_d is denoted by $\mathbb{D}_d$ having r_d rows and c columns. The union of the data sets of all the peers is $\cup_{d=1}^{S} \mathbb{D}_d = \mathbb{D}$, which is the global dataset. The inner product matrix at peer P_d, denoted by $\mathbb{A}_d$, is a $c \times c$ matrix whose $(i,j)^{th}$ entry is the inner product between the i^{th} and j^{th} feature vector in $\mathbb{D}_d$. In matrix notation, $\mathbb{A}_d$ can be computed as $\mathbb{A}_d = \mathbb{D}_d^T \mathbb{D}_d$. The global inner product matrix, denoted by $\mathbb{A}$, can be formed by pointwise addition of all the inner product matrices of all the peers. In other words, the $(i,j)^{th}$ entry of $\mathbb{A}$ is $A[i,j] = \sum_{d=1}^{S} A_d[i,j]$. Since the inner product matrix is symmetric about the diagonal and the

diagonal elements are the inner product of the feature vectors with themselves, we consider only the upper triangular matrix excluding the diagonal. Thus we have $\frac{c^2-c}{2}$ distinct entries in the set of inner products that we consider at each site. Henceforth, any reference to $\mathbb{A}$ (or $\mathbb{A}_d$'s) would indicate the upper triangular inner product matrix excluding the diagonal elements. We also assume that the entries of all the inner product matrices ($\mathbb{A}$ or $\mathbb{A}_d$'s) are labeled with a single index. For example, the $(i,j)^{th}$ entry of $\mathbb{A}$, $A[i,j]$ is now denoted by $A\left[(i-1)\times(c-\frac{i}{2})+(j-i)\right]$, $1<i<j<\frac{c^2-c}{2}$.

B. Problem definition

Without loss of generality we assume that $A[1] \geq A[2] \geq ... \geq A\left[(i-1)\times(c-\frac{i}{2})+(j-i)\right] \geq ... \geq A[\frac{c^2-c}{2}]$ is the non-increasing ordering of the values of the global inner product matrix $\mathbb{A}$. Given such an ordering and a value p (between 1 and 100), the top-p percentile of the inner product entries consist of the following set $\mathcal{F} = \left\{A[1], A[2], ..., A[\frac{p}{100} \times \frac{c^2-c}{2}]\right\}$ such that $|\mathcal{F}| = k$. Now, given a connected and undirected graph $G(V,E)$ with $|V| = S$ and $|E| = e$ and each node having its local inner product matrix $\mathbb{A}_d$ (as defined in the previous section), our goal is to identify some l elements from $\mathcal{F}$ using local inner product matrices $\mathbb{A}_d$ and some locally exchanged information among the peers.

C. Overview of the algorithm

Having discussed the notations and problem definition, we are now in a position to present an overview of our algorithm. We develop an approximate *local* algorithm to solve the problem, which relies on random sampling in the network to avoid traversing all the nodes for collecting data. At the heart of this algorithm are the ordinal approximation based on theories from order statistics and cardinal approximation using Hoeffding bound.

Order statistics provides a lower bound on the number of samples required to identify the top percentile of a data distribution with a user-specified confidence level. Therefore, it can be used to compute the number of samples (the number of global inner products) required to determine the top-l inner product entries. We call this *ordinal sampling* since we are primarily interested in estimating the relative ordering in this case. However, since the value of each sample (i.e., the global value of each attribute-wise inner product) is distributed at different sites, we have to estimate it by doing a second round of sampling. We call this the *cardinal sampling*. These random samplings are done in the network using random walks. A node in the network that wants to identify some of the highest inner product entries of the global inner product matrix, launches random walks to collect the ordinal and cardinal samples. Once the initiator node gets back the estimates of the ordinal samples, it can then arrange the elements in a non-increasing order. Then, depending on the *threshold* determined by applying ordinal decision theory, the node can make decisions about the top-l inner product entries in the global data set. Thus, the initiator node could conclude about the globally most related features in the dataset without actually getting every other nodes' data.

IV. Building Blocks

This section elaborates on some building blocks that are necessary to understand our distributed algorithm for identifying significant inner product entries.

A. Decomposable inner product computation

Let $\mathbf{x}$ and $\mathbf{y}$ be two τ-dimensional feature vectors. The inner product between $\mathbf{x}$ and $\mathbf{y}$ is defined as:

$$< \mathbf{x}, \mathbf{y} > \quad = \quad \sum_{i=1}^{\tau} x_i y_i.$$

Now in our scenario, the values of $\mathbf{x}$ and $\mathbf{y}$ are distributed over the network. The inner product of those two vectors are:

$$< \mathbf{x}, \mathbf{y} > \quad = \quad \sum_{i=1}^{\tau} x_i y_i = \sum_{d=1}^{S} \left[\sum_{j=1}^{r_d} x_j y_j \right] = \sum_{d=1}^{S} I_d,$$

where peer P_d has an r_d-dimensional vector, which is P_d's contribution towards the inner product between $\mathbf{x}$ and $\mathbf{y}$. I_d is the local inner product of the P_d-th peer. Visiting all the peers is infeasible especially in large systems and hence we resort to sampling from a subset of peers in order to estimate $< \mathbf{x}, \mathbf{y} >$.

B. Ordinal approximation

Given a data set horizontally partitioned among peers, we want to find some top-l entries which are in the top-p percentile of the population. A trivial approach to this problem would be to collect the entire data set from all peers and compare all the pairwise inner products among the features. This simple approach, however, does not work in a large-scale distributed P2P environment because the overhead of communication would be extremely high. Order statistics is an excellent choice in this case, since, by considering only a small set of samples from the entire population, we can still produce a reasonably good solution with probabilistic performance guarantees. Order statistics has been applied in a number of different fields such as classifier learning [17], sensor networks [18], and discrete event optimization [19]. Next we discuss the application of order statistics in our framework.

Let $\mathbf{X}$ be a continuous random variable with a strictly increasing cumulative density function (CDF) $F_{\mathbf{X}}(x)$. Let ξ_p be the population percentile of order p, *i.e.* $F_{\mathbf{X}}(\xi_p) = Pr\{x \leq \xi_p\} = p$, *e.g.* $\xi_{0.5}$ is called the median of the distribution. Suppose we take n independent samples from the given population $\mathbf{X}$ and write the ordered samples as $x_1 < x_2 < \cdots < x_n$. We are interested in computing the value of n that guarantees

$$Pr\{x_n > \xi_p\} > q, \text{for a given constant } q.$$

Lemma 4.1 (Ordinal Approximation): Let $x_1, x_2, ..., x_n$ be n i.i.d. samples drawn from an underlying distribution. They are arranged such that $x_1 < x_2 < ... < x_n$. Then $P(x_n > \xi_p) = 1 - p^n$, where ξ_p is the p^{th} percentile of the population.

Proof:

$$P(x_n > \xi_p) = 1 - P(x_n \leq \xi_p) = 1 - F_n(\xi_p) = 1 - p^n.$$

■

Now if the above probability is bounded by a confidence q, we can rewrite the above equation as

$$1 - p^n > q \Rightarrow n \geq \left\lceil \frac{\log(1-q)}{\log(p)} \right\rceil. \quad (1)$$

For example, for $q = 0.95$ and $p = 0.80$, the value of n obtained from the above expression is 14. That is, if we took 14 independent samples from any distribution, we can be 95% confident that 80% of the population would be below the largest order statistic x_{14}. In other words, any sample with value greater or equal to x_{14} would be in the top 20 percentile of the population with 95% confidence. Note that, the value of n decreases by decreasing p. For detailed treatment of this subject we refer the reader to David's book [5].

When $\mathbf{X}$ is discrete, the equation $F_{\mathbf{X}}(x) = p$ does not have a unique solution. However, ξ_p can still be defined by $Pr\{x < \xi_p\} \leq p \leq Pr\{x \leq \xi_p\}$. This gives ξ_p uniquely unless $F_{\mathbf{X}}(\xi_p)$ equals p, in which case ξ_p again lies in an interval. It can be shown that in this case, $Pr\{x_n < \xi_p\} \leq I_p(n, 1) = p^N$, where $I_p(n, 1)$ is the incomplete beta function. Therefore, in the discrete scenario, we have

$$\begin{aligned} Pr\{x_n \geq \xi_p\} &= 1 - Pr\{x_n < \xi_p\} \\ &\geq 1 - p^n > q. \end{aligned}$$

This does not change the conclusion in Equation 1.

