SUBLINEAR TIME APPROXIMATION OF THE COST OF A METRIC K-NEAREST NEIGHBOR GRAPH*

ARTUR CZUMAJ † AND CHRISTIAN SOHLER ‡

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.

Key words. sublinear algorithms, approximation algorithm, nearest neighbors

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, 10, 11, 17, 19] 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 [32], 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 [33]). 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, [25, 31] 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, e.e., [30]). 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 space.

*A preliminary version appeared in *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*, pages 2973–2992, Salt Lake City, UT, USA, January 5–8, 2020.

†Department of Computer Science and Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick (A.Czumaj@warwick.ac.uk). Research partially supported by the Centre for Discrete Mathematics and its Applications (DIMAP), by IBM Faculty Award, and by EPSRC award EP/N011163/1.

‡Department of Mathematics and Computer Science, University of Cologne (sohler@cs.uni-koeln.de). Work partially done while the author was visiting researcher at Google Research, Switzerland.

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 [7] 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) [28].

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. [11], 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 [11, 24], they do not come with guarantees. In order to evaluate the quality of the computed solution it is thus desirable 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 (e.g., 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 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 task 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.

THEOREM 1.1. *Algorithm* APPROXIMATENNLARGEK(n, ε, k) *computes with probability at least* $\frac{2}{3}$ *a value* $\overline{\mathtt{cost}}$ *with*

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

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

In particular, Theorem 1.1 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 (checking all $\binom{n}{2}$ distances). Even if this dependency is undesirable, it is easy to see that such dependency (or a similar one) is necessary at least in the basic when $k = 1$. 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 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 1.2) and in $\widetilde{O}\Big(\frac{nk^{3/2}}{\varepsilon^6}\Big)$ 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.

THEOREM 1.2. *Algorithm k-NNSIZEAPPROXIMATION(n, ε) in expected time $O\Big(\frac{nk^{3/2}\log^3 n \log^2 k}{\varepsilon^6}\Big)$ returns a value $\overline{\mathfrak{cost}}$ such that with probability at least $\frac{2}{3}$, we have*

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

The algorithms above can be combined to compute in expected $\widetilde{O}\Big(\min\Big\{\frac{nk^{3/2}}{\varepsilon^6}, \frac{n^2}{\varepsilon^2 k}\Big\}\Big)$ 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 $\widetilde{O}\Big(\frac{n^{8/5}}{\varepsilon^{18/5}}\Big)$.

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 (for $k > 1$) 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 1.3. *Let $\mathfrak{c}$ be any positive constant, $\mathfrak{c} \leq k$. Any algorithm that for a given metric space (X, d) of size n, with probability at least $\frac{2}{3}$ estimates the cost of a k-NN graph to within an additive error term $\mathfrak{c} \cdot \mathfrak{cost}(X)$ requires $\Omega\Big(\frac{n^2}{k}\Big)$ queries.*[1]

THEOREM 1.4. *Let $0 < \varepsilon \leq \frac{1}{2}$. Any algorithm that for a given metric space (X, d) of size n, with probability at least $\frac{5}{6}$ estimates the cost of a k-NN graph to within an additive error term $\varepsilon \cdot (\mathfrak{cost}(X) + \mathfrak{mst}(X))$ requires $\Omega\Big(\min\Big\{\frac{nk^{3/2}}{\varepsilon}, \frac{n^2}{k}\Big\}\Big)$ queries.*

1.2. Our techniques. We develop two algorithms, one of them is more suitable for large (Theorem 1.1) and the other for small (Theorem 1.2) 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] In fact, the claim holds even for superconstant $\mathfrak{c}$; all what we need is $\mathfrak{c} \leq k$.

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 (cf. Theorem 1.1) relies on a combination of two random sampling routines. It first takes $\widetilde{O}\big(\frac{n}{k}\big)$ 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 incident to all vertices of distance less than the length of the median edge.

Similar properties are also true for edges 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 $\widetilde{O}\big(\frac{n}{\varepsilon^2 k}\big)$ 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 $\widetilde{O}\big(\frac{n}{\varepsilon^2 k}\big)$ 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 3 for more details).

1.2.2. Algorithm for small k. Our algorithm from Theorem 1.2 that for small k in time $\widetilde{O}\Big(\frac{nk^{3/2}}{\varepsilon^6}\Big)$ estimates $\mathfrak{cost}(G)$ with an additive error term $\varepsilon\cdot(\mathfrak{mst}(X)+\mathfrak{cost}(X))$ is more complex and technically more involved, and 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 $(1+\varepsilon)$. We argue in Appendix A why this does not change the problem significantly (essentially, we are using existing sublinear-time algorithms to approximate the diameter [20] and the weight of the minimum spanning tree [9] 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 $d(u,v)\le(1+\varepsilon)^i$. We then show that we can use these "threshold" graphs to derive a concise formula for the cost of the k-NN graph. 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. [4, 9]). In our case, we have a surprisingly simple formula that only depends on vertex degrees. We will use the well-known identity $\sum_{i=0}^{\mathfrak{r}-1}(1+\varepsilon)^i=\frac{(1+\varepsilon)^{\mathfrak{r}}-1}{\varepsilon}$, where $\mathfrak{r}$ is the exponent of the maximum edge weight, 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 apply the identity above to obtain the following *central formula*:

$$\mathfrak{cost}(G)=nk+\varepsilon\cdot\sum_{i=0}^{\mathfrak{r}-1}\left((1+\varepsilon)^i\sum_{v\in V}(k-\deg^{(i)}(v))\right)\quad. \tag{1.1}$$

With (1.1) at hand, our problem reduces to the estimation of 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_j^i, $0\le j\le\mathfrak{t}$ with $\mathfrak{t}=1+\lfloor\log_{1+\varepsilon}k\rfloor$, such that V_0^i contains vertices with $\deg^{(i)}(v)=k$, and for $1\le j\le\mathfrak{t}$, all vertices in V_j^i satisfy $(1+\varepsilon)^{j-1}\le k-\deg^{(i)}(v)<(1+\varepsilon)^j$. This

leads to a modification of formula (1.1) stating that

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

Now the main challenge is to approximate the sizes of the sets V_j^i 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_j^i in this setting? We observe that since $V_0^i = V \setminus \{u\}$ for all $i < \mathfrak{r}$, there is exactly a single set V_j^i (with $j = \mathfrak{t}$) which is not empty and contains only a single point u. If we wanted to find u by random sampling, we would 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 $\widetilde{O}\Big(\frac{n}{\deg^{(i)}(x)}\Big)$ queries. We remark that a somewhat similar filtering strategy has been used for approximating the cost of a metric minimum spanning tree [9].

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 (and 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 [9])!

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 algorithm for small k. See Section 4.4.2 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 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 $\mathfrak{cost}(G) \sim 0$ 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 1.3). 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 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 $\Omega\left(\frac{n^2}{k}\right)$ 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 $\Omega\left(\frac{n^2}{k}\right)$ 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 $\Omega\left(\frac{n^2}{k}\right)$ time (cf. Theorem 1.4).

For smaller values of k, the arguments above (for approximation with the additive error term of $\varepsilon(\mathfrak{cost}(G) + \mathfrak{mst}(X))$) do not hold and so we revise our construction. In the case $k = O((\varepsilon n)^{2/5})$, we partition the input point set into $\Theta\left(\frac{\varepsilon n}{k^{5/2}}\right)$ clusters of size $k+1+\sqrt{k}$, the same number of clusters of size $k+1-\sqrt{k}$, and the remaining all but $\Theta\left(\frac{\varepsilon n}{k^{3/2}}\right)$ 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 $\Omega\left(\frac{nk^{3/2}}{\varepsilon}\right)$ 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 $\Omega\left(\frac{nk^{3/2}}{\varepsilon}\right)$ time (cf. Theorem 1.4).

The analysis above relies on a central result showing that $\Omega(nh)$ time is required 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 are presented in details in Sections 5–6.

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 [20]. The question of how to approximate the cost of a minimum spanning tree has been studied both in the metric [9] and non-metric [4] 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 $\widetilde{O}(n/\varepsilon^7)$ [9]. Also, the Euclidean minimum spanning tree problem has been studied in a setting where one can access the input via certain spatial data structure [8]. 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 $\widetilde{O}(nk)$ [23] as well as bi-criteria approximation algorithms [20]. The metric maximum cut problem can also be solved in sublinear time [13, 20]. A recent result on linear sampling from metric spaces can be used to improve a number of previous results [13] like, for example, reduce the query complexity of maxcut. The cost of metric TSP [5, 6], the size of a maximum matching [26, 35] and the vertex cover size [26, 27] can also be approximated using sublinear time approximation

algorithms. The average degree of a graph can be efficiently approximated as well [14, 18].

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 [15] 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.

2.1. Model of computation. We assume that we are given oracle access to a finite metric space (X, d) with $X = \{1, \dots, n\}$: Our algorithm receives n as an input. The algorithm can specify pairs of points $x, y \in X$ (i.e., two numbers from $\{1, \dots, n\}$) 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 assume that all distances in X are distinct; if this is not the case we use a lexicographical ordering of the edges as tiebreaker.

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. Observe that in general, this graph is *not* symmetric, i.e., if u is a k-nearest neighbor of v then not necessarily vertex v is a k-nearest neighbor of u. We 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 consider the problem of *approximating* the k-NN graph. For this purpose, we phrase the problem as an optimization problem. Given an input metric space, find a graph $G = (V, E)$ that minimizes

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

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 consider the question of approximating the *cost* of the k-NN graph in sublinear time.

3. $(1+\varepsilon)$-approximation of $\mathtt{cost}(G)$ with $\widetilde{O}_\varepsilon\left(\frac{n^2}{k}\right)$ queries.

In this section we develop a sampling algorithm ApproximatekNNLargek(n, ε, k) that with $\widetilde{O}_\varepsilon\left(\frac{n^2}{k}\right)$ queries returns a $(1+\varepsilon)$-approximation of the cost k-NN, $\mathtt{cost}(G)$.

Let $G = (V, E)$ be a k-NN graph. For any vertex $v \in V$, let us 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 vertex 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 $\frac{k}{2}$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, $E_S = \bigcup_{v\in V}\{[v,u\rangle \in E : d(v,u) \le 10d(v,u_v)\}$ 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 3.2) 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.

We begin with a formal description of our main algorithm to estimate $\mathtt{cost}(G)$.[2]

ApproximatekNNLargek(n,ε,k)
{Returns a $(1+\varepsilon)$-approximation of $\mathtt{cost}(G)$; cf. Lemma 1.1}
for each $v\in V$ **do**
Sample $\mathfrak{s} = \left\lceil \frac{1000(n-1)\log n}{k}\right\rceil$ 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 Sections 3.1–3.3 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 3.4, where we will conclude the analysis of the properties of algorithm ApproximatekNNLargek in the final Theorem 1.1.

3.1. 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 3.1. *Let v be a vertex in $H=(V,E)$ and let its neighbors $q_1,\dots,q_{n-1}$ be sorted by distance to v, i.e., $d(v,q_i)\le d(v,q_j)$ 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*

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

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

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, $\mathbf{E}[X_i] = \frac{\lfloor k/4\rfloor}{n-1}$. 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 $\sum_{i=1}^{\mathfrak{s}} X_i \ge \lceil 500\log n\rceil$. Notice the following,

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

Using the above inequality and then a Chernoff bound, we get that

$$\mathbf{Pr}[u_v\in\mathcal{S}] = \mathbf{Pr}\left[\sum_{i=1}^{\mathfrak{s}} X_i \ge \lceil 500\log n\rceil\right] \le \mathbf{Pr}\left[\sum_{i=1}^{\mathfrak{s}} X_i \ge \tfrac{4}{3}\cdot\mathbf{E}[\sum_{i=1}^{\mathfrak{s}} X_i]\right] \le e^{-\mathbf{E}[\sum_{i=1}^{\mathfrak{s}} X_i]/27}\ .$$

[2] 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

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

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 $\mathbf{E}[Y_i] = \frac{\lfloor 3k/4 \rfloor}{n-1} \geq \frac{3k-3}{4(n-1)}$. 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,

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

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

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

Finally, it follows from our lower bound on $\mathbf{E}[\sum_{i=1}^{\mathfrak{s}} Y_i]$ 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 $1 - \frac{1}{n^4}$. ∎

3.2. Approximating the sum of lengths 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 3.2. $\mathcal{S}_v \leq 10k \cdot d(v,u_v)$.

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 3.3}
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 3.3. *Algorithm* 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 $\mathbf{E}[\vartheta_i] = \sum_{v\in V} p_v \cdot \frac{\mathcal{S}_v}{p_v} = \sum_{v\in V} \mathcal{S}_v$ and hence $\mathbf{E}[Z_1] = \mathbf{E}[\frac{1}{\mathfrak{a}} \cdot \sum_{i=1}^{\mathfrak{a}} \vartheta_i] = \sum_{v\in V} \mathcal{S}_v$, and so Z_1 is an unbiased estimator. Next we observe that by Claim 3.2 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

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

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

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

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

3.3. Approximating the sum of lengths 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 3.4. *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 3.1 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 3.4 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

$$d(v,w_i) \leq \tfrac{10}{9} \cdot d(x,w_i) \ . \tag{3.1}$$

For a fixed $x \in \mathcal{C}_v$, consider all k-nearest neighbors $y_1, \ldots, y_k$ of x in G, and 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 hence, by triangle inequality, $d(v,w_i) \leq d(v,w_k) \leq d(v,y_i) \leq d(v,x) + d(x,y_i)$. Next, since $d(v,x) \leq d(v,u_v)$ and $10d(v,u_v) < d(v,w_i)$ for $i \geq \ell$, we get for $i \geq \ell$,

$$d(v,w_i) \leq d(v,x) + d(x,y_i) \leq d(v,u_v) + d(x,y_i) \leq \tfrac{1}{10} d(v,w_i) + d(x,y_i) \leq \tfrac{10}{9} d(x,y_i) \ . \tag{3.2}$$

If we combine (3.1) and (3.2), 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

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

$\mathcal{L}_v \leq \frac{10}{9}(\mathcal{S}_x + \mathcal{L}_x)$ holds for every x in $\mathcal{C}_v$. By Lemma 3.1, we know that $|\mathcal{C}_v| \geq k/4$, and therefore,

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

what yields the claim. ∎

With Claim 3.4 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 $\varepsilon \cdot \mathfrak{cost}(G)$; cf. Lemma 3.5}
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 3.5. *Algorithm* 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 $\mathbf{E}[\mathfrak{l}_i] = \frac{1}{n} \cdot \sum_{v \in V} \mathcal{L}_v$ and so $\mathbf{E}[Z_2] = \mathbf{E}[\frac{n}{\mathfrak{b}} \cdot \sum_{i=1}^{\mathfrak{b}} \mathfrak{l}_i] = \sum_{v \in V} \mathcal{L}_v$. Next, by Claim 3.4, we obtain the following,

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

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

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

Now we apply Chebyshev's inequality to obtain

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

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

3.4. Proof of Theorem 1.1 about the performance of ApproximateNNLargek. Now we are ready to complete the analysis of algorithm ApproximateNNLargek(n, ε, k) and prove that it performs $O\left(\frac{n^2 \log n}{\varepsilon^2 k}\right)$ queries to the distance oracle and computes with probability at least $\frac{2}{3}$ a value $\overline{\mathfrak{cost}}$ with $|\mathfrak{cost}(G) - \overline{\mathfrak{cost}}| \leq \varepsilon \cdot \mathfrak{cost}(G)$.

