Sublinear time approximation of the cost of a metric k-nearest neighbor graph*

Artur Czumaj†
University of Warwick
United Kingdom

Christian Sohler‡
University of Cologne
Germany

Abstract

Let (X, d) be an n-point metric space. We assume that (X, d) is given in the distance oracle model, that is, $X = \{1, \ldots, n\}$ and for every pair of points x, y from X we can query their distance $d(x, y)$ in constant time. A *k-nearest neighbor (k-NN) graph* for (X, d) is a directed graph $G = (V, E)$ that has an edge to each of v's k nearest neighbors. We use $\mathtt{cost}(G)$ to denote the sum of edge weights of G.

In this paper, we study the problem of approximating $\mathtt{cost}(G)$ in sublinear time, when we are given oracle access to the metric space (X, d) that defines G. Our goal is to develop an algorithm that solves this problem faster than the time required to compute G.

We first present an algorithm that in $\widetilde{O}_\varepsilon(n^2/k)$ time with probability at least $\frac{2}{3}$ approximates $\mathtt{cost}(G)$ to within a factor of $1 + \varepsilon$. Next, we present a more elaborate sublinear algorithm that in time $\widetilde{O}_\varepsilon(\min\{nk^{3/2}, n^2/k\})$ computes an estimate $\overline{\mathtt{cost}}$ of $\mathtt{cost}(G)$ that satisfies with probability at least $\frac{2}{3}$

$$|\mathtt{cost}(G) - \overline{\mathtt{cost}}| \leq \varepsilon \cdot (\mathtt{cost}(G) + \mathtt{mst}(X)) \ ,$$

where $\mathtt{mst}(X)$ denotes the cost of the minimum spanning tree of (X, d).

Further, we complement these results with near matching lower bounds. We show that any algorithm that for a given metric space (X, d) of size n, with probability at least $\frac{2}{3}$ estimates $\mathtt{cost}(G)$ to within a $1 + \varepsilon$ factor requires $\Omega(n^2/k)$ time. Similarly, any algorithm that with probability at least $\frac{2}{3}$ estimates $\mathtt{cost}(G)$ to within an additive error term $\varepsilon \cdot (\mathtt{mst}(X) + \mathtt{cost}(X))$ requires $\Omega_\varepsilon(\min\{nk^{3/2}, n^2/k\})$ time.

1 Introduction

Computing or approximating nearest neighbors is a fundamental task in many areas of computer science, including machine learning, data mining and information retrieval and has been studied extensively (see [1, 7, 8, 14, 16] for a few examples). In many applications that involve nearest neighbors there is an input set of objects together with a distance or similarity measure and the idea is that objects that are near to each other are similar to each other. This is, for example, used in by the k-nearest neighbor algorithm, which predicts class labels based on the class labels of the k-nearest neighbors of the query object in a training set.

One way to describe nearest neighbor relations of a set of objects with a distance or similarity measure is to use the *k-nearest neighbor graph*. In this directed graph the vertices represent a set of input objects and there is a directed edge from v to u, if the object represented by u is among the k nearest neighbors of v.

The k-nearest neighbor graph (k-NN) is a fundamental data structure to represent proximity relations within a set of objects. It is used as part of the non-linear dimensionality reduction algorithm ISOMAP [29], which first computes a k-nearest neighbor graph (or, alternatively, an ε-neighborhood graph) then computes the shortest path distances between points and finally uses multi dimensional scaling to embed the points into fewer dimensions.

In unsupervised learning, the k-nearest neighbor graph is used in the context of spectral clustering (see, for example, the survey [30]). Spectral clustering describes a class of graph based clustering algorithm that exploit spectral properties of the graph Laplacian to compute the clusters (see, for example, [22, 28] for some well-known variants). If the set of input objects comes with a distance or similarity measure, then a standard approach is to run spectral clustering algorithms on the k-nearest neighbor graph.

Density estimation is an important topic in statistics (see, for example, the book [27]). It deals with the problem of estimating the density of a distribution from empirical samples. The k-nearest neighbor graph can be used in this context to provide a non-parametric approach (the underlying distribution is not described by a parameterized statistical model) for this problem. Essentially, the density is predicted from the distance of the k-th nearest neighbor and the dimensionality of the

*Full version is available on the first author's web page.

†Research partially supported by the Centre for Discrete Mathematics and its Applications (DIMAP), by IBM Faculty Award, and by EPSRC award EP/N011163/1.

‡Work partially done while the author was visiting researcher at Google Research, Switzerland.

space.

The variety of the above applications motivates us to study k-NN graphs as a fundamental graph structure. As already described above, computing a k-NN graph requires some distance or similarity measure. In this paper, we will assume that our points come from a discrete metric space and we are given access to the distances through a distance oracle, i.e., we assume that we have a description of the input point set and we can ask an oracle in constant time for the distance between any pair of input points. Since computing or approximating the k-NN graph in such a general setting requires quadratic time, we will consider the question whether it is at least possible to *approximate the weight* of the k-NN graph in subquadratic time, which would be sublinear in the full description size of the input space. We will also try to find natural conditions under which we can obtain a better approximation algorithm.

The study of the weight of the k-NN graph is motivated by its use in the context of estimation of basic statistical properties of a point set. For example, Costa and Hero [4] use the (appropriately scaled) weight of a k-NN graph for powers of Euclidean distances as an estimator for the intrinsic dimension and entropy of a data set. The related generalized nearest neighbor graph (which connects to a subset of the k-nearest neighbors) has been used as an estimator for entropy and mutual information (again after appropriate scaling) [25].

Since we cannot compute an approximate nearest neighbor graph in subquadratic time, for very large data sets we have to use a heuristic approach such as the NN-Descent algorithm by Dong et al. [8], that starts with a random graph and then locally improves the solution by considering neighbors of neighbors. While this and similar graph-based approaches have been empirically shown to perform well [8, 21], they do not come with guarantees. In order to evaluate the quality of the computed solution is it therefore helpful to be able to quickly approximate the cost of the k-NN graph.

Besides of these concrete applications, from a theoretical point of view, our studies are motivated by the general question of understanding how (and under which assumptions) we can estimate fundamental properties of very large structured data sets (in this case, metric spaces) from random samples.

In order to study such and similar questions, we pursue the following approach. We formulate the problem of computing a k-nearest neighbor graph as an optimization problem on a metric space (X, d) with the objective to compute a directed graph $G = (V, E)$ with vertex set $V = X$ such that every vertex has outdegree exactly k and such that the sum of edge weights is minimized. For a k-nearest neighbor graph G we will use $\mathtt{cost}(G)$ to denote the sum of its edge weights. In this paper we are interested in the question of approximating $\mathtt{cost}(G)$, when we are given oracle access to the metric space (X, d) that defines G.

1.1 Our results. It is not difficult to see that to compute exactly the cost of a k-nearest neighbor (k-NN) graph G of a metric space (X, d) one requires $\Theta(n^2)$ oracle access queries to the metric space (X, d) that defines G, where $n = |X|$. Similarly, finding even an approximation of a k-NN graph G of a metric space (X, d) also requires $\Theta(n^2)$ queries. However, we show that in a *sublinear time* one can find a very good approximation of $\mathtt{cost}(G)$, and we complement these results by showing that the complexity of our algorithms is almost optimal.

We develop two randomized *sublinear time* algorithms to approximate $\mathtt{cost}(G)$, the cost of a k-NN graph G of a metric space (X, d). We assume that our input is given in the *distance oracle model*, i.e., the set X of n input points is indexed from 1 to n and we can evaluate the distance between any pair of points in constant time per evaluation.

Notice that while the directed k-nearest neighbor graph G has nk edges, for a given metric space (X, d) graph G is given only implicitly, and a naïve algorithm to find G or even to compute $\mathtt{cost}(G)$ would query the distances between all $\Theta(n^2)$ pairs of points in X.

Our first algorithm performs best for large values of k (cf. **Theorem 6.1**) and in $\widetilde{O}\left(\frac{n^2}{\varepsilon^2 k}\right)$ time[1] computes a value $\overline{\mathtt{cost}}$, such that with probability at least $\frac{2}{3}$,

$$|\mathtt{cost}(G) - \overline{\mathtt{cost}}| \leq \varepsilon \cdot \mathtt{cost}(G) \ .$$

In particular, our result shows that for any $k = \omega(\varepsilon^{-2} \log n)$, one can estimate to within a $(1 \pm \varepsilon)$ factor the cost of a k-NN graph G of a metric space (X, d) with a *sublinear* number of oracle access queries to (X, d). And when k is almost linear in n, the running time is also *almost linear*.

While this algorithm runs in sublinear time for large values of k, in arguably the most interesting case when k is relatively small, the running time is close to $O(n^2)$, which can be easily achieved even by a naïve algorithm. Even if this dependency is undesirable, such dependency (or a similar one) is known to be necessary for $k = 1$. Indeed, consider a set X of n points, n even, and a random perfect matching on X. Assign to every matching edge a cost close to 0 and to every other edge a cost 1. It is easy to verify that this is a metric space. Now, assign with probability $\frac{1}{2}$ a weight 1 to a

[1] Throughout the paper we will assume that ε is arbitrary parameter satisfying $0 < \varepsilon \leq 1$.

random matching edge. Notice that if weight 1 has been assigned to a random matching edge then $\mathfrak{cost}(X) \sim 1$, whereas $\mathfrak{cost}(X) \sim 0$ otherwise. Therefore, in order to approximate the cost of the 1-nearest neighbor graph with an additive error term of $\mathfrak{cost}(X)$, any algorithm would have to find out if such an edge of weight 1 exists. This requires to find a constant fraction of the matching edges, which is known to require $\Omega(n^2)$ time [2].

To bypass the negative dependency on k, we present the second algorithm that performs very well for small values of k (cf. **Theorem 5.1**) and in $\widetilde{O}\left(\frac{nk^{3/2}}{\varepsilon^6}\right)$ time computes a value $\overline{\mathfrak{cost}}$ such that with probability at least $\frac{2}{3}$, we have

$$|\mathfrak{cost}(G) - \overline{\mathfrak{cost}}| \leq \varepsilon(\mathfrak{cost}(G) + \mathfrak{mst}(X)) \ ,$$

where $\mathfrak{mst}(X)$ denotes the cost of the minimum spanning tree of (X, d), that is, the cost of a minimum spanning tree of the complete weighted graph induced by (X, d). Observe that while this algorithm requires a sublinear $o(n^2)$ time for values of k up to $\widetilde{o}_\varepsilon(n^{2/3})$, its approximation has an additive error term that depends on the cost of the minimum spanning tree of (X, d). We remark that this is a fairly weak condition. For example, if the k-NN graph is connected, then this implies that the approximation becomes a classical $(1+\varepsilon)$-approximation.

The algorithms above can be combined to compute in expected $\widetilde{O}(\min\{\frac{nk^{3/2}}{\varepsilon^6}, \frac{n^2}{\varepsilon^2 k}\})$ time an estimate $\overline{\mathfrak{cost}}$ for the cost of the k-NN graph G that satisfies with probability at least $\frac{2}{3}$,

$$|\mathfrak{cost}(G) - \overline{\mathfrak{cost}}| \leq \varepsilon(\mathfrak{cost}(G) + \mathfrak{mst}(X)) \ .$$

The running time is *sublinear for every* k, and is always at most $\widetilde{O}\left(\frac{n^{8/5}}{\varepsilon^{18/5}}\right)$.

The bounds above show that one can estimate the cost of the k-NN graph in sublinear time, but it is natural to ask whether one can obtain even stronger bounds and faster algorithms. For example, maybe it is possible to provide a good estimation in $\widetilde{O}(nk)$ time or even in $\widetilde{O}(n)$ time? We will prove that this is impossible; we will complement our algorithmic results and provide *matching lower bounds* showing that the complexity of our algorithms is essentially optimal.

THEOREM 1.1. *Let* $\mathfrak{c}$ *be any positive constant,* $\mathfrak{c} \leq k$. *Any algorithm that for any metric space* (X, d) *of size* n *with probability at least* $\frac{2}{3}$ *estimates the cost* $\mathfrak{cost}(X)$ *of a* k-NN *graph to within an additive error term* $\mathfrak{c} \cdot \mathfrak{cost}(X)$ *requires* $\Omega\left(\frac{n^2}{k}\right)$ *queries.*

THEOREM 1.2. *Let* $\varepsilon \leq \frac{1}{2}$. *Any algorithm that for any metric space* (X, d) *of size* n *with probability at least* $\frac{5}{6}$ *estimates the cost* $\mathfrak{cost}(X)$ *of a* k-NN *graph to within an additive error term* $\varepsilon \cdot (\mathfrak{cost}(X) + \mathfrak{mst}(X))$ *requires* $\Omega(\min\{\frac{nk^{3/2}}{\varepsilon}, \frac{n^2}{k}\})$ *queries.*

Because of space constraints, the proofs of Theorems 1.1 and 1.2 are deferred to the full version of this paper.

1.2 Our techniques. We develop two algorithms, one of them is more suitable for large and the other for small values of k. By choosing the better of the algorithms we obtain the running time claimed in the abstract. Furthermore, we will complement these bounds and show that these algorithms are essentially optimal. In the following, we describe the main ideas behind both algorithms, and then will briefly discussed the ideas behind our lower bounds.

1.2.1 Algorithm for large k and $(1 \pm \varepsilon)$-approximation of $\mathfrak{cost}(G)$. Our $(1 \pm \varepsilon)$-approximation of $\mathfrak{cost}(G)$ that works efficiently for large k relies on a combination of two random sampling routines. It first takes $\widetilde{O}\left(\frac{n}{k}\right)$ samples for each point to determine an *approximation of the median cost* of an edge of the k-NN graph (that is, the distance to the $\frac{k}{2}$-nearest neighbor). The median edge has several useful properties that we will use:

(a) $\Omega(k)$ times its cost is a lower bound for the contribution of the vertex at hand, and

(b) if there is an edge that is, say, 10 times longer than the median edge, then there is also an edge of similar length at all vertices of distance less than the length of the median edge.

Similar properties are also true for an edge with rank close to the median rank of its k nearest neighbors.

Once we have determined an approximation for the median edge, we partition the edges in G into *short* and *long* edges. Short edges have length at most 10 times the length of the (approximate) median edge and long edges are longer (all edges of G that are not short). We will separately estimate the total cost of all short edges and the cost of all long edges.

To approximate the contribution of the short edges, we use a form of importance sampling: We sample each point with probability proportional to the length of its median edge and compute the sum of short edges incident to the sample point. Then we scale the weight by the inverse of the sampling probability and the sample size. We analyze the quality of the sampling process using Chebyshev's inequality. The approach

is required to detect points that are far away from everything: For example, we may have a cluster of $n-1$ points that are close to each other and a single point that is far away from the cluster and that dominates the cost of the solution. In order to find this point, we need to apply non-uniform sampling.

To estimate the cost of the long edges, we use a uniformly random sample of $\widetilde{O}(\frac{n}{\varepsilon^2 k})$ points. By property (b) outlined above, we can ensure that long edges cannot "hide," i.e., for every long edge, we have $\Omega(k)$ long edges at other points. This can be used to show that by taking a sample of $\widetilde{O}(\frac{n}{\varepsilon^2 k})$ random points and determining the costs of the long edges incident to them, we will get a good estimation of the total cost of all long edges.

