Brief Announcement: Efficient Distributed Algorithms for the K-Nearest Neighbors Problem

Reza Fathi
University of Houston
Houston, Texas, USA.
rfathi@uh.edu

Anisur Rahaman Molla*
Indian Statistical Institute
Kolkata, India
molla@isical.ac.in

Gopal Pandurangan†
University of Houston
Houston, Texas, USA.
gopalpandurangan@gmail.com

ABSTRACT

The K-nearest neighbors is a basic problem in machine learning with numerous applications. In this problem, given a (training) set of n data points with labels and a query point q, we want to assign a label to q based on the labels of the K-nearest points to the query. We study this problem in the *k-machine model*,[1] a model for distributed large-scale data. In this model, we assume that the n points are distributed (in a balanced fashion) among the k machines and the goal is to compute an answer given a query point to a machine using a small number of communication rounds.

Our main result is a randomized algorithm in the k-machine model that runs in $O(\log K)$ communication rounds with high success probability (regardless of the number of machines k and the number of points n). The message complexity of the algorithm is small taking only $O(k \log K)$ messages. Our bounds are essentially the best possible for comparison-based algorithms. We also implemented our algorithm and show that it performs well in practice.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**; • **Mathematics of computing → Probabilistic algorithms**; *Discrete mathematics*;

KEYWORDS

K-Nearest Neighbors, Randomized selection, k-Machine Model, Distributed Algorithm, Round complexity, Message complexity

ACM Reference format:
Reza Fathi, Anisur Rahaman Molla, and Gopal Pandurangan. 2020. Brief Announcement: Efficient Distributed Algorithms for the K-Nearest Neighbors Problem. In *Proceedings of Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15–17, 2020 (SPAA '20),* 3 pages.
https://doi.org/10.1145/3350755.3400268

*Research supported by DST Inspire Faculty research grant DST/INSPIRE/04/2015/002801.
†Supported, in part, by NSF grants IIS-1633720, CCF-1540512, and CCF-1717075, and by BSF grant 2016419.
[1]Note that parameter k stands for the number of machines in the k-machine model and is independent of K-nearest points.

SPAA '20, July 15–17, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-6935-0/20/07.
https://doi.org/10.1145/3350755.3400268

1 INTRODUCTION

The K-nearest neighbors is a well-studied problem in machine learning with numerous applications. (e.g., [10]). It is a non-parametric method used for classification and regression, especially in application such as pattern recognition. The algorithmic problem is as follows. We are given a (training) set of n data points (n can be potentially very large and/or each point can be in a high dimensional space) with labels and a query point q. The goal is to assign a label to q based on the labels of the *K-nearest points to the query*. In the *classification* problem, one can use the majority of the labels of the K-nearest neighbors to assign a label to q. In the *regression* problem, one can assign the average of the labels (assuming that these are values) to q.

In this paper (see full version [4]), we study distributed algorithms for the K-nearest neighbors problem motivated by Big Data and privacy applications. When the data size is very large or naturally distributed at k-sites (e.g., patients data in different hospitals), then distributed computation using multiple machines is helpful.

Model: We study the K-nearest neighbors problem in the *k-machine model*, a model for distributed large-scale data. (Henceforth, to avoid confusion, between K and k, which are unrelated we will say ℓ-nearest neighbors). The k-machine model was introduced in [5] and further investigated in [1, 2, 7, 8]. The model consists of a set of $k \geq 2$ machines $\{M_1, M_2, \ldots, M_k\}$ that are pairwise interconnected by bidirectional point-to-point communication links. Each machine executes an instance of a distributed algorithm. The computation advances in synchronous rounds where, in each round, machines can exchange messages over their communication links and perform some local computation. Each link is assumed to have a bandwidth of B bits per round, i.e., B bits can be transmitted over each link in each round; unless otherwise stated, we assume $B = \Theta(\log n)$. Machines do not share any memory and have no other means of communication. We assume that each machine has access to a private source of true random bits.

Local computation within a machine is considered to happen instantaneously at zero cost, while the exchange of messages between machines is the costly operation. However, we note that in all the algorithms of this paper, every machine in every round performs lightweight computations. The goal is to design algorithms that take as few communication rounds as possible.