Proof of Theorem 1.1. We first note that by Lemma 3.1, with probability at least $1 - 1/n^3$ we have,

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

Thus, using ApproximateShortEdges and ApproximateLongEdges with parameters $\varepsilon/8$ and $\varepsilon/2$, respectively, by Lemmas 3.3 and 3.5, 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 3.3 and 3.5. ∎

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.

4. Bypassing $\widetilde{\Omega}_\varepsilon\left(\frac{n^2}{k}\right)$ query complexity by allowing error of $\varepsilon \cdot \mathfrak{mst}(X)$. In this section we develop another algorithm to approximate $\mathfrak{cost}(G)$ which improves upon the query complexity bound from Section 3 for small values of k. The improvement is obtained by allowing the additive error term in the estimation of $\mathfrak{cost}(G)$ to be $\varepsilon \cdot (\mathfrak{mst}(X) + \mathfrak{cost}(G))$, and so, the error term depends also on the cost of the minimum spanning tree of (X, d). Our new algorithm k-NNSizeApproximation(n, ε) performs $\widetilde{O}_\varepsilon(nk^{3/2})$ queries to approximate the cost of a k-NN graph G to within the additive error term of $\varepsilon \cdot (\mathfrak{mst}(X) + \mathfrak{cost}(G))$ (cf. Theorem 1.2). (Since the problem can be trivially solved with $\Theta(n^2)$ queries, in this section we assume that $k = o(n)$.)

4.1. A formula for the cost of a k-NN graph. We begin with deriving 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 $0 \leq j \leq \mathfrak{r}$ 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 [20] and the cost of the minimum spanning tree [9] 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 Appendix A. The time to approximate the diameter within a factor of 2 is $O(n)$ [20] and the time required to approximate the cost of the minimum spanning tree within a factor of 2 is $\widetilde{O}(n)$ [9]. We will assume that edges with the same weight are ordered lexicographically.

4.1.1. 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., $E^{(i)} = \{[u, v\rangle \in E : d(u, v) \leq (1+\varepsilon)^i\}$. We use $\deg^{(i)}(v)$ to denote the outdegree of vertex v in $G^{(i)}$.

Lemma 4.1. *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 $0 \leq j \leq \mathfrak{r}$, for some $\mathfrak{r} \in \mathbb{N}$. Let $\varepsilon > 0$. Then we can write*

$$\mathfrak{cost}(G) = nk + \varepsilon \cdot \sum_{i=0}^{\mathfrak{r}-1} \left((1+\varepsilon)^i \sum_{v \in V} \left(k - \deg^{(i)}(v) \right) \right) \ . \tag{4.1}$$

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

$$\sum_{i=0}^{\ell-1}(1+\varepsilon)^i = \frac{(1+\varepsilon)^\ell - 1}{\varepsilon} \tag{4.2}$$

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 (4.2), we can write

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

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

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

∎

4.2. Outline: applying Lemma 4.1 to approximate the cost of a k-NN graph. Our goal is to rely on the formula from Lemma 4.1 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, $0\le i\le\mathfrak{r}$, and let $\mathfrak{t}=1+\lfloor\log_{1+\varepsilon}k\rfloor$. In order to estimate $\sum_{v\in V}(k-\deg^{(i)}(v))$, 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 $1\le j\le\mathfrak{t}$,

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

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 $0\le i\le\mathfrak{r}$, $1\le j\le\mathfrak{t}$.

Lemma 4.2.

$$\mathfrak{cost}(G)\le nk+\varepsilon\sum_{0\le i\le\mathfrak{r},1\le j\le\mathfrak{t}}(1+\varepsilon)^{i+j}\cdot|V_j^i|\le(1+\varepsilon)\cdot\mathfrak{cost}(G)\ . \tag{4.3}$$

Proof. It immediately follows from Lemma 4.1 that

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

$$\leq nk + (1+\varepsilon)\cdot\varepsilon\cdot\sum_{i=0}^{\mathfrak{r}-1}(1+\varepsilon)^i\sum_{j=1}^{\mathfrak{t}}\sum_{v\in V_j^i}(k-\deg^{(i)}(v))$$

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

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

Hence, by the first inequality and the last identity, we have

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

what can be simplified to (4.3), concluding the claim. ∎

Thanks to Lemma 4.2, 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 (4.3) 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 4.3. *Let (X,d) be a metric space and assume 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 hold:*

- $E^{(i)} \subseteq F^{(i)}$,
- *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 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 denote by $\deg_{H^{(i)}}(v)$. Notice that $\deg^{(i)}(v) = \min\{\deg_{H^{(i)}}(v), k\}$.

4.3. The main sampling algorithm to estimate $\mathfrak{cost}(G)$. We now describe the main sampling algorithm k-NNSizeApproximation following the framework described by the inequalities from Lemma 4.2. 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 cost(G) to within ε · (mst(X) + cost(G); cf. Theorem 1.2}
  for i = 0 to r do
    for j = 1 to t do
      X_{i,j} = EstimateSetSize(n, ε, i, j)
  return nk + ε · Σ_{0≤i≤r, 1≤j≤t} (1+ε)^{i+j} X_{i,j}
```

EstimateSetSize(n, ε, i, j)
{Estimates $|V_j^i|$; cf. Lemma 4.8}
Sample set S of $s = \left\lceil \frac{100n(1+\varepsilon)^{i+j}\mathtt{rt}}{\mathtt{mst}(X)} \right\rceil$ 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 $O\left(\frac{n \log n}{k + \deg_{H^{(i)}}(v)}\right)$ queries, see Lemma 4.4. Thus, if a vertex has a high degree, $\deg_{H^{(i)}}(v)) \geq k$, we will dismiss it quickly, with $O\left(\frac{n \log n}{\deg_{H^{(i)}}(v)}\right)$ queries. We remark that a somewhat similar procedure has been the crux of the algorithm approximating the cost of the minimum spanning tree [9].

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 the estimate is within its guaranteed bounds. 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. This way, we may think of the sets V_j^i as being 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 4.3 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 [9], with only $\widetilde{O}(n)$ queries we can estimate $\mathtt{mst}(X)$ up to a constant, with probability at least $1 - 1/n^{10}$, and we will therefore pick a sample of size $\widetilde{O}\left(\frac{nk(1+\varepsilon)^{i+j}}{\mathtt{mst}(X)}\right)$. 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 $\varepsilon \cdot \mathtt{mst}(X)$ (more details follow in the analysis in Section 4.4). 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.

4.4. Approximating $\mathtt{cost}(G)$ by estimating the sizes of V_j^i. 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 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 4.2. 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 think of EstimateVertexDegree to return a correct estimate; the precise description of algorithm EstimateVertexDegree is given in Section 4.5. (We remark that we can replace $\mathtt{mst}(X)$ with its 2-approximation.)

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, if $(1+\varepsilon)^j > \sqrt{k}$, then we use a special sampling routine NeighborhoodSize(n, γ, k, v, i) to estimate $\deg_{H^{(i)}}(v)$ to within γk; cf. Section 4.5.1. We will first describe the filtering algorithm and then proceed to the analysis of our algorithm.

4.4.1. An auxiliary tool: 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. The idea to use geometrically increasing sample sizes to approximate the vertex degree ℓ with $O(n \log n/\ell)$ queries has been previously used in [9]. We modify their method for our purposes.

```
Filter(n, k, v, i)
    {Determines whether deg_{H^(i)}(v) = O(k); cf. Lemma 4.4}
  q = n − 1
  repeat
    a = 1000 ((n−1)/q) log n
    q = (2/3) q
    Sample v_1, ..., v_a independently and uniformly at random from V \ {v}
    Let q' be the number of vertices in v_1, ..., v_a that are neighbors of v in H^(i)
  until q' ≥ 1000 log n or q ≤ 2k
  if q ≤ 2k then return True
  else return False
```

Lemma 4.4. *Given access to a vertex v, algorithm* Filter(n, k, v, i) *asks in expectation $O\left(\frac{n \log n}{k+\deg_{H^{(i)}}(v)}\right)$ queries and with probability at least $1 - \frac{1}{n^{10}}$ returns the following value:*[3]

- 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

[3] As we will make it explicit later, when presenting algorithm EstimateVertexDegree (cf. Lemma 4.7), one could trivially obtain in expectation $O\left(\min\left\{n, \frac{n \log n}{k+\deg_{H^{(i)}}(v)}\right\}\right)$ queries by switching to computing $\deg_{H^{(i)}}(v)$ exactly when one performs more than n distance queries.

$\mathbf{Pr}[X_j] = \frac{\deg_{H^{(i)}}(v)}{n-1}$ and hence,

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

Chernoff's bound implies that

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

This implies that if $\deg_{H^{(i)}}(v) \geq 4k$, then algorithm FILTER returns FALSE in time $O\left(\frac{n \log n}{\deg_{H^{(i)}}(v)}\right)$ 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

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

We apply Chernoff bound (if $\lambda \geq 2\mathbf{E}[\sum_{j=1}^{\mathfrak{a}} X_j]$ 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 $O\left(\frac{n \log n}{k}\right)$ time. □

4.4.2. An auxiliary tool: The clustering connection and MST. In this section we connect a term used in the running time of parts of our algorithms with $\mathfrak{mst}(X)$.

LEMMA 4.5. *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)
    {Computes greedily a partition of X into clusters of radius σ and
        returns the number of clusters 𝔠; cf. Lemma 4.5}
  𝔠 = 0
  for each x ∈ X do
    𝔠 = 𝔠 + 1
    let C_𝔠 be the set of points (including x) with distance at most σ from x
    remove C_𝔠 from X
  return 𝔠
```

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,

$$\frac{1}{3}(\mathfrak{c}-1)\sigma \leq \mathfrak{mst}(X) \ . \tag{4.4}$$

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

$$\deg_{H^{(i)}}(v) \geq |C| - 1 \ . \tag{4.5}$$

Therefore 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 (4.4) to obtain the following,

$$(1+\varepsilon)^i \left(\sum_{v\in V} \frac{1}{\deg_{H^{(i)}}(v)+1} - 1 \right) = 4\sigma \left(\sum_{v\in V} \frac{1}{\deg_{H^{(i)}}(v)+1} - 1 \right) \leq 4\sigma(\mathfrak{c}-1) \leq 12 \cdot \mathfrak{mst}(X) \ . \tag{4.6}$$

With (4.6) at hand, we can conclude the first claim as follows:

$$\sum_{v\in V:\deg_{H^{(i)}}(v)\geq 1} \frac{1}{\deg_{H^{(i)}}(v)} \leq 2 \cdot \sum_{v\in V} \frac{1}{\deg_{H^{(i)}}(v)+1} \leq^{\text{by (4.6)}} 2 + \frac{24 \cdot \mathfrak{mst}(X)}{(1+\varepsilon)^i} \ .$$

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 (4.5), 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 (4.4) to simplify this bound as follows:

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

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 $\frac{120 \cdot k \cdot \mathfrak{mst}(X)}{(1+\varepsilon)^i}$. ∎

4.5. 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. page 15) to estimate the sizes of the sets V_j^i, which in turns is then used to analyze our main algorithm k-NNSizeApproximation(n,ε) that estimates $\mathfrak{cost}(G)$.

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

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 4.4 and Section 4.4.1) 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 4.5.1.

4.5.1. 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 4.6}
Let $\mathfrak{b} = \left\lceil \frac{500 n \ln n}{k\gamma^2} \right\rceil$
Sample $u_1,\dots,u_{\mathfrak{b}}$ independently and uniformly at random from $V \setminus \{v\}$
For every $1 \le \ell \le \mathfrak{b}$, let $X_\ell = 1$ if u_ℓ is a neighbor of v in $H^{(i)}$, and 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 4.6. *Let $\deg_{H^{(i)}}(v) \le 4k$. Given v as input,* NeighborhoodSize(n,γ,k,v,i) *in time $O\left(\frac{n \log n}{k\gamma^2}\right)$ 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 $\mathbf{E}[X_\ell] = \frac{\deg_{H^{(i)}}(v)}{n-1}$ for $1 \le \ell \le \mathfrak{b}$, and thus $\mathbf{E}[\mathfrak{X}] = \frac{\mathfrak{b}\cdot\deg_{H^{(i)}}(v)}{n-1} \ge \frac{500 \ln n \deg_{H^{(i)}}(v)}{k\gamma^2}$, $\mathbf{E}[\mathcal{Y}] = \deg_{H^{(i)}}(v)$.

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:

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

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

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

Furthermore,

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

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

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

EstimateVertexDegree(n,ε,k,v,i,j)
{Estimates $k - \deg^{(i)}(v)$; cf. Lemma 4.7}
if Filter(n,k,v,i) = False **then return** 0
else
if $(1+\varepsilon)^j \le \sqrt{k}$ **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 4.7. *Let $j \le \mathfrak{t}$.* ESTIMATEVERTEXDEGREE$(n, \varepsilon, k, v, i, j)$ *satisfies the following:*

- *If* $\deg_{H^{(i)}}(v) \ge 4k$ *then* ESTIMATEVERTEXDEGREE *runs in* $O\Big(\min\Big\{n, \frac{n \log n}{\deg_{H^{(i)}}(v)}\Big\}\Big)$ *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* $(1+\varepsilon)^j \le \sqrt{k}$, *or* $O\Big(\min\Big\{n, \frac{nk \log n}{\varepsilon^2 (1+\varepsilon)^{2j}}\Big\}\Big)$ *if* $(1+\varepsilon)^j > \sqrt{k}$, *and*
  - ◇ *either* $\deg_{H^{(i)}}(v) \ge k + \varepsilon \cdot (1+\varepsilon)^j$ *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* $|\widehat{d} - \deg_{H^{(i)}}(v)| \le \varepsilon \cdot (1+\varepsilon)^j$.

Proof. Consider two separate cases, depending on whether FILTER(n, k, v, i) returns FALSE or not.

By Lemma 4.4, 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, ESTIMATEVERTEXDEGREE$(n, \varepsilon, k, v, i, j)$ returns the correct value 0 of $k - \deg^{(i)}(v)$. Further, by Lemma 4.4, algorithm FILTER runs in expected time $O\Big(\frac{n \log n}{k + \deg_{H^{(i)}}(v)}\Big)$, 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 $O\Big(\min\Big\{n, \frac{n \log n}{\deg_{H^{(i)}}(v)}\Big\}\Big)$ (the $\min\{n, \star\}$ 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 4.4, 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 $O\Big(\min\Big\{n, \frac{n \log n}{k}\Big\}\Big)$, but ESTIMATEVERTEXDEGREE will still perform some more work.

If ε, j, and k satisfy $(1+\varepsilon)^j \le \sqrt{k}$, 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 $O\Big(\min\Big\{n, \frac{n \log n}{k}\Big\}\Big)$.

