Approximate Top-k Queries in Sensor Networks*
(Extended Abstract)

Boaz Patt-Shamir and Allon Shafrir

Dept. of Electrical Engineering
Tel Aviv University
Tel Aviv 69978, Israel
boaz@eng.tau.ac.il, shafrir@eng.tau.ac.il

Abstract. We consider a distributed system where each node has a local count for each item (similar to elections where nodes are ballot boxes and items are candidates). A top-k query in such a system asks which are the k items whose sum of counts, across all nodes in the system, is the largest. In this paper we present a Monte-Carlo algorithm that outputs, with high probability, a set of k candidates which approximates the top-k items. The algorithm is motivated by sensor networks in that it focuses on reducing the individual communication complexity. In contrast to previous algorithms, the communication complexity depends only on the global scores and not on the partition of scores among nodes. If the number of nodes is large, our algorithm dramatically reduces the communication complexity when compared with deterministic algorithms. We show that the complexity of our algorithm is close to a lower bound on the cell-probe complexity of any non-interactive top-k approximation algorithm. We show that for some natural global distributions (such as the Geometric or Zipf distributions), our algorithm needs only polylogarithmic number of communication bits per node.

1 Introduction

Possibly one of the clearest examples of the difference between "global" and "local" can be seen in elections: each ballot box has a local score for each candidate, but the result we care about is the global scores, i.e., how many votes does each candidate have overall. A top-k query in this case is "Which are the k globally most popular candidates?". Other examples for the top-k task abound: in peer-to-peer file-sharing networks (such as Gnutella), users may wish to find which are today's most popular downloads; in sensor networks, a sensor may count the number of occurrences of different species of birds, and a user might be interested in the most frequent species observed over the whole instrumented area; in a server farm with several gateways, denial-of-service (DoS) attacks are a major concern. The first question to be answered in this case is which are the most frequent sources of requests; and many others.

* This research was supported in part by Israel Ministry of Science and Technology contract 3-941.

P. Flocchini and L. Gąsieniec (Eds.): SIROCCO 2006, LNCS 4056, pp. 319–333, 2006.

In general, a top-k query returns the k items having largest global score in a distributed system, where each item has a set of local scores. The global score of an item is just the sum of its local scores, over all locations. The main difficulty is that the scores may be divided arbitrarily among the different locations. In elections, for example, it may be the case that the most popular global candidate has the lowest (positive) count in each ballot box.

When computing top-k queries in a distributed system, a key question is how to minimize the *communication complexity* required to provide an answer. This issue is particularly important in sensor networks, where the communication subsystem is by far the largest energy consumer at the nodes. An algorithm which allows us to trade communication for local computation may have a decisive effect on the longevity of node batteries and hence on the usability of the system (see, e.g. [14]). This observation has established the measure of *individual* communication complexity as a key performance criterion in sensor networks [14, 16, 24, 7, 20]. In this work, we adopt this measure to evaluate the complexity of top-k computation. Observe that deterministic algorithms are sensitive to the way scores are partitioned among the different nodes and for some partitions they may communicate all scores to a single node.

Our Results. In this paper we propose a simple and effective way to overcome the problem of adversarial partition of the scores among the nodes. Our algorithm is Monte Carlo (it may err with some arbitrarily small probability), and its results are only approximate: using very little communication, the algorithm can tell, roughly, which items are in the top-k set. We focus on the worst-case individual communication complexity, i.e., our goal is to minimize the maximal number of bits communicated (sent or received) by any single node.

Our basic tool is random sampling. Done in the right way, sampling strips away the difficulties due to geographical distribution of scores which are the main difficulties for deterministic and Las Vegas algorithms. The basic idea is compounded with techniques adapting it to the specific input at hand. The performance of the algorithm depends on how popular are the top-k items, and on how "flat" is the distribution of scores. Specifically, suppose that the global scores adhere to the Zipf distribution with parameter $a > 1$ (namely the relative popularity of the i^{th} popular item is proportional to i^{-a}). Then our Algorithm R guarantees that the communication complexity is bounded by $O(\frac{k}{\varepsilon^2})$ times a polylogarithmic factor, where ε is the required approximation accuracy. This case is quite important, as it is widely believed that the statistics of many phenomena are well approximated by the Zipf distribution (see, e.g., [3, 12]).

We note that the communication complexity of our algorithms scales very well compared with previous algorithms [11, 6]. Our simulations demonstrate that the performance of our algorithm is significantly superior to the best previously known algorithms.

We give some evidence showing that our algorithm is close to optimal. In particular, we demonstrate the optimality of its *cell-probe* complexity [23, 13] among a limited class of single-round Monte Carlo algorithms.

Previous Work. In [11], Fagin, Lotem and Naor introduced the Threshold Algorithm (TA) in the context of databases. They define a notion of 'instance-optimality,' and prove that TA incurs at most n accesses times the optimum, where n is the number of nodes in the system (they also show that an $\Omega(n)$ factor blowup is unavoidable for any deterministic or Las-Vegas algorithm). In [6], Cao and Wang propose the TPUT algorithm to reduce latency and save communication for the case where the *local* inputs are generated by Zipf-like distributions. As expected from deterministic algorithms, the performance of TPUT and TA depend crucially on the partition of scores to nodes. Other related work include variations of TA and TPUT optimized for certain network models [17, 25, 5].