The Selection Problem: We note that the ℓ-nearest neighbors problem really boils down to the *selection* problem, where the goal is to find the ℓ-smallest value in a set of n values. The selection problem has a (somewhat non-trivial) linear time deterministic algorithm [3] as well as simple randomized algorithm in the sequential

setting. For the ℓ-nearest neighbors, one can reduce it to the selection problem by computing the distance of the query point to all the points and then finding the ℓ-smallest distance among these n distance values. All these can be done in $O(n)$ time sequentially.

Our Results: In this paper, we present efficient bounds for the ℓ-nearest neighbors or equivalently to the ℓ-selection problem. Our main result is a randomized algorithm in the k-machine model that runs in $O(\log \ell)$ communication rounds with high probability (regardless of the number of machines k). The message complexity of the algorithm is also small taking only $O(k \log \ell)$ messages. Note that if ℓ is not very large (which is generally true in practice), then these bounds imply very fast algorithms requiring only a small number of rounds regardless of the number of points and the number of sites (machines).

Our bounds are essentially the best possible for comparison-based[2] algorithms, i.e., algorithms that use only comparison operations ($\leq, \geq, =$) between elements to distinguish the ordering among them. This is due to the existence of a lower bound of $\Omega(\log n)$ communication rounds (if only one element is allowed to be exchanged per round) for finding the *median* of $2n$ elements distributed evenly among two processors [9].

We also implement and test our algorithm in a distributed cluster, and show that it performs well compared to a simple algorithm that sends ℓ nearest points from each machine to a single machine which then computes the answer.

Definitions: We use the notation $dis(p, q)$ to denote the distance between two given points p and q where it can be any absolute norm $||p - q||$.

Definition 1.1 (ℓ-NN problem). Given an input data set D, a query data point q, and a number ℓ while $\ell \leq |D|$, the ℓ-Nearest Neighbors (ℓ-NN) problem is finding a set of data points S such that $(S \subset D) \wedge (|S| = \ell) \wedge (dis(p_i, q) \leq dis(p_j, q), \forall p_i \in S, p_j \in D \setminus S)$.

2 THE ALGORITHM

First we present a distributed algorithm to solve a more general *selection* problem: finding ℓ-smallest points among n points. Suppose n points are distributed over k machines arbitrarily. The problem is to find the ℓ-smallest points among those n points. In the end, each machine i outputs a set of points S_i such that $\cup_{i=1}^{k} S_i$ contains the ℓ-smallest points. Then we use this algorithm to solve the ℓ-nearest neighbors problem. For simplicity, let us assume that the points are all distinct; later we explain a simple extension in the algorithm to work for non-distinct points set. To solve this problem we implement the idea of randomized selection in the k-machine model.

We point out an implementation issue on the size of the messages used by our algorithm for the nearest neighbors problem. For the purpose of analysis, we can assume that each point (or value) is of size $O(\log n)$ bits and hence can be sent through an edge per round in the k-machine model. However, for the ℓ-nearest neighbor problem, points can be high-dimensional and can incur a lot of bits. But it is easy to see that one need not actually transfer points, but only *distances* between the query point to the given (training set) points. In fact, one can use randomization to choose a unique ID for each of the n points (choose a random number between say $[1, n^3]$ and they will be unique with high probability). Then one needs to transfer only the ID of the point (of size $O(\log n)$ bits) and its corresponding value (distance between the point and the query point) which we assume can be represented in $O(\log n)$ bits, i.e., all distances are polynomial in n.[3] Note that choosing unique IDs also takes care of non-distinct points as we can use IDs to break ties between points of equal distances.

2.1 Distributed Selection Algorithm

This algorithm is a distributed implementation of a well-known randomized (sequential) selection algorithm (see e.g.,[3]). The algorithm first elects a leader machine (among the k machines) which propagates the queries and controls the search process. Since the machines have unique IDs, the leader (say, the minimum ID machine) can be elected in a constant number of rounds and $O(\sqrt{k} \log^{3/2}(k))$ messages [6]. The leader repeatedly computes a random pivot which partitions the points set into two parts and reduces the search space, i.e., the set of points on which the algorithm executes. Let us now discuss how the leader computes a random pivot and partitions the search space in $O(1)$ rounds. This constitutes one "iteration" of the selection algorithm. The leader maintains two boundary variables, namely, min and max such that the search points belong to the range [min, max]. Initially, min and max are assigned respectively the minimum (denoted by min) and maximum (denoted by max) value among all the data points. Notice that the leader can get this global minimum and maximum point by asking all the machines their local minimum and maximum in 2 rounds.