Combining the two estimates yields the required bound (cf. Section 6 for more details).

1.2.2 Algorithm for small k. Our algorithm for small k that in time $\widetilde{O}\left(\frac{nk^{3/2}}{\varepsilon^6}\right)$ estimates $\mathfrak{cost}(G)$ with an additive error term $\varepsilon \cdot (\mathfrak{mst}(X) + \mathfrak{cost}(X))$ uses a different approach. We first derive a formula for the cost of a k-NN graph. We will assume that the distances are non-negative powers of $(1+\varepsilon)$. One can easily argue (for details, see the full version) that this does not change the problem significantly (essentially, we are using existing sublinear-time algorithms to approximate the diameter [17] and the weight of the minimum spanning tree [6] to be able to appropriately scale the edge lengths and then to round them).

Let $G = (V, E)$ be a k-NN graph. We define subgraphs $G^{(i)} = (V, E^{(i)})$ to consist of all edges $(u, v) \in E$ with distance $d(u, v) \leq (1+\varepsilon)^i$. We then show that we can use these "threshold" graphs to derive the following formula for the cost of the k-NN graph, where $\mathfrak{r}$ is the exponent of the maximum edge weight. The idea to express the cost of a graph structure in terms of the structure of threshold graphs has been used before in the area of sublinear algorithm, for example, in the context of the approximation of the cost of a minimum spanning tree (cf. [3, 6]). In our case, we have a surprisingly simple formula that only depends on vertex degrees. We will use the well-known identity $\sum_{i=0}^{\ell-1}(1+\varepsilon)^i = \frac{(1+\varepsilon)^\ell - 1}{\varepsilon}$ to express the weight of an edge as a weighted sum over the corresponding edges in the graphs $G^{(i)}$ in which the edge is missing. Using $\deg^{(i)}(v)$ to denote the degree of v in $G^{(i)}$, we will apply the identity above to obtain the following *central formula*:

$$\mathfrak{cost}(G) = nk + \varepsilon \cdot \sum_{i=0}^{\mathfrak{r}-1} \left((1+\varepsilon)^i \sum_{v \in V} (k - \deg^{(i)}(v)) \right) \ .$$

With this formula at hand, our problem has been reduced to that of estimating the sum $\sum_{v \in V}(k - \deg^{(i)}(v))$, which we will do by a suitable complex random sampling. For this purpose, for any fixed i, we partition the vertices according to their values of $k - \deg^{(i)}(v)$ into sets V^i_j, $0 \leq j \leq \mathfrak{t}$ with $\mathfrak{t} = 1 + \lfloor \log_{1+\varepsilon} k \rfloor$, such that V^i_0 contains vertices with $\deg^{(i)}(v) = k$, and for $1 \leq j \leq \mathfrak{t}$, all vertices in V^i_j satisfy $(1+\varepsilon)^{j-1} \leq k - \deg^{(i)}(v) < (1+\varepsilon)^j$. This leads to a modification of formula above, stating that

$$\mathfrak{cost}(G) \approx nk + \varepsilon \sum_{0 \leq i \leq \mathfrak{r}-1, 1 \leq j \leq \mathfrak{t}} (1+\varepsilon)^{i+j} |V^i_j| \ .$$

Now the main challenge is to approximate the sizes of the sets V^i_j and, most importantly, the sets with large i, i.e., sets that potentially contribute a lot to the sum. In order to cope this challenge and analyze our approach, we have to combine several ingredients.

The first idea can be illustrated on the following example. Consider an input set of points that can be divided into two clusters. One cluster contains a single point u and the other one contains the remaining $n-1$ points. The intra-cluster distances are 1 and the distances between the clusters are n^2 each, which results in $\mathfrak{cost}(G) \sim n^2 k$. How can we correctly approximate the sizes of the sets V^i_j in this setting? We observe that since $V^i_0 = V \setminus \{u\}$ for all $i < \mathfrak{r}$, in that case there is exactly a single set V^i_j (with $j = \mathfrak{t}$) which is not empty and contains only a single point u. If we would like to find u by random sampling, we need to sample $\Omega(n)$ points. If we were computing the k nearest neighbors of each sample point exactly, this would result in $\Omega(n^2)$ running time. What saves us is that in this instance we can quickly verify that a given point x is in the big cluster: We sample a few neighbors of x and if most of them have distance 1, we know that we are in the big cluster and we can drop the point x as it cannot contribute to the cost function. Our first ingredient to the final algorithm is therefore a *sampling based filtering* routine that allows us to drop such points when $\deg^{(i)}(x) = \Omega(k)$ with just $\widetilde{O}(\frac{n}{\deg^{(i)}(x)})$ queries. We remark that a somewhat similar filtering strategy has been used in the context of approximating the cost of a metric minimum spanning tree [6].

Now let us consider as a second example a metric space in which all points have pairwise distance 1 or $\mathfrak{d}$, for some large $\mathfrak{d} \gg k^2$. We have $m \cdot k^{3/2}$ clusters consisting of $k+1$ points and m clusters with $k - \sqrt{k}$ points (and hence, the total number of points is $n \sim mk^{5/2}$). The intra-cluster distance is 1 and all remaining distances are $\mathfrak{d}$. The cost of this k-NN graph is $\Theta(mk^{7/2} + \mathfrak{d}\, m\, k^{3/2}) = \Theta(\mathfrak{d}\, m\, k^{3/2})$. In this case it seems to be impossible to use random sampling to distinguish quickly between the case that a point is in a cluster with

$k+1$ points or is in the smaller cluster. The reason is that the standard deviation of a sample of linear size is still $\Omega(\sqrt{k})$. However, since there are $\Omega(mk)$ points with degree smaller than k (only those contribute to the sets V_j^i) we can relatively easily estimate their number by random sampling roughly $O(k^{3/2})$ points and computing the nearest neighbors for each sample point. The interesting observation is now that in this case the minimum spanning tree has a relatively large cost as well. And this is not a coincidence. In fact, our sampling algorithm to estimate the sizes of the sets V_j^i uses a sample size that depends inversely on the cost of the minimum spanning tree (which we can approximate using an algorithm from [6])!

The connection to the spanning tree problem might look surprising at first glance, but as we will see, there are relatively simple arguments why this works. Consider a fixed value ℓ and (for the analysis) greedily cluster the graph by selecting an arbitrary point as a center and assign all points at distance at most ℓ to it. Since the number of cluster centers have pairwise distance at least ℓ, we can derive a lower bound on the cost of the minimum spanning tree from the number of so greedily constructed clusters. The key property is that if there are many clusters, then the cost of the minimum spanning tree is high, which allows us to deal with a bigger estimation error and hence a smaller sampling size. On the other hand, if there are few clusters then most of them contain many points and thus do not contribute to our cost function; furthermore this can be also checked quickly by random sampling. This allows us to relate the expected running time of our sampling approach to the number of clusters and further to the cost of the minimum spanning tree. Finally, our analysis shows that this cancels out the dependence on the minimum spanning tree cost in the sample size. From this point we still need to do a few other tricks to obtain our first result, the algorithm for small k. See Section 5.1.2 for more details.

1.2.3 Lower bounds. In order to derive the lower bound for any algorithm approximating the cost of a k-NN graph, for a given parameter k, we will construct two families of problem instances whose cost differ substantially and show that no algorithm that queries the oracle less than T times can distinguish between these instances, for an appropriate value of T.

One family of problem instances essentially consists of point sets (X, d) partitioned into clusters of size $k+1$ each, where all points within the same cluster are at a very small distance from each other (say, after scaling, at distance 0), and the distance between any pair of points in different clusters is very large (say, after scaling, at distance 1). It is not difficult to see that in such case, we have $\mathfrak{cost}(G) \sim 0$ and $\mathfrak{mst}(X) \sim \frac{n}{k}$.

The other family of problem instances depend on the parameter k, and on whether we want to consider the approximation bound to be independent or dependent on $\mathfrak{mst}(X)$.

Let us first consider the case where we want to provide a lower bound for any algorithm giving a $(1+\varepsilon)$-approximation of $\mathfrak{cost}(G)$ (cf. Theorem 1.1). Then, the second family of problem instances consists of point sets (X, d) partitioned into clusters of size $k+1$, as above, except that we take a one random cluster and split it into one cluster of size k and another consisting of an isolated point (and as before, the inner-cluster distances are 0 and the outer-cluster distances are 1). Notice that in this case we have $\mathfrak{cost}(G) = 2k$. For such two problem instances, we will show (in a rather complex proof) that any algorithm distinguishing between inputs from these two families of problem instances requires $\Omega(\frac{n^2}{k})$ queries to the input oracle. The first straightforward implication is that this result proves that any $(1+\varepsilon)$-approximation algorithm for $\mathfrak{cost}(G)$ requires $\Omega(\frac{n^2}{k})$ time. However, we can extend this analysis also to the case when we allow an additive error term of at most $\varepsilon(\mathfrak{cost}(G) + \mathfrak{mst}(X))$. Indeed, in such problem instances we have not only $\mathfrak{cost}(G) = 2k$, but also $\mathfrak{mst}(X) \sim \frac{n}{k}$. Therefore, if we have $k = \Omega(\sqrt{n})$, then the arguments above yield that any algorithm that approximates $\mathfrak{cost}(G)$ to within an additive error term of $\varepsilon(\mathfrak{cost}(G) + \mathfrak{mst}(X))$ requires $\Omega(\frac{n^2}{k})$ time.

For smaller values of k, the arguments above (for approximation algorithms with the additive error term of $\varepsilon(\mathfrak{cost}(G) + \mathfrak{mst}(X))$) do not hold and so we have to revise our construction. Therefore in the case when $k = O((\varepsilon n)^{2/5})$, we partition the input point set into $\Theta(\frac{\varepsilon n}{k^{5/2}})$ clusters of size $k+1+\sqrt{k}$, the same number of clusters of size $k+1-\sqrt{k}$, and the remaining $\Theta(\frac{\varepsilon n}{k^{3/2}})$ points are partitioned into clusters of size $k+1$; as before, the inner-cluster distances are 0 and the outer-cluster distances are 1. Notice that $\mathfrak{cost}(G) \sim \frac{\varepsilon n}{k}$ and $\mathfrak{mst}(X) \sim \frac{n}{k}$. Similarly as before, we will show that to distinguish between inputs from these two families of problem instances requires $\Omega(\frac{nk^{3/2}}{\varepsilon})$ time. As an immediate consequence, this implies that for $k = O((\varepsilon n)^{2/5})$, any algorithm that approximates $\mathfrak{cost}(G)$ to within an additive error of $\varepsilon(\mathfrak{cost}(G)+\mathfrak{mst}(X))$ requires $\Omega(\frac{nk^{3/2}}{\varepsilon})$ time. This is formalized in Theorem 1.2.

The analysis above relies on a central result showing that it requires $\Omega(nh)$ time to distinguish between our input instances, (i) one with $r \cdot (h+2)$ clusters of size $k+1$, and (ii) another where $r \cdot h$ clusters are of size $k+1$ and r clusters are of size $k+1+\sqrt{k}$ and the same

number of r clusters are of size $k+1-\sqrt{k}$. This claim is the core of our analysis and its proof is elaborate. The first central intuition is that in order to determine $\Omega(k)$ of the points from any single cluster in either instance one needs to perform essentially $O(n)$ queries related to that cluster. Another key intuition is that if one knows only any set of $o(k)$ points from a single cluster, then one cannot determine with good confidence whether the size of that cluster is $k+1$ or $k+1\pm\sqrt{k}$. By combining these two properties, and by noticing that in the second problem instance there is one cluster of size $k+1\pm\sqrt{k}$ per $\Theta(h)$ clusters of size $k+1$, we can then argue that in order to find a single cluster of size $k+1\pm\sqrt{k}$, if there is one, one needs to perform $\Omega(nh)$ oracle queries. While these intuitions are rather simple, their formalization requires some elaborate arguments that we will present in details in the full version of the paper.

1.3 Further related work. Sublinear algorithm for problems in metric spaces have received significant attention in the last several years. It is known that one can compute in sublinear time approximations for the diameter, maximum travelling salesperson, maximum spanning tree, minimum routing cost spanning tree, average distance [17]. The question of how to approximate the cost of a minimum spanning tree has been studied both in the metric [6] and non-metric [3] setting. For our work, the algorithm for metric spaces is more relevant and it computes a $(1+\varepsilon)$ approximation of the cost of the metric minimum spanning tree in time $\widetilde{O}(n/\varepsilon^7)$ [6]. Also, the Euclidean minimum spanning tree problem has been studied in a setting where one can access the input via certain spatial data structure [5]. The cost of the (uniform) facility location problem can be approximated in sublinear time [2]. For the k-center and k-median problem there are approximation algorithms with a running time of $\widetilde{O}(nk)$ [20] as well as bi-criteria approximation algorithms [17]. The metric maximum cut problem can also be solved in sublinear time [10, 17]. A recent result on linear sampling from metric spaces can be used to improve a number of previous results [10] like, for example, reduce the query complexity of max-cut. The size of a maximum matching [23, 31] and the vertex cover size [23, 24] can also be approximated using sublinear time approximation algorithms. The average degree of a graph can be efficiently approximated as well [11, 15].

Besides the area of sublinear approximation algorithms, random sampling approaches from graphs have been studied within the framework of property testing. The most relevant result from this area is a recent work by Fichtenberger and Rhode [12] who studied the problem of testing whether a graph is a k-nearest neighbor graph for the Euclidean distance and showed both theoretically and empirically that one can efficiently solve the property testing version of the problem where one wants to accept any true k-nearest neighbor graph and reject every graph that differs from a k-nearest neighbor graph in more than an ε-fraction of its edges.

2 Preliminaries

We start by introducing basic notation and the problem description.

2.1 Model of computation. We assume that we are given oracle access to a finite metric space (X, d) with $X = \{1, \ldots, n\}$: Our algorithm receives n as an input. The algorithm can specify pairs of points $x, y \in X$ (i.e., two numbers from $\{1, \ldots, n\}$) and query the oracle for their distance $d(x, y)$. Answering each query takes constant time. We remark that the full description size of the input space is $\Theta(n^2)$ and so the algorithms presented in this paper have a sublinear running time in the size of the input object. We will assume that all distances in X are distinct; if this is not the case we use a lexicographical ordering of the edges as tie breaker.

2.1.1 Graph representation of (X, d). We will often view the metric space (X, d) as a weighted *complete* undirected graph $H = (V, F)$ with vertex set $V = X$ and edge set F. The weight of an edge $(u, v) \in F$ equals the corresponding distance in (X, d), i.e., $d(u, v)$. This allows us to extend the definition of basic graph theoretic concepts like minimum spanning trees or k-nearest neighbor graphs to metric spaces. With this in mind, throughout the paper we will use interchangeably the notation $\mathtt{mst}(X)$ and $\mathtt{mst}(H)$ to denote the cost of the minimum spanning tree of H.

2.2 Approximating the k-NN graph. The *k-nearest neighbor graph* is a *directed graph* $G = (V, E)$ that has a directed edge $[u, v\rangle$, if v is in the set of k nearest neighbors of u, i.e., the set of k vertices that have k smallest distance to u. We observe that in general, this graph is *not* symmetric, i.e., if a vertex u is a k-nearest neighbor of v then not necessarily vertex v is a k-nearest neighbor of u. We will refer to G as the *k-NN graph of* (X, d).