C. Cardinal approximation

Ordinal decision theory, as presented in the previous section, provides a bound on the number of samples that needs to be drawn from any population so that the highest-valued sample is in the top-p percentile of the population. However, in order to apply ordinal approximation, we need to estimate each of these ordinal samples using another round of sampling. We refer to this as *cardinal sampling*. In our distributed scenario, the samples are the inner product entry at each node. Therefore we need to visit a number of nodes for estimating each ordinal sample. In order to derive bounds on the number of peers to sample (m) for estimating each of these ordinal samples, we have used the Hoeffding Bound [6] which bounds the tail probability of a distribution.

Lemma 4.2 (Hoeffding Bound): Let x_i, $i \in \{1, \ldots, m\}$ be m independent samples of a random variable $\mathbf{X}$ with values in the range $[a, b]$. Let the sample mean be $Q_m = \frac{1}{m}\sum_i x_i$. Then for any $\epsilon > 0$, we have

$$Pr\{Q_m - E(\mathbf{X}) \geq \epsilon\} \leq \exp\left(-\frac{2m\epsilon^2}{(b-a)^2}\right),$$
$$Pr\{E(\mathbf{X}) - Q_m \geq \epsilon\} \leq \exp\left(-\frac{2m\epsilon^2}{(b-a)^2}\right).$$

Next, we show how the Hoeffding bound can be used to derive an upper bound on the value of m.

Lemma 4.3 (Cardinal Approximation): Let x_i, $i \in \{1, \ldots, m\}$ be m independent samples drawn from a population $\mathbf{X}$ with values in the range $[a, b]$. Let $Q_m = \frac{1}{m}\sum_i x_i$ be the sample mean. Then, when $m \geq \frac{(b-a)^2 ln(q')}{2\epsilon^2}$, we have

$$Pr\{Q_m - E(\mathbf{X}) \geq \epsilon\} \leq q',$$
$$Pr\{E(\mathbf{X}) - Q_m \geq \epsilon\} \leq q'.$$

Proof: Following Lemma 4.2, we have

$$Pr\{Q_m - E(\mathbf{X}) \geq \epsilon\} \leq \exp\left(-\frac{2m\epsilon^2}{(b-a)^2}\right) \leq q'.$$

Therefore,

$$-\frac{2m\epsilon^2}{(b-a)^2} \leq ln\,(q') \Longrightarrow m \geq \frac{(b-a)^2 ln\left(\frac{1}{q'}\right)}{2\epsilon^2}. \quad (2)$$

■

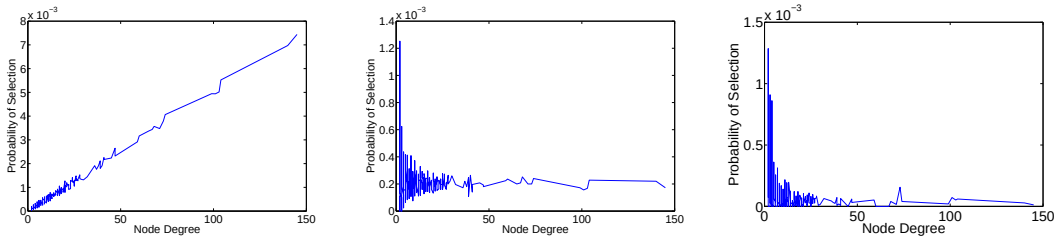
Note that $0 < q' < 1$, $0 < \epsilon < 1$ and both are parameters determined by the user. For example, if $b - a = 5$, $q' = 0.05$ and $\epsilon = 0.5$, we have $m \geq 150$. In other words, if we take at least 150 samples for estimating the mean of a random variable having a range 5, the probability that the difference between the true mean and the mean of the population is greater than 0.5 is less than by 0.05 (*i.e.* $Pr\,(Q_m - E[\mathbf{X}] \geq 0.5) \leq 0.05$ and $Pr\,(E[\mathbf{X}] - Q_m \geq 0.5) \leq 0.05$). Note that, as both ϵ and q' decreases, m increases.

In a distributed scenario, the peer which initiates the random walk needs to estimate this value of *m*. For each attribute c_i, it can compute the value of m_i using only the range of each attribute. Then m can be set to the maximum of all the individual m_i's *i.e.* $m = \max_{i=1}^{c}\{m_i\}$, where c is the number of attributes as defined in Section III-A.

D. Random sampling and random walk

The cardinal sampling process that we just discussed requires collecting samples from the peers. Random walk is a popular technique for random sampling from the network. It can be performed by modeling the network as an undirected graph with transition probability on each edge, and defining a corresponding Markov chain. Random walks of prescribed length on this graph produce a stationary state probability vector and the corresponding random sample. The simplest random walk algorithm chooses an outgoing edge at every node with equal probability, *e.g.* if a node has degree five, each of the edges is traversed with a probability 0.2. However, it can be shown that this approach does not yield a uniform sample of the network unless the degrees of all nodes are equal (see [20] for example). Since typical large-scale P2P network tends to have non-uniform degree distribution, this approach will generate a biased sample in most practical scenarios. Figure 1(a) shows the non-uniform selection probability using a power-law graph of 5000 nodes.

Fortunately, the elegant Metropolis-Hastings algorithm [21], [22] implies a simple way to modify the transition probability so that it leads to a uniform stationary state distribution, and therefore results in uniform sample. Such a technique has been used by Datta et al. [23] to generate uniform samples from a P2P network. In this paper, we use an adaptation [24] of this classical algorithm. Next we briefly introduce the Metropolis-Hastings algorithm for random walk.



(a) Simple Random Walk (b) Metropolis Hastings (c) Degree Balanced Random Walk

Fig. 1. Performance of three different random walks on a power law topology of 5000 Nodes.

Let $G(V, E)$ be a connected undirected graph with $|V| = S$ nodes and $|E| = e$ edges. Let d_i denote the degree of a node i, $1 \leq i \leq S$. The set of neighbors of node i is given by $\Gamma(i)$ where $\forall j \in \Gamma(i)$, edge $(i, j) \in E$. Let $T = \{p_{ij}\}$ represent the $n \times n$ transition probability matrix, where p_{ij} is the probability of walking from node i to node j in one message hop ($0 \leq p_{ij} \leq 1$ and $\sum_j p_{ij} = 1$). Algorithm 1 gives the basic protocol for generating this T in a distributed fashion using the Metropolis Hastings protocol. Note that peers need not know the entire matrix T in order to a random walk. All that peer P_i needs is one row of this matrix T_i, which gives the transition from node P_i to all other nodes.

Algorithm 1 Distributed Metropolis-Hastings (DMH) [22], [24]

Input of peer P_i: Its degree d_i

Output of peer P_i: A row (T_i) of transition matrix T

On initialization: P_i sends out a $Degree$ message to all $P_j \in \Gamma(P_i)$

On receiving a message ($Degree$): If it has received the degree information from all $P_j \in \Gamma(P_i)$ it can compute p_{ij} as follows:

$$p_{ij} = \begin{cases} 1/\max(d_i, d_j) & \text{if } i \neq j \text{ and } j \in \Gamma(i) \\ 1 - \sum_{j \in \Gamma(i)} p_{ij} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Termination: Once the p_{ij}'s have been populated set $T_i \leftarrow [p_{i1}\ p_{i2} \cdots\ p_{iS}]$. Terminate DHM.

This algorithm generates a symmetric transition probability matrix and has proven to produce uniform sampling via random walk. Lovász [20] showed that the length of random walk (λ) necessary to reach to stationary state is of the order of $O(\log S)$. Empirical results show that when the length of walk is $10 \times \log S$, this algorithm converges to uniform distribution. Figure 1(b) shows the probability of selection using the Metropolis-Hastings algorithm over a simulated network with 5000 nodes. As can be easily seen, the probability of selection is near uniform for nodes with different degrees. We also compared this technique with the Degree Balanced Random Walk (DRW) proposed by Orponen et al. [25]. Experiments (Figure 1(c)) shows that the probability is nearly uniform in this case as well. However, this technique requires a relatively long walk length in order to achieve stationarity. Therefore, we choose the MH algorithm for collecting samples from the network.