The leader asks the number of points that each machine holds in the range [min, max]. The leader randomly picks a machine i with probability proportional to the number of points a machine holds within the range of [min, max], i.e., with probability $n_i / \sum_{i=1}^{k} n_i$, where n_i is the number of points machine i holds in the range. The selected machine i chooses a point p randomly from its set of points in the range [min, max]. Then it replies back to the leader machine with the pivot p. In the next round, the leader asks the number of points each machine holds within the range [min, p]. Then it gathers all machines' count n_i and accumulates it to $s = \sum_{i=1}^{k} n_i$. If $s = \ell$, it found the correct upper boundary value and terminates the search process. If $s < \ell$, it means the algorithm needs to increase the lower boundary min to p and adjust the ℓ value by subtracting s from ℓ, i.e., $\ell = \ell - s$. On the other hand, if $s > \ell$, it can discard all the points greater than p by setting max to p. The leader iterates this process until it finds the correct upper boundary. Once the leader finds the correct upper bound (max), it broadcasts a 'finished' message with parameter max so that each machine outputs all the points less than or equal to max from its input set.

Correctness: In Lemma 2.1, we show that the leader machine computes the pivot p uniformly at random among all the search points in the range. The algorithm updates boundary values min, max and the ℓ-value according to the randomized selection algorithm. The boundary initialization makes sure that it includes all the data points in the beginning. Thus the algorithm correctly computes the ℓ-smallest points.

[2] We conjecture that the lower bound holds even for non-comparison based algorithms.

[3] We note that if distances are very large, one can use scaling to work with approximate distances which will be accurate with good approximation.

Algorithm 1 Distributed ℓ-NN Computation

Input: Query point q, the parameter ℓ.
Output: ℓ-nearest neighbors to the query point q.

1: Elect a leader machine among k machines (using the leader election algorithm in [6]).
2: If a machine i has more than ℓ data points, it keeps ℓ points whose distance from q is minimum and discards other points. Let's denote this remaining points set by S_i.
3: Each machine i samples $12\log(\ell)$ points randomly and independently from the set S_i.
4: Each machine sends its sampled points to the leader machine.
5: Leader sorts these $12k\log(\ell)$ based on their distance from q and stores in an array. Let r be the point at index $21\log(\ell)$ in the sorted array.
6: Leader broadcasts point r.
7: Each machine i removes any point larger than r from the set S_i.
8: Each machine i computes $d_{ij} = dis(p_{ij}, q)$ for all $p_{ij} \in S_i$ and stores them as $(p_{ij},\ d_{ij})$.
9: The leader machine runs the distributed selection algorithm where the input to the algorithm is those d_{ij} points.
10: Each machine outputs the p_{ij} points corresponding to the output points d_{ij} of the selection algorithm.

Lemma 2.1. *The leader machine selects the pivot p uniformly at random from all the points in the range* [min, max].

Using the above lemma, we show (in the full paper [4]) that the number of elements in the search process (i.e., in the range [min, max]) drops by a constant factor with constant probability. This implies that the algorithm stops in $O(\log n)$ rounds with high probability.

Theorem 2.2. *The above selection Algorithm computes the ℓ-smallest points among the n points in the k machine model in $O(\log n)$ rounds with high probability, and incurs $O(k \log n)$ messages with high probability.*

2.2 Distributed ℓ-NN Algorithm

We extend the above algorithm to compute ℓ-nearest neighbors (or, ℓ-NN) of a given query point q from a large data set D distributed over the k machines. Assume the machine i gets the set of points D_i as input. We assume that $|D_i| \leq \ell$ for all the machines, since, if a machine i gets more than ℓ data points as input, it keeps only ℓ points whose distance from q is minimum and discards the rest of the data points. Each machine i locally computes the distance $d_{ij} = dis(p_{ij}, q)$ such that all the points $p_{ij} \in D_i$ and maintains the pair $(p_{ij},\ d_{ij})$. Then we apply the selection algorithm on the distance values $\cup_{i=1}^{k} d_{ij}$ and output the corresponding points p_{ij}s. This takes $O(\log(k\ell)) = O(\log k + \log \ell)$ rounds, since the number of candidate points is at most $k\ell$.