The k-NN graph can be computed in time $O(n^2)$ by computing the vertex at rank k (according to distances) in the list of neighbors of each vertex v and then doing a linear scan over all neighbors to compute the set of k neighbors for every vertex. It is also not too hard to see that computing the k-NN graph exactly requires $\Omega(n^2)$ time: Pick a metric space where every node has pairwise distance 2 except for a single pair that has

distance 1. This pair belongs to the k-NN graph and finding it requires $\Omega(n^2)$ time.

For very large data sets, a quadratic running time is typically not feasible. Therefore, we will consider the problem of *approximating* the k-NN graph. For this purpose, we will phrase the problem as an optimization problem. Given an input metric space, find a graph $G = (V, E)$ that minimizes

$$\mathfrak{cost}(G) = \sum_{[u,v\rangle \in E} d(u, v)$$

subject to the outdegree of every vertex $u \in V$ to be exactly k in G.

Once we have defined an optimization problem, an α-approximation algorithm is an algorithm that computes a solution with cost at most $\alpha \cdot \mathsf{Opt}$, where Opt is the cost of an optimal solution. In this paper, we will be interested in the question of approximating the *cost* of the k-NN graph in sublinear time.

3 A formula for the cost of a k-NN graph

In this section we will derive a central tool in our algorithm for small k: a formula for the (approximate) cost of a k-NN graph. In fact, the formula can be used to express the cost of any weighted directed graph with outdegree exactly k. To simplify our exposition we will assume that the edge weights are of the form $(1+\varepsilon)^j$ with integer $0 \le j \le \mathfrak{r}$ and $\mathfrak{r} = O(\log n/\varepsilon)$. We can always transform our input on the fly into such a space by first approximating the diameter of the metric space [17] and the cost of the minimum spanning tree [6] and then rescale the space and round edge weights to the nearest power of $1+\varepsilon$ (rescaling and rounding is done on the fly). This only slightly affects the triangle inequality by introducing an additional $(1+\varepsilon)$ factor. Details can be found in the full version of the paper. The time to approximate the diameter within a factor of 2 is $O(n)$ [17] and the time required to approximate the cost of the minimum spanning tree within a factor of 2 is $\widetilde{O}(n)$ [6]. We will assume that edges with the same weight are ordered lexicographically.

Threshold graphs $G^{(i)}$. Let G be a graph with outdegree k at every vertex and with edge weights that are of the form $(1+\varepsilon)^j$ for integer values of j. Let $G^{(i)} = (V, E^{(i)})$ be the subgraph of G that contains all edges of weight less than or equal to $(1+\varepsilon)^i$, i.e., $E^{(i)} = \{[u, v\rangle \in E : d(u, v) \le (1+\varepsilon)^i\}$. We use $\deg^{(i)}(v)$ to denote the outdegree of vertex v in $G^{(i)}$.

LEMMA 3.1. *Let $G = (V, E)$, $|V| = n$, be a weighted directed graph such that every vertex has outdegree k and such that the edge weights are of the form $(1+\varepsilon)^j$ for integer $0 \le j \le \mathfrak{r}$, for some $\mathfrak{r} \in \mathbb{N}$. Let $\varepsilon > 0$. Then we can write*

$$\mathfrak{cost}(G) = nk + \varepsilon \cdot \sum_{i=0}^{\mathfrak{r}-1} \left((1+\varepsilon)^i \sum_{v \in V} \left(k - \deg^{(i)}(v) \right) \right) .$$

Proof. Let $[u, v\rangle \in E$ be an edge of weight $(1+\varepsilon)^\ell$. We will use the well-known identity

$$\sum_{i=0}^{\ell-1} (1+\varepsilon)^i = \frac{(1+\varepsilon)^\ell - 1}{\varepsilon} \tag{3.1}$$

to express the weight of an edge as a weighted sum over the corresponding edges in the graphs $G^{(i)}$ in which the edge is missing. By (3.1), we can write

$$\begin{aligned}(1+\varepsilon)^\ell &= 1 + \varepsilon \cdot \sum_{i=0}^{\ell-1} (1+\varepsilon)^i \\ &= 1 + \varepsilon \cdot \sum_{i=0}^{\mathfrak{r}-1} \left((1+\varepsilon)^i \cdot \mathbb{1}([u, v\rangle \notin E^{(i)}) \right) ,\end{aligned}$$

where we use $\mathbb{1}(B)$ to be 1, if expression B is true and 0, otherwise.

We can sum this formula up over all edges in E to obtain

$$\begin{aligned}\mathfrak{cost}(G) &= \sum_{[u,v\rangle \in E} \left(1 + \varepsilon \cdot \sum_{i=0}^{\mathfrak{r}-1} (1+\varepsilon)^i \cdot \mathbb{1}([u, v\rangle \notin E^{(i)} \right) \\ &= kn + \varepsilon \cdot \sum_{i=0}^{\mathfrak{r}-1} \left((1+\varepsilon)^i \cdot \sum_{[u,v\rangle \in E} \mathbb{1}([u, v\rangle \notin E^{(i)}) \right) \\ &= kn + \varepsilon \cdot \sum_{i=0}^{\mathfrak{r}-1} \left((1+\varepsilon)^i \cdot |E \setminus E^{(i)}| \right) \\ &= kn + \varepsilon \cdot \sum_{i=0}^{\mathfrak{r}-1} \left((1+\varepsilon)^i \cdot \sum_{v \in V} (k - \deg^{(i)}(v)) \right) .\end{aligned}$$

□

4 Outline of the algorithm with $\widetilde{O_\varepsilon}(nk^{3/2})$ queries

In this section we present main ideas behind our algorithm k-NNSIZEAPPROXIMATION(n, ε) that performs $\widetilde{O_\varepsilon}(nk^{3/2})$ queries to approximate the cost of a k-NN graph G (cf. Theorem 5.1). (Since the problem can be trivially solved with $\Theta(n^2)$ queries, in this section we will assume that $k = o(n)$.) Our goal is to rely on the formula from Lemma 3.1 to approximate the cost of a k-NN graph G by estimating $\sum_{v \in V} (k - \deg^{(i)}(v))$ for every $i, 0 \le i \le \mathfrak{r}$.

Let us fix i, $0 \leq i \leq \mathfrak{r}$, and let $\mathfrak{t} = 1 + \lfloor \log_{1+\varepsilon} k \rfloor$. In order to estimate the sum $\sum_{v \in V}(k - \deg^{(i)}(v))$, we partition the vertex set V into subsets $V_0^i, V_1^i, \ldots, V_{\mathfrak{t}}^i$ such that

$$V_0^i = \{v \in V : \deg^{(i)}(v) = k\} \ ,$$

and for $1 \leq j \leq \mathfrak{t}$,

$$V_j^i = \{v \in V : (1+\varepsilon)^{j-1} \leq k - \deg^{(i)}(v) < (1+\varepsilon)^j\} \ .$$

We begin with an auxiliary lemma that shows that in order to approximate $\mathfrak{cost}(G)$ it suffices to estimate well the sizes of all sets V_j^i with $0 \leq i \leq \mathfrak{r}$, $1 \leq j \leq \mathfrak{t}$.

LEMMA 4.1.

$$\mathfrak{cost}(G) \leq nk + \varepsilon \sum_{\substack{0 \leq i \leq \mathfrak{r} \\ 1 \leq j \leq \mathfrak{t}}} (1+\varepsilon)^{i+j} |V_j^i| \leq (1+\varepsilon) \cdot \mathfrak{cost}(G).$$

Proof. It immediately follows from Lemma 3.1 that

$$\begin{aligned}
\mathfrak{cost}(G) &= nk + \varepsilon \sum_{i=0}^{\mathfrak{r}-1} (1+\varepsilon)^i \sum_{v \in V} (k - \deg^{(i)}(v)) \\
&= nk + \varepsilon \sum_{i=0}^{\mathfrak{r}-1} (1+\varepsilon)^i \sum_{j=1}^{\mathfrak{t}} \sum_{v \in V_j^i} (k - \deg^{(i)}(v)) \\
&\leq nk + \varepsilon \sum_{i=0}^{\mathfrak{r}-1} (1+\varepsilon)^i \sum_{j=1}^{\mathfrak{t}} (1+\varepsilon)^j |V_j^i| \\
&= nk + (1+\varepsilon)\varepsilon \sum_{i=0}^{\mathfrak{r}-1} (1+\varepsilon)^i \sum_{j=1}^{\mathfrak{t}} (1+\varepsilon)^{j-1} |V_j^i| \ .
\end{aligned}$$

Next, we use that $(1+\varepsilon)^{j-1} \leq k - \deg^{(i)}(v)$ to get,

$$\begin{aligned}
\sum_{j=1}^{\mathfrak{t}} (1+\varepsilon)^{j-1} |V_j^i| &\leq \sum_{j=1}^{\mathfrak{t}} \sum_{v \in V_j^i} (k - \deg^{(i)}(v)) \\
&= \sum_{v \in V} (k - \deg^{(i)}(v)) \ ,
\end{aligned}$$

giving the following,

$$\begin{aligned}
\mathfrak{cost}(G) &\leq nk + (1+\varepsilon)\varepsilon \sum_{i=0}^{\mathfrak{r}-1} (1+\varepsilon)^i \sum_{j=1}^{\mathfrak{t}} (1+\varepsilon)^{j-1} |V_j^i| \\
&\leq nk + (1+\varepsilon)\varepsilon \sum_{i=0}^{\mathfrak{r}-1} (1+\varepsilon)^i \sum_{v \in V} (k - \deg^{(i)}(v)) \\
&\leq (1+\varepsilon)\mathfrak{cost}(G) \ .
\end{aligned}$$

Hence, we obtain the following,

$$\begin{aligned}
\mathfrak{cost}(G) &\leq nk + \varepsilon \cdot \sum_{i=0}^{\mathfrak{r}-1} (1+\varepsilon)^i \sum_{j=1}^{\mathfrak{t}} (1+\varepsilon)^j |V_j^i| \\
&\leq (1+\varepsilon) \cdot \mathfrak{cost}(G) \ ,
\end{aligned}$$

concluding the proof of the claim. ∎

Thanks to Lemma 4.1, in order to estimate $\mathfrak{cost}(G)$ it is enough to estimate the number of vertices in each of the sets V_j^i and then use expression from Lemma 4.1 to estimate the cost of a k-NN graph. The problem is that we do not know G. In principle, we are given G implicitly, because we can identify the outgoing edges of a vertex v in $O(n)$ time by querying for all of its neighbors, but this can be way too expensive. Instead, we will consider subgraphs $H^{(i)} = (V, F^{(i)})$ of H that contains all edges $[u, v\rangle$ with $d(u, v) \leq (1+\varepsilon)^i$, that is, $F^{(i)} = \{[u, v\rangle \in F : d(u, v) \leq (1+\varepsilon)^i\}$. We make the following straightforward observation.

OBSERVATION 4.1. *Let (X, d) be a metric space and assume that all edge weights are at least 1 and are powers of $(1+\varepsilon)$. Let $G^{(i)} = (V, E^{(i)})$ and $H^{(i)} = (V, F^{(i)})$ be defined as above. Let $E_v^{(i)}$ and $F_v^{(i)}$ be the outgoing edges of a vertex v in $G^{(i)}$ and $H^{(i)}$, respectively. Then the following statements are true:*

- $E^{(i)} \subseteq F^{(i)}$,
- *if $|F_v^{(i)}| \leq k$ then $E_v^{(i)} = F_v^{(i)}$, and*
- *if $|F_v^{(i)}| > k$ then $|E_v^{(i)}| = k$.*

The above observation allows us to make statements about $G^{(i)}$, and hence G, by considering $H^{(i)}$. The challenge here is to find the right tradeoffs between the sample size required to obtain a good estimation and the time spent on each sample vertex to approximate $\deg^{(i)}(G)$ via approximating the corresponding degree in $H^{(i)}$, which we will denote by $\deg_{H^{(i)}}(v)$. Notice that $\deg^{(i)}(v) = \min\{\deg_{H^{(i)}}(v), k\}$.

4.1 The main sampling algorithm. We now describe the main sampling algorithm k-NNSIZE-APPROXIMATION following the framework described by the inequalities from Lemma 4.1. It uses a subroutine ESTIMATESETSIZE to compute an approximation $X_{i,j}$ for the sizes of the sets V_j^i. We assume that the input is normalized as earlier discussed. The precise value of $\mathfrak{r} = O(\log n/\varepsilon)$ follows from scaling the input and the value of $\mathfrak{t} = O(\log k/\varepsilon)$ has been set up earlier as the maximum index j for V_j^i ($\mathfrak{t}$ is the smallest integer such that $k < (1+\varepsilon)^{\mathfrak{t}}$, i.e., $\mathfrak{t} = 1 + \lfloor \log_{1+\varepsilon} k \rfloor$).

k-NNSIZEAPPROXIMATION(n, ε)
{Approximation of $\mathfrak{cost}(G)$; cf. Theorem 5.1}

for $i = 0$ **to** $\mathfrak{r}$ **do**
  for $j = 1$ **to** $\mathfrak{t}$ **do**
    $X_{i,j}$ = ESTIMATESETSIZE(n, ε, i, j)
return $nk + \varepsilon \cdot \sum_{0 \leq i \leq \mathfrak{r}, 1 \leq j \leq \mathfrak{t}} (1+\varepsilon)^{i+j} X_{i,j}$

EstimateSetSize(n, ε, i, j)
{Estimates $|V_j^i|$; cf. Lemma 5.5}

Sample set S of $s = \lceil \frac{100n(1+\varepsilon)^{i+j}\mathfrak{rt}}{\mathfrak{mst}(X)} \rceil$ vertices $u_1, \ldots, u_s$ uniformly at random
for $\ell = 1$ **to** s **do**
  if EstimateVertexDegree$(n, \varepsilon, k, u_\ell, i, j) \in [(1+\varepsilon)^{j-1}, (1+\varepsilon)^j)$ **then** $Y_\ell = 1$
  else $Y_\ell = 0$
return $\frac{n}{s} \cdot \sum_{\ell=1}^{s} Y_\ell$

The main challenge now is to approximate the sizes of the sets V_j^i. A standard approach would be to sample vertices uniformly at random and verify membership in V_j^i for each sample vertex and then to extrapolate. Unfortunately, such a simple approach does not work as we typically cannot afford to ask $\Theta(n)$ queries to compute all neighbors of a sample vertex. Indeed, the example from the previous section suggests that there are cases where we cannot spend much more than constant time per sample. Therefore, we proceed as follows. Firstly, for every sample vertex we run the procedure Filter(n, k, v, i), which rejects, if a given vertex v has degree $\deg_{H^{(i)}}(v)$ significantly more than k (in which case v does not contribute to our objective function). The central feature of our implementation of Filter is that we can run Filter(n, k, v, i) with the expected $O(\frac{n \log n}{k + \deg_{H^{(i)}}(v)})$ queries, see Lemma 5.1. Thus, if a vertex has a high degree, we will dismiss it quickly, with $O\left(\frac{n \log n}{\deg_{H^{(i)}}(v)}\right)$ queries if $\deg_{H^{(i)}}(v)) \geq k$. We remark that a somewhat similar procedure has been the crux of the algorithm approximating the cost of the minimum spanning tree [6].