Otherwise, let us consider the case $(1+\varepsilon)^j \le \sqrt{k}$, 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 4.6, algorithm NEIGHBORHOODSIZE$(n, \varepsilon(1+\varepsilon)^j/k, k, v, i)$ in time $O\Big(\frac{nk \log n}{\varepsilon^2 (1+\varepsilon)^{2j}}\Big)$ returns a value $\widehat{d}$ that with probability at least $1 - \frac{2}{n^{10}}$ satisfies $|\widehat{d} - \deg_{H^{(i)}}(v)| \le \varepsilon(1+\varepsilon)^j$. Therefore, the expected running time of algorithm ESTIMATEVERTEXDEGREE$(n, \varepsilon, k, v, i, j)$ in this case is $O\Big(\min\Big\{n, \frac{n \log n}{k} + \frac{nk \log n}{\varepsilon^2 (1+\varepsilon)^{2j}}\Big\}\Big)$, which for $j \le \mathfrak{t}$ (and hence $\varepsilon(1+\varepsilon)^j = O(k)$) simplifies to $O\Big(\min\Big\{n, \frac{nk \log n}{\varepsilon^2 (1+\varepsilon)^{2j}}\Big\}\Big)$.

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

4.6. Analysis of the running time of algorithm ESTIMATESETSIZE. With Lemma 4.7 at hand, we are ready to analyze algorithm ESTIMATESETSIZE from p. 15.

LEMMA 4.8. *The expected runtime of* ESTIMATESETSIZE(n, ε, i, j) *is* $O\Big(\frac{nk^{3/2} \log^2 n \log k}{\varepsilon^4}\Big)$.

Proof. (Let us remind that we have assumed that $k = o(n)$, and hence Lemma 4.5 (which requires $k < \frac{n-1}{4}$) holds.) We first observe that the guarantee from Lemma 4.7 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 4.6.1 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 4.4). Let us partition V into two sets V_1 and V_2, with $V_1 = \{v \in V : \deg_{H^{(i)}}(v) > 4k\}$ and $V_2 = \{v \in V : \deg_{H^{(i)}}(v) \le 4k\}$. We will split our analysis into two separate cases, depending on whether $(1+\varepsilon)^j \le \sqrt{k}$ 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 single 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 4.5, 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 by EstimateSetSize, $s = \left\lceil \frac{100n(1+\varepsilon)^{i+j}\mathfrak{rt}}{\mathfrak{mst}(X)} \right\rceil$):

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

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

$$O\left(\frac{n(k+\log n)\sqrt{k} \log n \log k}{\varepsilon^2} \right) = O\left(\frac{nk^{3/2} \log^2 n \log k}{\varepsilon^2} \right) . \tag{4.7}$$

Case 2. Next, we consider the case when $(1+\varepsilon)^j > \sqrt{k}$. By Lemma 4.7, the expected running time to evaluate a single sample vertex by algorithm EstimateVertexDegree$(n, \varepsilon, k, v, i, j)$ satisfies the following:

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

Using Lemma 4.5, this bound can be simplified to

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

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

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

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

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

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

$$O\left(\frac{nk^{3/2} \log^2 n \log k}{\varepsilon^4} \right) . \tag{4.8}$$

We can combine the two cases in (4.7) and (4.8) to conclude the proof of Lemma 4.8. ∎

4.6.1. 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).

4.7. 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 4.9. *For every $0 \le i \le \mathfrak{r}$, $1 \le j \le \mathfrak{t}$,*

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

Proof. V_j^i is the set of vertices v with $(1+\varepsilon)^{j-1} \le k - \deg^{(i)}(v)$. 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 4.10. *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 $s = s(i,j) = \left\lceil \frac{100n(1+\varepsilon)^{i+j}\mathfrak{rt}}{\mathfrak{mst}(X)} \right\rceil$ 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 4.9 and the last inequality follows from the fact that $s = \left\lceil \frac{100n(1+\varepsilon)^{i+j}\mathfrak{rt}}{\mathfrak{mst}(X)} \right\rceil$. ∎

4.7.1. Proof of Theorem 1.2 about the performance of k-NNSizeApproximation. Now, we prove Theorem 1.2 and show that k-NNSizeApproximation(n,ε) in $O\left(\frac{nk^{3/2}\log^3 n \log^2 k}{\varepsilon^6}\right)$ expected time returns with probability $\frac{2}{3}$ a value $\overline{\mathfrak{cost}}$ with $|\mathfrak{cost}(G) - \overline{\mathfrak{cost}}| \le \varepsilon \cdot (\mathfrak{mst}(X) + \mathfrak{cost}(G))$.

Proof of Theorem 1.2. The running time of algorithm k-NNSizeApproximation(n,ε) follows immediately from Lemma 4.8 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 4.10. 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 4.7, this happens with probability at least $1 - 1/n^{10}$.

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

$$\mathbf{Pr}\left[\left|\sum_{i=0}^{\mathfrak{r}} \sum_{j=1}^{\mathfrak{t}} (1+\varepsilon)^{i+j} X_{i,j} - \mathbf{E}\left[\sum_{i=0}^{\mathfrak{r}} \sum_{j=1}^{\mathfrak{t}} (1+\varepsilon)^{i+j} X_{i,j}\right]\right| \ge \mathfrak{mst}(X) + \mathfrak{cost}(G))\right]$$

$$\leq \frac{\mathbf{Var}[\sum_{i=0}^{\mathfrak{r}}\sum_{j=1}^{\mathfrak{t}}\left((1+\varepsilon)^{i+j}X_{i,j}\right)]}{(\mathfrak{mst}(X)+\mathfrak{cost}(G))^2} = \frac{\sum_{i=0}^{\mathfrak{r}}\sum_{j=1}^{\mathfrak{t}}\left((1+\varepsilon)^{2(i+j)}\mathbf{Var}[X_{i,j}]\right)}{(\mathfrak{mst}(X)+\mathfrak{cost}(G))^2}$$

$$\leq^{\text{(by Lemma 4.10)}} \frac{\sum_{i=0}^{\mathfrak{r}}\sum_{j=1}^{\mathfrak{t}}\left((1+\varepsilon)^{2(i+j)}\cdot\left(\frac{\mathfrak{mst}(X)\cdot\mathfrak{cost}(G)}{100\cdot(1+\varepsilon)^{2(i+j)}\cdot\mathfrak{r}\cdot\mathfrak{t}}\right)\right)}{2\cdot\mathfrak{mst}(X)\cdot\mathfrak{cost}(G)}$$

$$= \frac{(\mathfrak{r}+1)\cdot\mathfrak{t}}{200\cdot\mathfrak{r}\cdot\mathfrak{t}} \leq \frac{1}{100} \ .$$

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 (4.3) to bound $\mathfrak{cost}(G) - \overline{\mathfrak{cost}}$. Let $\mathfrak{X} = \sum_{i=0}^{\mathfrak{r}}\sum_{j=1}^{\mathfrak{t}}(1+\varepsilon)^{i+j}X_{i,j}$ and notice that $\mathbf{E}[\mathfrak{X}] = \sum_{i=0}^{\mathfrak{r}}\sum_{j=1}^{\mathfrak{t}}(1+\varepsilon)^{i+j}\cdot|V_j^i|$ and that $\overline{\mathfrak{cost}} = nk + \varepsilon\cdot\mathfrak{X}$. By Lemma 4.2 we have,

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

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

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

what is equivalent to

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

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

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

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

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

5. Lower bound of $\Omega\left(\frac{n^2}{k}\right)$ for $(1+\varepsilon)$-approximation of $\mathfrak{cost}(X)$. In this section we prove Theorem 1.3: we show that any approximation algorithm for the cost of a k-NN graph that for any metric space (X,d) of size n, computes with probability at least $\frac{2}{3}$ an estimate $\overline{\mathfrak{cost}}(X)$ with $|\mathfrak{cost}(X) - \overline{\mathfrak{cost}}(X)| \leq \varepsilon\cdot\mathfrak{cost}(X)$ has complexity $\Omega\left(\frac{n^2}{k}\right)$.

We begin with an auxiliary framework of balanced clustered graphs in Sections 5.1 and 5.2, and then, in Section 5.3, we incorporate this framework (and in particular, Theorem 5.4) to prove the main result of this section, Theorem 1.3.

5.1. Revealing vertices in balanced clustered graphs. Let $G=(V,E)$, $V=\{1,\ldots,n\}$, be an undirected and *unweighted* graph with $n = m\cdot\ell$ vertices that is a collection of m disjoint cliques of size ℓ. Without loss of generality, we assume that the vertices $i\ell+1,\ldots,(i+1)\ell$ form a clique. Let A be the adjacency matrix of G. Let $\mathbb{S}_n$ be the permutation group of $\{1,\ldots,n\}$. For a given permutation $\pi\in\mathbb{S}_n$ we use G_π to denote the graph that is obtained by applying π to V. Also, let A_π be the corresponding matrix, i.e., $A_\pi[i,j] = A[\pi^{-1}(i),\pi^{-1}(j)]$.

Now consider an arbitrary algorithm Alg that has access to the adjacency matrix A of an input graph G. Alg learns the input graph by querying unordered pairs of vertices i,j, and each such a query returns 1 if i and j are adjacent in G, and returns 0 otherwise. We use $\square$ to denote the entries in A which have not been queried by the algorithm. We can now define a *query history*.

Definition 5.1. *An $n\times n$ symmetric matrix with entries from $\{0,1,\square\}$ is called **query history**.*

Definition 5.2. *At any point of time the **query history of algorithm Alg** is the query history H that satisfies,*

- *$H[i,j]=\square$, if Alg did not query entry $\{i,j\}$, and*

- $H[i,j] = x$ *for* $x \in \{0,1\}$*, if* Alg *queried entry* $\{i,j\}$ *and received answer* x*.*

For the analysis we allow Alg to make some queries for *free* and assume that the algorithm always makes all free queries. Queries that are not free are called *paid queries*. An entry from $\{0,1\}$ in the *query history is free*, if it was obtained by a free query and is *paid*, if is was obtained by a paid query.

Definition 5.3. *We say an algorithm* Alg ***reveals*** *a vertex* $i \in V$ *if and only if the* i*-th row of its query history* H *contains at least one entry* 1*.*

Now, we are ready to state our main technical result of this section.

Theorem 5.4. *Let* G *be an undirected graph on vertex set* $\{1,\dots,n\}$ *that is a collection of* $m \geq 1$ *disjoint cliques of size* $\ell \geq 1$ *each. Let* $\pi \in \mathbb{S}_n$ *be a uniformly random permutation of* $\{1,\dots,n\}$*,* G_π *be the graph that is obtained by applying* π *to the vertex set of* G*, and* A_π *be the adjacency matrix of* G_π*. Then any algorithm that queries* A_π *in at most* T *places reveals in expectation* $O\left(\frac{T\ell}{n}\right)$ *vertices.*

Before we proceed with the proof, let us first discuss the intuitions behind this theorem. Theorem 5.4 says that, informally and approximately, in order to reveal a vertex $i \in V$, any algorithm has to query (in expectation) at least $\Omega\left(\frac{n}{\ell}\right)$ neighbors of i. Intuitively, this is what we should expect: if a vertex does not know any of its neighbors in its clique, then Alg can only query for "random" neighbors of i, and since only $\ell - 1$ of them are in the same clique, we expect $\Omega\left(\frac{n}{\ell}\right)$ queries to detect the first neighbor from the same clique. A formal proof formalizing this claim is complex, and is presented in Section 5.2 below.

5.2. Proof of Theorem 5.4. In order to prove Theorem 5.4 we need further definitions.

Definition 5.5. *A* ***basic event*** *is any matrix* $B := A_\pi$ *obtained by applying to the adjacency matrix* A *an arbitrary permutation* π *to the vertices of* V*.*

Definition 5.6. *Let* H *be a query history and* B *be a basic event.* H ***and*** B ***are consistent****, if* B *and* H *agree on the entries of* H *that are not* $\square$*.*

We call a query history H ***consistent*** *if it is consistent with at least one basic event.*

In our analysis we have to consider explicit information revealed by the algorithm, as well as its implications that provide some implicit knowledge. For example, we have to carefully consider the case when for some vertices $i,j \in V$ we have $H[i,j] = 1$, since then we also know that for any vertex $r \in V$, if $H[i,r] = 1$ then also $A_\pi[j,r] = 1$, and if $H[i,r] = 0$ then also $A_\pi[j,r] = 0$. Similarly, we have to deal carefully with the scenarios when Alg has queried many neighbors of a given vertex: if almost all entries in row i are in $\{0,1\}$, then this can reveal some additional information about the $\square$-entries. To quantify these observations, we introduce the notions of fully explored vertices, saturated vertices, and explored vertices.

Definition 5.7. *We say an algorithm* Alg ***fully explores*** *a vertex* $i \in V$ *if and only if the* i*-th row of its query history contains only entries from* $\{0,1\}$*.*

Notice that if Alg fully explores a vertex $i \in V$ then we know the entire clique of i.

Definition 5.8. *Let* H *be a query history,* B *be a basic event that is consistent with* H*, and* $i \in V$ *be a vertex that is not fully explored. Let* $C(i)$ *be the set of vertices in the same clique of* G *as vertex* i *in* B*. Let* $N_B(C(i)) = \{j \in V : \exists_{i' \in C(i)} H[i',j] \neq \square\}$*. We call vertex* $i \in V$ ***saturated*** *if* $|N_B(C(i))| \geq \frac{n}{10}$*.*

While Definition 5.8 refers to cliques, it has also implications for individual vertices.

Claim 5.9. *Let* H *be a query history and let* $i \in V$*. If vertex* i *is not saturated for some basic event* B *that is consistent with* H*, then* $|\{j \in V : H[i,j] \neq \square\}| < \frac{n}{10}$*.*

The notion of saturated vertices depends on both, on the query history H and on a given basic event B that is consistent with H. Our next notion of explored vertices relies only on H.

Definition 5.10. *We call a vertex* $i \in V$ ***explored****, if conditioned on a consistent query history, vertex* i *is saturated with probability at least* $\frac{1}{4}$ *(the probability is over all basic events consistent with the query history).*

In order to focus our analysis on "nice" query histories, and with this, to cope with implicit information available to the algorithm Alg, we will give Alg additional power: the oracle *shows for free* all entries of explored vertices. If a vertex i is explored then we make it at once fully explored. Notice that then all vertices from its clique $C(i)$ become explored and then fully explored. Therefore, once a vertex i becomes

explored we perform the operation of *uncovering* the entire clique $C(i)$ at once, making all vertices from $C(i)$ fully explored. Notice that uncovering gives away $O(\ell n)$ *free queries*. In fact, our construction ensures that all free queries are caused by the uncovering operation — all other queries in ALG are paid.

If multiple vertices become explored at the same point of time, then we break ties by uncovering for free the entire clique $C(i)$ of the smallest explored vertex i (we assume $V = \{1, \ldots, n\}$). This process is repeated until no explored vertices are left. Observe that this operation transforms every explored vertex and its clique into fully explored vertices.

Now we describe a class of query histories of our interest.

DEFINITION 5.11. *A query history H is **nice** iff H is consistent with at least one basic event and there are no explored vertices.*

From now on, we will focus only on nice query histories and on algorithms that always perform free queries. Let us list some straightforward properties of fully explored vertices.