We now present a randomized algorithm (Algorithm 1) whose running time is $O(\log(\ell))$ rounds, which is independent of the number of machines k. The main idea of the algorithm is to apply a sampling technique to reduce the search space (i.e., candidate points) from (at most) $k\ell$ to $O(\ell)$. Then we apply the distributed selection algorithm on these reduced set of candidate points to obtain our main result (proof in full paper [4]).

Theorem 2.3. *Algorithm 1 computes ℓ-NN in $O(\log(\ell))$ rounds and uses $O(k \log(\ell))$ messages with high probability.*

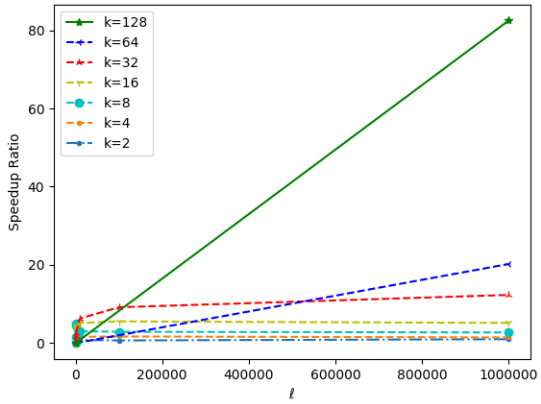

Figure 1: Run-time performance of our algorithm 1 compared to the simple method. X-axis shows the number of ℓ-nearest neighbors w.r.t. a query point and Y-axis shows the execution time ratio of the simple method over our algorithm 1. It shows that the higher the ratio, the higher the algorithm's speedup.

3 EXPERIMENTAL RESULTS

We ran the Algorithm 1 using Crill cluster from the University of Houston [4] which has 16 NLE Systems nodes. Each node has four 2.2 GHz 12-core AMD Opteron processor (48 cores total) and 64 GB main memory. We used a (synthetic) random data set. Each process generated 2^{22} random points independently between 0 and $2^{32} - 1$.

We compare the performance of our ℓ-NN algorithm with the following simple method: each machine finds its local ℓ-NN. Then it transfers all of them to a leader machine that finds the final ℓ-NN among those points. For each simulation, the leader machine chooses a random number between 0 and $2^{32} - 1$ as the query point. We ran each simulation 30 times.

REFERENCES

[1] S. Bandyapadhyay, T. Inamdar, S. Pai, and S. V. Pemmaraju. Near-optimal clustering in the k-machine model. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, page 15. ACM, 2018.
[2] F. Chung and O. Simpson. Distributed algorithms for finding local clusters using heat kernel pagerank. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 177–189. Springer, 2015.
[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.
[4] R. Fathi, A. R. Molla, and G. Pandurangan. Efficient distributed algorithms for the k-nearest neighbors problem. *CoRR*, abs/2005.07373, 2020.
[5] H. Klauck, D. Nanongkai, G. Pandurangan, and P. Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 391–410. Society for Industrial and Applied Mathematics, 2015.
[6] S. Kutten, G. Pandurangan, D. Peleg, P. Robinson, and A. Trehan. Sublinear bounds for randomized leader election. *Theoretical Computer Science*, 561:134–143, 2015.
[7] G. Pandurangan, P. Robinson, and M. Scquizzato. Fast distributed algorithms for connectivity and mst in large graphs. In *In Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 429–438. ACM, 2016.
[8] G. Pandurangan, P. Robinson, and M. Scquizzato. On the distributed complexity of large-scale graph computations. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, pages 405–414. ACM, 2018.
[9] M. Rodeh. Finding the median distributively. *Journal of Computer and System Sciences*, 24(2):162–166, 1982.
[10] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[4] http://pstl.cs.uh.edu/resources/crill-access