Once we know that a vertex has degree $O(k)$ in $H^{(i)}$, we call it a *candidate vertex*. For each candidate vertex we would like to determine whether it is contained in the current set V_j^i. Again, it would be way too slow to decide exactly whether $v \in V_j^i$, since it would require $\Theta(n)$ queries per every single vertex. Therefore, we will do it approximately by using fine-tuned random sampling procedure on the input graph $H^{(i)}$ to estimate the vertex degree. One technical obstacle is that if we use estimates to determine membership in V_j^i instead of exact values, the sets V_j^i will depend on the randomness used. Fortunately, there is a simple argument why this is not an issue, provided that we have a good estimation. Instead of considering an algorithm that takes a random sample of vertices and then runs an estimation procedure on the sample, we can assume for the analysis that at every vertex we run the estimation procedure first and then we sample from the set of resulting estimates. Thus, we may think of the sets V_j^i as independent of the randomness used for the sampling.

We now discuss our approach to test whether $v \in V_j^i$. We will sample vertices from $V \setminus \{v\}$ and query their distance to v to determine whether they are neighbors of v in $H^{(i)}$. The sample sizes used by our estimation algorithm will depend on the index j of the set V_j^i. The reason is that while we would like to obtain an approximation for the value of $k - \deg^{(i)}(v)$ up to an additive error of $\varepsilon \cdot (k - \deg^{(i)}(v))$, our sampling algorithm estimates only $\deg_{H^{(i)}}(v)$, and not $\varepsilon \cdot (k - \deg^{(i)}(v))$ (in fact, it estimates the number of edges of length at most $(1+\varepsilon)^i$ in $H^{(i)}$, but we can use Observation 4.1 to transform this into an approximation of $\deg^{(i)}(v)$ for the relevant range of parameters). This means that when $k - \deg^{(i)}(v)$ is small, then we need a very good approximation of $\deg_{H^{(i)}}(v)$. Indeed, if $(1+\varepsilon)^j$ is smaller than $\sqrt{k}$, we simply consider all neighbors of the current vertex. For larger values of j, we can use random samples of different sizes, which results in different query complexities. Fortunately, the number of queries for larger values of $k - \deg^{(i)}(v)$ is smaller, so that we can take larger samples for items that contribute a lot to the cost function, which allows us to reduce the variance of the process. The sweet spot is when $k - \deg^{(i)}(v) = \Theta(\sqrt{k})$, where we require $\Omega(n)$ queries per sample.

What remains is to determine the initial sample size. Here we will exploit the fact that thanks to [6], with only $\widetilde{O}(n)$ queries we can estimate $\mathfrak{mst}(X)$ up to a constant, with probability at least $1 - 1/n^{10}$, and we will therefore pick a sample of size $\widetilde{O}(\frac{nk(1+\varepsilon)^{i+j}}{\mathfrak{mst}(X)})$. Since every vertex can contribute between 0 and $k \cdot (1+\varepsilon)^i$ to our formula for the k-NN graph cost, such a sample size will suffice to get an additive error of $\varepsilon \cdot \mathfrak{mst}(X)$ (more details follow in the analysis in Section 5). In order to analyze the expected number of queries of the algorithm, we will then define a simple greedy clustering procedure and use the resulting clustering to obtain the desired bound on the expected number of queries of the filtering procedure.

5 Approximating $\mathfrak{cost}(G)$ with $\widetilde{O}_\varepsilon(nk^{3/2})$ queries

We continue the discussion of our $\widetilde{O_\varepsilon}(nk^{3/2})$-queries algorithm k-NNSizeApproximation to approximate the cost of a k-NN graph G. We will prove in Theorem 5.1 that Algorithm k-NNSizeApproximation (n, ε) in expected time $O\left(\frac{nk^{3/2} \log^3 n \log^2 k}{\varepsilon^6}\right)$ returns a value $\overline{\mathfrak{cost}}$ such that with probability at least $\frac{2}{3}$, we have

$$|\mathfrak{cost}(G) - \overline{\mathfrak{cost}}| \leq \varepsilon \cdot (\mathfrak{mst}(X) + \mathfrak{cost}(G)) \ .$$

We will now describe in details how the estimation of the sizes of the sets V_j^i is done and then show how to approximate $\mathtt{cost}(G)$ using Lemma 4.1. Our algorithm EstimateSetSize uses a subroutine EstimateVertexDegree to approximate the degree $\deg_{H^{(i)}}(v)$ of a given vertex in $H^{(i)}$. For now, it will be convenient to just think of EstimateVertexDegree to return a correct estimate; the precise description of algorithm EstimateVertexDegree is given in Section 5.2.

5.1 Tools for the analysis of EstimateVertexDegree.

Before we will present details of our algorithm EstimateVertexDegree$(n, \varepsilon, k, v, i, j)$ to approximate the degree $\deg_{H^{(i)}}(v)$ of a vertex in $H^{(i)}$, we will first provide some basic tools useful for the analysis. Our algorithm EstimateVertexDegree performs two steps: It first checks by a sampling procedure called Filter whether the vertex degree is significantly more than k. If this is the case, the vertex does not contribute to the cost function and we can ignore it. If the vertex passes our test, we know that with high probability its degree is no more than $4k$. In this case, we continue our analysis depending on the relation between j and k; if $(1+\varepsilon)^j \leq \sqrt{k}$ then we compute its degree exactly with $O(n)$ queries, and otherwise, we use a special sampling routine NeighborhoodSize(n, γ, k, v, i) to estimate $\deg_{H^{(i)}}(v)$ to within γk (Section 5.3). We will first describe the filtering algorithm and then proceed to the main analysis.

5.1.1 Filtering vertices with many neighbors.

Our first subroutine will be used to quickly filter vertices that have more than k neighbors and thus do not contribute to our cost function. The algorithm draws random samples of increasing size until it obtains a good estimate of the number of neighbors of a given query vertex or with high probability the vertex has $O(k)$ neighbors. A similar idea of using geometrically increasing sample sizes has been previously used in [6].

Filter(n, k, v, i)
{Determines if $\deg_{H^{(i)}}(v) = O(k)$; cf. Lemma 5.1}

$\mathfrak{q} = n - 1$
repeat
  $\mathfrak{a} = 1000\frac{n-1}{\mathfrak{q}} \log n$; $\mathfrak{q} = \frac{2}{3}\mathfrak{q}$
  Sample $v_1, \ldots, v_{\mathfrak{a}}$ i.u.r. from $V \setminus \{v\}$
  Let $\mathfrak{q}'$ be the number of vertices in $v_1, \ldots, v_{\mathfrak{a}}$ that are neighbors of v in $H^{(i)}$
until $\mathfrak{q}' \geq 1000 \log n$ **or** $\mathfrak{q} \leq 2k$
if $\mathfrak{q} \leq 2k$ **then return** True
else return False

Lemma 5.1. *Given access to a vertex v, algorithm* Filter(n, k, v, i) *performs in expectation $O(\frac{n \log n}{k + \deg_{H^{(i)}}(v)})$ queries and with probability at least $1 - 1/n^{10}$ returns the following value:*[2]

- False*: if* $\deg_{H^{(i)}}(v) \geq 4k$*;*
- True*: if* $\deg_{H^{(i)}}(v) \leq k$*;*
- *either* True *or* False*, otherwise.*

Proof. We will consider separately two cases, when $\deg_{H^{(i)}}(v) \geq 4k$ and $\deg_{H^{(i)}}(v) \leq k$.

Let us first assume that $\deg_{H^{(i)}}(v) \geq 4k$. Then at some point the algorithm reaches an iteration of the **for**-loop, such that $\mathfrak{q} \leq 3k$ or it has returned False before within the running time bound of the lemma. We now let X_j to be the indicator variable for the event that vertex v_j is a neighbor of v. We have $\mathbf{Pr}[X_j] = \frac{\deg_{H^{(i)}}(v)}{n-1}$ and hence,

$$\mathbf{Ex}\left[\sum_{j=1}^{\mathfrak{a}} X_j\right] = \mathfrak{a} \cdot \frac{\deg_{H^{(i)}}(v)}{n-1} = \frac{1000 \log n \cdot \deg_{H^{(i)}}(v)}{\mathfrak{q}} \geq \frac{4}{3} \cdot 1000 \log n \ .$$

Chernoff's bound implies that

$$\mathbf{Pr}\left[\sum_{j=1}^{\mathfrak{a}} X_j < 1000 \log n\right] \leq \mathbf{Pr}\left[\sum_{j=1}^{\mathfrak{a}} X_j \leq \frac{3}{4}\mathbf{Ex}[\sum_{j=1}^{\mathfrak{a}} X_j]\right] \leq e^{-\mathbf{Ex}[\sum_{j=1}^{\mathfrak{a}} X_j]/32} \leq \frac{1}{n^{10}} \ .$$

This implies that if $\deg_{H^{(i)}}(v) \geq 4k$, then algorithm Filter returns False in time $O(\frac{n \log n}{\deg_{H^{(i)}}(v)})$ with probability at least $1 - 1/n^{10}$.

Now we assume that $\deg_{H^{(i)}}(v) \leq k$. We observe that for $\mathfrak{q} > 2k$, we have

$$\mathbf{Ex}\left[\sum_{j=1}^{\mathfrak{a}} X_j\right] = \mathfrak{a} \cdot \frac{\deg_{H^{(i)}}(v)}{n-1} = \frac{1000 \log n \cdot \deg_{H^{(i)}}(v)}{\mathfrak{q}} \leq \frac{1000 \log n \cdot k}{2k} = 500 \log n \ .$$

[2] As we will make it explicit later, when presenting algorithm EstimateVertexDegree (cf. Lemma 5.4), one could trivially obtain in expectation $O(\min\{n, \frac{n \log n}{k + \deg_{H^{(i)}}(v)}\})$ queries by switching to computing $\deg_{H^{(i)}}(v)$ exactly when one performs more than n distance queries.

We apply Chernoff bound (if $\lambda \geq 2\mathbf{Ex}[\sum_{j=1}^{\mathfrak{a}} X_j]$ then $\mathbf{Pr}[\sum_{j=1}^{\mathfrak{a}} X_j \geq \lambda] \leq e^{-\lambda/6}$) to get,

$$\mathbf{Pr}\left[\sum_{j=1}^{\mathfrak{a}} X_j \geq 1000 \log n\right] \leq e^{-166 \log n} \leq \frac{1}{n^{10}} \ .$$

Therefore the probability that the algorithm returns FALSE is at most $\frac{1}{n^{10}}$. The running time is dominated by the last iteration of the **for**-loop, which requires $O(\frac{n \log n}{k})$ time. ∎

5.1.2 An auxiliary tool: The clustering connection and MST. In this section we show how to connect a term used in the running time of some parts of our algorithms with the cost of the minimum spanning tree for the input set of points X.

LEMMA 5.2. *Let $k < \frac{n-1}{4}$. For every i, $0 \leq i \leq \mathfrak{r}$, the following holds,*

$$\sum_{v \in V : \deg_{H^{(i)}}(v) \geq 1} \frac{1}{\deg_{H^{(i)}}(v)} \leq 2 + \frac{24 \cdot \mathfrak{mst}(X)}{(1+\varepsilon)^i} \ .$$

Furthermore, the number of vertices with degree at most $4k$ in $H^{(i)}$ is at most $\frac{120 \cdot k \cdot \mathfrak{mst}(X)}{(1+\varepsilon)^i}$.

Proof. We begin with an auxiliary clustering algorithm:

GREEDYCLUSTERING(n, σ)
{Greedy partition of X into clusters of radius σ and returns the number of clusters $\mathfrak{c}$; cf. Lemma 5.2}

$\mathfrak{c} = 0$
for each $x \in X$ **do**
  $\mathfrak{c} = \mathfrak{c} + 1$
  let $C_{\mathfrak{c}}$ be the set of points (including x) with distance at most σ from x
  remove $C_{\mathfrak{c}}$ from X
return $\mathfrak{c}$

We observe that all cluster centers have pairwise distance at least σ, and therefore if $\mathfrak{c}$ is the returned number of clusters, then $\frac{1}{3}(\mathfrak{c}-1)\sigma$ is a lower bound on the cost of the minimum spanning tree, that is,

$$\frac{1}{3}(\mathfrak{c}-1)\sigma \leq \mathfrak{mst}(X) \ . \tag{5.2}$$

This follows, because the minimum spanning tree connects all cluster centers and using additional vertices can only reduce the cost of the tree by a factor of 3. We remark that in this place we are applying the triangle inequality in the original metric space, that is, after we scale but before we round the distances to powers of $(1+\varepsilon)$. Therefore, we are losing slightly more than the usual factor of 2 (because edges are rounded afterwards).

Let us run GREEDYCLUSTERING(n, σ) on our input instance (X, d) with $\sigma = \frac{1}{4}(1+\varepsilon)^i$. Notice that if any two points are in the same cluster, then the distance between them is at most 4σ (which is also true in our space of rounded distances). Therefore for two such points, their distance is at most $(1+\varepsilon)^i$ and thus if we consider the graph $H^{(i)}$, then these two points are adjacent in $H^{(i)}$. Hence, for any cluster C and any point $v \in C$, we have

$$\deg_{H^{(i)}}(v) \geq |C| - 1 \ . \tag{5.3}$$

This means that for every cluster C, we have $1 = \sum_{v \in C} \frac{1}{|C|} \geq \sum_{v \in C} \frac{1}{\deg_{H^{(i)}}(v)+1}$ and so $\mathfrak{c} \geq \sum_{v \in V} \frac{1}{\deg_{H^{(i)}}(v)+1}$. We can combine this bound with (5.2) to obtain the following,

$$\begin{aligned}(1+\varepsilon)^i \left(\sum_{v \in V} \frac{1}{\deg_{H^{(i)}}(v)+1} - 1\right) & \\ = 4\sigma \left(\sum_{v \in V} \frac{1}{\deg_{H^{(i)}}(v)+1} - 1\right) & \\ \leq 4\sigma(\mathfrak{c}-1) \leq 12 \cdot \mathfrak{mst}(X) \ . &\end{aligned} \tag{5.4}$$

With (5.4) at hand, we can conclude the first claim as follows:

$$\begin{aligned}\sum_{v \in V : \deg_{H^{(i)}}(v) \geq 1} \frac{1}{\deg_{H^{(i)}}(v)} &\leq 2 \cdot \sum_{v \in V} \frac{1}{\deg_{H^{(i)}}(v)+1} \\ &\leq^{\text{by (5.4)}} 2 + \frac{24 \cdot \mathfrak{mst}(X)}{(1+\varepsilon)^i} \ .\end{aligned}$$

Next let us consider the second claim, and we want to bound the number of vertices of degree at most $4k$ in $H^{(i)}$. Assume first that $\mathfrak{c} \geq 2$. Since by (5.3), for any cluster C and any vertex $v \in C$, we have $\deg_{H^{(i)}}(v) \geq |C| - 1$, any cluster C contains at most $4k+1$ vertices of degree at most $4k$ in $H^{(i)}$. Therefore, in particular, the number of vertices of degree at most $4k$ in $H^{(i)}$ is at most $\mathfrak{c} \cdot (4k+1)$. Next, we use (5.2) to simplify this bound as follows:

$$\begin{aligned}\mathfrak{c} \cdot (4k+1) \leq 2(\mathfrak{c}-1) \cdot (5k) \leq 10k \cdot \frac{3 \cdot \mathfrak{mst}(X)}{\sigma} & \\ = \frac{120 \cdot k \cdot \mathfrak{mst}(X)}{(1+\varepsilon)^i} \ , &\end{aligned}$$

which yields the second claim (notice that in the first inequality we use $\mathfrak{c} \geq 2$).