CLAIM 5.12. *Let* ALG *be an algorithm running on A_π. Then the following properties hold:*
(a) If ALG *fully explores $i \in V$ and $A_\pi[i,j] = 1$ for some $j \in V$, then* ALG *fully explores j.*
(b) If ALG *fully explored r vertices then these vertices are coming from $\frac{r}{\ell}$ cliques in A_π.*
(c) If ALG *fully explored $s\ell$ vertices then* ALG *performed at most $s\ell n$ free queries.*

5.2.1. Properties of deterministic algorithms. We begin with our first technical lemma that describes properties of a query about two vertices i, j with $H[i,j] = \square$. For a nice query history H we write $\mathbf{Pr}[A_\pi[i,j] = 1 \mid H]$ to denote the probability of $A_\pi[i,j] = 1$ conditioned on A_π agreeing with the query history H.

LEMMA 5.13. *Let H be a nice query history and sssume that the number of* revealed *vertices is at most $\frac{1}{2}n$. Let $i \in V$ and let $j \in V$ be a not revealed vertex with $H[i,j] = \square$. Then,*
- $\mathbf{Pr}[$*vertex i or j is saturated* $\mid A_\pi[i,j] = 1, H] \geq \frac{1}{2}$*, or*
- $\mathbf{Pr}[A_\pi[i,j] = 1 \mid H] \leq \frac{100\ell}{n}$.

Proof. If $\ell \geq \frac{n}{100}$ the statement of Lemma 5.13 is trivial. Therefore, let us assume that $\ell < \frac{n}{100}$.

Let $\mathcal{B}$ be the set of basic events that are consistent with our query history H. Observe that we can write the event "$A_\pi[i,j] = 1$" conditioned on H as the union of basic events $B \in \mathcal{B}$ with $B[i,j] = 1$. Call the latter set $\mathcal{B}^{(1)}$ and let $\mathcal{B}^{(0)} = \mathcal{B} \setminus \mathcal{B}^{(1)}$. Notice that $\mathbf{Pr}[A_\pi[i,j] = 1 \mid H] = \frac{|\mathcal{B}^{(1)}|}{|\mathcal{B}|}$.

Let $\mathcal{B}^{(1)}_S$ be the subset of $\mathcal{B}^{(1)}$ such that i or j is saturated. Observe that

$$\mathbf{Pr}[\text{vertex } i \text{ or } j \text{ is saturated} \mid A_\pi[i,j] = 1, H] = \frac{|\mathcal{B}^{(1)}_S|}{|\mathcal{B}^{(1)}|} \ .$$

Therefore, if $|\mathcal{B}^{(1)}_S| \geq \frac{1}{2}|\mathcal{B}^{(1)}|$ then $\mathbf{Pr}[$vertex i or j is saturated $\mid A_\pi[i,j] = 1, H] \geq \frac{1}{2}$ and we are done. Hence, we will assume that $|\mathcal{B}^{(1)} \setminus \mathcal{B}^{(1)}_S| > \frac{1}{2}|\mathcal{B}^{(1)}|$ and our goal is to prove that in that case $\mathbf{Pr}[A_\pi[i,j] = 1 \mid H] = \frac{|\mathcal{B}^{(1)}|}{|\mathcal{B}|} \leq \frac{100\ell}{n}$, which we do by showing that $\frac{|\mathcal{B}^{(1)} \setminus \mathcal{B}^{(1)}_S|}{|\mathcal{B}|} \leq \frac{50\ell}{n}$.

For a given basic event $B \in \mathcal{B}^{(1)} \setminus \mathcal{B}^{(1)}_S$ we will want to *swap* vertex j with some other vertex z, and we consider the case when such a swap gives a basic event[4] $B_z \in \mathcal{B}^{(0)}$ (and hence, that is consistent with H and has $B_z[i,j] = 0$). We have the following auxiliary claim.

CLAIM 5.14. *Let $\ell < \frac{n}{100}$. Let B be a basic event from $\mathcal{B}^{(1)} \setminus \mathcal{B}^{(1)}_S$. There are at least $\frac{29n}{100}$ vertices $z \in V$ such that the basic event B_z is consistent with H and has $B_z[i,j] = 0$.*

Proof. In order to ensure that the basic event B_z is consistent with H with $B_z[i,j] = 0$, we must be able to swap vertices j and z without violating entries that are determined in H, so that in B_z vertex j belongs to the clique of z in B (call it $C(z)$), in B_z vertex z belongs to the clique of j in B (call it $C(j)$), and the clique of j is distinct from the clique of z. For that, we will prove that for a given basic event $B \in \mathcal{B}^{(1)} \setminus \mathcal{B}^{(1)}_S$,

[4] That is, for any $x, y \in V$, we have $B_z[x,y] = \begin{cases} B[x,y] & \text{if } x, y \in V \setminus \{j, z\}, \\ B[x,z] & \text{if } y = j, \\ B[x,j] & \text{if } y = z. \end{cases}$

if we choose vertex z uniformly at random from V, then with constant probability j and z are in different cliques and we can swap j and z to obtain a basic event B_z that is consistent with H.

We will prove this property by bounding the success probability for five operations for B_z to be consistent with H: that j and z are in different cliques in B, that j can be removed from its clique $C(j)$ in B, that z can be removed from its clique $C(z)$ in B, that j can join $C(z)$, and that z can join $C(j)$.

- The probability that a random vertex $z \in V$ is in the same clique as j in B is $\frac{\ell}{n} < \frac{1}{100}$.
- Since j is not revealed, j is not "committed" in H to any clique (i.e., $H[j,x] \in \{0, \square\}$ for every $x \in V$) and thus j can be removed from $C(j)$ in B without violating H.
- Since there are at most $\frac{1}{2}n$ revealed vertices, vertex z is not revealed with probability at least $\frac{1}{2}$, in which case z can be removed from its clique in B without violating H.
- Since j is not saturated, for all except at most $\frac{n}{10}$ vertices $x \in V$, we have $H[y,x] = \square$ for all vertices $y \in C(j)$. Hence, $H[j,x] \neq \square$ for at most $\frac{n}{10}$ vertices $x \in V$, and so with probability at most $\frac{1}{10}$ we cannot connect j to the clique of a random $z \in V$.
- Similarly, since j is not saturated, for all except at most $\frac{n}{10}$ vertices $x \in V$, we have $H[y,x] = \square$ for all vertices $y \in C(j)$. Hence, for all except at most $\frac{n}{10}$ vertices z, we have $H[x,z] = \square$ for all vertices $x \in C(j)$, and so with probability at most $\frac{1}{10}$ we cannot connect a random vertex $z \in V$ to the clique of j without violating H.

Combining all cases above, we obtain that with probability at least $\frac{29}{100}$, we can swap j with a random vertex $z \in V$ without violating H, as required. ◻

Claim 5.14 ensures that for every $B \in \mathcal{B}^{(1)} \setminus \mathcal{B}^{(1)}_S$ there are at least $\frac{29}{100}n$ basic events $B_z \in B^{(0)}$. Next, we observe that for any fixed $z \in V$, the events B_z obtained from the different $B \in \mathcal{B}^{(1)} \setminus B^{(1)}_S$ are distinct (because B_z is obtained from $B \in \mathcal{B}^{(1)} \setminus B^{(1)}_S$ by swapping vertices j and z). Therefore, if we choose a random vertex $z \in V$, then with a positive constant probability, for a constant fraction of basic events $B \in \mathcal{B}^{(1)} \setminus B^{(1)}_S$ we will have basic events $B_z \in B^{(0)}$ with $B_z[i,z] = 1$, and all these basic events will be distinct. In particular, there must be $\frac{1}{5}n$ vertices $z \in V$ such that

$$\mathbf{Pr}[A_\pi[i,j] = 1 \mid H] \leq 20 \cdot \mathbf{Pr}[A_\pi[i,z] = 1 \mid H] \ . \tag{5.1}$$

Since the ith row of any basic event B has at most ℓ entries 1, we must have that $\sum_{z \in V} \sum_{B \in \mathcal{B}} \mathbb{1}_{B[i,z]=1} \leq |\mathcal{B}| \cdot \ell$. Therefore, for vertex $z^* \in V$ with the $\frac{1}{5}n$-th largest value of the sum $\sum_{B \in \mathcal{B}} \mathbb{1}_{B[i,z^*]=1}$, we have $\sum_{B \in \mathcal{B}} \mathbb{1}_{B[i,z^*]=1} \leq \frac{5 \cdot |\mathcal{B}| \cdot \ell}{n}$. This bound combined with (5.1) yields

$$\mathbf{Pr}[A_\pi[i,j] = 1 \mid H] \leq 20 \cdot \mathbf{Pr}[A_\pi[i,z^*] = 1 \mid H] = 20 \cdot \frac{\sum_{B \in \mathcal{B}} \mathbb{1}_{B[i,z^*]=1}}{|\mathcal{B}|} \leq \frac{100 \cdot \ell}{n} \ . \qquad ◻$$

5.2.2. Completing the proof of Theorem 5.4 (bounding revealed vertices). We are now ready to prove Theorem 5.4. Our analysis relies on Yao's principle [34], which implies that in order to prove a lower for the running time of a randomized algorithm it suffices to show a lower bound for the average running time of a deterministic algorithms over an adversarially chosen distribution of inputs. We will show that no deterministic algorithm with at most T queries reveals $\omega(\frac{T\ell}{n})$ vertices in expectation. This will imply the result also for randomized algorithms.

In what follows, we consider deterministic algorithms for a random input instance of a graph $G = (V, E)$, $V = \{1, \ldots, n\}$, with $n = m \cdot \ell$ vertices that form $m \geq 1$ disjoint cliques of size $\ell \geq 1$. The input's randomness comes from the random choice of the permutation of the vertices.

Consider an arbitrary deterministic algorithm Alg that makes at most T queries and suppose that $T \leq \frac{n^2}{6400\,\ell}$. We assume that the algorithm always makes all free queries, and hence we restrict ourselves to nice query histories (Definition 5.11). We begin with an auxiliary lemma that bounds the number of fully explored vertices by any deterministic algorithm.

Lemma 5.15. *Let $\ell \leq \frac{n}{4500}$. Consider a deterministic algorithm* Alg *that asks at most T paid queries with $T \leq \frac{n^2}{6400\ell}$. Then* Alg *fully explores at most $\frac{320\ell T}{n}$ vertices in expectation.*

Proof. Algorithm Alg is querying about the input graph and is gradually exploring some vertices. Every time a vertex $i \in V$ becomes explored by Alg, we uncover its entire clique, making all vertices from the

clique first explored and then fully explored. This will make ℓ new fully explored vertices and give away up to ℓn free queries (cf. Claim 5.12).

We split the run of ALG into two phases: *first phase* lasts until at most $\frac{n}{20}$ vertices become fully explored by ALG, after which there is the *second phase*.

Let us focus on the first phase and consider the case when some vertex $i \in V$ becomes explored as the first vertex from its clique. Since i becomes explored, in this moment, with probability at least $\frac{1}{4}$ vertex i is saturated. We claim that if i is saturated, then at least $\frac{n}{20}$ paid queries have been asked about the neighbors in the clique $C(i)$ of i (queries of the form $\{x, y\}$ with $x \in C(i)$). Indeed, since we are in the first phase, there are at most $\frac{n}{20}$ fully explored vertices and since vertex i is saturated, $C(i)$ has at least $\frac{n}{10}$ distinct neighbors ($|N_B(C(i))| \geq \frac{n}{10}$), out of which at most $\frac{n}{20}$ might have been obtained via free queries. Therefore, with probability at least $\frac{1}{4}$, at least $\frac{n}{20}$ paid queries were made to vertices in $C(i)$ before we uncovered the clique $C(i)$.

Notice that the arguments above imply that Lemma 5.15 is trivial for $T < \frac{n}{20}$: if ALG asks less than $\frac{n}{20}$ paid queries then no vertex can become explored. Hence, we assume that $T \geq \frac{n}{20}$.

Every time algorithm ALG explores a vertex as the first vertex in its clique, ALG uncovers the entire clique and with probability at least $\frac{1}{4}$, ALG asks at least $\frac{n}{20}$ paid queries to its vertices (and a single query is about two vertices). Therefore, in the first phase, the expected number of uncovered cliques is at most $\frac{2T}{\frac{1}{4}\cdot\frac{n}{20}} = \frac{160T}{n}$ (the factor 2 is because the T paid queries are about T *pairs* of vertices), and so the expected number of fully explored vertices in the first phase is at most $\frac{160\ell T}{n}$.

Next, let us consider the second phase. Notice that in the second phase, ALG can fully explore as many as all n vertices (minus $\frac{n}{20}$ vertices fully explored in the first phase).

CLAIM 5.16. *Let $\ell \leq \frac{n}{4500}$, $\frac{n}{20} \leq T \leq \frac{n^2}{6400\ell}$. The second phase happens with probability at most $\frac{160\ell T}{n^2}$, that is, the probability that* ALG *fully explores more than $\frac{n}{20}$ vertices is at most $\frac{160\ell T}{n^2}$.*

Proof. By our arguments above, in the first phase, every uncovering of a clique is caused by a vertex that is saturated with probability at least $\frac{1}{4}$, and every saturated vertex can be charged to at least $\frac{n}{20}$ paid queries. Let U_i be the number of paid queries charged to the vertices in the ith uncovered clique. (Since a single query is about two vertices, every query is charged half to each of the two vertices in the query.) Notice that if a vertex from the ith uncovered clique is saturated, we must have $U_i \geq \frac{n}{40}$. Therefore, since that vertex is saturated with probability at least $\frac{1}{4}$, the random variables $U_1, U_2, \ldots$ are stochastically dominating (that is, $\mathbf{Pr}[U_i \geq x] \geq \mathbf{Pr}[S_i \geq x]$ for every real x) a sequence of independent random variables $S_1, S_2, \ldots$ such that

$$\mathbf{Pr}\left[S_i = \frac{n}{40}\right] = \frac{1}{4} \text{ and } \mathbf{Pr}\left[S_i = 0\right] = \frac{3}{4} \ .$$

Since ALG performs at most T paid queries, if ALG uncovered $\mathfrak{q}$ cliques and $\mathfrak{q} \leq \frac{n}{20\ell}$ (hence, ALG is still in the first phase), then we must have $U_1 + \ldots U_{\mathfrak{q}} \leq T$. Therefore,

$$\mathbf{Pr}[\text{ALG uncovered } \mathfrak{q} \text{ cliques}] \leq \mathbf{Pr}[U_1 + \cdots + U_{\mathfrak{q}} \leq T] \leq \mathbf{Pr}[S_1 + \cdots + S_{\mathfrak{q}} \leq T] \ .$$

Let $\mathfrak{q}^* = \frac{n}{20\ell}$. Observe that since $T \leq \frac{n^2}{6400\ell}$, $\mathbf{E}[U_1 + \cdots + U_{\mathfrak{q}^*}] \geq \mathbf{E}[S_1 + \cdots + S_{\mathfrak{q}^*}] = \frac{\mathfrak{q}^* n}{160} \geq 2T$, and thus the probability that ALG will enter the second phase should be low. To formalize these intuitions, let $X_i = \frac{40}{n} S_i$ for every i, $1 \leq i \leq \mathfrak{q}^*$. Notice that X_i is a 0–1 random variable with $\mathbf{E}[X_i] = \frac{1}{4}$. Let $\mathfrak{X} = \sum_{i=1}^{\mathfrak{q}^*} X_i$ and observe that $\mathbf{E}[\mathfrak{X}] = \frac{1}{4}\mathfrak{q}^* = \frac{n}{80\ell}$.