For random walk to reach a stationary state, we need an estimate of the network size. There exists several techniques in the literature to solve this problem. Examples include the capture-recapture method proposed by Mane et al. [26] and the aggregate computation as proposed by Bawa et al. [27].

V. P2P Algorithm for Identifying the Significant Inner Product Entries

Using the building blocks discussed in the previous section, we now describe our algorithm for doing distributed selection of some l elements from the top-p percentile of the population when there are k elements in the top-p percentile ($l < k$).

The process is started by the initiator node in the network that decides to find the top few entries in the distributed inner product matrix. Our algorithm needs to know three parameters – (1) number of ordinal samples to collect (n); (2) the number of peers to visit for estimating each sample (m); and (3) n indices of the inner product matrix corresponding to the n samples to collect. Based on the desired level of confidence (q), the percentile (p) of the population to monitor, the range R, the accuracy ϵ and q' (Section IV-C), the initiator knows the values of these parameters using the results of Section IV. It launches $m \times n$ random walks and after all these walks terminate, the samples are sent back to the initiator node. The initiator then needs to add all the samples having the same index. It then orders the n samples and the highest one is the $threshold$. Any inner product value greater than this threshold is expected to be in the top-p percentile of the population with the chosen confidence. Hence the overall approach consists of the following tasks:

1) *sample size computation*,
2) *sample collection*,
3) *threshold detection*, and
4) *some top-l inner product elements identification*

Each of these steps is further discussed below.

A. Sample size computation

The initiator P_d first selects a confidence level q and the order of population percentile p it would tolerate. Based on the bound derived in Section IV-B, the initiator calculates the number of samples (n) required to compute the threshold such that any inner product that is greater than this threshold is among the top-p percentile of the population of inner products. It also randomly generates n indices (each between $1 \leq i \leq$

$\frac{c^2-c}{2}$) which will be sampled for the set of all the inner product entries. The initiator also uses the Hoeffding bound (Section IV-C) to find the value of m, or the number of peers to visit for estimating each of these n ordinal samples. Thus, after this step, the initiator peer knows the value of m, n and the actual indices of the inner product entries to be sampled.

B. Sample collection

Given the sample size of n and the number of peers to visit m, the initiator invokes $m \times n$ random walks using the protocols described in Section IV-D to choose independent samples from the network. Since estimating one single inner product entry requires sampling m peers for the same indexed entry, each random walk carries with it the index number of the element to be sampled. Also each random walk carries the IP address and port number of the initiator node so that the terminal node of a random walk can send its inner product entry directly to the initiator node. At the end of these random walks P_d has $m \times n$ samples where there are n different indices and m inner product values for every index of the inner product.

C. Threshold detection

Once the initiator node gets all the samples, its next task is to identify the threshold. Since inner product is decomposable, for every index i, peer P_d sums up the all the m entries corresponding to the same index i. It then finds the largest of this n aggregated set of inner product entries and this is the threshold.

D. Some top-l inner product elements identification

The above technique would give the peer a way to identify one of the items in the top-k, where there are k elements in the top-p percentile of the population. We can extend this to find some l of the top-k elements ($l < k$). All that a peer P_d needs to do is to launch $n \times m \times l$ random walks. Now after aggregating the results we have nl elements and for every n element we can find a threshold. Thus we will have l thresholds. The ordinal framework guarantees that each of these *l* thresholds are in the top-p percentile of the population.

OrdSamp (Algorithm 2) presents the sample collection technique for a single random walk using the ordinal framework. The initiator sends a token (initialized to a value equal to the length of the random walk λ), its IP address, port number (*InitiatorNodeNum*) and the index of the element (*SampleIndex*) to sample for this random walk. When a node gets this token, it decrements its value by 1. If the value of the token becomes 0, the inner product entry indexed by *SampleIndex* is selected from the local data set and sent back to the initiator node.

Algorithm 2 Distributed selection of samples ($OrdSamp$)

Input of peer P_d: $\mathbb{D}_d$ - the local database, $\Gamma(d)$ - set of immediate neighbors of P_d, a row T_d of the transition matrix T

Output of peer P_d: Sends the sample if the random walk terminates at this peer

On receiving a message ($Token$):
$Token$ = $Token$ - 1
Fetch $SampleIndex$
Fetch $InitiatorNodeNum$
IP Address and Port number of the initiator node
if $Token = 0$ **then**
  Pick the element whose index is $SampleIndex$ from $\mathbb{D}_d$.
  Send $SampleIndex$ to the $InitiatorNodeNum$.
  Wait for new $Token$ messages for other random walks
else
  Send $SampleIndex$, $InitiatorNodeNum$ to a neighbor selected according to the transition matrix
end if

VI. Local Algorithm

In this section we first define *local* algorithms and then prove that the algorithm that we have developed in this paper is local.

Definition 6.1 (α-neighborhood of a vertex): Let $G = (V, E)$ be the graph representing the network where V denotes the set of nodes and E represents the edges between the nodes. The ***α-neighborhood of a vertex*** $v \in V$ is the collection of vertices at distance α or less from it in G: $\Gamma_\alpha(v, V) = \{u | dist(u, v) \leq \alpha\}$, where $dist(u, v)$ denotes the length of the shortest path in between u and v and the length of a path is defined as the number of edges in it.

Definition 6.2 (α-local query): Let $G = (V, E)$ be a graph as defined in last definition. Let each node $v \in V$ store a data set X_v. An ***α-local query*** by some vertex v is a query whose response can be computed using some function $f(X_\alpha(v))$ where $X_\alpha(v) = \{X_v | v \in \Gamma_\alpha(v, V)\}$.

Definition 6.3 ((α, γ)-local algorithm): An algorithm is called ***(α, γ)-local*** if it never requires computation of a β-local query such that $\beta > \alpha$ and the total size of the response to all such α-local queries sent out by a peer is bounded by γ. α can be a constant or a function parameterized by the size of the network while γ can be parameterized by both the size of the data of a peer and the size of the network.

We call such an (α, γ)-local algorithm ***efficient*** if both α and γ are either small constants or some slow growing functions (sub-linear) with respect to its parameters. The following lemma, Lemma 6.1 proves that Algorithm 2 is *local* according to this definition.

Lemma 6.1 (Locality): The $OrdSamp$ algorithm is $(O(logS), nml)$-local where S is the number of nodes in the network and the other items are as defined in Section IV.

Proof: We prove this using the property of random walks. The initiator node, launches $O(nml)$ independent random walks. Each random walk has a walk length of O(log$S$). So the maximum number of hops that a query can propagate for finding each samples is O(log$S$). While returning these samples, back to the initiator, it is a 1-hop process. Note that in the sample collection process, all the random walks

are launched using the same walk length. Hence the entire algorithm is an $(O(logS), nml)$-local since the number of queries is nml. ■

Note that the $OrdSamp$ algorithm is **efficient** since $\alpha = O(logS)$ is a slowly growing polynomial compared to the network size S and $\gamma = nml$ is a small number, independent of the network size. We have given typical example values of n, m and l in Sections IV-B, IV-C and V-D respectively.

Similarly we can show that the running time of our algorithm is $O(nml \times logS)$.

The previous definition discusses about the efficiency of such algorithms, it does not specify the quality of the result. There are two types of local algorithms in terms of accuracy: *exact* and *approximate*. In an exact local algorithm, once the computation terminates, the result computed by each peer is the same as that compared to a centralized execution [10]. However, such algorithms have only been developed till date for very simple thresholding functions (e.g., l_2-norm [15]). For more complicated tasks, researchers have proposed approximate local algorithms using probabilistic techniques (for example K-means [14]). Next, we define the notation for measuring the quality of local algorithms.

Definition 6.4 ((ϵ, δ) correct local algorithm): An local algorithm is (ϵ, δ) ***correct***, if it returns the result of a query within an ϵ-distance of its actual result with a probability of $(1 - \delta)$, where the actual result is computed on a centralized data and δ is the probability that the result is outside the ϵ radius.

The algorithm we have developed in this paper is both $(O(logS), nml)$-local and (ϵ, δ) correct, where $1 - \delta = q$, as defined in Section IV-B and ϵ corresponds to the error discussed in the next section.

VII. Error Bound and Message Complexity

In this section we analyze the error bound and the message complexity of our distributed algorithm.