The claim above assumed that $\mathfrak{c} \geq 2$ and let us now consider the case $\mathfrak{c} = 1$. In this case, since we

run GreedyClustering(n, σ) with $\sigma = \frac{1}{4}(1+\varepsilon)^i$ and ended up with a single cluster, there is a point $x \in X$ such that all other points in X are at distance at most $\sigma = \frac{1}{4}(1+\varepsilon)^i$ from x. Hence, the diameter of X is at most $2\sigma = \frac{1}{2}(1+\varepsilon)^i$ and thus $H^{(i)}$ is a clique on n vertices. This immediately implies that since we assumed that $4k < n-1$, the number of vertices with degree at most $4k$ in $H^{(i)}$ is zero, which is less than $\frac{120 \cdot k \cdot \mathtt{mst}(X)}{(1+\varepsilon)^i}$. ∎

5.2 Algorithm EstimateVertexDegree**.** In this section we present an algorithm EstimateVertexDegree to estimate the degree of a vertex, which will be later playing central role in algorithm EstimateSetSize (cf. Section 4.1) to estimate the sizes of the sets V_j^i, which in turns is used to analyze our main algorithm k-NNSizeApproximation(n, ε) to estimate $\mathtt{cost}(G)$.

We assume that the input is normalized as earlier discussed. The precise value of $\mathfrak{r} = O(\log n/\varepsilon)$ follows from scaling the input and the value of $\mathfrak{t} = O(\log k/\varepsilon)$ has been set up earlier as the maximum index j for V_j^i ($\mathfrak{t}$ is the smallest integer such that $k < (1+\varepsilon)^{\mathfrak{t}}$).

Our algorithm for estimating vertex degrees relies on a random sampling approach that approximates the size of the neighborhood of a given vertex in $H^{(i)}$ up to some additive error. The main point is that for larger values of $k - \deg_{H^{(i)}}(v)$ we need more samples (as there can be fewer points that together contribute significantly to the cost function), but we can use fewer samples to get a sufficient approximation for $k - \deg_{H^{(i)}}(v)$.

Our algorithm EstimateVertexDegree performs two steps: It first checks by a sampling procedure Filter (cf. Lemma 5.1 and Section 5.1.1) whether the vertex degree is significantly more than k. If this is the case, then the vertex does not contribute to the cost function and we can ignore it. If the vertex passes our test, then we know that with high probability its degree is no more than $4k$. In this case, we continue our analysis depending on the relation between j and k; if $(1+\varepsilon)^j \le \sqrt{k}$ then we compute its degree exactly, and otherwise, if $(1+\varepsilon)^j > \sqrt{k}$, then we call algorithm NeighborhoodSize(n, γ, k, v, i) to estimate $\deg_{H^{(i)}}(v)$ to within γk; cf. Section 5.3.

5.3 Algorithm NeighborhoodSize**.** We begin with description of our auxiliary algorithm NeighborhoodSize(n, γ, k, v, i) to estimate $\deg_{H^{(i)}}(v)$ to within γk using random sampling, assuming $\deg_{H^{(i)}}(v)$ is small.

NeighborhoodSize(n, γ, k, v, i)
{Estimates $\deg_{H^{(i)}}(v)$ to within γk; cf. Lemma 5.3}

Let $\mathfrak{b} = \lceil \frac{500 n \ln n}{k\gamma^2} \rceil$
Sample $u_1, \ldots, u_{\mathfrak{b}}$ i.u.r. from $V \setminus \{v\}$
For every $1 \le \ell \le \mathfrak{b}$,
    $X_\ell := 1$ if u_ℓ is a neighbor of v in $H^{(i)}$;
    $X_\ell := 0$ otherwise
return $\frac{n-1}{\mathfrak{b}} \sum_{\ell=1}^{\mathfrak{b}} X_\ell$

The following central lemma shows the properties of NeighborhoodSize.

Lemma 5.3. *Let $\deg_{H^{(i)}}(v) \le 4k$. Given v as input,* NeighborhoodSize(n, γ, k, v, i) *in time $O(\frac{n \log n}{k\gamma^2})$ returns a value $\widehat{d}$ that with probability at least $1 - \frac{1}{n^{10}}$ satisfies the following:*

$$|\widehat{d} - \deg_{H^{(i)}}(v)| \le \gamma k \ .$$

Proof. The running time of NeighborhoodSize follows from the value of s in the code.

To analyze the value of $\widehat{d}$, let $\mathfrak{X} = \sum_{\ell=1}^{\mathfrak{b}} X_\ell$ and $\mathcal{Y} := \widehat{d} = \frac{n-1}{\mathfrak{b}}\mathfrak{X}$. We have $\mathbf{Ex}[X_\ell] = \frac{\deg_{H^{(i)}}(v)}{n-1}$ for $1 \le \ell \le \mathfrak{b}$, and thus $\mathbf{Ex}[\mathfrak{X}] = \frac{\mathfrak{b} \cdot \deg_{H^{(i)}}(v)}{n-1} \ge \frac{500 \ln n \deg_{H^{(i)}}(v)}{k\gamma^2}$, $\mathbf{Ex}[\mathcal{Y}] = \deg_{H^{(i)}}(v)$.

Let us start with the case $\deg_{H^{(i)}}(v) \le \frac{1}{2}\gamma k$. We define $\delta = \frac{\gamma k}{2 \deg_{H^{(i)}}(v)} \ge 1$. By Chernoff inequality we get the following:

$$\begin{aligned}\mathbf{Pr}[\mathcal{Y} > \gamma k] &\le \mathbf{Pr}[\mathcal{Y} > (1+\delta) \cdot \mathbf{Ex}[\mathcal{Y}]] \\ &= \mathbf{Pr}[\mathfrak{X} > (1+\delta) \cdot \mathbf{Ex}[\mathfrak{X}]] \le e^{-\delta \cdot \mathbf{Ex}[\mathfrak{X}]/3} \\ &\le n^{-10} \ .\end{aligned}$$

Next, consider the case $\deg_{H^{(i)}}(v) > \frac{1}{2}\gamma k$. We again define $\delta = \frac{\gamma k}{2 \deg_{H^{(i)}}(v)} \le 1$. We get

$$\begin{aligned}\mathbf{Pr}[\mathcal{Y} - \mathbf{Ex}[\mathcal{Y}] > \gamma k] &\le \mathbf{Pr}[\mathcal{Y} > (1+\delta) \cdot \mathbf{Ex}[\mathcal{Y}]] \\ &= \mathbf{Pr}[\mathfrak{X} > (1+\delta)\mathbf{Ex}[\mathfrak{X}]] \\ &\le e^{-\delta^2 \mathbf{Ex}[\mathfrak{X}]/3} \le n^{-10}/2 \ .\end{aligned}$$

Furthermore,

$$\begin{aligned}\mathbf{Pr}[\mathbf{Ex}[\mathcal{Y}] - \mathcal{Y} > \gamma k] &\le \mathbf{Pr}[\mathcal{Y} < (1-\delta) \cdot \mathbf{Ex}[\mathcal{Y}]] \\ &= \mathbf{Pr}[\mathfrak{X} < (1-\delta)\mathbf{Ex}[\mathfrak{X}]] \\ &\le e^{-\delta^2 \mathbf{Ex}[\mathfrak{X}]/2} \le n^{-10}/2 \ .\end{aligned}$$

By the union bound, this implies the claim in the second case when $\deg_{H^{(i)}}(v) > \frac{1}{2}\gamma k$. ∎

5.3.1 Algorithm EstimateVertexDegree **and its properties.** Now we can introduce our algorithm to estimate the vertex degrees.

EstimateVertexDegree$(n, \varepsilon, k, v, i, j)$
{Estimates $k - \deg^{(i)}(v)$; cf. Lemma 5.4}

if Filter$(n, k, v, i) =$ False **then return** 0
else
  if $(1+\varepsilon)^j \leq \sqrt{k}$ **then**
    compute $\widehat{d} = \deg_{H^{(i)}}(v)$ exactly
  else
    $\widehat{d} =$NeighborhoodSize$(n, \varepsilon(1+\varepsilon)^j/k, k, v, i)$
  return $\max\{k - \widehat{d}, 0\}$
(if at any moment one queries $\Theta(n)$ distances then stop and compute $k - \deg_{H^{(i)}}(v)$ exactly)

Lemma 5.4. *Let $j \leq \mathfrak{t}$.* EstimateVertexDegree $(n, \varepsilon, k, v, i, j)$ *satisfies the following:*

- *If $\deg_{H^{(i)}}(v) \geq 4k$ then* EstimateVertexDegree *runs in $O(\min\{n, \frac{n \log n}{\deg_{H^{(i)}}(v)}\})$ expected time and with probability at least $1 - \frac{1}{n^{10}}$ returns 0.*
- *If $\deg_{H^{(i)}}(v) < 4k$ then*
  - ◇ *the expected running time of* EstimateVertexDegree *is either $O(n)$ if $(1+\varepsilon)^j \leq \sqrt{k}$, or $O(\min\{n, \frac{nk \log n}{\varepsilon^2(1+\varepsilon)^{2j}}\})$ if $(1+\varepsilon)^j > \sqrt{k}$, and*
  - ◇ *either $\deg_{H^{(i)}}(v) \geq k + \varepsilon \cdot (1+\varepsilon)^j$ and* EstimateVertexDegree *with probability at least $1 - \frac{2}{n^{10}}$ returns 0, or*
  - ◇ $\deg_{H^{(i)}}(v) < k + \varepsilon \cdot (1+\varepsilon)^j$ *and* EstimateVertexDegree *determines $\widehat{d}$ such that with probability at least $1 - \frac{2}{n^{10}}$ it holds that $|\widehat{d} - \deg_{H^{(i)}}(v)| \leq \varepsilon \cdot (1+\varepsilon)^j$.*

Proof. We consider two separate cases, when Filter(n, k, v, i) returns False and when it returns True.

By Lemma 5.1, if Filter(n, k, v, i) returns False then with probability at least $1 - \frac{1}{n^{10}}$ we have that $\deg_{H^{(i)}}(v) > k$. Therefore, indeed, algorithm EstimateVertexDegree$(n, \varepsilon, k, v, i, j)$ returns the correct value 0 of $k - \deg^{(i)}(v)$. Further, by Lemma 5.1, algorithm Filter runs in expected time $O(\frac{n \log n}{k + \deg_{H^{(i)}}(v)})$, and so since with probability at least $1 - \frac{1}{n^{10}}$ we have that $\deg_{H^{(i)}}(v) > k$, algorithm EstimateVertexDegree$(n, \varepsilon, k, v, i, j)$ runs in expected time $O(\min\{n, \frac{n \log n}{\deg_{H^{(i)}}(v)}\})$ (the $\min\{n, \}$ term is because of the last line of the code).

Next, let us consider the case when Filter(n, k, v, i) returns True. By Lemma 5.1, then with probability at least $1 - \frac{1}{n^{10}}$ we have that $\deg_{H^{(i)}}(v) < 4k$; let us now condition on that $\deg_{H^{(i)}}(v) < 4k$. In that case, the expected running time of algorithm Filter is $O(\min\{n, \frac{n \log n}{k}\})$, but EstimateVertexDegree will still perform some more work.

If ε, j, and k satisfy $(1+\varepsilon)^j \leq \sqrt{k}$, then algorithm EstimateVertexDegree will spend $O(n)$ time and exactly compute the value of $\deg_{H^{(i)}}(v)$. Therefore in this case, the expected running time of EstimateVertexDegree$(n, \varepsilon, k, v, i, j)$ is $O(\min\{n, \frac{n \log n}{k}\})$.

Otherwise, let us consider the case $(1+\varepsilon)^j \leq \sqrt{k}$, with $\deg_{H^{(i)}}(v) < 4k$. Then, after calling Filter, we invoke algorithm NeighborhoodSize$(n, \varepsilon(1+\varepsilon)^j/k, k, v, i)$. By Lemma 5.3, algorithm NeighborhoodSize$(n, \varepsilon(1+\varepsilon)^j/k, k, v, i)$ in time $O(\frac{nk \log n}{\varepsilon^2(1+\varepsilon)^{2j}})$ returns a value $\widehat{d}$ that with probability at least $1 - \frac{2}{n^{10}}$ satisfies $|\widehat{d} - \deg_{H^{(i)}}(v)| \leq \varepsilon(1+\varepsilon)^j$. Therefore, the expected running time of algorithm EstimateVertexDegree$(n, \varepsilon, k, v, i, j)$ in this case is $O(\min\{n, \frac{n \log n}{k} + \frac{nk \log n}{\varepsilon^2(1+\varepsilon)^{2j}}\})$, which for $j \leq \mathfrak{t}$ (and hence $\varepsilon(1+\varepsilon)^j = O(k)$) simplifies to $O(\min\{n, \frac{nk \log n}{\varepsilon^2(1+\varepsilon)^{2j}}\})$.

We obtain the main claim by combining the cases above. ∎

5.4 Analysis of the running time of algorithm EstimateSetSize**.** With Lemma 5.4 at hand, we are ready to analyze algorithm EstimateSetSize from Section 4.1.

Lemma 5.5. *The expected runtime of algorithm* EstimateSetSize(n, ε, i, j) *is $O\left(\frac{nk^{3/2} \log^2 n \log k}{\varepsilon^4}\right)$.*

Proof. (Let us remind that we have assumed that $k = o(n)$, and hence Lemma 5.2 (which requires $k < \frac{n-1}{4}$) holds.) We first observe that the guarantee from Lemma 5.4 suffices to make sure that with probability at least $1 - \frac{2}{n^{10}}$ a vertex is only counted towards membership in V_j^i when it is either in the class or a neighboring class (see Section 5.4.1 for a brief discussion how to avoid double counting in different classes).

Let us fix i and j, and we will analyze the expected running time to evaluate a single sample vertex by algorithm EstimateVertexDegree$(n, \varepsilon, k, v, i, j)$. We assume that the expected running time is the average over all vertices as the expected running time of the filtering algorithm holds with probability $(1 - 1/n^{10})$ in the worst case (cf. Lemma 5.1). Let us partition V into two sets V_1 and V_2, with $V_1 = \{v \in V : \deg_{H^{(i)}}(v) > 4k\}$ and $V_2 = \{v \in V : \deg_{H^{(i)}}(v) \leq 4k\}$. We will split our analysis into two separate cases, depending on whether $(1+\varepsilon)^j \leq \sqrt{k}$ or $(1+\varepsilon)^j > \sqrt{k}$.