From the sensor networks perspective, top-k queries are viewed as a special case of aggregate queries (see, e.g., [16, 24]). Typically, it is assumed that data is routed on a spanning tree, and each node does some aggregation en-route. Simple aggregates (such as counting the number of items, summing numbers etc.) can be done with $O(\log n)$ bits per node. Considine et al. [7], and Gibbons et al. [18], present methodologies for robust approximation of aggregates in sensor networks. In [18] they also present a sketch of a top-k-approximation algorithm that appears promising, but the algorithm is not fully specified, and no formal statement or analysis is given.

Techniques for efficient monitoring of aggregates in sensor networks are studied by [21, 4, 8, 2]. The main question in these works is how to efficiently update the results under some assumptions on the way the input changes.

Paper Organization. The remainder of the paper is organized as follows. Section 2 describes our model, problem definition and a few known results about efficient counting. Section 3 presents our algorithms with formal analysis results. Simulation results are presented in Sect. 4. In Sect. 5, we present our lower bound, and we conclude in Sect. 6. Due to lack of space, proofs are omitted from this extended abstract.

2 Model and Preliminaries

System Model. The system is modeled as a communication graph $G(V, E)$ with $n \triangleq |V|$, where each node models a classical RAM machine with access to its local input and to an infinite tape of random bits. A distinguished node $v^* \in V$ is the *root node* and is assumed to have a special write-once output register.

The system executes distributed algorithms according to the standard asynchronous message passing model (see, e.g., [1, 15]). Very briefly, in this model an event (such as arrival of a message to node u) triggers a state-transition (e.g., u computes a response message and inserts it to the link buffer). An execution in our model is considered terminated when the root-node has written the result to the output register.

Let M denote some finite string of bits. We assume that the system contains a message passing infrastructure supporting the following facilities:

- Each node $u \in V$ may send a message M to any other node $v \in V$. This causes each node along a path from u to v to send and receive $\Theta(|M|)$ bits.

– Each node $u \in V$ can broadcast a message M to all other nodes. This causes every node in V to send and receive $\Theta(|M|)$ bits.

While the particular way in which these actions are implemented is immaterial for our purposes, we note that these assumptions can be justified by the existence of a spanning-tree of constant degree for message passing (see, e.g., [14, 16, 24, 7, 20]).

Input Model and Problem Statement. Let $\mathcal{I}$ denote the set of possible *items*. We assume $\mathcal{I}$ is finite. An instance of the problem, denoted by X, is a vector of multisets of $\mathcal{I}$: one multiset, denoted X_v, for each node $v \in V$. We sometimes slightly abuse notation and use X to also denote the multiset $\bigcup_{v \in V} X_v$.

It is convenient to imagine each multiset as a set of *cells*, where each cell contains a single item, so that an item with multiplicity w has w replicas, one in a cell. The *weight* of item i in node v, denoted $w_v(i)$, is the multiplicity of i in X_v. The *weight of a node* v is the total number of cells in v, formally $W_v \triangleq |X_v|$. The *weight of an item* $i \in \mathcal{I}$, is the sum of its multiplicities over all nodes, i.e., $w(i) \triangleq \sum_{v \in V} w_v(i)$. The input size is defined to be the total number of cells, $W(X) \triangleq \sum_{v \in V} |X_v|$. The *empirical probability*, or *frequency*, of an item $i \in \mathcal{I}$ in X, is defined by $p_X(i) \triangleq \frac{w(i)}{W}$. When the context is clear we omit the subscript.

Using this notation, we define the top-k set of X as follows.

Definition 1. *Let k be a natural number, and suppose that $|\mathcal{I}| \geq k$. The top-k set of X, denoted* $\mathrm{top}(k, X)$*, is a subset of $\mathcal{I}$ of size k containing the items with the maximal weights.*

Following [11], we extend Definition 1 to the concept of approximate top-k sets.

Definition 2. *Let $\varepsilon \geq 0$. An ε-approximation of the top-k set of X is a set* top_ε *of k items, such that for all items $i \in \mathrm{top}_\varepsilon$ and $j \notin \mathrm{top}_\varepsilon$, we have $p_X(j) \leq (1+\varepsilon) p_X(i)$.*

We will mainly be interested in small values of ε so without loss of generality, we assume henceforth that $\varepsilon \leq 1$.

It turns out that the following quantity has a central role in the complexity of computing top-k (and approximate top-k) queries. For a given input, the *critical frequency* of the instance, denoted $p^*(X, k)$, is the empirical probability of the least popular item in $\mathrm{top}(k, X)$, i.e., given input X and a natural number k, we define $p^*(X, k) \triangleq \min\{p_X(i) \mid i \in \mathrm{top}(k, X)\}$.

Throughout the paper, we assume instances having n nodes, total weight W, and critical frequency p^*. We denote the set of all such instances by $\mathcal{X}(W, n, p^*)$.

Complexity Measures. We evaluate the performance of certain algorithms using a worst-case measure per node. Specifically, the communication complexity of an algorithm is the maximum, over all inputs and over all nodes, of the total number of bits transmitted and received by a node throughout the execution of the algorithm. Formally, $c_A(X, v)$ denotes the total number of bits transmitted and received by node v, throughout the execution of algorithm A, for the input

X; $c_A(X)$ denotes the maximal node-communication of algorithm A on input X, i.e., $c_A(X) \triangleq \max\{c_A(X,v) \mid v \in V\}$. Finally, given a collection $\mathcal{X}$ of possible inputs, $C_A(\mathcal{X})$ denotes the worst-case communication complexity of algorithm A over all inputs in $\mathcal{X}$, i.e., $C_A(\mathcal{X}) \triangleq \max\{c_A(X) \mid X \in \mathcal{X}\}$.