A. Error bound

In our distributed algorithm there are two sources of error – (1) error due to ordinal sampling and (2) due to cardinal sampling. Let $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n$ denote the samples as found by the distributed algorithm (the subscripts correspond to the indexing scheme defined in III-A). Note that each of these $\tilde{x}_d$-s are estimated by aggregating the values of the d^{th} entry of the inner product matrix from m peers. The value of the d^{th} entry for the i^{th} peer is given by $A_i[d]$. Therefore, $\tilde{x}_d = \sum_{i=1}^{m} A_i[d]$. Let $\bar{A}[d] = \frac{\sum_{i=1}^{m} A_i[d]}{m}$ denote the mean of the estimates, $\forall d \in \{1, \ldots, n\}$. Lemma 7.1 derives the probability that the threshold *i.e.* $\tilde{x}_n$ is greater than the p^{th} percentile of the population.

Lemma 7.1 (Error): Let $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n$ be the n samples found by the distributed algorithm. They are ordered such that $\tilde{x}_1 < \tilde{x}_2 < \cdots < \tilde{x}_n$. Then, $P(\tilde{x}_n > \xi_p) = 1 - \prod_{d=1}^{n} \Phi\left(\left[\frac{\xi_p}{m} - \mu_d\right] \frac{\sqrt{m}}{\sigma_d}\right)$, where μ_d and σ_d are the mean and standard deviation of the feature of the population corresponding to $\tilde{x}_d$, ξ_p is the population percentile of order p and $\Phi(.)$ is the area under the standard normal curve.

Proof:

$$\begin{aligned}
P(\tilde{x}_n > \xi_p) &= 1 - P(\tilde{x}_n \leq \xi_p) \\
&= 1 - \prod_{d=1}^{n} P\left(\tilde{x}_d \leq \xi_p\right) \\
&= 1 - \prod_{d=1}^{n} P\left(\sum_{i=1}^{m} A_i[d] \leq \xi_p\right) \\
&= 1 - \prod_{d=1}^{n} P\left(\frac{\sum_{i=1}^{m} A_i[d]}{m} \leq \frac{\xi_p}{m}\right) \\
&= 1 - \prod_{d=1}^{n} P\left(\bar{A}[d] \leq \frac{\xi_p}{m}\right) \\
&= 1 - \prod_{d=1}^{n} P\left(\frac{\bar{A}[d] - \mu_d}{\frac{\sigma_d}{\sqrt{m}}} \leq \frac{\frac{\xi_p}{m} - \mu_d}{\frac{\sigma_d}{\sqrt{m}}}\right) \\
&= 1 - \prod_{d=1}^{n} P\left(\mathbf{Z} \leq \left[\frac{\xi_p}{m} - \mu_d\right] \frac{\sqrt{m}}{\sigma_d}\right) \\
&= 1 - \prod_{d=1}^{n} \Phi\left(\left[\frac{\xi_p}{m} - \mu_d\right] \frac{\sqrt{m}}{\sigma_d}\right).
\end{aligned}$$

■

Step 2 follows directly from step 1. Now since $\tilde{x}_d$ is a sum of all the elements obtained by visiting m peers, we must have $\tilde{x}_d = \sum_{i=1}^{m} A_i[d] \ \forall \ d$. Finally, since $\sum_{i=1}^{m} A_i[d]$ is a sum of random variables we have used Central Limit Theorem to derive the final expression.

Hence the probability of error is $\prod_{d=1}^{n} \Phi\left(\left[\frac{\xi_p}{m} - \mu_d\right] \frac{\sqrt{m}}{\sigma_d}\right)$. This shows that as n increases, the error decreases since each term of the product is $\Phi(.)$, which is the area under a unit Normal variable and is less than or equal to 1. Also as m increases, the expression inside Φ decreases and thus the overall probability of error decreases. For a special case in which all the μ_d's and σ_d's are equal to say μ and σ, the error becomes $\Phi\left(\left[\frac{\xi_p}{m} - \mu_d\right] \frac{\sqrt{m}}{\sigma_d}\right)^n$ – hence as n increases, the error decreases exponentially.

B. Message complexity

The distributed algorithm that we just described launches $n \times m \times l$ parallel random walks each of length λ such that each random walk will return a single element. The coordinator node can then aggregates these samples, and finds the l thresholds. We will use this model to analyze the message complexity.

For each such a random walk, the initiator node needs to send the following four information in the message:

1) Token Number - Integer 32 bits
2) Index of the inner product entry to sample - Integer 32 bits
3) IP Address - Integer 32 bits
4) Port Number - Integer 32 bits

The message complexity for this step is : $128 \times m \times n \times l \times \lambda = 128mnl\lambda$ bits. Since at the end of each random walk, the terminal node needs to send the sampled element back to the initiator node, it would need 64 bits (assuming that each entry

of the inner product matrix can be represented as a double number). Thus, the overall message complexity for the entire sample collection process is: $128mnl\lambda + 64nml$ = O($mnl\lambda$) bits. Substituting the values of n and m from equations 1 and 2 respectively, and using $10 * log(S)$ as the value of λ, the message complexity can be rewritten as,

$$[1 + 20log(S)] \left[64l\frac{(b-a)^2 ln(1/q')log(1-q)}{2\epsilon^2 log(p)}\right] \text{ bits},$$

where the symbols are defined in the respective sections. Note that this expression is independent of the number of features c, the number of rows r_i and is logarithmic with respect to the number of nodes.

Now, considering the centralized algorithm, if each peer has a dataset of size $r_i \times c$ =O($r_i c$), then the total message complexity for the centralized scheme can be written as : $64 \times r_i \times c \times S$ = O($r_i cS$) bits. Hence, the communication complexity of the centralized algorithm is dependent linearly on the size of the data set (r_i and c) and network (S).

VIII. Experiments and Performance Evaluation

In this section, we study the performance of the proposed inner product identification algorithm.

A. Network topology, simulator and data generation

Our network topology is generated using the ASWaxman Model from BRITE [28], a universal topology generator. The generator initially assigns node degrees from a power-law distribution and then proceeds to interconnect the nodes using Waxman's probability model. Power-law random graph is often used in the literature to model large non-uniform network topologies. It is believed that P2P networks conform to such power law topologies [29]. We use the Distributed Data Mining Toolkit (DDMT) [30] developed by the DIADIC research lab at UMBC to simulate the distributed computing environment.

The experimental data consists of tuples generated from different random distributions. Each column of the data is generated from a fixed uniform distribution (with a fixed range). Thus, there are as many different distributions as the number of features. The centralized data set is then uniformly split (so that each peer has the same number of tuples) among all the peers to simulate a horizontally partitioned scenario.

B. Performance

We study the applicability of the ordinal approximation theories in our distributed environment by comparing the results produced by the centralized algorithm. By a centralized algorithm we mean centralizing the entire data set of all peers and running the ordinal approximation on this data set. Our measurement metric consisted of two quantities – (1) **Quality** and (2) **Cost**. By quality we measure the thresholds detected both in the distributed and centralized scenario as compared to the actual percentile of the population. Cost refers to the message exchanged in Kilobytes (KB) for doing the computation with reference to a centralized scheme.

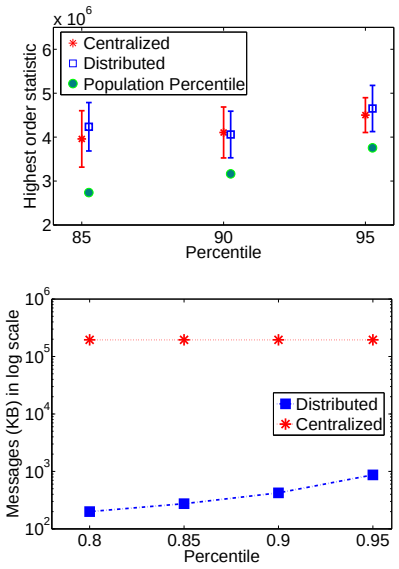


Fig. 2. Relative values of the estimated highest order statistic (distributed and centralized experiments) with corresponding values of actual population percentile (top figure) and corresponding cost (bottom figure).

We report here three sets of experiments - (1) performance of the algorithm when monitoring increasing percentile of population, (2) the scalability of our algorithm, and (3) the effect of increasing the cardinal sampling (m). We have reported both the quality and cost whenever appropriate. Unless otherwise noted we have the following default values for the different parameters: (1) S=500, (2) c=100, (3) n=19 (p=85% and q=95%), (4) $m = 35$ ($R = 5, q' = 0.5, \epsilon = 0.5$), (5) l=1, (6) $\lambda = 10 \times logS$, and (7) r_i (number of data rows for each peer) = 500. Each random experiment was run for 100 trials and the we plot both the average and the standard deviation.