Case 1: We begin with the case when $(1+\varepsilon)^j \leq \sqrt{k}$. The *expected* running time to evaluate a sin-

gle sample vertex by algorithm EstimateVertexDegree$(n, \varepsilon, k, v, i, j)$ is

$$\frac{1}{n} \cdot \left(\sum_{v \in V_1} O\left(\frac{n \log n}{\deg_{H^{(i)}}(v) + k} \right) + \sum_{v \in V_2} O(n) \right) .$$

Using Lemma 5.2, this bound can be simplified to $O\left(\frac{\mathfrak{mst}(X) \log n}{(1+\varepsilon)^i} + \frac{k \cdot \mathfrak{mst}(X)}{(1+\varepsilon)^i}\right)$. Plugging this into the bound above, we obtain an expected running time of EstimateSetSize(n, ε, i, j) (with s being the number of sampled vertices, $s = \lceil \frac{100n(1+\varepsilon)^{i+j}\mathfrak{rt}}{\mathfrak{mst}(X)} \rceil$):

$$\begin{aligned} & s \cdot O\left(\frac{\mathfrak{mst}(X) \log n}{(1+\varepsilon)^i} + \frac{k \cdot \mathfrak{mst}(X)}{(1+\varepsilon)^i} \right) \\ &= \frac{n(1+\varepsilon)^{i+j}\mathfrak{rt}}{\mathfrak{mst}(X)} \cdot \left(\frac{(k + \log n) \cdot \mathfrak{mst}(X)}{(1+\varepsilon)^i} \right) \\ &= O(n(k + \log n)(1+\varepsilon)^j \mathfrak{rt}) . \end{aligned}$$

Since $\mathfrak{r} = O(\log n/\varepsilon)$, $\mathfrak{t} = O(\log k/\varepsilon)$, $(1+\varepsilon)^j \leq \sqrt{k}$, we can simplify this bound to conclude that the expected running time of EstimateSetSize(n, ε, i, j) is

$$O\left(\frac{n(k + \log n)\sqrt{k} \log n \log k}{\varepsilon^2} \right) = O\left(\frac{nk^{3/2} \log^2 n \log k}{\varepsilon^2} \right) . \tag{5.5}$$

Case 2: Next, we consider the case when $(1+\varepsilon)^j > \sqrt{k}$. By Lemma 5.4, the expected running time to evaluate a single sample vertex by algorithm EstimateVertexDegree$(n, \varepsilon, k, v, i, j)$ satisfies the following:

$$\frac{1}{n} \left(\sum_{v \in V_1} O\left(\frac{n \log n}{\deg_{H^{(i)}}(v) + k} \right) + \sum_{v \in V_2} O\left(\frac{nk \log n}{\varepsilon^2 (1+\varepsilon)^{2j}} \right) \right) .$$

Using Lemma 5.2, this bound can be simplified to

$$\begin{aligned} & O\left(\frac{\mathfrak{mst}(X) \log n}{(1+\varepsilon)^i} + \frac{k^2 \cdot \mathfrak{mst}(X) \log n}{\varepsilon^2 (1+\varepsilon)^{i+2j}} \right) \\ &= O\left(\frac{\mathfrak{mst}(X) \log n}{(1+\varepsilon)^i} \cdot \left(1 + \frac{k^2}{\varepsilon^2 (1+\varepsilon)^{2j}} \right) \right) . \end{aligned}$$

Since $(1+\varepsilon)^j > \sqrt{k}$ and $\varepsilon \cdot (1+\varepsilon)^j \leq \varepsilon \cdot (2k) \leq 2k$, we can simplify it further to

$$\begin{aligned} & O\left(\frac{\mathfrak{mst}(X) \log n}{(1+\varepsilon)^i} \cdot \left(1 + \frac{k^2}{\varepsilon^2 (1+\varepsilon)^{2j}} \right) \right) \\ &= O\left(\frac{\mathfrak{mst}(X) \log n}{(1+\varepsilon)^i} \cdot \frac{k^{3/2}}{\varepsilon^2 (1+\varepsilon)^j} \right) . \end{aligned}$$

Using this bound, we obtain the expected running time of EstimateSetSize(n, ε, i, j) :

$$\begin{aligned} & s \cdot O\left(\frac{k^{3/2} \cdot \mathfrak{mst}(X) \log n}{\varepsilon^2 \cdot (1+\varepsilon)^{i+j}} \right) \\ &= O\left(\frac{n(1+\varepsilon)^{i+j}\mathfrak{rt}}{\mathfrak{mst}(X)} \cdot \left(\frac{k^{3/2} \cdot \mathfrak{mst}(X) \log n}{\varepsilon^2 \cdot (1+\varepsilon)^{i+j}} \right) \right) \\ &= O\left(\frac{nk^{3/2}\mathfrak{rt} \log n}{\varepsilon^2} \right) . \end{aligned}$$

Since $\mathfrak{r} = O(\log n/\varepsilon)$, $\mathfrak{t} = O(\log k/\varepsilon)$, the expected running time of EstimateSetSize is

$$O\left(\frac{nk^{3/2} \log^2 n \log k}{\varepsilon^4} \right) . \tag{5.6}$$

We can combine the two cases in (5.5) and (5.6) to conclude the proof of Lemma 5.5. ∎

5.4.1 Consistency. In order to avoid double counting, we need to make sure that our answers are consistent. In order to ensure this with high probability, we will assume as a thought experiment that we run algorithm EstimateVertexDegree for different values of j. We will use the estimate for the largest value of j such that the error interval is such that the vertex could be placed into at most two different sets V_j^i. Once this happens, we put the vertex into the set that is determined by its estimate. If all estimates are correct, then we will be at most "one class" off. Note that whenever the confidence interval intersects more than one class, we will assume that the vertex is not contained in the corresponding class, i.e., the vertex does not contribute to our estimate. We observe that we can always simulate this process in the same running time as before, if a vertex is in more than one sample set. We start by evaluating the largest j and proceed in decreasing order until the class is determined (or all j have been evaluated).

5.5 Analysis of the performance of k-NNSizeApproximation. In this section we analyze the running time and the approximation guarantee of algorithm k-NNSizeApproximation. For this purpose, we will assume that algorithm EstimateVertexDegree always provides the correct answer. This happens with probability at least $1 - 1/n^{10}$. We first analyze the quality of EstimateSetSize.

Lemma 5.6. *For every $0 \leq i \leq \mathfrak{r}$, $1 \leq j \leq \mathfrak{t}$,*

$$|V_j^i| \cdot (1+\varepsilon)^{i+j} \leq \mathfrak{cost}(G) .$$

Proof. V_j^i is the set of vertices v with $(1+\varepsilon)^{j-1} \leq k - \deg^{(i)}(v)$. Therefore any vertex $v \in V_j^i$ has at least

$(1+\varepsilon)^{j-1}$ neighbors in G whose cost is strictly greater than $(1+\varepsilon)^i$, and thus at least $(1+\varepsilon)^{i+1}$. Hence, a vertex in V_j^i contributes at least $(1+\varepsilon)^{i+j}$ to $\mathfrak{cost}(G)$, which yields the result. ∎

LEMMA 5.7. *For every $0 \le i \le \mathfrak{r}$, $1 \le j \le \mathfrak{t}$, $X_{i,j}$ in k-NNSIZEAPPROXIMATION(n,ε) is a random variable that satisfies the following inequality:*

$$\mathbf{Var}[X_{i,j}] \le \frac{\mathfrak{mst}(X)\cdot\mathfrak{cost}(G)}{100\cdot(1+\varepsilon)^{2(i+j)}\cdot\mathfrak{r}\cdot\mathfrak{t}} \ .$$

Proof. Fix i and j, $0 \le i \le \mathfrak{r}$, $1 \le j \le \mathfrak{t}$. For a fixed i,j, ESTIMATESETSIZE(n,ε,i,j) samples $s = s(i,j) = \lceil \frac{100n(1+\varepsilon)^{i+j}\mathfrak{rt}}{\mathfrak{mst}(X)} \rceil$ random vertices $u_1,\ldots,u_s$, for which it then calls ESTIMATEVERTEXDEGREE. Let Y_ℓ be the number returned by ESTIMATEVERTEXDEGREE when applied to the sampled vertex u_ℓ in ESTIMATESETSIZE(n,ε,i,j). Y_ℓ is an indicator random variable for the event $u_\ell \in V_j^i$ (for an i.u.r. choice of u_ℓ in V), and thus $\mathbf{Pr}[Y_\ell = 1] = \mathbf{Pr}[u_\ell \in V_j^i] = \frac{|V_j^i|}{n}$ and $\mathbf{Var}[Y_\ell] \le \mathbf{Pr}[Y_\ell = 1] = \frac{|V_j^i|}{n}$. Hence, since $X_{i,j} = \sum_{\ell=1}^{s} Y_\ell$, we obtain,

$$\begin{aligned}\mathbf{Var}[X_{i,j}] &= \mathbf{Var}\left[\frac{n}{s}\cdot\sum_{\ell=1}^{s} Y_\ell\right] = \frac{n^2}{s^2}\cdot\sum_{\ell=1}^{s}\mathbf{Var}[Y_\ell] \\ &\le \frac{n^2}{s^2}\cdot s\cdot\frac{|V_j^i|}{n} = \frac{n}{s}\cdot|V_j^i| \le \frac{n}{s}\cdot\frac{\mathfrak{cost}(G)}{(1+\varepsilon)^{i+j}} \\ &\le \frac{\mathfrak{mst}(X)\cdot\mathfrak{cost}(G)}{100\cdot(1+\varepsilon)^{2(i+j)}\cdot\mathfrak{r}\cdot\mathfrak{t}} \ ,\end{aligned}$$

where the penultimate inequality follows from Lemma 5.6 and the last inequality follows from the fact that $s = \lceil \frac{100n(1+\varepsilon)^{i+j}\mathfrak{rt}}{\mathfrak{mst}(X)} \rceil$. ∎

THEOREM 5.1. *Algorithm k-NNSIZEAPPROXIMATION (n,ε) in expected time $O\left(\frac{nk^{3/2}\log^3 n\log^2 k}{\varepsilon^6}\right)$ returns a value $\overline{\mathfrak{cost}}$ such that with probability at least $\frac{2}{3}$, we have*

$$|\mathfrak{cost}(G) - \overline{\mathfrak{cost}}| \le \varepsilon\cdot(\mathfrak{mst}(X) + \mathfrak{cost}(G)) \ .$$

Proof. The running time of algorithm k-NNSIZEAPPROXIMATION(n,ε) follows immediately from Lemma 5.5 and since $\mathfrak{r} = O(\log n/\varepsilon)$ and $\mathfrak{t} = O(\log k/\varepsilon)$.

Next, let us analyze the performance guarantee of algorithm k-NNSIZEAPPROXIMATION(n,ε). Algorithm k-NNSIZEAPPROXIMATION(n,ε) returns a value

$$\overline{\mathfrak{cost}} = nk + \varepsilon\cdot\sum_{i=0}^{\mathfrak{r}}\sum_{j=1}^{\mathfrak{t}}(1+\varepsilon)^{i+j}X_{i,j} \ ,$$

where $X_{i,j}$ are random numbers studied in Lemma 5.7. Since the first term is deterministic, our goal is to analyze the concentration of $\sum_{i=0}^{\mathfrak{r}}\sum_{j=1}^{\mathfrak{t}}(1+\varepsilon)^{i+j}X_{i,j}$ around its mean.

We condition on the event that all calls to algorithm ESTIMATEVERTEXDEGREE provide the correct answer; by Lemma 5.4, this happens with probability at least $1 - 1/n^{10}$.

Let $\mathfrak{X} = \sum_{i=0}^{\mathfrak{r}}\sum_{j=1}^{\mathfrak{t}}(1+\varepsilon)^{i+j}X_{i,j}$. We use Chebyshev's inequality to analyze the concentration of $\mathfrak{X}$:

$$\begin{aligned}&\mathbf{Pr}\left[|\mathfrak{X} - \mathbf{Ex}[\mathfrak{X}]| \ge \mathfrak{mst}(X) + \mathfrak{cost}(G))\right] \\ &\le \frac{\mathbf{Var}[\mathfrak{X}]}{(\mathfrak{mst}(X)+\mathfrak{cost}(G))^2} \\ &= \frac{\mathbf{Var}[\sum_{i=0}^{\mathfrak{r}}\sum_{j=1}^{\mathfrak{t}}\left((1+\varepsilon)^{i+j}X_{i,j}\right)]}{(\mathfrak{mst}(X)+\mathfrak{cost}(G))^2} \\ &= \frac{\sum_{i=0}^{\mathfrak{r}}\sum_{j=1}^{\mathfrak{t}}\left((1+\varepsilon)^{2(i+j)}\mathbf{Var}[X_{i,j}]\right)}{(\mathfrak{mst}(X)+\mathfrak{cost}(G))^2} \\ &\le \frac{\sum_{i=0}^{\mathfrak{r}}\sum_{j=1}^{\mathfrak{t}}\left((1+\varepsilon)^{2(i+j)}\cdot\left(\frac{\mathfrak{mst}(X)\cdot\mathfrak{cost}(G)}{100\cdot(1+\varepsilon)^{2(i+j)}\cdot\mathfrak{r}\cdot\mathfrak{t}}\right)\right)}{2\cdot\mathfrak{mst}(X)\cdot\mathfrak{cost}(G)} \\ &= \frac{(\mathfrak{r}+1)\cdot\mathfrak{t}}{200\cdot\mathfrak{r}\cdot\mathfrak{t}} \\ &\le \frac{1}{100} \ .\end{aligned}$$

We remark that the bound also holds when we use a factor 2 approximation for $\mathfrak{mst}(X)$ in the sample size. Finally, we need to rescale ε by some constant to take care of the additional errors introduced by rounding and the formula using the set V_j^i.

Let us apply Lemma 4.1 to bound $\mathfrak{cost}(G) - \overline{\mathfrak{cost}}$. Notice that $\mathbf{Ex}[\mathfrak{X}] = \sum_{i=0}^{\mathfrak{r}}\sum_{j=1}^{\mathfrak{t}}(1+\varepsilon)^{i+j}\cdot|V_j^i|$ and that $\overline{\mathfrak{cost}} = nk + \varepsilon\cdot\mathfrak{X}$. By Lemma 4.1 we have,

$$\mathfrak{cost}(G) \le nk + \varepsilon\cdot\mathbf{Ex}[\mathfrak{X}] \le (1+\varepsilon)\cdot\mathfrak{cost}(G) \ ,$$

and by our analysis above, with probability at least $1 - \frac{1}{100}$ we have

$$|\mathfrak{X} - \mathbf{Ex}[\mathfrak{X}]| < \mathfrak{mst}(X) + \mathfrak{cost}(G) \ ,$$

what is equivalent to

$$|(nk + \varepsilon\mathfrak{X}) - (nk + \varepsilon\mathbf{Ex}[\mathfrak{X}])| < \varepsilon(\mathfrak{mst}(X) + \mathfrak{cost}(G)) \ ,$$

yielding the following (with probability at least $1 - \frac{1}{100}$):

$$|\overline{\mathfrak{cost}} - (nk + \varepsilon\cdot\mathbf{Ex}[\mathfrak{X}])| < \varepsilon\cdot(\mathfrak{mst}(X) + \mathfrak{cost}(G)) \ .$$

This implies that with probability at least $1 - \frac{1}{100}$ the following holds:

$$|\mathfrak{cost}(G) - \overline{\mathfrak{cost}}| \le \varepsilon\cdot(\mathfrak{mst}(X) + \mathfrak{cost}(G)) \ .$$

∎

6 Bypassing dependency on $\mathtt{mst}(X)$: $(1+\varepsilon)$-approximation of $\mathtt{cost}(X)$ with $\widetilde{O}_\varepsilon(\frac{n^2}{k})$ queries

In this section we prove Theorem 6.1: we develop another algorithm to approximate $\mathtt{cost}(G)$ with a smaller number of queries when k is large, improving upon the algorithm from Theorem 5.1 when $k = \widetilde{\omega}_\varepsilon(n^{2/5})$. Furthermore, unlike the $\widetilde{O}_\varepsilon(nk^{3/2})$-time algorithm from Theorem 5.1 which approximates $\mathtt{cost}(G)$ with an additive error term of $\varepsilon \cdot (\mathtt{mst}(X) + \mathtt{cost}(G))$, our new algorithm with $\widetilde{O}_\varepsilon(n^2/k)$ queries finds the approximation guarantee that differs from the optimal value within an $\varepsilon \cdot \mathtt{cost}(G)$ additive term, i.e., independently of $\mathtt{mst}(X)$.