Note that our communication complexity measure is individual in the sense that we measure the maximal number of bits communicated by any single node. The motivation for such a measure is that in sensor networks, each node has an individual energy source, and the longevity of the system often depends on the longevity of the weakest sensors (see, e.g., [14]). Furthermore, assuming a spanning tree of bounded degree, we can disregard many aspects of wireless communication and focus on the net communication used by the algorithm.

Loglog Counting. Let us present a known result which we use. First we define the following concept.

Definition 3. *Let Z be a positive number we wish to estimate, let $\varepsilon \geq 0$ and $\sigma \geq 0$ be real numbers. A random variable $\hat{Z}$ is a (ε, σ^2)-estimate of Z if $\frac{1}{Z}|E[\hat{Z}] - Z| \leq \varepsilon$, and $\frac{1}{Z^2} Var[\hat{Z}] \leq \sigma^2$.*

Durand and Flajolet [10] prove a result which, specialized to our system model, can be stated as follows.

Fact 4 ([10]). *There exists an algorithm A_{loglog} which outputs an (ε, σ^2)-estimate of W with $\varepsilon = 10^{-6}$ and $\sigma = 1$, using $O(\log\log W)$ bits of communication.*

To get bounds that hold with high probability, we iterate Algorithm A_{loglog} and use Bernstein's Inequality (see, e.g., [9]).

Algorithm BoundCount (Input: ε, δ, i)

1. $M \leftarrow (6/\varepsilon^2) \ln 1/\delta$.
2. Broadcast a filtering message indicating that only input cells holding item i should be considered in Step 3.
3. for $\ell = 1$ to M, run A_{loglog} obtaining an independent estimate $\hat{w}_\ell$ of $w(i)$.
4. Output $\hat{w} \triangleq \frac{1}{M}\sum_{\ell=1}^{M} \hat{w}_\ell$.

Corollary 5 (high-probability estimates). *For $10^{-5} \leq \varepsilon \leq 1$ and $\delta > 0$, the output $\hat{w}$ of Algorithm BoundCount satisfies $\Pr\left\{\frac{1}{w(i)}|\hat{w} - w(i)| < \varepsilon\right\} \geq 1 - \delta$. The individual communication complexity of the algorithm is of order $O\left(\log|\mathcal{I}| + \frac{1}{\varepsilon^2}\log\frac{1}{\delta}\log\log w(i)\right)$. Also, if the algorithm ran M iterations in Step 3, then for any $\zeta > 10^{-5}$, $\Pr\left\{\frac{1}{w(i)}|\hat{w} - w(i)| < \zeta\right\} \geq 1 - \exp\left(-\Omega\left(M\zeta^2\right)\right)$.*

3 Algorithms

In this section we present our main result, namely a randomized algorithm for computing top-k. In our algorithm, the basic idea is to view each cell (representing a unit of weight, or score) as a "vote," and to *sample each vote independently.*

Thus, the expected number of sampled votes for candidate i is proportional to the total number of votes candidate i has in the input *regardless of their partition into nodes.* The sampling results provide a good indication which items are globally popular, so that counting can be applied only to these items.

Next, we need to determine the sample size. Let p^* denote the frequency of the least popular of the top-k items. Clearly, if we sample once, the sample size should be proportional to $O(1/p^*)$ (or else the sample will fail to find all top k items). A more refined analysis shows what should be the sample size as a function of p^*, the approximation parameter ε, and the confidence parameter δ. To deal with unknown p^*, we augment the basic sampling algorithm with a technique to find the right sample size. Intuitively, we have a simple test which can prove whether the sample size is sufficiently large; if it isn't, we double the sample size.

Finally, we address the issue of very small p^* values: while $\Omega(1/p^*)$ sample size cannot be avoided for worst-case inputs, a much better bound can be obtained if the popularity of items decreases relatively rapidly. Consider, for example, the case where the global scores are close to the geometric distribution, e.g., when the frequency of the ℓ^{th} popular element is about $2^{-\ell}$. Then we have $p^* = 2^{-k}$, and therefore the sample size should be $\Omega(2^k)$. This cost can be reduced exponentially by utilizing the following simple idea: Whenever a sample is taken, the top item in the sample is by far the most popular (it is expected to have half of the weight in the sample). Therefore it is safe to add the top item to the output list, remove it from further consideration, and take another sample *of the same size.* In the geometric case, this approach has communication cost linear in k. This intuition leads us to our final algorithm, called Algorithm R. In essence, the idea is to iteratively discover the very top items, "shave them off," and to continue recursively with the remainder of the population. The algorithm combines this idea with an additional way to verify that the top-k items have been discovered. Algorithm R is far better than naive sampling for some common input distributions, such as Zipf distribution.

We start, in Sect. 3.1, by describing the basic algorithm we later use as a building block. Section 3.2 presents Algorithm S which uses adaptive sample size. Section 3.3 presents Algorithm R, which is our main result.

3.1 Algorithm *B*: Basic Sampling

Consider basic sampling: if the top-k items occupy a constant fraction of the total weight, then a log-size sample is sufficient to detect them for any input size (the logarithm is of the inverse of the error probability).

It is convenient to first analyze the following sampling routine.

Algorithm A (Input: $\mathtt{P_{SAMPLE}}$)

1. The root sends $\mathtt{P_{SAMPLE}}$ to all other nodes.
2. Each node sends each cell to the root with probability $\mathtt{P_{SAMPLE}}$.