1) Experiments with different percentile of population: In this experiment we compared the accuracy of the distributed algorithm with the centralized one. We have experimented with three different percentile (p) values of 95, 90 and 85 for which the number of samples (n) required are 59, 29 and 19 respectively. Figure 2 shows the effect on quality and cost with changes in population percentile. In Figure 2 (top), the circular points represent the actual p^{th} percentile of the population, whereas the blue square error bars and the red star error bars represent the threshold for the same confidence and percentile for the distributed and centralized scenario respectively using ordinal approximation. The distance between the red (stars) error bars and the green circular dots represents the error due to ordinal approximation whereas the difference between the red (stars) error bars and the blue (squares) error bars in the graph can be attributed to the cardinal approximation introduced in the distributed environment. We notice that in both the centralized and distributed scenario, the threshold is greater than the actual p^{th} percentile of the population. This means that there will be no false positives in ordinal estimation.

Figure 2 (bottom) compares the communication of our algorithm with that of the centralized version for monitoring different percentiles of population (p) plotted in the log-

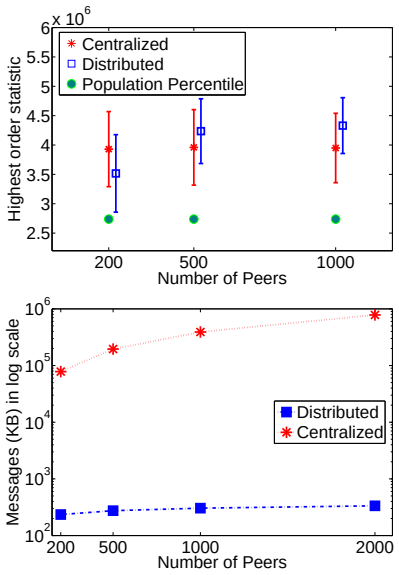


Fig. 3. Variation of the threshold detected by the centralized and distributed algorithms (top figure) and cost (bottom figure) with changes in the size of the network.

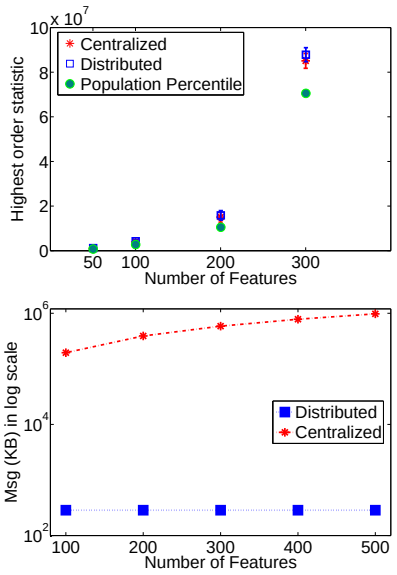


Fig. 4. Quality and cost with changes in number of features.

scale. Since the number of features $c = 100$, $r_i = 500$ and $S = 500$ remain constant, messages for the centralized experiments for different percentiles does not change. In the distributed scenario, the expression in Section VII-B is used for finding the number of messages. In all cases, our algorithm outperforms the centralizing scheme in terms of message complexity.

2) Scalability: We test the scalability of our algorithm both with respect to the number of nodes and number of features of the dataset. In both cases we plot the quality and cost of the algorithm.

For the scalability w.r.t. the number of peers, we keep the number of data points per peer constant (500). Figure 3 (top figure) shows the effect on the threshold detected as the size of the network is changed (all the other parameters are at their default values). As can be seen from the figure, the threshold detected by both the centralized and distributed experiments using order statistics are greater than the p^{th} percentile of the population. Moreover, the centralized and distributed estimates are quite close for different sizes of the network. This shows that our proposed distributed algorithm has good accuracy w.r.t scalability.

Figure 3 (bottom figure) shows the cost of the algorithm (plotted in log-scale) with increasing number of nodes. For the centralized algorithm, the effect of the number of nodes (S) is linear. On the other hand, it is logarithmic for the distributed algorithm (refer to Section VII-B for details). This means that the proposed distributed algorithm is far more communication efficient than the centralized counterpart as corroborated by the experiments here.

In the other scalability experiment, we varied the number of features (c). The results are shown in Figure 4. Figure 4 (top figure) shows that the quality of our estimate is quite good – in all cases, the highest order statistic is greater than the actual percentile of the population. Also, the centralized and distributed estimates are very close. Since there is a large difference in the scale, the points are close (almost on top of each other). The number of features has no effect on the cost of the distributed algorithm, while the same for the centralized algorithm increases linearly as shown in Figure 4 (bottom figure).

3) Experiments with increasing m: This section presents the quality and cost of the algorithm as the percentage of cardinal sampling (m) increases. Figure 5 (top figure) shows the effect on the highest threshold detected with increasing sampling m. The trend is clear - as we increase the percentage of network sampled, the distributed threshold (red stars) approaches the centralized threshold (blue squares). In Figure 5 (bottom figure), plotted in the log-scale, the messages transmitted increase as the percentage of network sampled increases. On the other hand, for the centralized version the message complexity is a constant.

Overall, this experiment shows that the estimation of our algorithm is comparable to the corresponding centralized version at a cost which is far less than its centralized counterpart.

IX. Application

An interesting application of this technique is client-side web mining. In this section we discuss how we modify our order statistics based top-l item identification technique for this application. Interested readers are referred to the paper by Liu et al. [3] for a detailed discussion on this application.

A. Why P2P communities and client-side web mining ?

According to Maslow's theory [31], social motive, which drives people to seek contact with others and to build satisfying relations with them, is one of the most basic needs of human beings. The tendency to have affiliations with others is visible even in virtual environments such as the World Wide Web. Many online communities like Google and Yahoo! groups

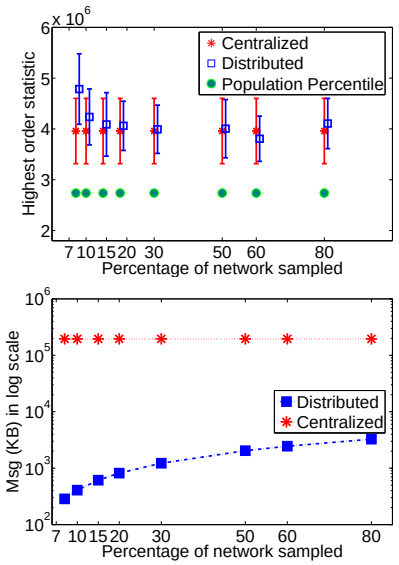


Fig. 5. Quality and cost with changes in cardinal sampling (m).

provide the user a place to share knowledge, and to request and offer services. Traditional web mining has spent lots of efforts on the web server side, *e.g.* to analyze the server log. We propose a framework that utilizes the client-side information, namely, the web browsing cache. In many cases the server-side web data is inaccessible to the user who generated the data – so no information about that data is available to the user. On the other hand, using the data at the source machine itself (which we call the client-side data), we can learn several interesting facts about the data and develop several systems (*e.g.* P2P community, recommender systems etc.). We define a P2P community as a collection of nodes in the network that share common interests. Communities can then exchange information for better query routing for example. Compared with other related work, our framework has the following specific features:

- It applies the order statistics-based algorithm already discussed to quantify the similarity between peers over the network. This approach allows a peer to build a community with hierarchical structure.
- Any technique in which the similarity between two peers can be expressed in metric space (vectors, trees and the like) can be plugged into our framework.

B. Related work: P2P communities

Generally speaking, the research on self-formation of P2P communities can be grouped into four major categories: 1) ontology matching-based approach; 2) attribute similarity-based approach; 3) trust-based approach; and 3) link analysis-based approach. We briefly introduce each of them as follows.

Castano et al. addressed the problem of formation of semantic P2P communities [32]. Each peer is associated with an ontology which gives a semantically rich representation of the interests that the peer exposes to the network. The advantage of this approach is that peers do not have to agree on the same predefined ontology, and therefore they have lots of flexibility of describing their interests. However, the gain of flexibility comes at the price of accuracy because of the uncertainty of concepts. We refer the reader to [33] for a brief survey of existing ontology matching approaches.