Let $G = (V, E)$ be a k-NN graph. For any $v \in V$, order all other vertices in V according to the distance from v, and call the ith vertex in this order the ith nearest neighbor of v, or the v's neighbor of *rank* i. Notice that in the k-NN graph G any v is connected by an edge $[v, u\rangle$ only to vertices u with rank at most k.

Our algorithm uses the following idea. First, for each vertex v, we approximate a *median neighbor* u_v, which is a vertex which is approximately the $k/2^{\text{th}}$ nearest neighbor of v. Then we divide the edges into two sets. The set E_S of *short edges* contains all edges of distance at most $10d(v, u_v)$ and the set E_L of *long edges* contains the remaining edges of G; that is, $E_S = \bigcup_{v \in V}\{[v, u\rangle \in E : d(v, u) \leq 10d(v, u_v)\}$ and $E_L = E \setminus E_S$. We will separately estimate the sum of lengths of short edges and then the sum of lengths of long edges. We will approximate the contribution of the edges from E_S by sampling vertices v with probability proportional to $d(v, u_v)$, relying on the property (cf. Claim 6.1) that by the choice of u_v, we have $k \cdot \sum_{v \in V} d(v, u_v) = \Omega(\mathtt{cost}(G))$. The contribution of the long edges is approximated by uniform sampling of vertices and computing the length of all incident edges in E_L; the central property here is that for every long edge there is also another edge of roughly the same length from every neighbor of v that has rank smaller than the rank of u_v. We will give more details in the remainder of this section.

ApproximatekNNLargek(n, ε, k)
{Returns a $(1+\varepsilon)$-approx. of $\mathtt{cost}(G)$; cf. Lemma 6.1}

Let $\mathfrak{s} := \lceil \frac{1000(n-1)\log n}{k} \rceil$
for each $v \in V$ **do**
  Sample $\mathfrak{s}$ vertices i.u.r. from $V \setminus \{v\}$
  Check the distances of the sampled vertices to v
  Let u_v be the vertex of rank $\lceil 500 \log n \rceil$ in this set
$Z_1 =$ ApproximateShortEdges$(n, \varepsilon/8, k)$
$Z_2 =$ ApproximateLongEdges$(n, \varepsilon/2, k)$
return $Z_1 + Z_2$

In the reminder of this section, we will show that the above algorithm ApproximatekNNLargek(n, ε, k) is a $(1+\varepsilon)$-approximation of $\mathtt{cost}(G)$.[3]

In Sections 6.1–6.3 we will prove three central properties of ApproximatekNNLargek, that the vertices u_v are good approximations of the median neighbors, and that routines ApproximateShortEdges and ApproximateLongEdges provide good approximations of the sum of lengths of short edges and long edges, respectively. We will combine all these claims together in Section 6.4, where we will conclude the analysis of the properties of algorithm ApproximatekNNLargek in the final Theorem 6.1.

6.1 Approximating median neighbors.

Our first lemma shows that for each vertex $v \in V$, vertex u_v found by algorithm ApproximatekNNLargek is an approximate median neighbor v.

Lemma 6.1. *Let v be a vertex in $H = (V, E)$ and let its neighbors $q_1, \ldots, q_{n-1}$ be sorted by distance to v, i.e., $d(v, q_i) \leq d(v, q_j)$ for $i < j$. Let u_v be the vertex as defined in algorithm* ApproximatekNNLargek. *Then with probability at least $1 - \frac{1}{n^4}$, we have*

$$\frac{k}{4} \leq \mathrm{rank}(u_v) \leq \frac{3k}{4} \ ,$$

where $\mathrm{rank}(u_v)$ *is to the rank of u_v in $q_1, \ldots, q_{n-1}$.*

Proof. Fix $v \in V$. Let $\mathcal{S}$ be the set of vertices with rank smaller than or equal to $\frac{k}{4}$. Let us use X_i to denote the indicator random variable that we sample a vertex from $\mathcal{S}$ in the i-th sampling step; clearly, $\mathbf{Ex}[X_i] = \frac{\lfloor k/4 \rfloor}{n-1}$. Observe that $\mathrm{rank}(u_v) < \frac{k}{4}$ iff $u_v \in \mathcal{S}$, i.e., if the sampled multi-set of $\mathfrak{s}$ random vertices from $V \setminus \{v\}$ contains at least $\lceil 500 \log n \rceil$ vertices from $\mathcal{S}$. For that to happen, we need $\sum_{i=1}^{\mathfrak{s}} X_i \geq \lceil 500 \log n \rceil$. Notice the following,

$$\mathbf{Ex}\left[\sum_{i=1}^{\mathfrak{s}} X_i\right] \leq \mathfrak{s} \cdot \frac{k}{4(n-1)} \leq 251 \log n \ .$$

Using the above inequality and then a Chernoff bound,

$$\begin{aligned}\mathbf{Pr}[u_v \in \mathcal{S}] &= \mathbf{Pr}\left[\sum_{i=1}^{\mathfrak{s}} X_i \geq \lceil 500 \log n \rceil\right] \\ &\leq \mathbf{Pr}\left[\sum_{i=1}^{\mathfrak{s}} X_i \geq \tfrac{4}{3} \cdot \mathbf{Ex}[\sum_{i=1}^{\mathfrak{s}} X_i]\right] \\ &\leq e^{-\mathbf{Ex}[\sum_{i=1}^{\mathfrak{s}} X_i]/27} \ .\end{aligned}$$

[3] In algorithm ApproximatekNNLargek, while for every vertex v we sample $\mathfrak{s}$ vertices i.u.r., and hence *with replacement*, we compute the rank in the *multi-set* of the sampled vertices, that is, to compute the rank we take each sampled copy into account.

Next, we observe that

$$\mathbf{Ex}\left[\sum_{i=1}^{\mathfrak{s}} X_i\right] \geq \mathfrak{s} \cdot \frac{k-3}{4(n-1)} \geq 125 \log n \ ,$$

where we assume $k \geq 6$ (if $k < 6$ we can afford to compute the k-NN graph using brute force). Plugging in the bound on the expectation into the previous inequality yields

$$\mathbf{Pr}\left[\text{rank}(u_v) < \frac{k}{4}\right] = \mathbf{Pr}[u_v \notin \mathcal{S}] \geq 1 - \frac{1}{2n^4} \ .$$

We proceed similarly to prove the second inequality. Consider the set $\mathcal{R}$ of all vertices of rank at most $\frac{3k}{4}$. Let Y_i be the indicator random variable that we sample a vertex from $\mathcal{R}$ in the i-th sampling step; clearly $\mathbf{Ex}[Y_i] = \frac{\lfloor 3k/4 \rfloor}{n-1} \geq \frac{3k-3}{4(n-1)}$. Notice that $\text{rank}(u_v) > \frac{3k}{4}$ iff $u_v \notin \mathcal{R}$, i.e., if the sampled multi-set of $\mathfrak{s}$ random vertices from $V \setminus \{v\}$ contains strictly less than $\lceil 500 \log n \rceil$ vertices from $\mathcal{R}$, that is, if $\sum_{i=1}^{\mathfrak{s}} Y_i < \lceil 500 \log n \rceil$. Notice the following,

$$\mathbf{Ex}\left[\sum_{i=1}^{\mathfrak{s}} Y_i\right] = \mathfrak{s} \cdot \frac{\lfloor 3k/4 \rfloor}{n-1} \geq \mathfrak{s} \cdot \frac{3k-3}{4(n-1)} \geq 625 \log n \ ,$$

assuming that $k \geq 6$. We combine the above inequality with a Chernoff bound and get

$$\begin{aligned}
\mathbf{Pr}[u_v \notin \mathcal{R}] &= \mathbf{Pr}\left[\sum_{i=1}^{\mathfrak{s}} Y_i < \lceil 500 \log n \rceil\right] \\
&\leq \mathbf{Pr}\left[\sum_{i=1}^{\mathfrak{s}} Y_i \leq \tfrac{4}{5} \cdot \mathbf{Ex}[\sum_{i=1}^{\mathfrak{s}} Y_i]\right] \\
&\leq e^{-\mathbf{Ex}[\sum_{i=1}^{\mathfrak{s}} Y_i]/50} \ .
\end{aligned}$$

Finally, it follows from our lower bound on $\mathbf{Ex}[\sum_{i=1}^{\mathfrak{s}} Y_i]$ that the right hand side inequality of the lemma with probability $1 - \frac{1}{2n^4}$, giving the following

$$\mathbf{Pr}\left[\text{rank}(u_v) > \frac{3k}{4}\right] = \mathbf{Pr}[u_v \in \mathcal{R}] \geq 1 - \frac{1}{2n^4} \ .$$

The union bound then yields that the lemma holds with probability at least $1 - \frac{1}{n^4}$. ∎

6.2 Approximating total length of short edges.

In this section, we present our sampling routine ApproximateShortEdges$(n, \varepsilon/8, k)$ that approximates the sum of lengths of short edges, that is, $\sum_{[v,u\rangle \in E_S} d(v,u)$.

We assume that for every vertex $v \in V$ we have found a vertex $u_v \in V \setminus \{v\}$ using the routine from ApproximatekNNLargek(n, ε, k). Then, $E_S = \{[v,u\rangle \in E : d(u,v) \leq 10 d(v,u_v)\}$. For every $v \in V$, let $\mathcal{S}_v$ denote the sum of distances to the k nearest neighbors of v in G that are at distance at most $10d(v,u_v)$, that is, $\mathcal{S}_v = \sum_{[v,u\rangle \in E_S} d(v,u)$. We use the following simple claim.

Claim 6.1. $\mathcal{S}_v \leq 10k \cdot d(v,u_v)$.

Proof. There are at most k outgoing edges incident to v in G and all such edges in E_S have distance at most $10 \cdot d(v,u_v)$. ∎

Now we define the sampling algorithm.

ApproximateShortEdges(n, ε, k)
{Estimates the length of *short edges* to within $\varepsilon \cdot \mathtt{cost}(G)$; cf. Lemma 6.2}

for each $v \in V$ compute $p_v = \frac{d(v,u_v)}{\sum_{w \in V} d(w,u_w)}$
for $i = 1$ **to** $\mathfrak{a} = \lceil 800/\varepsilon^2 \rceil$ **do**
  Sample a vertex v according to the distribution $\mathbf{Pr}[v = u] = p_u$
  Compute $\mathcal{S}_v$
  Let $\vartheta_i = \mathcal{S}_v / p_v$
return $Z_1 = \frac{1}{\mathfrak{a}} \cdot \sum_{i=1}^{\mathfrak{a}} \vartheta_i$

Lemma 6.2. ApproximateShortEdges(n, ε, k) *with $O(n/\varepsilon^2)$ queries returns an estimate Z_1 such that with probability at least $\frac{7}{8}$, we have*

$$|Z_1 - \sum_v \mathcal{S}_{v \in V}| \leq \varepsilon k \sum_{v \in V} d(v,u_v) \ .$$

Proof. Notice that $\mathbf{Ex}[\vartheta_i] = \sum_{v \in V} p_v \cdot \frac{\mathcal{S}_v}{p_v} = \sum_{v \in V} \mathcal{S}_v$ and hence $\mathbf{Ex}[Z_1] = \mathbf{Ex}[\frac{1}{\mathfrak{a}} \cdot \sum_{i=1}^{\mathfrak{a}} \vartheta_i] = \sum_{v \in V} \mathcal{S}_v$, and so Z_1 is an unbiased estimator. Next we observe that by Claim 6.1 we get,

$$\frac{\mathcal{S}_v}{p_v} \leq \frac{10k \cdot d(v,u_v)}{p_v} = 10 \cdot k \cdot \sum_{w \in V} d(w,u_w) \ .$$

Next, for every $1 \leq i \leq \mathfrak{a}$, using the inequality above we have

$$\begin{aligned}
\mathbf{Var}[\vartheta_i] \leq \mathbf{Ex}[\vartheta_i^2] &= \sum_{v \in V} p_v \cdot \left(\frac{\mathcal{S}_v}{p_v}\right)^2 \\
&\leq \sum_{v \in V} p_v \cdot \left(10 \cdot k \cdot \sum_{w \in V} d(w,u_w)\right)^2 \\
&= 100 \cdot k^2 \cdot \left(\sum_{w \in V} d(w,u_w)\right)^2 \cdot \sum_{v \in V} p_v \\
&= 100 \cdot k^2 \cdot \left(\sum_{w \in V} d(w,u_w)\right)^2 \ .
\end{aligned}$$

Therefore, by independence of the ϑ_i it follows that

$$\mathbf{Var}[Z_1] = \mathbf{Var}\left[\frac{1}{\mathfrak{a}} \cdot \sum_{i=1}^{\mathfrak{a}} \vartheta_i\right] = \frac{1}{\mathfrak{a}^2} \cdot \sum_{i=1}^{\mathfrak{a}} \mathbf{Var}[\vartheta_i]$$
$$\leq \frac{1}{\mathfrak{a}} \cdot 100 \cdot k^2 \cdot \left(\sum_{w \in V} d(w, u_w)\right)^2 \ .$$

Now, we apply Chebyshev's inequality to obtain

$$\mathbf{Pr}\left[|Z_1 - \mathbf{Ex}[Z_1]| \geq \varepsilon k \sum_{v \in V} d(v, u_v)\right]$$
$$\leq \frac{\mathbf{Var}[Z_1]}{\varepsilon^2 k^2 \left(\sum_{v \in V} d(v, u_v)\right)^2} \leq \frac{100}{\varepsilon^2 \mathfrak{a}} \ .$$

Finally, the result follows from our choice of $\mathfrak{a} = \lceil 800/\varepsilon^2 \rceil$ in the algorithm. □

6.3 Approximating total length of long edges. In this section, we present our sampling routine ApproximateShortEdges$(n, \varepsilon/8, k)$ that approximates the sum of lengths of long edges, that is, $\sum_{[v,u\rangle \in E_L} d(v,u)$ with $E_L = \{[v,u\rangle \in E : d(u,v) > 10d(v,u_v)\}$. For every $v \in V$, let $\mathcal{L}_v$ denote the sum of distances to the k nearest neighbors of v in G that are at distance greater than $10d(v,u_v)$ from v, that is, $\mathcal{L}_v = \sum_{[v,u\rangle \in E_L} d(v,u)$. We start with a simple auxiliary claim.

Claim 6.2. *With probability at least $1 - \frac{1}{n^3}$, for every vertex $v \in V$ we have,*

$$\mathcal{L}_v \leq \frac{40}{9k} \cdot \mathtt{cost}(G) \ .$$

Proof. Let us condition on that the bounds of Lemma 6.1 are satisfied, that is, that for every vertex $v \in V$ we have $\frac{k}{4} \leq \mathrm{rank}(u_v) \leq \frac{3k}{4}$. This happens with probability at least $1 - 1/n^3$ for all vertices.