Lemma 6. *There exists a function* $S^*(p^*, \varepsilon, \delta) = \Theta\left(\frac{1}{p^*\,\varepsilon^2} \cdot \ln\frac{1}{p^*\,\delta}\right)$ *, such that for any input* $X \in \mathcal{X}(W, n, p^*)$*, the top-k elements of a random sample of size at least* $S^*(p^*, \varepsilon, \delta)$ *is a top-k* ε*-approximation of* X *with probability at least* $1 - \delta$*.*

The proof (like all others) is omitted from this extended abstract. Intuitively, the argument is as follows. Define a 'swap' to be a pair of items i, j such that i is more popular than j in the input but less popular than j in the sample. We identify which item-pairs may not be swapped in an ε-approximation, and bound the probability that such a swap occurs using the Chernoff-like bound for *self-weakening* random variables presented in [19]. The probability bound is used to deduce the required sample size. The exact definition of S^* used by our algorithms is $S^*(p^*, \varepsilon, \delta) \triangleq \frac{g(\varepsilon)}{p^*} \cdot \left(\ln\frac{1+\varepsilon}{p^*\delta} + 4\right)$, where $g(\varepsilon) \triangleq \frac{(1+\varepsilon)\,\ln(1+\varepsilon)}{\varepsilon\,\ln\left(\frac{\varepsilon}{\ln(1+\varepsilon)}\right)+\ln(1+\varepsilon)-\varepsilon}$.

Algorithm B, presented below, first determines the sampling probability $\mathtt{P_{SAMPLE}}$ using the function S^* from Lemma 6. Each cell (i.e., unit of weight) is then sent to the root with probability $\mathtt{P_{SAMPLE}}$, and the root outputs the top k items in the sample as an approximation of the top k items in the complete input.

Algorithm B (Input: $W, p^*, k, \varepsilon, \delta$)

1. The root computes $\mathtt{P_{SAMPLE}} \leftarrow 2 \cdot S^*(p^*, \varepsilon, \frac{\delta}{2})/W$.
2. Execute Algorithm A with parameter $\mathtt{P_{SAMPLE}}$ to get a sample $\mathcal{S}$.
3. Output $\mathrm{top}(k, \mathcal{S})$.

When running Algorithm B as described above, each sampled vote is sent to the root separately incurring communication $\log|\mathcal{I}|$. An obvious optimization is to aggregate votes for the same candidate along the way, for example by sending the *count* of votes for each candidate. While such optimization is very worthwhile to implement, it would not help much when the partition of votes to nodes is adversarial. We therefore ignore such optimizations in our upper bounds.

Theorem 7. *Let* $X \in \mathcal{X}(W, n, p^*)$ *be an instance. Provided that* p^**,* ε*,* δ *and* W *are known, Algorithm* B *outputs a top-k* ε*-approximation with probability at least* $1 - \delta$ *and communication* $O\left(\frac{1}{p^*\,\varepsilon^2} \cdot \ln\frac{1}{p^*\delta} \cdot \log|\mathcal{I}|\right)$*.*

The proof is rather standard; we omit it due to lack of space. Note the $1/p^*$ factor: It is unavoidable because if the sample is to contain the top k elements, it should contain the least popular of them, and hence the sample size must be $\Omega(1/p^*)$. A stronger bound is proved in Sect. 5.

3.2 Algorithm S: Adaptive Sample Size

Algorithm B requires knowing the values of W and p^*. While estimating W is straightforward and cheap (by deterministic or randomized counting, at the cost of $O(\log W)$ or $O(\log\log W)$ communication, respectively), obtaining a lower bound on p^* seems less trivial. We solve this problem as follows.

First, we note that by counting the weight of *any* k items, we obtain a lower bound on p^*: the least popular among any k items is certainly no more popular than the least popular among the top-k items. Second, we note that exact counting is not necessary: we can use high-probability estimates as described in Corollary 5 to get a lower bound on p^* that holds with high probability. However, if we simply use an arbitrary set of k items to bound p^*, that value can be smaller than the true value of p^* by an arbitrary factor, resulting in communication cost that is higher than the bound in Theorem 7 by an arbitrary factor.

Our solution, in Algorithm S below, combines the ideas described above with an iterative approach that avoids unbounded 'overshoots.' The algorithm uses a variable $\hat{p}$ as an estimate of p^*. The algorithm repeatedly halves $\hat{p}$ while improving its lower bound on p^*, stopping when $\hat{p}$ is smaller than the lower bound.

Algorithm S (Input: k, ε, δ)

1. $\hat{W} \leftarrow \text{BoundCount}(\varepsilon = 1, \delta/5, \text{'ALL'})$
2. $\hat{p} \leftarrow 1$ ($\hat{p}$ is the current estimate of p^*)
3. Repeat
   (a) $\hat{p} \leftarrow \frac{\hat{p}}{2}$
   (b) Execute Algorithm B with parameters $(\hat{W}/2, \hat{p}, \varepsilon, \delta/5)$, to get a candidate top-k-set T.
   (c) For each item $i \in T$,
   $\hat{w}(i) \leftarrow \text{BoundCount}(\varepsilon = 1, \delta/5, i)$
   Until $\hat{p} < \frac{1}{8} \min \left\{ \hat{w}(i)/\hat{W} \mid i \in T \right\}$.
4. Output the set T computed at Step 3b of the last iteration.

Theorem 8. *For any input $X \in \mathcal{X}(W, n, p^*)$, with probability at least $1 - \delta$, Algorithm S outputs a top-k ε-approximation with communication complexity* $C_S = O\left(\frac{1}{p^* \varepsilon^2} \cdot \log \frac{1}{p^* \delta} \cdot \log |\mathcal{I}| \; + \; k \log \frac{1}{p^*} \log \frac{1}{\delta} \log \log W\right)$.