Khambatti et al. proposed a P2P community discovery approach where each peer is associated with a set of attributes that represent the interests of that peer [34]. These attributes are chosen from a controlled vocabulary that each peer agrees with. In this paper, we also assume each peer has a set of attributes, which we call as profile vector. The difference is that each interest in the profile vector can be given a weight to show its importance. Moreover, we do not simply check the intersection of attributes, instead, we quantitatively compute the similarity between profile vectors (using inner product), and we use an order statistics-based algorithm that can tell how similar a pair of peers are to each other in the whole network.

Trust-based community formation is usually discussed in the scenario of file sharing and service providing. The notation "trust" is a measure used by a peer to evaluate other peer's capability of providing a good quality service or resource. This trust is based on information about the peer's past behavior. We refer the reader to [35] as a starting point on this topic. In this paper, we are interested in forming a community based on peers' interests without considering the past interactions of peers.

There exists another area of research that focuses on the link structure analysis of network to identify patterns of interaction. For example, Scott identified the various cliques, components and circles into which networks are formed [36]. The drawback of link analysis-based approach is that it depends on the stable link structure of the network, and therefore precludes a peer from being a member of more than one community simultaneously.

C. Peer profiles

A crucial issue in forming P2P communities is to create peer profiles that accurately reflect a peer's interests. These interests can be either explicitly claimed by a peer, or implicitly discovered from the peer's behaviors. A peer's profile is usually represented by a keyword/concept vector. Trajkova et al. proposed techniques to implicitly build ontology-based user profiles by automatically monitoring the user's browsing habits [37]. Figure 6 shows a sample ontology for user profile. We point out that any approach that represents a peer's profile in a feature vector can be used in our framework. In this paper, we use the frequency of the web domains a peer has visited during a period of time as the peer's profile vector. To avoid the uncertainty of ontology matching, we expect all peers to agree on the same ontology defined by a controlled vocabulary. In this paper, this means that all peers agree on a superset of web domain names.

D. Similarity measurement

The goal of community formation is to find peers sharing similar interests. However, if we choose a similarity measurement Ω, and simply setup a subjective threshold such that peers

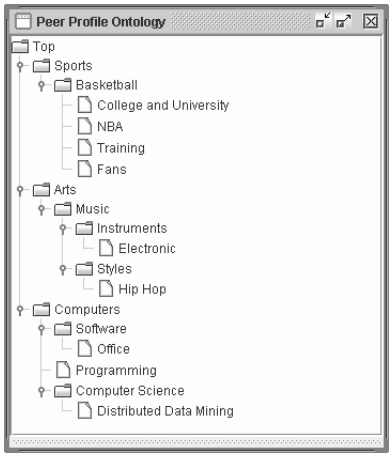


Fig. 6. A sample ontology for user profile.

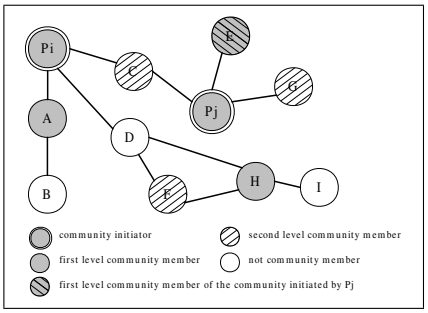


Fig. 7. Example of P2P communities.

with similarities greater than this threshold can be grouped together, we can't represent the essential characteristics of a social community, namely, *hierarchy*. In a social network, a person may have multi-level friends, where the first level might be family members and closest friends, the second level might be some colleagues who are not so familiar with. A person could also have indirect friends from his/her friends' social network. A P2P community from one peer's perspective should also have such a kind of hierarchical structure. That is, some peers share more interests with this peer, while others share less.

To achieve this goal, we use our order statistics-based approach which enables a peer to know how similar the other peer is to itself. In other words, our statistical measurement guarantees that if the similarity between peer P_i and P_j is above a threshold, P_i can determine with confidence level q that P_j is among the top-p percentile most similar peers of P_i's. As a running example, let us assume there are 5 peers $\{P_1, P_2, P_3, P_4, P_5\}$ in the network, and the similarity measures between P_1 and all other peers are $\{1, 3, 2, 4\}$, respectively, where the higher the value, the higher the similarity. If P_1 knows the similarity between her and P_5 is 4, our approach will enable P_1 to know with high confidence that P_5 is among the top 25% most similar peers of P_1's in the network, without computing all the similarity values.

Now we formally define a P2P community based on our above discussion.

Definition 9.1 ((Ω, p, q)-P2P Community): A (Ω, p, q)-P2P community from peer P_i's view is a collection of peers in the network, denoted by $\mathcal{C}$, such that the similarity measures Ω between P_i and all the members in $\mathcal{C}$ are among the top-p percentile of the population of similarity measures between P_i and all the peers in the network, with confidence level q.

Definition 9.2 (Extended (Ω, p, q)-P2P Community): An extended (Ω, p, q)-P2P community from peer P_i's view is the union of $\mathcal{C}$ (defined by the above Definition) and all the peers from the (Ω, p, q)-P2P community of each member in $\mathcal{C}$.

These two definitions implicitly capture the hierarchical characteristics of the community. When a peer finds a similar buddy, she could compute the percentile value and determine which area this buddy belongs to. A peer could also specify a p value and only invite those belonging to top-p percentile area to be her community members. The community could be expanded to include members from members's community. For example, in Figure 7, Peer A, P_j, H are the first level members (with larger p) of community initiated by P_i. Peer C, F and G are the second level members (with smaller p) of community. Note that P_j is also a initiator of another community, and it has E as its first level community member. Peer A, P_j, H, E compose an extended P2P community initiated by P_i.

We use the scalar product between two profile vectors to quantify the similarity between two peers. Other similarity metrics such as Euclidean distance, graphs, trees can also be applied in our framework without any hurdle. In the next subsection we discuss how the community is actually formed. Note than in this application, since each peer has the entire vector, there is no need for cardinal sampling. We can simply do an ordinal sampling of the entries of the inner product entries in order to identify the top few.

E. Community formation process

We address the P2P community formation process under the assumptions that: 1) each peer can be a member of multiple virtual communities; 2) peers interact with each other by submitting or replying queries to determine the potential members of a given community; and 3) there is no super peer as a centralized authority.

The P2P community emerges as a peer, P_i, called community initiator, invokes a community discovery process which consists of the following tasks: *sample size computation, percentile estimation, member identification, member notification and acceptance,* and *community expansion.*

Sample Size Computation: The initiator P_i first selects a confidence level q and the order of population percentile p it would tolerate. It can then find the sample size n as discussed in Section IV-B. Note that for this scenario a peer does not need to do a cardinal sampling since we are dealing with a special case of the distributed inner product computation here – when each peer has only one feature vector and not a matrix of local inner product elements.

Percentile Estimation: Given the sample size n, the initiator invokes n random walks using the protocols described in Section IV-D to choose independent sample peers in the network. Whenever a new peer P_j is chosen, it replies to P_i

with its address and port number, and builds an end-to-end connection with P_i. Then P_i computes the scalar product of its profile vector and P_j's profile vector. After P_i collects all the n scalar products, it finds the largest one as the threshold for percentile of order p. These two steps are very similar to the first two steps of the algorithm discussed in Section V.

Member Identification: The initiator P_i composes a discovery message containing its address and port number, as well as a time-to-live (TTL) parameter defining the maximum number of hops allowed for the discovery propagation. Then the discovery message is sent to all P_i neighbors. When a peer P_j receives this message, it replies to P_i with its address and port number. P_i then invokes a scalar product computation with P_j to get the similarity value. If TTL ≥ 0, P_j forwards the discovery message to all its neighbors, except for the peer from which the message has been received. Each peer discards duplicate copies of the same discovery message possibly received.

Member Invitation and Acceptance: The initiator P_i evaluates the quality of the discovered peers by comparing the similarity values with the different levels of threshold. If the similarity is above the threshold, P_i sends an invitation message to that peer. If the similarity is below the first level of threshold, P_i still could analyze, with the same confidence level and order of percentile p'. Given this information, P_i can decide whether to send an invitation to a peer with less similarity. Once a peer P_j receives an invitation message, it decides whether to accept it or not by replying with an acceptance message. Receiving the acceptance message, P_i records P_j in its cache.