Let $w_1, \ldots, w_k$ be the set of the k nearest neighbors of v in G, sorted in order of increasing distance from v, that is, $d(v,w_1) \leq \cdots \leq d(v,w_k)$. Since Claim 6.2 trivially holds when $\mathcal{L}_v = 0$, let us assume that $\mathcal{L}_v > 0$ and define ℓ such that $d(v,w_{\ell-1}) \leq 10 \cdot d(v,u_v) < d(v,w_\ell)$. Let $\mathcal{C}_v = \{z \in V : d(v,z) \leq d(v,u_v)\}$. Let x be an arbitrary vertex in $\mathcal{C}_v$. Notice that for $i \geq \ell$, since $d(v,x) \leq d(v,u_v)$ and $d(v,w_i) > 10 \cdot d(v,u_v)$, we have

$$d(v,x) \leq d(v,u_v) \leq \tfrac{1}{10} \cdot d(v,w_i) \ .$$

This inequality, when combined with triangle inequality, immediately yields

$$d(v,w_i) \leq d(v,x) + d(x,w_i) \leq \tfrac{1}{10} \cdot d(v,w_i) + d(x,w_i) \ ,$$

and hence

$$d(v,w_i) \leq \tfrac{10}{9} \cdot d(x,w_i) \ . \tag{6.7}$$

For a fixed $x \in \mathcal{C}_v$, let us consider all k-nearest neighbors $y_1, \ldots, y_k$ of x in G, and let us order them so that if a vertex w_i is among the k-nearest neighbors of x then $y_i = w_i$. By definition, if $y_i \neq w_i$, then $d(v,w_k) \leq d(v,y_i)$, and therefore, by triangle inequality, $d(v,w_i) \leq d(v,w_k) \leq d(v,y_i) \leq d(v,x) + d(x,y_i)$. Further, since $d(v,x) \leq d(v,u_v)$, and if $i \geq \ell$ then $10d(v,u_v) < d(v,w_i)$, we will get that for $i \geq \ell$,

$$d(v,w_i) \leq d(v,x) + d(x,y_i) \leq d(v,u_v) + d(x,y_i)$$
$$\leq \tfrac{1}{10} d(v,w_i) + d(x,y_i) \ ,$$

giving,

$$d(v,w_i) \leq \tfrac{10}{9} d(x,y_i) \ . \tag{6.8}$$

If we combine (6.7) and (6.8), then we obtain that for every $i \geq \ell$, it holds

$$d(v,w_i) \leq \tfrac{10}{9} d(x,y_i) \ ,$$

and therefore

$$\mathcal{L}_v = \sum_{i=\ell}^{k} d(v,w_i) \leq \sum_{i=\ell}^{k} \tfrac{10}{9} d(x,y_i) \leq \tfrac{10}{9} \sum_{i=1}^{k} d(x,y_i)$$
$$= \tfrac{10}{9}(\mathcal{S}_x + \mathcal{L}_x) \ .$$

The inequality $\mathcal{L}_v \leq \frac{10}{9}(\mathcal{S}_x + \mathcal{L}_x)$ holds for every x in $\mathcal{C}_v$. By Lemma 6.1, we know that $|\mathcal{C}_v| \geq k/4$, and therefore,

$$\tfrac{k}{4}\mathcal{L}_v \leq \sum_{x \in \mathcal{C}_v} \mathcal{L}_v \leq \sum_{x \in \mathcal{C}_v} \tfrac{10}{9}(\mathcal{S}_x + \mathcal{L}_x) \leq \sum_{x \in V} \tfrac{10}{9}(\mathcal{S}_x + \mathcal{L}_x)$$
$$\leq \tfrac{10}{9} \cdot \mathtt{cost}(G) \ ,$$

what yields the claim. □

With Claim 6.2 at hand, we can now first describe and then analyze the algorithm to estimate the sum of lengths of long edges ApproximateLongEdges. It samples vertices i.u.r. and uses their contribution to the sum of lengths of long edges as an estimate.

ApproximateLongEdges(n, ε, k)
{Estimates the length of *long edges* to within $\varepsilon \cdot \mathtt{cost}(G)$; cf. Lemma 6.3}

for $i = 1$ **to** $\mathfrak{b} = \lceil \frac{36n}{\varepsilon^2 k} \rceil$ **do**
  Sample a vertex $v \in V$ i.u.r.
  Compute $\mathcal{L}_v$ and set $\mathfrak{l}_i = \mathcal{L}_v$
return $Z_2 = \frac{n}{\mathfrak{b}} \cdot \sum_{i=1}^{\mathfrak{b}} \mathfrak{l}_i$

LEMMA 6.3. APPROXIMATELONGEDGES(n, ε, k) *with* $O\left(\frac{n^2}{\varepsilon^2 k}\right)$ *queries returns an estimate* Z_2 *such that with probability at least* $\frac{7}{8}$*, we have*

$$|Z_2 - \sum_{v \in V} \mathcal{L}_v| \leq \varepsilon \cdot \mathfrak{cost}(G) \ .$$

Proof. The number of queries of $O\left(\frac{n^2}{\varepsilon^2 k}\right)$ of algorithm APPROXIMATELONGEDGES(n, ε, k) follows immediately from our choice of $\mathfrak{b}$ in the algorithm.

In order to analyze the quality of the estimate Z_2, notice first that $\mathbf{Ex}[\mathfrak{l}_i] = \frac{1}{n} \cdot \sum_{v \in V} \mathcal{L}_v$ and so $\mathbf{Ex}[Z_2] = \mathbf{Ex}[\frac{n}{\mathfrak{b}} \cdot \sum_{i=1}^{\mathfrak{b}} \mathfrak{l}_i] = \sum_{v \in V} \mathcal{L}_v$. Next, by Claim 6.2, we obtain the following,

$$\begin{aligned}
\mathbf{Var}[\mathfrak{l}_i] \leq \mathbf{Ex}[\mathfrak{l}_i^2] &= \frac{1}{n} \cdot \sum_{v \in V} (\mathcal{L}_v)^2 \\
&\leq \frac{1}{n} \cdot \sum_{v \in V} \left(\mathcal{L}_v \cdot \frac{40 \cdot \mathfrak{cost}(G)}{9k} \right) \\
&= \frac{40 \cdot \mathfrak{cost}(G)}{9kn} \cdot \sum_{v \in V} \mathcal{L}_v \leq \frac{40 \cdot (\mathfrak{cost}(G))^2}{9kn} \ .
\end{aligned}$$

Then, using the independence of the $\mathfrak{l}_i$, we obtain the following,

$$\begin{aligned}
\mathbf{Var}[Z_2] = \mathbf{Var}\left[\frac{n}{\mathfrak{b}} \cdot \sum_{i=1}^{\mathfrak{b}} \mathfrak{l}_i \right] &= \frac{n^2}{\mathfrak{b}^2} \cdot \sum_{i=1}^{\mathfrak{b}} \mathbf{Var}[\mathfrak{l}_i] \\
&\leq \frac{n^2}{\mathfrak{b}} \cdot \frac{40(\mathfrak{cost}(G))^2}{9kn} = \frac{40n(\mathfrak{cost}(G))^2}{9k\mathfrak{b}} \ .
\end{aligned}$$

Now we apply Chebyshev's inequality to obtain

$$\begin{aligned}
\mathbf{Pr}\left[|Z_2 - \mathbf{Ex}[Z_2]| \geq \varepsilon \cdot \mathfrak{cost}(G) \right] &\leq \frac{\mathbf{Var}[Z_2]}{\varepsilon^2 \cdot \mathfrak{cost}(G)^2} \\
&\leq \frac{40n}{9\varepsilon^2 k \mathfrak{b}} \ .
\end{aligned}$$

Finally, the result follows from our choice of $\mathfrak{b}$ in the algorithm. ∎

6.4 Completing the analysis. Now we are ready to complete the analysis of our algorithm APPROXIMATENNLARGEK.

THEOREM 6.1. *Algorithm* APPROXIMATENNLARGEK (n, ε, k) *computes with probability at least* $\frac{2}{3}$ *a value* $\overline{\mathfrak{cost}}$ *with*

$$|\mathfrak{cost}(G) - \overline{\mathfrak{cost}}| \leq \varepsilon \cdot \mathfrak{cost}(G) \ .$$

The algorithm performs $O(\frac{n^2 \log n}{\varepsilon^2 k})$ *queries to the distance oracle.*

Proof. We first observe that by Lemma 6.1, with probability at least $1 - 1/n^3$ we have that

$$\mathfrak{cost}(G) \geq \frac{k}{4} \cdot \sum_{v \in V} d(v, u_v) \ .$$

Thus, using APPROXIMATESHORTEDGES and APPROXIMATELONGEDGES with parameters $\varepsilon/8$ and $\varepsilon/2$, respectively, by Lemmas 6.2 and 6.3, our approximation of Z_1 and Z_2 has an additive error of at most $\varepsilon \cdot \mathfrak{cost}(G)$ with probability at least $1 - \frac{3}{4} - \frac{1}{n^3}$. This yields the first part of the theorem.

The number of queries of algorithm APPROXIMATENNLARGEK follows immediately from our setting of $\mathfrak{s} = O\left(\frac{n \log n}{k}\right)$ and from Lemmas 6.2 and 6.3. ∎

Repeating the algorithm $O(\log n)$ times and returning the median estimate will provide this approximation with probability at least $1 - \frac{1}{n^{10}}$. We remark that this algorithm does not require the cost of the minimum spanning tree to be small.

7 Conclusions

In this paper, we present a rather complete picture of the complexity of the problem of approximating $\mathfrak{cost}(G)$ in sublinear time to within an additive error term $\varepsilon \cdot \mathfrak{cost}(G)$ or $\varepsilon \cdot (\mathfrak{mst}(X) + \mathfrak{cost}(G))$, when we are given oracle access to the metric space (X, d) that defines G.

We present two sublinear-time algorithms. First, in Theorem 5.1, we show that with $\widetilde{O_\varepsilon}(nk^{3/2})$ queries one can approximate $\mathfrak{cost}(G)$ to within an additive error term $\varepsilon \cdot (\mathfrak{mst}(X) + \mathfrak{cost}(G))$. Then, in Theorem 6.1, we show that one can find a $(1+\varepsilon)$-approximation of $\mathfrak{cost}(G)$ with $\widetilde{O}_\varepsilon(n^2/k)$ queries.

Further, we complement these results with near matching lower bounds. In Theorem 1.1, we show that any algorithm that for any metric space (X, d) of size n, with probability at least $\frac{2}{3}$ estimates $\mathfrak{cost}(G)$ to within an ε factor requires $\Omega(n^2/k)$ time. Similarly, in Theorem 1.2, we show that any algorithm that with probability at least $\frac{2}{3}$ estimates $\mathfrak{cost}(G)$ to within an additive error term $\varepsilon \cdot (\mathfrak{mst}(X) + \mathfrak{cost}(X))$ requires $\Omega_\varepsilon(\min\{nk^{3/2}, n^2/k\})$ time.

References

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1): 117–122, 2008.

[2] M. Bădoiu, A. Czumaj, P. Indyk, and C. Sohler. Facility location in sublinear time. *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, pp. 866–877, 2005.

[3] B. Chazelle, R. Rubinfeld, and L. Trevisan. Estimating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing*, 34(6): 1370–1379, 2005.

[4] J. Costa and A. O. Hero III. Entropic graphs for manifold learning. *Proceedings of the 37th Asilomar Conference on Signals, Systems & Computers*, pp. 316–320, 2003.

[5] A. Czumaj, F. Ergün, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, and C. Sohler. Approximating the weight of the Euclidean minimum spanning tree in sublinear time. *SIAM Journal on Computing*, 35(1): 91–109, 2005.

[6] A. Czumaj and C. Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM Journal on Computing*, 39(9): 904–922, 2009.

[7] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. *Proceedings of the 20th Symposium on Computational Geometry (SoCG'04)*, pp. 253–262, 2004.

[8] W. Dong, M. Charikar, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. *Proceedings of the 20th International Conference on World Wide Web (WWW'11)*, pp. 577–586, 2011.

[9] W. Ehm. Binomial approximation to the Poisson binomial distribution. *Statistics & Probability Letters*, 11(1): 7–16, 1991.

[10] H. Esfandiari and M. Mitzenmacher. Metric sublinear algorithms via random sampling. *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS'18)*, pp. 11–22, 2018.

[11] U. Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35(4): 964–984, 2006.

[12] H. Fichtenberger and D. Rhode. A theory-based evaluation of nearest neighbor models put into practice. *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeuRIPS'18)*, pp. 6743–6754, 2018.

[13] D. Freedman. A remark on the difference between sampling with and without replacement. *Journal of the American Statistical Association*, 72(359):681, 1977.

[14] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. *Proceedings of the 25th International Conference on Very Large Databases (VLDB'99)*, pp. 518–529, 1999.

[15] O. Goldreich and D. Ron. Approximating average parameters of graphs. *Random Structures & Algorithms*, 32(4): 473–493, 2008.

[16] P. Indyk. On approximate nearest neighbors in non-Euclidean spaces. *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS'98)*, pp. 148–155, 1998.

[17] P. Indyk. Sublinear time algorithms for metric space problems. *Proceedings of the 31st Annual Symposium on Theory of Computing (STOC'99)*, pp. 428–434, 1999.

[18] P. Indyk. *High-Dimensional Computational Geometry.* Doctoral Dissertation, Stanford University, 2001.

[19] H. R. Künsch. The difference between the hypergeometric and the binomial distribution. Note, ETH Zürich, May 1998.

[20] R. Mettu and G. Plaxton. Optimal time bounds for approximate clustering. *Machine Learning*, 56(1–3): 35–60, 2004.

[21] B. Naidan, L. Boytsov, and E. Nyberg. Permutation search methods are efficient, yet faster search is possible. *Proceedings of the VLDB Endowment*, 8(12): 1618–1629, 2015.

[22] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Proceedings of the 14th Conference on Neural Information Processing Systems (NIPS'01)*, pp. 849–856, 2001.

[23] H. Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. *Proceedings of the 49th Annual Symposium on Foundations of Computer Science(FOCS'08)*, pp. 327–336, 2008.

[24] K. Onak, D. Ron, M. Rosen, and R. Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*, pp. 1123–1131, 2012.

[25] D. Pál, B. Póczos, and C. Szepesvári. Estimation of Rényi entropy and mutual information based on generalized nearest-neighbor graphs. *Proceedings of the 23rd Conference on Neural Information Processing Systems (NIPS'10)*, pp. 1849–1857, 2010.

[26] L. Paninski. A coincidence-based test for uniformity given very sparsely-sampled discrete data. *IEEE Transactions on Information Theory*, 54:4750–4755, 2008.

[27] B. W. Silverman. *Density Estimation for Statistics and Data Analysis.* Monographs on Statistics and Applied Probability 26, Chapman and Hall, 1986.

[28] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8): 888–905, 2000.

[29] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500): 2319–2323, 2000.

[30] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing* 17(4): 395–416, 2007.

[31] Y. Yoshida, M. Yamamoto, and H. Ito. An improved constant-time approximation algorithm for maximum matchings. *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC'09)*, pp. 225–234, 2009.