Note that when $|\mathcal{I}| \geq \log W$ and $k < O\left(1/(-p^* \log p^*)\right)$, the communication complexity of Algorithm S is within a constant factor of the complexity of Algorithm B.

The general idea in the proof is that the high-probability estimates ensure that we run approximately $\log \frac{1}{p^*}$ iterations. As a result, the last iteration is similar to an execution of Algorithm B with the correct parameters and its communication complexity is as specified in Theorem 7.

3.3 Algorithm *R*: Sample and Remove

As already mentioned above, it appears that the $1/p^*$ factor is unavoidable when we want all the top-k items to be included in the sample (because p^* is the empirical probability of the k^{th} popular item). However, when the popularity of the popular items is far from uniform, one can do much better, as mentioned for the geometric distribution example in the beginning of this section: the $1/p^*$ factor in the communication complexity can be replaced by a factor of k. Algorithm R

Algorithm R (Input: k, ε, δ)

1. $Q \leftarrow \emptyset$ (Q holds all items whose weights were already estimates)
2. $\hat{p} \leftarrow 1$, $\ell \leftarrow 0$. ($\hat{p}$ is the current estimate of p^*, ℓ is the iteration index)
3. Repeat
   (a) $\hat{W} \leftarrow \text{BoundCount}(\varepsilon = 1, \delta/7, \text{'ALL'})$.
   (b) $\ell \leftarrow \ell + 1$; $\hat{W}[\ell] \leftarrow \hat{W}$.
   (c) $\hat{p} \leftarrow \hat{p} \cdot \frac{1}{2} \cdot \hat{W}[\ell - 1]/\hat{W}[\ell]$.
   (d) $\texttt{P}_{\texttt{SAMPLE}} \leftarrow 2S^*(\hat{p}, \varepsilon/4, \delta/7) \,/\, \hat{W}$.
   (e) Execute Algorithm A with parameter $\texttt{P}_{\texttt{SAMPLE}}$ to get sample $\mathcal{S}$.
   (f) $\eta \leftarrow (\delta/7) \cdot (1/ - \log p_{lo}(\text{top}(k, Q), \hat{W}))$;
   $T^{\mathcal{S}} \leftarrow \text{top}(|\mathcal{S}| \log |\mathcal{I}| \,/ (\log |\mathcal{I}| + \log\log \hat{W}),\ \mathcal{S})$.
   (g) For each $i \in T^{\mathcal{S}}$, $\hat{w}(i) \leftarrow \text{BoundCount}(\varepsilon = 1, \eta, i)$.
   (h) $Q \leftarrow Q \cup T^{\mathcal{S}}$; remove elements of $T^{\mathcal{S}}$ from the input.
   Until $\texttt{safe}(\hat{W}, Q, \mathcal{S}, \hat{p}, \varepsilon, \delta)$.
4. If $|T^{\mathcal{S}}| < k$, then for each $i \in \text{top}(k, \mathcal{S})$, do $\hat{w}(i) \leftarrow \text{BoundCount}(\varepsilon = 1, \eta, i)$ and add i to Q.
5. $Q \leftarrow \left\{ i \in Q \mid \hat{w}(i)/\hat{W} > p_{lo}(\text{top}(k, Q), \hat{W}) \right\}$
6. For each $i \in Q$, $\hat{w}(i) \leftarrow \text{BoundCount}(\varepsilon/4, \frac{\delta}{5|Q|}, i)$.
7. Output the top k items in Q according to the new estimates.

Function p_{lo} (Input: Q, W)

1. Return $\frac{1}{4W} \min\{\hat{w}(i) \mid i \in Q\}$.

Predicate `safe` (Input: $W, Q, \mathcal{S}, \hat{p}, \varepsilon, \delta$)

1. If $\hat{p} \cdot (W[\ell]/W[1]) < p_{lo}(\text{top}(k, Q), W)$ return TRUE.
2. $r_{hi} \leftarrow p_{lo}(\mathcal{S} \cap Q, W)$
3. $q \leftarrow (1 + \varepsilon/2) \cdot p_{lo}(\text{top}(k, Q), W)$; $\alpha \leftarrow q/r_{hi}$.
4. If $\alpha > 1$ AND $\exp\left(-|\mathcal{S}| q \left(\frac{\alpha-1}{\alpha}\right)^2\right) < \frac{q\delta}{5}$ then return TRUE else return FALSE.

Fig. 1. Algorithm R

achieves this improvement for both geometric and Zipf distributions. In a nutshell, the idea is still to cut the estimate of p^* by half in each iteration; however, while Algorithm S achieves this by doubling the sample size, Algorithm R tries also to reduce the size of the relevant population.

In more detail, the algorithm works as follows (see Fig. 1). In each iteration, the algorithm samples with probability $\texttt{P}_{\texttt{SAMPLE}}$, obtained from the current estimate $\hat{p}$ of p^*. Then, in Step 3f, the algorithm counts (approximately) some of the top items in the sample: the number of items is such that the cost of counting equals the cost of sampling. The counted items (recorded in Q) are not considered part of the input anymore. In Step 3c, $\hat{p}$ is adjusted so that its value *in the full input* is halved (but since the input is smaller now, $\hat{p}$ is multiplied by a factor larger than 1/2). The loop is executed until one of the stopping rules specified in Predicate `safe` is met. These rules use a lower bound on p^* computed by function p_{lo}: it is a high-probability lower bound on the k^{th} smallest

value among all values counted so far. Using p_{lo}, the stopping rules are defined as follows.