We use the following Chernoff bound for the sum of $\mathfrak{q}^*$ 0–1 independent random variables,

$$\mathbf{Pr}\left[\mathfrak{X} \leq \frac{1}{2} \cdot \mathbf{E}[\mathfrak{X}]\right] \leq e^{-\frac{1}{8}\mathbf{E}[\mathfrak{X}]} \ .$$

Our goal is to prove that $e^{-\frac{1}{8}\mathbf{E}[\mathfrak{X}]} \leq \frac{160\ell T}{n^2}$, which will yield the claim. Let $\alpha = \frac{n}{640\ell}$ and let us analyze function $\psi(\alpha) = \frac{Te^\alpha}{4\alpha n}$, to show that with our parameters we have $\psi(\alpha) \geq 1$ (this is equivalent to $e^{-\alpha} \leq \frac{T}{4\alpha n}$, and hence, $e^{-\frac{1}{8}\mathbf{E}[\mathfrak{X}]} \leq \frac{160\ell T}{n^2}$). Notice that since $\psi'(\alpha) = \frac{e^\alpha(\alpha-1)T}{4\alpha^2 n} > 0$ for $\alpha > 1$, $\psi(\alpha)$ is increasing for $\alpha > 1$. Further, observe that for $T \geq \frac{n}{20}$, we have $\psi(7) \geq 1$, and hence if $T \geq \frac{n}{20}$ then $\frac{Te^\alpha}{4\alpha n} \geq 1$ for every

$\alpha \geq 7$. Therefore, in particular, if $T \geq \frac{n}{20}$ and $\ell \leq \frac{n}{4500}$ then $e^{-\frac{n}{640\ell}} \leq \frac{160\ell T}{n^2}$. This immediately yields the following final claim.

$$\begin{aligned}\mathbf{Pr}[\textsc{Alg} \text{ uncovered } \mathfrak{q}^* \text{ cliques}] &\leq \mathbf{Pr}\left[\sum_{i=1}^{\mathfrak{q}^*} S_i \leq T\right] \leq \mathbf{Pr}\left[\sum_{i=1}^{\mathfrak{q}^*} S_i \leq \frac{n^2}{6400\ell}\right] = \mathbf{Pr}\left[\mathfrak{X} \leq \frac{n}{160\ell}\right] \\ &= \mathbf{Pr}\left[\mathfrak{X} \leq \frac{1}{2}\mathbf{E}[\mathfrak{X}]\right] \leq e^{-\frac{1}{8}\mathbf{E}[\mathfrak{X}]} = e^{-\frac{n}{640\ell}} \leq \frac{160\ell T}{n^2} \ . \end{aligned}$$

□

With Claim 5.16 at hand, we can conclude that the expected number of fully explored vertices by Alg is at most $\frac{160\ell T}{n}$ in the first phase plus $n \cdot \frac{160\ell T}{n^2}$ in the second phase, and hence the expected number of fully explored vertices by Alg is at most $\frac{320\ell T}{n}$. □

Now we are ready to bound the number of vertices revealed by any deterministic algorithm.

Lemma 5.17. *Let $\ell \leq \frac{n}{4500}$. Let Alg be an arbitrary deterministic algorithm that makes at most T paid queries with $T \leq \frac{n^2}{6400\ell}$. Then, in expectation, Alg reveals $O\left(\frac{T\ell}{n}\right)$ vertices.*

Proof. Consider a deterministic algorithm Alg that makes at most T paid queries. Algorithm Alg queries the input graph and is gradually revealing and exploring some vertices. We split the runtime of algorithm Alg into two phases: *first phase* is until at most $\frac{1}{2}n$ vertices are revealed and until at most $\frac{n}{20}$ vertices become fully explored by Alg; the *second phase* starts after the first phase finishes. We will upper bound the expected number of vertices revealed in the first phase, and then we will show that the second phase is unlikely to happen.

Let us consider the first phase. We will analyze the process of revealing new vertices by Alg, first for paid queries and then for free queries.

Let us fix the current nice query history H and consider any *paid* query $\{i, j\}$; clearly, we can assume without loss of generality that $H[i, j] = \square$. Furthermore, if the query returns 0 then there are no changes in the set of revealed vertices and so we will only consider the case when $A_\pi[i, j] = 1$, combined with the bound for the probability that this happens.

Firstly, if both i and j are already revealed in H, then no new vertex is revealed by the query. Therefore we only have to consider the case when at least one of vertices i or j is not revealed; without loss of generality, suppose that j is not revealed.

Since in the first phase at most $\frac{1}{2}n$ vertices are revealed in H, all pre-conditions of Lemma 5.13 are satisfied, and so $\mathbf{Pr}[\text{vertex } i \text{ or } j \text{ is saturated } \mid A_\pi[i, j] = 1, H] \geq \frac{1}{2}$ or $\mathbf{Pr}[A_\pi[i, j] = 1 \mid H] \leq \frac{100\ell}{n}$.

If $\mathbf{Pr}[\text{vertex } i \text{ or } j \text{ is saturated } \mid A_\pi[i, j] = 1, H] \geq \frac{1}{2}$ then in this case one of vertices i or j are explored by Alg if $A_\pi[i, j] = 1$; in that case all vertices in the clique of i and j will be uncovered. Since Alg performs at most T paid queries, by Lemma 5.15 this case can happen in expectation not more than $\frac{320T}{n}$ times. Hence, in expectation, the number of vertices revealed by saturating one of vertices i or j is at most $\frac{320\ell T}{n}$, across the entire Alg.

Otherwise, we know that $\mathbf{Pr}[A_\pi[i, j] = 1 \mid H] \leq \frac{100\ell}{n}$, and so vertices i and j will be revealed with probability at most $\frac{100\ell}{n}$. Therefore with at most T such queries, in expectation we reveal at most $2T \cdot \frac{100\ell}{n} = \frac{200\ell T}{n}$ such vertices.

Next, let us consider free queries. Notice that all free queries will return 1 only for the fully explored vertices (cf. Claim 5.12). And since Alg performs at most T paid queries, by Lemma 5.15, in expectation, the number of fully explored vertices is at most $\frac{320\ell T}{n}$. Therefore, in expectation, the number of vertices revealed by free queries is at most $\frac{320\ell T}{n}$.

In summary, if we consider Alg until it fully explores at most $\frac{n}{20}$ vertices and reveals at most $\frac{1}{2}n$ vertices, in expectation, the number of revealed vertices is at most $\frac{840\ell T}{n}$.

Next, let us consider the second phase. Notice that in the second phase, Alg can reveal as many as all n vertices (minus $\frac{1}{2}n$ vertices revealed in the first phase). We bound the probability that the second phase will happen (that is, that either more than $\frac{1}{2}n$ vertices are revealed by Alg or that more than $\frac{n}{20}$ vertices are fully explored by Alg).

Claim 5.18. *Let $\ell \leq \frac{n}{4500}$ and $\frac{n}{20} \leq T \leq \frac{n^2}{6400\ell}$. The second phase happens with probability at most $\frac{161\ell T}{n^2}$, that is, the probability that Alg reveals more than $\frac{1}{2}n$ vertices or that Alg fully explores more than*

$\frac{n}{20}$ *vertices is at most* $\frac{161\ell T}{n^2}$.

Proof. First let us recall that by Claim 5.16, the probability that ALG fully explores more than $\frac{n}{20}$ vertices is at most $\frac{160\ell T}{n^2}$. Therefore from now on, we will condition on that at most $\frac{n}{20}$ vertices have been fully explored by ALG and we will bound the probability that ALG reveals more than half of the vertices with at most T paid queries.

By our analysis above, some vertices will be revealed by saturating vertices or in the process of uncovering some clique, and some other vertices will be revealed by paid queries $\{i,j\}$ with neither i nor j saturated. The former will take place at most $\frac{n}{20}$ times, since at most $\frac{n}{20}$ vertices have been fully explored by ALG. Therefore, we only have to consider the case of revealing a new vertex j via a query $\{i,j\}$.

We will mimic the analysis from the proof of Claim 5.16. By Lemma 5.13, if we condition on the fact that until now at most $\frac{1}{2}n$ vertices have been revealed and that at most $\frac{n}{20}$ vertices have been fully explored, the probability that a new vertex will be revealed is at most $\frac{100\ell}{n}$. Therefore, the probability that more than $\frac{1}{2}n$ vertices will be revealed (conditioned on that at most $\frac{n}{20}$ vertices are fully explored) is stochastically majorized by the probability that the sum of identically distributed independent 0–1 random variables $X_1,\ldots,X_T$ with $\mathbf{E}[X_i]=\frac{100\ell}{n}$ satisfies $\sum_{i=1}^{T} X_i > \frac{1}{2}n$.

Let $\mathfrak{X}=\sum_{i=1}^{T} X_i$. By Chernoff bound, for any $\kappa \geq 6\cdot\mathbf{E}[\mathfrak{X}]$, we have $\mathbf{Pr}[\mathfrak{X}\geq\kappa]\leq 2^{-\kappa}$. Notice that since $T\leq\frac{n^2}{6400\ell}$, in our case $\mathbf{E}[\mathfrak{X}]=\frac{100\ell T}{n}\leq\frac{n}{64}$, and hence $\frac{1}{2}n\geq 6\cdot\mathbf{E}[\mathfrak{X}]$. Therefore, using the fact that $n^2\leq 2^{n/2}$ for $n\geq 16$, we obtain that for $n\geq 16$:

$$\mathbf{Pr}\left[\mathfrak{X}\geq\frac{1}{2}n\right]\leq 2^{-n/2}\leq n^{-2}\leq\frac{\ell T}{n^2}\ .$$

This immediately imply that the probability that more than $\frac{1}{2}n$ vertices will be revealed (conditioned on that at most $\frac{n}{20}$ vertices have been fully explored) is upper bounded by $\mathbf{Pr}[\mathfrak{X}\geq\frac{1}{2}n]\leq\frac{\ell T}{n^2}$.

Therefore, we have proven that the probability that ALG fully explores more than $\frac{n}{20}$ vertices is at most $\frac{160\ell T}{n^2}$, and conditioned on that, the probability that more than $\frac{1}{2}n$ vertices will be revealed is at most $\frac{\ell T}{n^2}$. This implied that the probability that the second phase happens is at most $\frac{161\ell T}{n^2}$. ∎

Now we are ready to complete the proof of Lemma 5.17. By Claim 5.18, the expected number of vertices revealed by ALG is at most $\frac{840\ell T}{n}$ in the first phase plus $n\cdot\frac{161\ell T}{n^2}$ in the second phase, and hence the expected number of vertices revealed by ALG is at most $\frac{1001\ell T}{n}$. ∎

Now we can conclude the proof of Theorem 5.4. Observe that if $\ell>\frac{n}{4500}$ then Theorem 5.4 is trivial, since then $O\big(\frac{T\ell}{n}\big)$ is $\Omega(T)$ and any T-steps randomized algorithm can reveal at most $O(T)$ vertices. Similarly, Theorem 5.4 is trivial for $T>\frac{n^2}{6400\ell}$, since then $O\big(\frac{T\ell}{n}\big)$ is $\Omega(n)$, and any randomized algorithm can reveal at most $O(n)$ vertices (since there are only n vertices). Otherwise, if $\ell\leq\frac{n}{4500}$ and $T\leq\frac{n^2}{6400\ell}$, then by Lemma 5.17 the expected number of vertices revealed by a deterministic algorithm that performs T queries is $O\big(\frac{T\ell}{n}\big)$, and therefore Yao's principle implies that the result holds also for randomized algorithms.

5.3. Lower bound for $(1+\varepsilon)$-approximation of $\mathfrak{cost}(\mathbf{X})$: proof of Theorem 1.3. Now we are ready to prove the following implication of Theorem 5.4: Theorem 1.3, which states that for any $\mathfrak{c}\leq k$, any algorithm that for any metric space (X,d) of size n, with probability at least $\frac{2}{3}$ estimates the cost of a k-NN graph within an additive error term $\mathfrak{c}\cdot\mathfrak{cost}(X)$ requires $\Omega\Big(\frac{n^2}{k}\Big)$ queries.

Proof of Theorem 1.3. For the simplicity of presentation, we will assume that n is a multiple of $k+1$; it is easy to extend the arguments to the general case though.

In what follows, we will consider metric spaces (X,d) defined by partitioning X into *clusters*, such that two points in the same cluster are at distance 0 apart, and two points from distinct clusters are at distance 1 apart. (To ensure that so defined (X,d) is a metric space, one should think about distance 0 like about an arbitrary small positive number $\ll 1$.)

Consider two problem instances (cf. Figure 5.1):

- $(\mathcal{X},d)$: consists of $\frac{n}{k+1}$ clusters of size $k+1$ each, and
- $(\mathcal{X}',d)$: consists of $\frac{n}{k+1}-1$ clusters of size $k+1$ each, one cluster of size k and one isolated point (cluster of size 1).

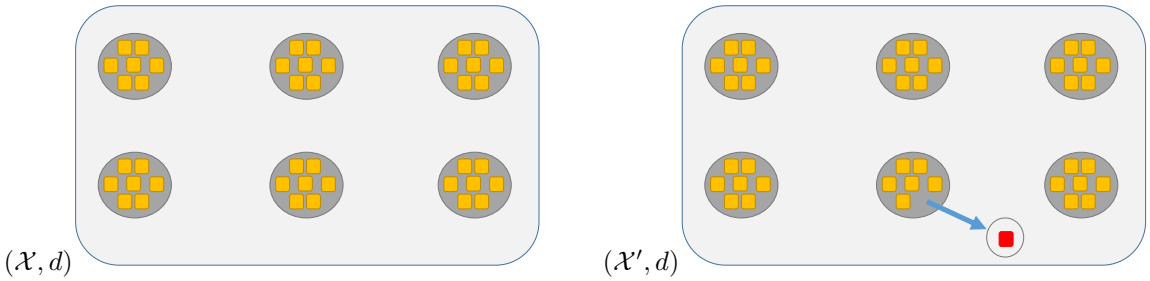

FIG. 5.1. *Instance $(\mathcal{X}, d)$ and instance $(\mathcal{X}', d)$ obtained from $(\mathcal{X}, d)$ by taking a point from its cluster.*

In short, instance $(\mathcal{X}', d)$ is obtained by taking one of the clusters in the instance $(\mathcal{X}, d)$ and removing from it a single point. Notice that the cost of the first instance is $\mathfrak{cost}(\mathcal{X}) = 0$, whereas the cost of the second instance is $\mathfrak{cost}(\mathcal{X}') = 2k$.

We claim that Theorem 5.4 implies that any algorithm that can distinguish between these two instances requires $\Omega\left(\frac{n^2}{k}\right)$ queries.