Community Expansion: When a peer P_j accepts the invitation, it replies to the initiator with an acceptance message, as well as with the member lists in its local cache. These members are from the P2P community or extended P2P community initiated by P_j. The peers that P_i receives from P_j, however, belong to the lowest level of friends of P_i in its hierarchical network, since they are not directly discovered by P_i and are just part of its social network through association. As a reward, P_i sends the current member list in its local cache to P_j. In this way, each peer has an extended P2P community.

F. Experiments

In this section, we study the performance of the proposed framework for P2P community formation.

1) Data Preparation: We use the web domains a peer has browsed to create the profile vector. Each element of the vector corresponds to the frequency that the domain has been visited by the peer during a period of time. The data was collected from the IE history files of volunteers from UMBC and Johns Hopkins University. There are totally 40842 browsing history records in our data set, and 3318 unique web domains. These records are randomly split and distributed to peers in our network simulator so that each peer can compute its own profile vector. As we have stated previously, we assume all the peers agree on the same profile ontology, *i.e.* the same set of domain names, and therefore, all the profile vectors have the same size - 3318. Figure 9 shows a snapshot of a peer's profile.

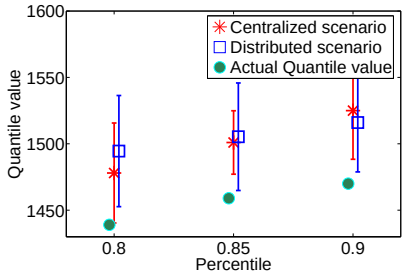


Fig. 8. Quality value w.r.t. the order of percentile.

TABLE I
AVERAGE NUMBER OF COMMUNITY MEMBERS FOUND BY THE INITIATOR WITH DIFFERENT TTL VALUES.

TTL	Friends/peer	Messages/peer
1	5	30
2	22	31
3	55	30

2) Performance: Having discussed about the data and the simulator setup we are in a position to report the experimental results.

Random Sampling and Percentile Estimation: This experiment evaluates the accuracy of random sampling and percentile estimation. We chose three different p values - $80\%, 85\%$ and 90%. In all the three cases, the confidence level q was set to 95%, and the size of the network was fixed at 100 nodes. Let P_i be the community initiator. The population can be defined as the set of all pairwise scalar products between P_i and all the other peers. Now, if P_i wants to find similar peers who are in the top-p percentile of the population, it launches n random walks. The terminal peer for each random walk refers to a sample and P_i computes the scalar product between its own vector and the vector owned by the sample. P_i sorts all the n scalar products and finds the largest one as the threshold of percentile of order p. Figure 8 shows estimated threshold in the distributed experiment. To compare the results with centralized sampling, P_i first collects the pairwise scalar products between itself and all the peers in the network. P_i then performs a random sampling of size n and finds the largest scalar product. The threshold found by this approach is illustrated by the stars in Figure 8. Figure 8 also shows the actual population percentile of order p. As is evident from these results, the threshold found through random sampling and order statistics theory is above the actual population percentile. Therefore any scalar product greater than this threshold can be recognized as among the top-p percentile population with high confidence.

Community Formation: Once the threshold is detected, the next step is to form the communities. The size of the network was fixed to be 100. Table I shows the average number of members found by a community initiator with respect to different TTL values using the community expansion scheme. The table also shows the number of messages per peer. Since it remains a constant, we expect good scalability of our algorithm.

Table II presents the number of community members formed

TABLE II
AVERAGE NUMBER OF COMMUNITY MEMBERS FOUND BY THE INITIATOR WITH INCREASING NUMBER OF NODES.

#nodes	Friends/peer	Messages/peer
100	10	30
500	25	30
1000	23	30
2000	33	34

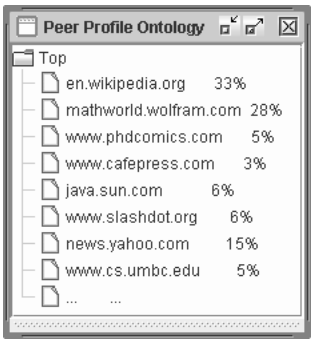


Fig. 9. Snapshot of a peer's profile.

for different network sizes. Here also since the number of messages per peer remains a constant, the algorithm is highly scalable.

X. DISCUSSION

In this section we compare the communication complexity of our algorithm with some existing distributed inner product computation techniques. One of the most widely used methods is random projection. Considering a vector of dimension $r_i \times 1$ at each site and a network of size S, the communication complexity of random projection method for finding the pairwise inner products is $\binom{S}{2} \times k = O(S^2k)$, where $k \times r_i$, is a random matrix such that $k < r_i$. Under a similar setting, the communication complexity of our algorithm is $O(n \times log(S) + n \times r_i)$ where n is the number of ordinal samples required (which in most cases is very small). The dominating factor for P2P networks is the size S; hence our algorithm scales well compared to the random projection method even if r_i is of the order of S. Egecioglu et al. [38] propose a technique in which the inner product can be computed using only two floating point numbers. Although this technique is very efficient, it is still a two-party protocol, and it cannot identify top inner products in a population distributed over many places without communicating with all the parties. However, we can adopt these efficient inner product protocols in our ordinal and cardinal approximation framework and achieve more efficient and effective solutions.

XI. CONCLUSIONS

In this paper we have developed a distributed algorithm for efficiently identifying top-l inner products from horizontally partitioned data. To achieve low communication overhead, we use an order statistics-based approach together with cardinal sampling. Ordinal statistics provides a general framework for estimating distribution free confidence intervals for population percentiles. Cardinal sampling helps to combine the inner product values that are distributed among the peers. Experimental results substantiate our claims regarding accuracy and message complexity of our algorithm.

Besides having direct algorithmic contributions, this paper suggests and adopts an important concept in large distributed computing systems – *local* approximate algorithms. Local algorithms are natural candidates for applications in large dynamic networks because of their good scalability. Local algorithms can be *exact* or *approximate*. However, the class of exact local algorithms that currently exist in the literature work for simple primitives such as average and L2-norm. For solving more complicated distributed problems, researchers have developed approximate solutions. The ordinal analysis technique developed in this paper belongs to this genre of approximate local algorithms. As demonstrated by the simulation results, our algorithm performs well both in terms of accuracy of results and communication intensity. In future, we hope to use this algorithm for solving real-life challenges in distributed settings such as the internet and sensor networks.

ACKNOWLEDGEMENTS

This research is supported by the United States National Science Foundation CAREER award IIS-0093353 and NASA Grant NNX07AV70G. The authors would also like to thank Phoung Nguyen, Chris Giannella and the anonymous reviewers for their valuable comments.

REFERENCES

[1] C. Giannella, K. Liu, T. Olsen, and H. Kargupta, "Communication Efficient Construction of Deicision Trees Over Heterogeneously Distributed Data," in *Proceedings of ICDM'04*, Brighton, UK, 2004, pp. 67–74.
[2] K. Liu, H. Kargupta, and J. Ryan, "Random Projection-Based Multiplicative Data Perturbation for Privacy Preserving Distributed Data Mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 92–106, 2006.
[3] K. Liu, K. Bhaduri, K. Das, P. Nguyen, and H. Kargupta, "Client-side Web Mining for Community Formation in Peer-to-Peer Environments," in *Proceedings of WebKDD'06*, Philadelphia, Pennsylvania, 2006.
[4] B. Babcock and C. Olston, "Distributed Top-k monitoring," in *Proceedings of SIGMOD'03*, California, 2003, pp. 28–39.
[5] H. A. David, *Order Statistics*. John Wiley and Sons, Inc., 1970.
[6] W. Hoeffding, "Probability for Sums of Bounded Random Variables," *Journal of the American Statistical Association*, no. 58, pp. 13–30, 1963.
[7] H. Kargupta and K. Sivakumar, *Existential Pleasures of Distributed Data Mining. Data Mining: Next Generation Challenges and Future Directions*. AAAI/MIT press, 2004.
[8] H. Kargupta and P. Chan, Eds., *Advances in Distributed and Parallel Knowledge Discovery*. MIT Press, 2000.
[9] R. I. Arriaga and S. Vempala, "An Algorithmic Theory of Learning: Robust Concepts and Random Projection," in *Proceedings of FOCS'99*, New York, 1999, pp. 616–623.
[10] R. Wolff and A. Schuster, "Association Rule Mining in Peer-to-Peer Systems," *IEEE Trasnsactions on Systems, Man and, Cybernetics Part B: Cybernetics*, vol. 34, no. 6, pp. 2426–2438, 2004.
[11] R. Fagin, "Combining Fuzzy Information from Multiple Systems," in *Proceedings of SIGMOD'96*, Montreal, Canada, 1996, pp. 216–226.
[12] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden, "Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks," in *Proceedings of ICDE'05*, Tokyo, Japan, 2005, pp. 174–185.
[13] F. Cuenca-Acuna, C. Peery, R. Martin, and T. Nguyen, "PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities," in *Proceedings of HPDC'03*, Seattle, Washington, 2003, pp. 236–249.