First (Step 1 of Predicate `safe`), we can stop if the value of $\hat{p}$ w.r.t. the full input is smaller than the lower bound on p^*. This test is done in Step 1. Second, we bound the probability that an "important" item did not arrive at the top of the current sample and therefore was not counted. More specifically, we bound the probability that an item with frequency q, where q is sufficiently larger than p^*, was not counted, while an item whose frequency is r_{hi} was counted. If the probability of this event is low, we know that the top items counted by the algorithm contain an ε-approximation to the top-k set (Step 4). This test is useless in some cases (say, the uniform distribution), but it is effective in some distributions (such as Zipf).

Theorem 9 shows some general bounds on its complexity. As expected, it shows that for some inputs, Algorithm R has very little advantage over Algorithm S. Theorem 10, however, presents an alternative analysis which, when applied to some natural input distributions, attains much better bounds than those of Theorem 9.

Theorem 9. *For any input $X \in \mathcal{X}(W, n, p^*)$, with probability at least $1-\delta$, Algorithm R outputs a top-k ε-approximation while the communication complexity satisfies $C_R = O\left(\frac{1}{p^*\varepsilon^2} \log \frac{1}{\delta p^*} (\log |\mathcal{I}| + \log\log W)\right)$.*

Again, when $|\mathcal{I}| < \log W$, the communication complexity of Algorithm R is within a constant factor of the complexity of Algorithm B.

Next we refine the analysis to depend more closely on the global distribution (rather than only on p^*). Let π_ℓ denote the probability that a random item has global frequency at most $2^{-\ell}$, namely π_ℓ is the fraction of the votes for candidates whose popularity is at most $2^{-\ell}$.

Theorem 10 (Main result). *Let $X \in \mathcal{X}(W, n, p^*)$. Then with probability at least $1-\delta$, Algorithm R outputs a top-k ε-approximation of X while using $O\left(\frac{1}{\varepsilon^2} \log \frac{1}{p^*\,\delta} \; (\log |\mathcal{I}| + \log\log W) \cdot \sum_{\ell=1}^{\log \frac{1}{p^*}} 2^\ell \pi_\ell\right)$ communication bits per node.*

Using Theorem 10, we can prove the following corollaries for specific distributions. The proof of Corollary 11 is straightforward; the proof of Corollary 12 is based on approximating the Zipf distribution by the Pareto distribution.

Corollary 11. *Let X be an instance where item frequencies are geometrically distributed, i.e., $p_X(i) = (1-\lambda)\lambda^{i-1}$ for some constant $0 < \lambda < 1$. Then Algorithm R, when run on input X with parameters $\delta, \varepsilon > 0$ has communication complexity $c_R(X) = O\left(\frac{k}{\varepsilon^2} \log \frac{1}{p^*\,\delta} (\log |\mathcal{I}| + \log\log W)\right)$.*

For the important case of Zipf distribution, we have the following corollary.

Corollary 12. *Let X be an instance where item frequencies have Zipf distribution with parameter $a > 1$, i.e., $p_X(i) \triangleq \dfrac{i^{-a}}{h}$, where $a > 1$ and h is the normalization*

factor. Then Algorithm R, when run on input X with parameters $\delta, \varepsilon > 0$ *has communication complexity* $c_R(X) = O\left(\frac{a+1}{a-1} \cdot \frac{k}{\varepsilon^2} \cdot \log \frac{1}{p^* \delta} \cdot (\log |\mathcal{I}| + \log \log W)\right)$.

The results in this section analyze the behavior of R for asymptotically large inputs. As described in the next section, we have tested our algorithms on realistic-size inputs to study the effect of the hidden constants. We have also compared R with other known algorithms from the literature. We remark that Algorithm R turns out to have significantly superior performance.

4 Simulations Results

To evaluate the performance of our algorithm in more realistic scenarios, we ran simulations examining the actual number of bits communicated throughout the execution. We compared our algorithm R with the best known algorithms, namely TA [11] and TPUT [6]. We compared the performance of the algorithms when the input is partitioned randomly and when it is partitioned adversarially. Finally, we evaluated the performance of R for various values of p^* (popularity of the k^{th} most popular item) and various values of ϵ (the approximation parameter). Informally, we find that Algorithm R offers a dramatic improvement over TA and TPUT unless the system is very small and the distribution is particularly favorable to TA and TPUT. Moreover, for typical inputs, R behaves better than predicted by our analytical bounds.

Simulated Instances. We used randomly generated inputs of various sizes, and our main focus was on inputs generated by Zipf distribution with exponent between 0.8 and 3; all shown charts used Zipf distribution with exponent 1.5 (different exponent values showed qualitatively similar behavior). Typical other parameters were $k = 5$, $\varepsilon = 0.5$ and $\delta = 0.05$. The total number of votes (W) was either 10^6 or 10^7 and the varying parameter is the number of nodes (n). The number of candidate-items was equal to the number of nodes ($|\mathcal{I}| = n$). All simulations measured the worst-case individual communication (in bits).