Let us transform the input instances into the framework of balanced clustered graphs from Section 5.1. We define an undirected unweighted graph $G = (X, E)$ such that two vertices are adjacent in G if and only if the corresponding points are in the same cluster in (X, d). Notice that for $(\mathcal{X}, d)$, the underlying graph G consists of $\frac{n}{k+1}$ cliques of size $k+1$ each, and for $(\mathcal{X}', d)$, one of the cliques in G is split into a clique of size k and an isolated vertex.

Let us randomly permute the vertices of G and consider the two instances above. By Theorem 5.4, any randomized algorithm ALG that considers the first instance $(\mathcal{X}, d)$ and queries the input in T places reveals in expectation $O\left(\frac{Tk}{n}\right)$ vertices. Now notice that the single point that is removed from a cluster to define the second instance is a random point, and so if algorithm ALG is run on the second instance $(\mathcal{X}', d)$, it will see a difference in its queries only if it reveals that single point in $(\mathcal{X}, d)$, and for that it requires (in expectation) $O\left(\frac{n^2}{k}\right)$ queries. Therefore, no algorithm ALG that queries the first instance $(\mathcal{X}, d)$ in $o\left(\frac{n^2}{k}\right)$ queries will able to distinguish it (with probability at least $\frac{2}{3}$) from the second instance $(\mathcal{X}', d)$. ∎

While the lower bound in Theorem 1.3 shows that our first algorithm performing $O\left(\frac{n^2 \log n}{\varepsilon^2 k}\right)$ queries (cf. Theorem 1.1) is asymptotically almost optimal, such a claim does not extend to the case when one allows the estimation error to be a function of $\mathfrak{mst}(X)$ and $k = O(\sqrt{n})$. Indeed, it is easy to see that the construction in Theorem 1.3 has $\mathfrak{mst}(\mathcal{X}) = \frac{n}{k+1} - 1$ for the first instance X_1, and $\mathfrak{mst}(\mathcal{X}') = \frac{n}{k+1}$ for the second instance $\mathcal{X}'$. Therefore, for $k = O(\sqrt{n})$ an error of $\Theta\left(\frac{n}{k+1}\right)$ is $\Omega(k)$, and hence it is of the same order as the cost of $(\mathcal{X}', d)$. (On the other hand, if $k = \omega(\sqrt{n})$ then the claim in Theorem 1.3 still holds even if we allowed an additive error of $\varepsilon \cdot \mathfrak{mst}(X)$.) To obtain an asymptotically tight lower bound that would contain the additive error term of the form $\varepsilon \cdot (\mathfrak{cost}(X) + \mathfrak{mst}(X))$, we need more elaborate arguments that we will present in the next section.

6. Lower bound of $\Omega\left(\min\left\{\frac{nk^{3/2}}{\varepsilon}, \frac{n^2}{k}\right\}\right)$ for approximating $\mathfrak{cost}(X)$ to within $\varepsilon \cdot (\mathfrak{cost}(X) + \mathfrak{mst}(X))$: Proof of Theorem 1.4. In this section we show Theorem 1.4, that any randomized algorithm that computes $\overline{\mathfrak{cost}}(X)$ with $|\mathfrak{cost}(X) - \overline{\mathfrak{cost}}(X)| \leq \varepsilon(\mathfrak{cost}(X) + \mathfrak{mst}(X))$ requires $\Omega\left(\min\left\{\frac{nk^{3/2}}{\varepsilon}, \frac{n^2}{k}\right\}\right)$ queries. Our construction relies on a design of two metric spaces that are undistinguishable with $o\left(\min\left\{\frac{nk^{3/2}}{\varepsilon}, \frac{n^2}{k}\right\}\right)$ queries, and for which finding an estimate for their costs would allow to distinguish between them.

Let h be an integer parameter (which we set to $h = O\left(\frac{k^{3/2}}{\varepsilon}\right)$ if $k \leq (4\varepsilon n)^{2/5}$, and $h = O\left(\frac{n}{k}\right)$ otherwise). Let r be an integer such that $r(h+2)(k+1) = n$; notice that $r = \Theta\left(\frac{n}{kh}\right)$. (In the arguments below we assume that r and $\sqrt{k}$ are integer; extension to the general case is straightforward.) In our analysis, the metric space has only two types of distances, 0 or 1.

Let us define two metric spaces $\mathcal{X}$ and $\mathcal{X}'$ on n points (see also Figure 6.1).

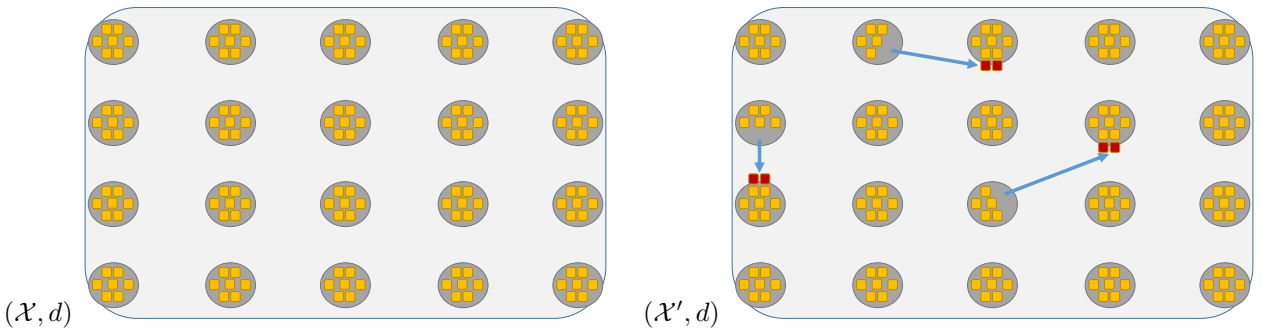

FIG. 6.1. *Instance $(\mathcal{X}, d)$ with $r \cdot (h+2) = 12$ cluster point sets of size $k+1$ each, and instance $(\mathcal{X}', d)$ obtained by moving in $(\mathcal{X}, d)$ $r = 3$ sets of $\sqrt{k}$ points between some clusters.*

Construction of two metric spaces $\mathcal{X}$ and $\mathcal{X}'$ on n points:

Constraints for positive parameters n, k, r, h: $\sqrt{k}, r, h \in \mathbb{N}$, $k \leq n$, $r(h+2)(k+1) = n$

- Let $\mathcal{X}$ consist of the following:
  - $r \cdot (h+2)$ cluster point sets of $k+1$ points each, with pairwise distance 0 and distance 1 to everybody else.
- Let $\mathcal{X}'$ consist of the following:
  - $r \cdot h$ cluster point sets of $k+1$ points each, with pairwise distance 0 and distance 1 to everybody else;
  - r cluster point sets with $k+1-\sqrt{k}$ points each, with pairwise distance 0 and distance 1 to everybody else;
  - r cluster point sets with $k+1+\sqrt{k}$ points each, with pairwise distance 0 and distance 1 to everybody else.

It is easy to see that so defined $\mathcal{X}$ and $\mathcal{X}'$ are two properly defined metric spaces on n points (assuming, as we do throughout the paper, that distance 0 is an arbitrary small positive number $\ll 1$.). One can construct $\mathcal{X}'$ from $\mathcal{X}$ by taking r random pairs of the clusters and then, in each pair, by moving $\sqrt{k}$ points from one of the cluster to the other cluster in the pair.

The following straightforward claim describes some basic properties of $\mathcal{X}$ and $\mathcal{X}'$.

CLAIM 6.1. $\mathtt{cost}(\mathcal{X}) = 0$ *and* $\mathtt{cost}(\mathcal{X}') = r \cdot (k+1-\sqrt{k}) \cdot \sqrt{k}$.
Further, the cost of a minimum spanning tree is $\mathtt{mst}(\mathcal{X}) = \mathtt{mst}(\mathcal{X}') = r \cdot (h+2) - 1$.

Next, we state our main claim that any algorithm that distinguishes between the input instances of metrics $\mathcal{X}$ and $\mathcal{X}'$ must have query complexity $\Omega(nh)$. We defer the proof of Lemma 6.2 (which relies on the approach from Section 5, cf. Theorem 5.4) to Section 6.1.

LEMMA 6.2 (**Main**). *Any algorithm that with probability at least $\frac{5}{6}$ distinguishes between $\mathcal{X}$ and $\mathcal{X}'$ requires $\Omega(nh)$ time.*

With Lemma 6.2 at hand, we can conclude with the proof of our main Theorem 1.4.

Proof of Theorem 1.4. Let us first notice that the claim trivially holds when $k^{3/2}/\varepsilon = O(1)$ or when $k = \Omega(n)$. To see that $\Omega(n)$ is always a lower bound (which takes care of the case $k = \Omega(n)$), consider the following two instances: one in which all points are clustered together, and another in which there is a single point at a large distance from all other points. It is known that there is no $o(n)$-time algorithm that can distinguish between these two input instances with probability at least $\frac{5}{6}$, and so *for any* k, no $o(n)$-time algorithm can (with probability at least $\frac{5}{6}$) approximate $\mathtt{cost}(X)$ with an additive error term $\frac{1}{2} \cdot (\mathtt{cost}(X) + \mathtt{mst}(X))$.

In view of that, from now on we assume that $\left\lceil \frac{k^{3/2}}{48\varepsilon} \right\rceil \geq 2$and $k = o(n)$ (to ensure the existence of $h \in \mathbb{N}$ satisfying $(h+2)(k+1) \leq n$).

We show that for an appropriate value for h, $\Omega\Big(\min\Big\{\frac{nk^{3/2}}{\varepsilon}, \frac{n^2}{k}\Big\}\Big)$ queries are necessary to distinguish

between the input instances $\mathcal{X}$ and $\mathcal{X}'$. We prove this by first showing that if $k \leq (4\varepsilon n)^{2/5}$ then one requires $\Omega\left(\frac{nk^{3/2}}{\varepsilon}\right)$ queries and then show that if $k > (4\varepsilon n)^{2/5}$ then one requires $\Omega\left(\frac{n^2}{k}\right)$ queries. This gives the required lower bound of $\Omega\left(\min\left\{\frac{nk^{3/2}}{\varepsilon}, \frac{n^2}{k}\right\}\right)$. (The difference between the two cases, when $k \leq (4\varepsilon n)^{2/5}$ and when $k > (4\varepsilon n)^{2/5}$, is that in the first case we use $r > 1$ (and $h = \Theta(k^{3/2}/\varepsilon)$) and in the second case we use $r = 1$ (and $h = \Theta(n/k)$, to ensure $r = \Theta\left(\frac{n}{kh}\right)$).)

Small k. Let us begin with the case $k \leq (4\varepsilon n)^{2/5}$, in which case we set $h = \left\lceil \frac{k^{3/2}}{48\varepsilon} \right\rceil - 1$ (and by the comments above, we have $h \geq 1$). Notice that then $(h+2)(k+1) \leq n/2$, and hence the constructions of $\mathcal{X}$ and $\mathcal{X}'$ are valid.

Assume, for the purpose of contradiction, that we have an algorithm that in time $o(nh)$, with probability at least $\frac{5}{6}$ estimates the cost of a k-nearest neighbor graph for an arbitrary metric space (X, d) of size n to within an error term $\varepsilon \cdot (\mathtt{cost}(X) + \mathtt{mst}(X))$. We will argue that this algorithm would also be able to distinguish our input instances $\mathcal{X}$ and $\mathcal{X}'$ in $o(nh)$ time, which by Lemma 6.2 would lead to contradiction.

If we run this algorithm to our input instances $\mathcal{X}$ and $\mathcal{X}'$, then with probability at least $\frac{2}{3}$ it would return two values $\overline{\mathtt{cost}}_{\mathcal{X}}$ and $\overline{\mathtt{cost}}_{\mathcal{X}'}$ that satisfy the following inequalities,

$$|\overline{\mathtt{cost}}_{\mathcal{X}} - \mathtt{cost}(\mathcal{X})| \leq \varepsilon \cdot (\mathtt{cost}(\mathcal{X}) + \mathtt{mst}(\mathcal{X})) \text{ and } |\overline{\mathtt{cost}}_{\mathcal{X}'} - \mathtt{cost}(\mathcal{X}')| \leq \varepsilon \cdot (\mathtt{cost}(\mathcal{X}') + \mathtt{mst}(\mathcal{X}')) \ .$$

In particular, with probability at least $\frac{2}{3}$ we have

$$\overline{\mathtt{cost}}_{\mathcal{X}} \leq (1+\varepsilon) \cdot \mathtt{cost}(\mathcal{X}) + \varepsilon \cdot \mathtt{mst}(\mathcal{X}) \quad \text{and} \quad \overline{\mathtt{cost}}_{\mathcal{X}'} \geq (1-\varepsilon) \cdot \mathtt{cost}(\mathcal{X}') - \varepsilon \cdot \mathtt{mst}(\mathcal{X}') \ .$$

By Claim 6.1, and since $\varepsilon \leq \frac{1}{2}$, the bound above implies that with probability at least $\frac{2}{3}$ we have

$$\overline{\mathtt{cost}}_{\mathcal{X}} \leq (1+\varepsilon) \cdot \mathtt{cost}(\mathcal{X}) + \varepsilon \cdot \mathtt{mst}(\mathcal{X}) = 0 + \varepsilon \cdot (r \cdot (h+2) - 1) \leq 3 \cdot \varepsilon \cdot r \cdot h \ , \tag{6.1}$$

and

$$\overline{\mathtt{cost}}_{\mathcal{X}'} \geq (1-\varepsilon) \cdot \mathtt{cost}(\mathcal{X}') - \varepsilon \cdot \mathtt{mst}(\mathcal{X}) \geq (\tfrac{1}{4} \cdot k^{3/2} - 3 \cdot \varepsilon \cdot h) \cdot r \ . \tag{6.2}$$

Next, notice that if we set $h = \left\lceil \frac{k^{3/2}}{48\varepsilon} \right\rceil - 1$, then we have that $3\varepsilon h < \frac{1}{8}k^{3/2} - 3\varepsilon h$ and thus,

$$\overline{\mathtt{cost}}_{\mathcal{X}} \leq 3 \cdot \varepsilon \cdot r \cdot h < \left(\tfrac{1}{8} \cdot k^{3/2} - 3 \cdot \varepsilon \cdot h\right) \cdot r \leq \overline{\mathtt{cost}}_{\mathcal{X}'} \ ,$$

where the first inequality follows from (6.1) and the last one follows from (6.2).

But this implies that with probability at least $\frac{2}{3}$ we have $\overline{\mathtt{cost}}_{\mathcal{X}} < \overline{\mathtt{cost}}_{\mathcal{X}'}$, and thus we can use our algorithm to distinguish the input instances $\mathcal{X}$ and $\mathcal{X}'$ in $o(nh)$ time, which by Lemma 6.2 leads to contradiction. Therefore, there is no $o(nh)$-time algorithm that with probability at least $\frac{5}{6}$ estimates the cost of a k-NN graph to within an additive error term $\varepsilon \cdot (\mathtt{cost}(X) + \mathtt{mst}(X))$, assuming $k \leq (4\varepsilon n)^{2/5}$ and $h = \left\lceil \frac{k^{3/2}}{48\varepsilon} \right\rceil - 1$.