[14] S. Datta, C. Giannella, and H. Kargupta, "K-Means Clustering over Large, Dynamic Networks," in *Proceedings of SDM'06*, Maryland, 2006, pp. 153–164.

[15] R. Wolff, K. Bhaduri, and H. Kargupta, "Local L2 Thresholding Based Data Mining in Peer-to-Peer Systems," in *Proceedings of SDM'06*, Maryland, 2006, pp. 430–441.

[16] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta, "Distributed Data Mining in Peer-to-Peer Networks," *IEEE Internet Computing Special Issue on Distributed Data Mining*, vol. 10, no. 4, pp. 18–26, 2006.

[17] K. Tumer and J. Ghosh, "Robust Combining of Disparate Classifiers through Order Statistics," *Pattern Analysis and Applications*, vol. 5, pp. 189–200, 2001.

[18] M. B. Greenwald and S. Khanna, "Power-Conserving Computation of Order-Statistics over Sensor Networks," in *Proceedings of PODS'04*, Paris, France, 2004, pp. 275–285.

[19] Y.-C. Ho, C. G. Cassandras, C.-H. Chen, and L. Dai, "Ordinal Optimization and Simulation," *Journal of Operations Research Society*, vol. 51, pp. 490–500, 2000.

[20] L. Lovász, "Random Walks on Graphs: A Survey," *Combinatorics*, vol. 2, no. 80, pp. 1–46, 1993.

[21] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equations of State Calculations by Fast Computing Machines," *Journal of Chemical Physics*, vol. 21, pp. 1087–1092, 1953.

[22] W. Hastings., "Monte Carlo Sampling Methods Using Markov Chains and Their Applications," *Biometrika*, vol. 57, pp. 97–109, 1970.

[23] S. Datta and H. Kargupta, "Uniform Data Sampling from a Peer-to-Peer Network," in *Proceedings of ICDCS'02*, Toronto, Ontario, 2007, p. 50.

[24] A. Awan, R. A. Ferreira, S. Jagannathan, and A. Grama, "Distributed Uniform Sampling in Unstructured Peer-to-Peer Networks," in *Proceedings of HICSS'06*, Kauai, Hawaii, 2006.

[25] P. Orponen and S. E. Schaeffer, "Efficient Algorithms for Sampling and Clustering of Large Nonuniform Networks," arXiv.org e-Print archive, Tech. Rep. cond-mat/0406048, 2004.

[26] S. Mane, S. Mopuru, K. Mehra, and J. Srivastava, "Network Size Estimation In A Peer-to-Peer Network," University of Minnesota, MN, Tech. Rep. 05-030, September 2005.

[27] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani, "Estimating Aggregates on a Peer-to-Peer Network," April 2003, Stanford University.

[28] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation," in *Proceedings of MASCOTS'01*, Ohio, 2001.

[29] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in *Proceedings of MMCN'02*, San Jose, CA, J 2002, pp. 156–170.

[30] "DDMT," http://www.umbc.edu/ddm/wiki/software/DDMT/.

[31] A. H. Maslow, *Motivation and Personality*, 3rd ed. HarperCollins Publishers, January 1987.

[32] S. Castano and S. Montanelli, "Semantic Self-Formation of Communities of Peers," in *Proceedings of ESWC'05*, Heraklion, Greece, 2005.

[33] N. Noy, "Semantic Integration: A Survey of Ontology-based Approaches," *ACM SIGMOD Record*, vol. 33, no. 4, pp. 65–70, 2004.

[34] M. Khambatti, K. D. Ryu, and P. Dasgupta, "Efficient Discovery of Implicitly Formed Peer-to-Peer Communities," *International Journal of Parallel and Distributed Systems and Networks*, vol. 5, no. 4, pp. 155–164, 2002.

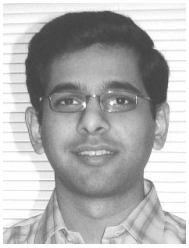
[35] Y. Wang and J. Vassileva, "Trust-Based Community Formation in Peer-to-Peer File Sharing Networks," in *Proceedings of WI'04*, Beijing, China, 2004, pp. 341–338.

[36] J. P. Scott, *Social Network Analysis: A Handbook*, 2nd ed. Sage Publications Ltd., March 2000.

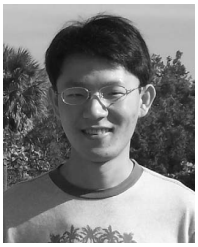
[37] J. Trajkova and S. Gauch, "Improving Ontology-Based User Profiles," in *Proceedings of RIAO*, Vaucluse, France, 2004, pp. 380–389.

[38] Ömer Egecioglu and H. Ferhatosmanoglu, "Dimensionality Reduction and Similarity Computation by Inner Product Approximations," in *Proceedings of CIKM'00*, New York, 2000, pp. 219–226.

Kamalika Das received her B.Tech degree in Computer Science and Engineering from Kalyani University, India, in 2003 and her MS degree in Computer Science from University of Maryland Baltimore County, in 2005. Since then, she has been a PhD student at the same university, working in the area of distributed data mining. Her research interests include privacy preserving data mining, web mining and P2P data mining. More information about her can be found at http://www.csee.umbc.edu/˜kdas1.



Kanishka Bhaduri received his B.E. in Computer Science and Engineering from Jadavpur University, India. Currently he is doing his PhD at University of Maryland Baltimore County. His research interests include distributed and P2P data mining, data stream mining, and statistical data mining. More information about him can be found at http://www.csee.umbc.edu/˜kanishk1.



Kun Liu is a member of the Intelligent Information Systems group at IBM Almaden Research Center. He received his Ph.D. in Computer Science from University of Maryland Baltimore County in 2007. His main research interests include privacy preserving data mining, distributed data mining and web mining. His paper client-side web mining for community formation in P2P environments with privacy constraints was selected as the most interesting paper of WebKDD'06. Kun regularly serves as a reviewer and/or external reviewer for SDM, KDD, ICDM conferences and TKDE journals. He is also in the organizing committee for the 2002 and 2007 NSF Workshop/Symposium on Next Generation of Data Mining. More information about him can be found at http://www.csee.umbc.edu/˜kunliu1.

Hillol Kargupta is an Associate Professor at the Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County. He received his Ph.D. in Computer Science from University of Illinois at Urbana-Champaign in 1996. He is also a co-founder of AGNIK LLC, a ubiquitous data intelligence company. His research interests include distributed data mining, data mining in ubiquitous environment, and privacy-preserving data mining. Dr. Kargupta won a US National Science Foundation CAREER award in 2001 for his research on ubiquitous and distributed data mining. He, along with his coauthors, received the best paper award at the 2003 IEEE International Conference on Data Mining for a paper on privacy-preserving data mining. He won the 2000 TRW Foundation Award, the 1997 Los Alamos Award for Outstanding Technical Achievement, 1996 SIAM Annual Best Student Paper Award. His research has been funded by the US National Science Foundation, US Air Force, Department of Homeland Security, NASA, and various other organizations. He has published more than 90 peer-reviewed articles in journals, conferences, and books. He has coedited two books: Advances in Distributed and Parallel Knowledge Discovery, AAAI/MIT Press, and Data Mining: Next Generation Challenges and Future Directions, AAAI/MIT Press. He is an associate editor of the IEEE Transactions on Knowledge and Data Engineering, the IEEE Transactions on Systems, Man, and Cybernetics, Part B, and the Statistical Analysis and Data Mining Journal. He regularly serves on the organizing and program committees of many data mining conferences. More information about him can be found at http://www.csee.umbc.edu/˜hillol. He is a senior member of the IEEE.