Comparison with Existing Algorithms. In Fig. 2 we compare algorithms R, TA and TPUT. We used the approximation version of the TA algorithm. We note

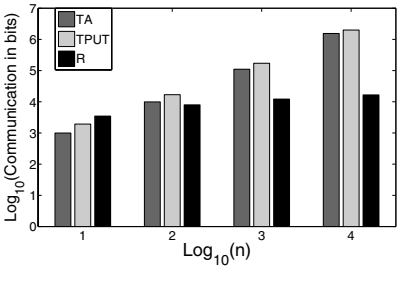

Fig. 2. *Communication costs of algorithm R compared to TA and TPUT for random partition of scores in various system scales*

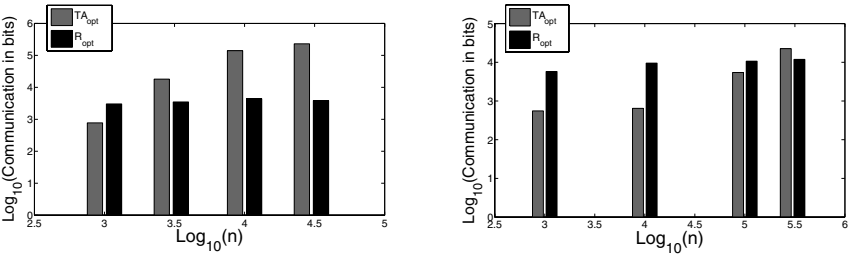

Fig. 3. *Communication costs of optimized algorithms for various system scales. Left: with an adversarial partition of scores among nodes. Right: with a random partition of scores among nodes.*

that TPUT has no approximation version, but in all simulations we ran, TPUT appeared less efficient than TA, even for high accuracy such as $\varepsilon = 0.01$.

Results for adversarial and random partition of scores were similar. As shown in Fig. 2, the performance of TA and TPUT for a random partition is rather poor for large systems and Algorithm R is superior even for a system of 100 nodes. R is overwhelmingly superior (by more than two orders of magnitude) for $n > 10000$.

Next, we modified the deterministic algorithms by using loglog counting and allowing them to err. We also assumed the existence of a bounded degree spanning tree, and used unlimited memory at each node (which allows simple aggregation while traversing the tree). These modifications resulted in drastic improvement in TA performance: see Fig. 3. The optimized version of TPUT is not shown because it performed far worse than the optimized version of TA. Note that when the input is adversarially partitioned, R beats TA for $n > 2000$. But even when the input is partitioned randomly (which is close to the best case of TA), R performs better for n large enough, where large enough means $n > 200000$ in this case.

The Effect of Accuracy and Critical Frequency on Performance. We examined the performance of Algorithms R and algorithm S while varying ε and p^*

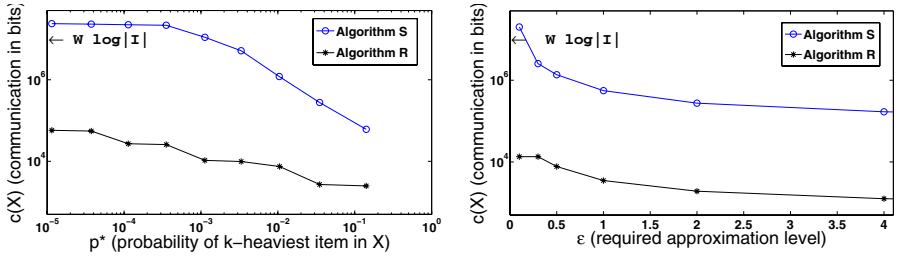

Fig. 4. *Communication costs of algorithms S and R. Left: as a function of p^*. Right: as a function of ε. The communication cost of reading the entire input is $W \log |\mathcal{I}|$, (W is the input size and $|\mathcal{I}|$ is the number of items).*

(for n fixed). The results are given in Fig. 4. Obviously, Algorithm R was always superior to Algorithm S. The results for p^* (Fig. 4, left) show that Algorithm R is less sensitive to p^* when the input is generated by Zipf distribution. We note that the flat left segment in the graph of S is due to the fact that for small values of p^*, the sample of Algorithm S consists of the entire input.

The effect of the approximation factor ε (Fig. 4, right) on the communication of both algorithms is proportional to $\left(\frac{1+\varepsilon}{\varepsilon}\right)^2$, as suggested by our upper bounds. Algorithm R is more efficient due to its improved termination criterion.

5 A Lower Bound on the Total Cell-Probe Complexity

In this section we give a lower bound on the *total cell probe complexity* [23, 13] of randomized algorithms for the top-k problem. For a restricted settings, we can deduce a lower bound on the individual communication complexity of such randomized algorithms.

We define the total cell probe complexity of an algorithm as the total number of input cells accessed by a all nodes for the worst-case input. Formally, $\mathrm{cp}_A(X, v)$ denotes the number of input cells accessed by node v, throughout the execution of A for input X; $\mathrm{cp}_A^T(X)$ denotes the total number of cell-probes for an instance X, i.e., $\mathrm{cp}_A^T(X) \triangleq \sum_{v \in V} \mathrm{cp}_A(X, v)$. The total cell-probe complexity of A is defined by $\mathrm{CP}_A^T(\mathcal{X}) \triangleq \max\left\{\mathrm{cp}_A^T(X) \mid X \in \mathcal{X}\right\}$. Note that the set of all probed cells can be viewed as a sample whose size is the total number of cell-probes.

Theorem 13. *For some natural k, let $\varepsilon > 0$, $0 < p^* \leq \frac{1}{2k}$, $0 < \delta < 0.014$, and $W = \Omega\left(\frac{1}{\delta(p^*)^2} \cdot \frac{(1+\varepsilon)^2}{\varepsilon^4}\right)$ and let A be any Monte Carlo algorithm. If A outputs a top-k ε-approximation with probability at least $1-\delta$ for any input in $\mathcal{X}(W, n, p^*)$, then its total cell-probe complexity satisfies $E\left[\mathrm{CP}_A^T\right] = \Omega\left(\frac{1}{p^*\,\varepsilon^2} \cdot \log\frac{1}{\delta}\right)$.*

The proof (omitted from this extended abstract) applies Yao's Minimax Principle to obtain a lower bound on Monte Carlo algorithms. To this end, we prove a lower bound on the expected sample size used by any deterministic algorithm when the input is drawn from a certain distribution we construct, so that there is a single correct output to the top-k ε-approximation query. Furthermore, under our distribution, the local view of each node is just a random subset of the items. We show that any algorithm which does not output the top-k set of its sample is doomed to err with probability at least $1/2$. Finally, we bound the required sample size for other algorithms.