Large k. The analysis above requires k to be not too large and we assumed $k \leq (4\varepsilon n)^{2/5}$ to ensure the necessary constraint $(h+2)(k+1) \leq \frac{1}{2}n$ with $h = \left\lceil \frac{k^{3/2}}{48\varepsilon} \right\rceil - 1$. When k is larger, we have to take consider a smaller value of h and we set $r = 1$.

Let $k > (4\varepsilon n)^{2/5}$ and $k = o(n)$ (to ensure the existence of $h \in \mathbb{N}$ satisfying $(h+2)(k+1) \leq n$). In our analysis above, assuming that $r = 1$ (which is what we have now), the claim relies on a single inequality, $3\varepsilon h < \frac{1}{8}k^{3/2} - 3\varepsilon h$. We set $h = \left\lfloor \frac{n}{12k} \right\rfloor$ to ensure that this inequality holds. Thus, as above, assuming $k > (4\varepsilon n)^{2/5}$, we obtain a lower bound of $\Omega(nh)$, which is $\Omega\left(\frac{n^2}{k}\right)$. ∎

6.1. Lemma 6.2 and testing hypergeometric distributions. In this section we prove Lemma 6.2. We will first provide some tools, and then will link the task of distinguishing between $\mathcal{X}$ and $\mathcal{X}'$ to the problem of testing hypergeometric distributions.

6.1.1. Testing hypergeometric distributions. We consider the standard model of *distribution testers* to access the unknown distribution by getting independent and identically distributed samples from it (see, e.g., [3]). Let $\mathcal{D}$ be a fixed distribution over some domain Ω. A sampling oracle for $\mathcal{D}$ is an oracle that

when queried, returns an element $x \in \Omega$, where the probability that x is returned is $\mathcal{D}(x)$, independently of all previous calls to the oracle. (Notice that this standard definition immediately implies that all distribution testers in this standard model are essentially non-adaptive.)

Denote by $\mathbb{H}_{N,M,S}$ the *hypergeometric distribution* with parameters N, M, and S, that is, for is a random variable $\mathcal{Z}$ with distribution $\mathbb{H}_{N,M,S}$, we have $\mathbf{Pr}[\mathcal{Z} = r] = \frac{\binom{M}{r}\cdot\binom{N-M}{S-r}}{\binom{N}{S}}$. Similarly, denote by $\mathbb{B}(S,p)$ the *binomial distribution* with parameters S and p, that is, for a random variable $\mathcal{Z}$ with distribution $\mathbb{B}(S,p)$, we have $\mathbf{Pr}[\mathcal{Z} = r] = \binom{S}{r} \cdot p^r \cdot (1-p)^{S-r}$.

Let us recall that the *total variation distance* between two discrete probability distributions $\mathcal{D}$ and $\mathcal{D}'$ over the same domain Ω is equal to the half of the first norm distance between these distributions, $\|\mathcal{D}' - \mathcal{D}\|_{TV} = \frac{1}{2}\|\mathcal{D}' - \mathcal{D}\|_1 = \frac{1}{2}\sum_{\omega\in\Omega} |\mathbf{Pr}_{\mathcal{D}'}[\omega] - \mathbf{Pr}_{\mathcal{D}}[\omega]|$.

It is known that the hypergeometric distribution and the binomial distribution are very similar. Indeed, the hypergeometric distribution is a discrete probability distribution that describes the probability of r successes (random draws for which the object drawn has a specified feature) in S draws, *without replacement*, from a finite population of size N that contains exactly M objects with that feature, wherein each draw is either a success or a failure. In contrast, the binomial distribution describes the probability of r successes in S draws *with replacement*, where the success probability p corresponds to the ratio of the number of objects with a given feature to the total population. To quantify this similarity between $\mathbb{H}_{N,M,S}$ and $\mathbb{B}(S, M/N)$, Freedman [16] showed that the total variation norm between sampling with and without replacement is small if S^2/N is small, and then Ehm [12] and Künsch [22] extended this to larger values of S:

LEMMA 6.3. *[12, 22] If $S \cdot M/N \cdot (1 - M/N) \geq 1$, then*

$$\|\mathbb{H}_{N,M,S} - \mathbb{B}(S, M/N)\|_{TV} \leq \frac{S-1}{N-1} \ .$$

Next, let us remind the result due to Paninski [29, Theorem 4] (we state this result only in the most basic case of the support size of the uniform distribution to be equal to 2).

LEMMA 6.4. *[29] If $S \cdot \varepsilon^2 = o(1)$, then*

$$\|\mathbb{B}(S, \tfrac{1}{2}) - \mathbb{B}(S, \tfrac{1}{2} + \varepsilon)\|_{TV} = o(1) \ .$$

In particular, no tester that samples only $o(\varepsilon^{-2})$ elements can distinguish with probability at least $\frac{2}{3}$ between the uniform distribution on two elements and a distribution that assigns probability $\frac{1}{2} + \varepsilon$ to one element and $\frac{1}{2} - \varepsilon$ to the other element.

Now, we combine Lemmas 6.3 and 6.4 to obtain our main Lemma 6.5. Observe that to distinguish between two different hypergeometric distributions, we have to consider a (non-adaptive) tester that out of a finite population of size N that contains exactly M objects with a special feature, samples S random elements from the population (equivalently, one performs S steps of sampling without replacement a single element from the population). Our goal is to determine the value of M and in our case, to distinguish between the case $M = \frac{1}{2}N$ and $M = \frac{1}{2}N + c\sqrt{N}$. This would mean that we can distinguish between two hypergeometric distributions $\mathbb{H}_{N,\frac{1}{2}N,S}$ and $\mathbb{H}_{N,\frac{1}{2}N+c\sqrt{N},S}$.

LEMMA 6.5. *Let c be an arbitrary positive constant. Let $\mathcal{X}$ be a random variable with hypergeometric distribution $\mathbb{H}_{N,\frac{1}{2}N,S}$ and let $\mathcal{Y}$ be a random variable with hypergeometric distribution $\mathbb{H}_{N,\frac{1}{2}N+c\sqrt{N},S}$. If $S = o(N)$, then $\|\mathcal{X} - \mathcal{Y}\|_{TV} = o(1)$.*

In particular, no tester that samples only $o(N)$ elements can distinguish with probability at least $\frac{2}{3}$ between $\mathbb{H}_{N,\frac{1}{2}N,S}$ and $\mathbb{H}_{N,\frac{1}{2}N+c\sqrt{N},S}$.

Proof. It is enough (cf. the arguments in [29]) to show only the first claim, that if $S = o(N)$, then $\|\mathcal{X} - \mathcal{Y}\|_{TV} = o(1)$.

We will use the triangle inequality on the norms which ensures the following:

$$\|\mathbb{H}_{N,N/2,S} - \mathbb{H}_{N,N/2+c\sqrt{N},S}\|_{TV} \leq \tag{6.3}$$
$$\|\mathbb{H}_{N,N/2,S} - \mathbb{B}(S, \tfrac{1}{2})\|_{TV} + \|\mathbb{B}(S, \tfrac{1}{2}) - \mathbb{B}(S, \tfrac{1}{2} + \tfrac{c}{\sqrt{N}})\|_{TV} + \|\mathbb{H}_{N,N/2+c\sqrt{N},S} - \mathbb{B}(S, \tfrac{1}{2} + \tfrac{c}{\sqrt{N}})\|_{TV} \ .$$

To use this bound, first we obtain by Lemma 6.3,

$$\|\mathbb{H}_{N,N/2,S} - \mathbb{B}(S, \tfrac{1}{2})\|_{TV} \leq \frac{S-1}{N-1}$$

and

$$\|\mathbb{H}_{N,N/2+c\sqrt{N},S} - \mathbb{B}(S, \tfrac{1}{2} + \tfrac{c}{\sqrt{N}})\|_{TV} \leq \frac{S-1}{N-1} \ .$$

Next, by Lemma 6.4, since $S = o(N)$, we have

$$\|\mathbb{B}(S, \tfrac{1}{2}) - \mathbb{B}(S, \tfrac{1}{2} + \tfrac{c}{\sqrt{N}})\|_{TV} = o(1) \ .$$

Since $S = o(N)$, we combine the bounds above with the triangle inequality (6.3) to obtain

$$\|\mathbb{H}_{N,N/2,S} - \mathbb{H}_{N,N/2+c\sqrt{N},S}\|_{TV} \leq \frac{S-1}{N-1} + o(1) + \frac{S-1}{N-1} = o(1) \ . \qquad \square$$

6.1.2. Testing two clusters. We will relate the problem of distinguishing between hypergeometric distribution $\mathbb{H}_{N,\frac{1}{2}N,S}$ and hypergeometric distribution $\mathbb{H}_{N,\frac{1}{2}N+c\sqrt{N},S}$ from Lemma 6.5 to a clustering testing problem.

Consider the following clustering problem. We are given a graph G with N vertices, which consists of two disjoint cliques, one of size M, $M < N$, and another of size $N - M$. Suppose that we know that either $M = \frac{1}{2}N$, or that $M = \frac{1}{2}N + c\sqrt{N}$ for some small positive constant c. Consider an algorithm ALG that is allowed only to query the adjacency matrix of G, i.e., to query whether a pair of vertices are connected by an edge. How many queries algorithm ALG must ask to distinguish between the two cases, when $M = \frac{1}{2}N$ and $M = \frac{1}{2}N + c\sqrt{N}$? We prove the following.

LEMMA 6.6. *Let c be a positive constant. Let G_1 and G_2 be two graphs on the vertex set $V = \{1, \ldots, N\}$, N even, such that G_1 consists of two disjoint cliques of size $\frac{1}{2}N$ each, and G_2 consists of two disjoint cliques of size $\frac{1}{2}N + c\sqrt{N}$ and $\frac{1}{2}N - c\sqrt{N}$, respectively. Then, no algorithm that queries entries of the adjacency matrix of the input graph only about $o(N)$ vertices can distinguish with probability at least $\frac{2}{3}$ between G_1 and G_2.*

Proof. The proof is by reduction to the problem of testing hypergeometric distribution.

Let us first set up our framework. Fix $N \in \mathbb{N}$. For any natural $M \leq N$, let $G^{\langle M \rangle}$ be a graph on the vertex set $V = \{1, 2, \ldots, N\}$ which consists of two disjoint cliques, one of size M and another of size $N - M$. Let ALG be any algorithm that queries the entries of the adjacency matrix of the input graph G^*, and on the basis of the queries decides whether $G^* = G^{\langle M \rangle}$ or $G^* = G^{\langle M' \rangle}$, for two distinct M and M'. We will consider any algorithm ALG that queries at most T vertices in the adjacency matrix of the input graph. We will show that if $M = \frac{1}{2}N$ and $M' = \frac{1}{2}N + c\sqrt{N}$, then in order to distinguish with probability at least $\frac{2}{3}$ between $G^{\langle M \rangle}$ and $G^{\langle M' \rangle}$, we must have $T = \Omega(N)$.

We first notice that in the case of two cliques, the behavior of any algorithm ALG can be slightly modified to follow a very simple format. One first queries an arbitrary pair of vertices x_0 and y_0:

- if $\{x_0, y_0\} \in E$ then we will assign both x_0 and y_0 to the first clique C_1;
- otherwise, if $\{x_0, y_0\} \notin E$ then we will assign x_0 to C_1 and y_0 to C_2.

Then, we will ensure that the following *invariant* is maintained:

- if we have already queried for any edge incident to a vertex x then we have already determined whether $x \in C_1$ or $x \in C_2$;
- however, if we have not queried yet for any edge incident to x, then we cannot determine the cluster of x (and the algorithm cannot be sure whether x is in C_1 or in C_2).

In order to maintain the invariant, we will *enhance* ALG: except for the initial, first query (which checks if $\{x_0, y_0\} \in E$), if ALG is querying for a pair x, y:

- we will give away one *free* query to ALG to determine whether $\{x, x_0\} \in E$.

Notice that since we know that $x_0 \in C_1$, then the additional query $\{x, x_0\} \in E$? uniquely determines if $x \in C_1$ or $x \in C_2$, and hence together with the query $\{x, y\} \in E$? it uniquely determines whether $y \in C_1$ or $y \in C_2$.

This ensures that the invariant is satisfied. (To ensure that this is always possible, we additionally require that ALG queries at most $\min\{M, N-M\}$ vertices, which is what we require from our input parameters.) Notice that since for any two vertices queried before we have already determined whether they are adjacent, algorithm ALG has no reason to ask queries involving two vertices queried earlier.

Now, suppose that there is an algorithm ALG that asks queries about at most T vertices to determine whether the input graph is G_1 or G_2. We can assume, without loss of generality, that every time ALG queries for an edge $\{x, y\}$ with x or y as a new yet non-visited vertex, then the new vertex is a random vertex among all non-visited-yet vertices (since no information about these vertices is known to the algorithms and since we have started with randomly permuting all vertices). Therefore, every time ALG queries a new vertex (and it can query up to two vertices with a single query), one can think that the algorithm checks the status of a new vertex.

We will show that ALG can be used to distinguish between hypergeometric distributions $\mathbb{H}_{N,\frac{1}{2}N,S}$ and $\mathbb{H}_{N,\frac{1}{2}N+c\sqrt{N},S}$, with $S \leq T$, and hence, by Lemma 6.5, we must have $T = \Omega(N)$.

Consider an instance of the problem of testing hypergeometric distribution: there is a finite population of size N that contains exactly M objects with a special feature, one performs S steps (with $S \leq T$) of sampling an element from the population — without replacement. Our goal is to determine the value of M, to distinguish between the case $M = \frac{1}{2}N$ and $M = \frac{1}{2}N + c\sqrt{N}$. We will model this task by considering a graph G on N vertices and distinguishing between the case when G is $G^{\langle \frac{1}{2}N \rangle}$ or $G^{\langle \frac{1}{2}N+c\sqrt{N} \rangle}$. Let us randomly permute the vertices of G, to ensure, as claimed above, that every time ALG queries a new vertex (and it can query up to two vertices with a single query), one can think that the algorithm checks the status of a new *random* vertex from G.

In order to distinguish between hypergeometric distributions $\mathbb{H}_{N,\frac{1}{2}N,S}$ and $\mathbb{H}_{N,\frac{1}{2}N+c\sqrt{N},S}$, we run ALG on G that is either $G^{\langle \frac{1}{2}N \rangle}$ or $G^{\langle \frac{1}{2}N+c\sqrt{N} \rangle}$. In each step, algorithm ALG queries for at most two new random vertices; a query about a new vertex x determines whether x is in cluster C_1 or in C_2. This can be modeled by choosing S *random* (distinct, so without replacement) vertices, $S \leq T$, and learning which of them belong to C_1 and which of them belong to C_2. Let ϑ_1 be the number of sampled vertices belonging to C_1 and ϑ_2 be the number of sampled vertices belonging to C_2. Notice that if G is equal to $G^{\langle M \rangle}$ then either ϑ_1 is a random variable with hypergeometric distribution $\mathbb{H}_{N,M,S}$, or $\vartheta_2 = S - \vartheta_1$ has hypergeometric distribution $\mathbb{H}_{N,M,S}$. Therefore, by distinguishing in T steps on whether G is either $G^{\langle \frac{1}{2}N \rangle}$ or $G^{\langle \frac{1}{2}N+c\sqrt{N} \rangle}$, we also would be able to distinguish on whether ϑ_1 (or, symmetrically, ϑ_2) has hypergeometric distribution $\mathbb{H}_{N,\frac{1}{2}N,S}$ or hypergeometric distribution $\mathbb{H}_{N,\frac{1}{2}N+c\sqrt{N},S}$, which is known to require $\Omega(N)$ random samples, by Lemma 6.5. This proves that algorithm ALG queries $T = \Omega(N)$ vertices. □

6.1.3. Testing many clusters. We now give a simple extention of the framework from Section 6.1.2 and Lemma 6.6 to the case of many clusters. We will prove the following lemma.

LEMMA 6.7. *Let c be a positive constant and r be a positive integer. Let H_1 and H_2 be two graphs on the vertex set $V = \{1, 2, \ldots, rN\}$, N even, such that*

- *H_1 consists of $2r$ disjoint cliques of size $\frac{1}{2}N$ each, and*
- *H_2 consists of $2r$ disjoint cliques: r cliques of size $\frac{1}{2}N + c\sqrt{N}$ and r cliques of size $\frac{1}{2}N - c\sqrt{N}$.*

Then, no algorithm that queries entries of the adjacency matrix of the input graph only about $o(N)$ vertices can distinguish with probability at least $\frac{2}{3}$ between H_1 and H_2.

Proof. For the purpose of contradiction, suppose that there is an algorithm ALG that queries entries of the adjacency matrix of the input graph about $o(N)$ vertices and distinguishes between H_1 and H_2 with probability at least $\frac{2}{3}$. We argue that ALG could be used to distinguish between the two instances, as defined in Lemma 6.6. Indeed, suppose we have two graphs G_1 and G_2 on the vertex set $V = \{1, 2, \ldots, N\}$, as in Lemma 6.6. Then H_1 is obtained by taking r disjoint copies of G_1 and H_2 is obtained by taking r disjoint copies of G_2 (with relabeling of the vertices). Hence, we can distinguish between G_1 and G_2 by applying ALG to distinguish between H_1 and H_2, without loss of complexity. This would distinguish G_1 from G_2 with $o(N)$ queries, contradicting Lemma 6.6. □

6.1.4. Completing the proof of Lemma 6.2. Now we are ready to complete the proof of Lemma 6.2, that to distinguish $\mathcal{X}$ from $\mathcal{X}'$ one needs $\Omega(nh)$ time. Our analysis relies on the tools from Section 5 (Theorem 5.4) and from Section 6.1.3.

Consider any algorithm ALG that distinguishes between the instances $\mathcal{X}$ and $\mathcal{X}'$ (cf. page 30 and Figure 6.1). For the purpose of contradiction, suppose that ALG performs $T = o(nh)$ queries.

One can model the input instances $\mathcal{X}$ and $\mathcal{X}'$ by an undirected graph $\mathcal{G}$ on $V = \{1, 2, \ldots, n\}$, such that two vertices are adjacent if and only if they are in the same cluster in the input instance. To use this setting, let us first consider the framework analyzed in Lemma 6.7. Let H_1 and H_2 be two graphs on $2(k+1)r$ vertices such that H_1 consists of $2r$ disjoint cliques of size $k+1$ each, and H_2 consists of $2r$ disjoint cliques: r cliques of size $k+1+\sqrt{k}$ and r cliques of size $k+1-\sqrt{k}$.

Observe that graph $\mathcal{G}$ consists of either $\frac{1}{2}h+1$ copies of graphs H_1, or $\frac{1}{2}h$ copies of graphs H_1 and one copy of H_2, depending on whether the input to ALG is $\mathcal{X}$ or $\mathcal{X}'$. Thus if ALG can distinguish between $\mathcal{X}$ or $\mathcal{X}'$, it can also distinguish between the instances for the undirected graph $\mathcal{G}$.

Let H be a graph that is isomorphic either to H_1 or H_2. By Lemma 6.7, there is no algorithm that queries entries of the adjacency matrix of H only about $o(k)$ vertices and that can distinguish with probability at least $\frac{2}{3}$ whether $H = H_1$ or $H = H_2$. Our proof of Lemma 6.2 is by contradiction: we will show (by observing $\mathcal{G}$) that any algorithm that distinguishes $\mathcal{X}$ from $\mathcal{X}'$ with $o(nh)$ queries can be used to distinguish H_1 from H_2 with $o(k)$ queries.

One can view $\mathcal{X}$ and $\mathcal{X}'$ in the context of *pairs* of clusters: $\mathcal{X}'$ is obtained from $\mathcal{X}$ by taking r random *pairs* of the clusters and then, in each pair, of moving $\sqrt{k}$ points from one of the cluster to the other cluster. Let us call a pair of clusters of size $k+1$ each *balanced* and a pair of clusters of sizes $k+1\pm\sqrt{k}$ *unbalanced*. In view of the above, for the sake of the analysis, we will consider both $\mathcal{X}$ and $\mathcal{X}'$ as defined by *pairs of clusters*: we assume that $\mathcal{X}$ consists of $\frac{1}{2}r(h+2)$ *random pairs of balanced clusters* and $\mathcal{X}'$ is obtained from $\mathcal{X}$ by taking r of these random pairs of clusters and then, in each pair, of moving $\sqrt{k}$ points from one of the cluster to the other cluster, making these r pairs unbalanced (any point is in the same pair of clusters in both problem instances).

The definition of pairs of clusters naturally extends to pairs of cliques in the graphs $\mathcal{G}$ and H. If there is a pair of clusters in $\mathcal{X}$ or $\mathcal{X}'$, then we pair the corresponding cliques in $\mathcal{G}$. Similarly, graphs H_1 and H_2 consist of r pairs of cliques, in H_1 the two cliques in each pair are of size $k+1$, and in H_2 the pair consists of a clique of size $k+1+\sqrt{k}$ and a clique of size $k+1-\sqrt{k}$.

Let us set up an instance for algorithm ALG corresponding to $\mathcal{G}$ as a single copy of the input graph H and $\frac{1}{2}h$ copies of graphs H_1. Permute the vertices using a random permutation $\pi \in \mathbb{S}_n$. As before, this describes precisely either the input instance $\mathcal{X}$ or $\mathcal{X}'$, and so if ALG distinguishes between $\mathcal{X}$ and $\mathcal{X}'$, then it also determines whether $H = H_1$ or $H = H_2$. In this framework we will *project* ALG on H: Every time ALG queries for two vertices from the selected pairs of clusters (corresponding to H), we will apply identical query to H (in our tester that aims to distinguish H_1 from H_2). However, all queries about at least one vertex from outside H are ignored in our tester.

Let us call a point x *revealed by* ALG, if ALG queried entry $\{x, y\}$ for another vertex y that is in the same *pair of clusters* as x. Let τ be the expected number of vertices revealed by algorithm ALG. Notice that by Theorem 5.4, we have $\tau = O\big(\frac{Tk}{n}\big) = o(kh)$.

We now apply algorithm ALG to distinguish H_1 from H_2 with $o(k)$ queries. We run ALG until it reaches the threshold of $\frac{20\tau}{h+2}$ revealed points in the r copies of the pairs of clusters on which $\mathcal{X}$ and $\mathcal{X}'$ differ (which we call *selected pairs of clusters*). Notice that with that, our algorithm queries entries of the adjacency matrix of the input graph H on at most $\frac{20\tau}{h+2} = o(k)$ vertices of H.

We consider two cases, depending on whether algorithm ALG reveals at least $\frac{20\tau}{h+2}$ points in the r copies of the selected pairs of clusters, or it does not.

When ALG is run on $\mathcal{X}$, then all pairs of clusters are balanced and look identical. Therefore, because of the random permuting of the vertices, since the expected number of vertices revealed by ALG is τ, every pair of clusters (there are $\frac{1}{2}r(h+2)$ such pairs) has in expectation $\frac{2\tau}{r(h+2)}$ revealed vertices. Therefore, by Markov's inequality, algorithm ALG reveals at least $\frac{20\tau}{h+2}$ vertices from the r copies of the selected pairs of clusters with probability at most $\frac{1}{10}$. Thus, if ALG reveals at least $\frac{20\tau}{h+2}$ points in the r copies of the selected pairs of clusters, then we return $H = H_2$. Indeed, since in this case the input to ALG is $\mathcal{X}$ with probability at most $\frac{1}{10}$, the probability that $H = H_2$ is at least $\frac{9}{10}$. Hence, our algorithm distinguishes between H_1 and H_2 with probability at least $\frac{9}{10}$.

Next, let us consider the other case, when algorithm ALG reveals less than $\frac{20\tau}{h+2}$ points in the r copies of the selected pairs of clusters. In this case we return H_1 when ALG returns $\mathcal{X}$, and return H_2 otherwise.

Since ALG with probability at least $\frac{2}{3}$ correctly distinguishes between $\mathcal{X}$ and $\mathcal{X}'$, our algorithm correctly distinguishes between H_1 and H_2 with probability at least $\frac{2}{3}$.

Let us summarize our analysis above. We have shown that if there is an algorithm ALG that performs $T = o(nh)$ queries and distinguishes with probability at least $\frac{2}{3}$ between the problem instances $\mathcal{X}$ and $\mathcal{X}'$, then there is an algorithm that queries the entries of the adjacency matrix of the input graph H about at most $\frac{20\tau}{h} = o(k)$ vertices and distinguishes with probability at least $\frac{2}{3}$ between H_1 and H_2. This contradicts Lemma 6.6, and thus any algorithm that with probability at least $\frac{2}{3}$ distinguishes between $\mathcal{X}$ and $\mathcal{X}'$ requires $\Omega(nh)$ queries. This yields Lemma 6.2.

REFERENCES

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

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

[3] Tuğkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D. Smith, and Patrick White. Testing closeness of discrete distributions. Journal of the ACM, 60(1):4:1-–4:25, February 2013.

[4] Bernhard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Estimating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing*, 34(6): 1370–1379, 2005.

[5] Yu Chen, Sampath Kannan, and Sanjeev Khanna. Sublinear algorithms and lower bounds for metric TSP cost estimation. *Proceedings of the 47th International Colloquium on Automata, Languages and Programming (ICALP'20)*, pp. 30:1–30:19, 2020.

[6] Yu Chen, Sanjeev Khanna, and Zihan Tan. Sublinear algorithms and lower bounds for estimating MST and TSP cost in general metrics. Arxiv, CoRR abs/2203.14798, 2022.

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

[8] Artur Czumaj, Funda Ergün, Lance Fortnow, Avner Magen, Ilan Newman, Ronitt Rubinfeld, and Christian Sohler. Approximating the weight of the Euclidean minimum spanning tree in sublinear time. *SIAM Journal on Computing*, 35(1): 91–109, 2005.

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

[10] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab 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.

[11] Wei Dong, Moses Charikar, and Kai 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.

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

[13] Hossein Esfandiari and Michael 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.

[14] Uriel 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.

[15] Hendrik Fichtenberger and Dennis 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.

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

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

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

[19] Piotr 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.

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

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

[22] Hans Rudolf Künsch. The difference between the hypergeometric and the binomial distribution. Note, ETH Zürich, May 1998.

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

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

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

[26] Huy Nguyen and Krzysztof 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.

[27] Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. A near-optimal sublinear-time algorithm for approximat-

ing the minimum vertex cover size. *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*, pp. 1123–1131, 2012.

[28] Dávid Pál, Barnabás Póczos, and Csaba 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.

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

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

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

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

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

[34] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science (FOCS'77)*, pp. 222—227, 1977.

[35] Yuichi Yoshida, Masaki Yamamoto, and Hiro 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.

Appendix A. Assumptions on the input space.

Let (X, d_X) be our input metric space. We will assume in various places of the paper that the minimum distance is 1 and the maximum distance is $n^{(O(1))}$ and that the cost of the k-NN graph is at least $\frac{nk}{\varepsilon}$. Furthermore, we will assume that all distances are powers of $(1+\varepsilon)$. In the following we show how we can always achieve this setting when we work with a slightly weaker form of the triangle inequality.

DEFINITION A.1. *We say that (Y, d) is an ε-near metric space, if it satisfies all properties of a metric space except for the triangle inequality, which is relaxed to be:*

$$d(x, y) \leq (1+\varepsilon) \cdot (d(x, z) + d(z, y))$$

for all $x, y, z \in Y$.

We first approximate the diameter D of the metric space within a factor of $\frac{1}{2}$ using a sublinear time algorithm of Indyk [20]. The algorithm takes an arbitrary vertex v and computes its furthest neighbor u and returns it as an estimate $\widehat{D}$ for the diameter. By the triangle inequality, $\widehat{D} = d(u, v) \geq D/2$. We normalize distances by multiplying them with $kn/(\varepsilon\widehat{D})$, which results in distances between 0 and $2kn/\varepsilon$. Let (X', d'_X) be the resulting metric space (which is just a scaled version of (X, d_X). We observe that the cost of the minimum spanning tree of the metric space (X', d'_X) is at least kn/ε. Then we round every distance smaller than 1 to be 1 and every distance with weight more than one is rounded up to the closest power of $(1+\varepsilon)$. Let (Y, d) be the resulting space. Let $G_{X'}$ be a k-NN graph of (X', d'_X) and let G_Y be a k-NN graph of (Y, d). Then we have the following.

LEMMA A.2.

$$\mathtt{cost}(G_{X'}) \leq \mathtt{cost}(G_Y) \leq (1+\varepsilon) \cdot \mathtt{cost}(G_{X'}) + \varepsilon \cdot \mathtt{mst}(G_{X'}) \ ,$$

and

$$\mathtt{mst}(G_{X'}) \leq \mathtt{mst}(G_Y) \leq (1+2\varepsilon) \cdot \mathtt{mst}(G_{X'}) \ .$$

Proof. Since we are rounding up all distances, the lower bounds are immediate. For the first upper bound, we observe that there are at most kn edges in the k-NN graph. Every edge that is rounded to 1 can increase the cost by at most 1, which gives an overall increase of kn. This is at most $\varepsilon \cdot \mathtt{mst}(G_{X'})$. The other term follows immediate since we round the remaining edge weights to the closest power of $(1+\varepsilon)$. For the second statement, we observe that the $\mathtt{mst}$ has at most $n-1$ edges. The rest of the argument is similar to above. ∎

LEMMA A.3. *(Y, d) is an ε-near metric space.*

Proof. We need to show that (Y, d) satisfies the relaxed triangle inequality. Consider three points $x, y, z \in Y$ and let x', y', z' be their counterparts in (X', d'_X). We note that the inequality is always true, when $d(x, y) = 1$. Hence we may assume that $d(x, y) > 1$ and so it was rounded up to the closest power of $(1+\varepsilon)$. We get

$$d(x, y) \leq (1+\varepsilon) d'_X(x', y') \leq (1+\varepsilon)(d'_X(x', z') + d'_X(z', y')) \leq (1+\varepsilon)(d(x, z) + d(z, y)) \ . \qquad \blacksquare$$

We will therefore assume throughout the entire paper that (Y, d) is an ε-near metric space. In fact, our formula for the cost of the k-NN graph will work for every graph with outdegree k and edges weights that are powers of $(1 + \varepsilon)$.