Star Topology and "Smart Dust" Systems. Theorem 13 demonstrates the optimality of Algorithm B with respect to the *total* cell-probe complexity but not the *individual* cell-probe complexity. Consider a setting where the system has star topology, i.e., all nodes are connected to a root node, and suppose that each node holds a single cell. This model is a reasonable abstraction of the setting in sensor networks using *passive communication*. In these systems, the only communication is between a powered base-station and a sensor, forming a star topology

(as opposed to the common spanning tree). Passive communication is suitable for very small devices, such as "smart dust" systems (see e.g., [22]). Now, for these systems, Theorem 13 says that any algorithm that satisfies the conditions of the theorem has individual communication complexity at least $\Omega\left(\frac{1}{p^* \varepsilon^2} \cdot \log \frac{1}{\delta}\right)$. We stress that while our model assumes certain routing capabilities, our algorithms only use a broadcast-convergecast scheme rooted by the root node. Such communication is suitable for star topology.

6 Conclusions and Future Work

In this paper we have proposed algorithms solving the top-k problem by adaptive sampling. The communication complexity of our algorithms does not depend on the way the input is partitioned in the network: only the global statistics affect the complexity. Our final algorithm performs particularly well when the global statistics are far from flat. We have tested our algorithm by simulation and found empirical support for our analytical claims. Although our study is mainly theoretical, simulation results indicate that our algorithm is rather practical and can be very useful in real-life scenarios. Future work may extend our algorithm to specific models, where spatial and temporal dependence among different sensors holds (e.g., nearby nodes have similar readings).

From the theoretical viewpoint, we think that it is very interesting to extend our lower bound to the case of interactive algorithms. We conjecture that our Algorithm R is nearly optimal in this more general model.

References

[1] H. Attiya and J. Welch. *Distributed Algorithms.* McGraw-Hill Publishing Company, UK, 1998.
[2] B. Babcock and C. Olston. Distributed top-k monitoring. In *Proc. 2003 ACM SIGMOD.*
[3] P. Bak. *How Nature Works: The science of self-organized criticality.* Springer-Verlag, New York, 1996.
[4] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive distributed top k retrieval in peer-to-peer networks. In *Proc. 21st Int. Conf. on Data Engineering*, 2005.
[5] N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accessible databases. In *Proc. 18th Int. Conf. on Data Engineering*, 2002.
[6] P. Cao and Z. Wang. Efficient top-k query calculation in distributed networks. In *Proc. 23rd Ann. ACM Symp. on Principles of Distributed Computing*, 2004.
[7] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. Apr. 2004.
[8] G. Cormode, M. N. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *Proc. 2005 ACM SIGMOD*, 2005.
[9] P. Dagum, R. M. Karp, M. Luby, and S. Ross. An optimal algorithm for Monte Carlo estimation. *SIAM J. Comput.*, 29(5), 2000.

[10] M. Durand and P. Flajolet. Loglog counting of large cardinalities (extended abstract). In *Algorithms: ESA 11th Ann. European Symp.*, 2003.
[11] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proc. 20th ACM Symp. on Principles of Database Systems*, 2001.
[12] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proc. SIGCOMM '99*, New York, NY, USA. ACM Press.
[13] M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, May 1989.
[14] M. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proc. 23rd ACM Symp. on Principles of Database Systems*, 2004.
[15] N. Lynch. *Distributed Algorithms.* Morgan Kaufmann, San Mateo, CA, 1995.
[16] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proc. 2003 ACM SIGMOD.*
[17] S. Michel, P. Triantafillou, and G. Weikum. Klee: A framework for distributed top-k query algorithms. In *Proc. 31st Int. Conf. on Very Large Data Bases*, 2005.
[18] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *SenSys '04: Proc. 2nd international conference on Embedded networked sensor systems*, 2004.
[19] A. Panconesi and A. Srinivasan. Fast randomized algorithms for distributed edge coloring (extended abstract). In *Proc. 11th Ann. ACM Symp. on Principles of Distributed Computing*, 1992.
[20] B. Patt-Shamir. A note on efficient aggregate queries in sensor networks. In *Proc. 23rd Ann. ACM Symp. on Principles of Distributed Computing*, 2004.
[21] A. Silberstein, R. Braynard, C. Ellis, K. Munagala, and J. Yang. A sampling-based approach to optimizing top-k queries in sensor networks. In *Proc. 22nd Int. Conf. on Data Engineering*, 2006.
[22] B. Warneke. Miniaturizing sensor networks with mems. In M. Ilyas and I. Mahgoub, editors, *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems.* CRC Press, 2004.
[23] A. C.-C. Yao. Should tables be sorted? *J. ACM*, 28(3), 1981.
[24] Y. Yao and J. Gehrke. The Cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record*, 31(3):9–18, Sept. 2002.
[25] D. Zeinalipour-Yazti, Z. Vagena, D. Gunopulos, V. Kalogeraki, V. Tsotras, M. Vlachos, N. Koudas, and D. Srivastava. The threshold join algorithm for top-k queries in distributed sensor networks. In *Proc. 2nd Int. Workshop on Data Management for Sensor Networks*, 2005.