Optimal Time Bounds for Approximate Clustering*

Ramgopal R. Mettu C. Greg Plaxton

Abstract

We give randomized constant-factor approximation algorithms for the k-median problem and an intimately related clustering problem. The input to each of these problems is a metric space with n weighted points and an integer k, $0 < k \leq n$. For any such input, let R_d denote the ratio between the maximum and minimum nonzero interpoint distances, and let R_w denote the ratio between the maximum and minimum nonzero point weights. We analyze the running time of our algorithms in terms of the parameters n, k, R_d, and R_w. We prove that over a wide range of parameter settings, the complexity of both problems is $\Theta(nk)$.

*Department of Computer Science, University of Texas at Austin, Austin, TX 78712. This research was supported by NSF Grant CCR–9821053. Email: {ramgopal, plaxton}@cs.utexas.edu. The second author is presently on leave at Akamai Techologies, Inc., Cambridge, MA 02139.

1 Introduction

Given a set of points and pairwise distances between the points, the goal of *clustering* problems is to partition the points into a number of sets such that points in each set are "close" with respect to some objective function. Clustering algorithms are widely used to organize large data sets in areas such as data mining and information retrieval. For example, we may wish to partition a set of web logs to infer certain usage patterns, or divide a corpus of documents into a small number of related groups. Given a set of points and associated interpoint distances, let the ***median*** of the set be the point in the set that minimizes the sum of distances to all other points in the set. (Remark: The median is essentially the discrete analog of the centroid, and is also called the *medoid* [9].) The clustering problem we consider asks us to partition n weighted points into k sets such that the sum, over all sets, of the weight of a point times the distance to the median of its set is minimized. Approaches to this type of clustering problem, such as the k-means heuristic, have been well-studied [4, 9]. We refer to this problem as the ***clustering variant*** of the classic ***k-median problem***; the k-median problem asks us to mark k of the points such that the sum over all points x of the weight of x times the distance from x to the nearest marked point is minimized. It is straightforward to see that we can convert a solution to the k-median problem into a solution for its clustering variant in $O(nk)$ time; thus we focus on the k-median problem when developing our upper bounds. We also restrict our attention to the ***metric*** version of the problems throughout this paper; the given distance matrix defines a metric space over the set of input points, that is, the distances are nonnegative, symmetric, satisfy the triangle inequality, and the distance between points x and y is zero if and only if $x = y$. (Remark: For the sake of brevity we write "k-median problem" to mean "metric k-median problem" throughout the remainder of the paper.)

Since problem instances in the application areas mentioned above tend to be large, we are motivated to ask how input characteristics such as the point weights and interpoint distances affect the complexity of the k-median problem and its clustering variant. Weighted points are useful in a number of applications; for example, we may wish to prioritize the objects in the input. We ask the following natural question: Does allowing inputs with arbitrary point weights incur a substantial time penalty? We note that even for moderate weights, say $O(n^2)$, the naive approach of viewing a weighted point as a collection of unit-weight points increases the input size dramatically. For certain applications, the interpoint distances may lie in a relatively small range. Thus we are motivated to ask: Does constraining distances to a small range admit substantially faster algorithms? We resolve both of the above questions for a wide range of input parameters by establishing a time bound of $\Theta(nk)$ for the k-median problem and its clustering variant. Thus, we show that in many cases, having large point weights does not incur a substantial time penalty, and, that we cannot hope to develop substantially faster algorithms even when the interpoint distances lie in a small range.

Before stating our results we introduce some useful terminology that we use throughout this paper. Let U denote the set of all points in a given instance of the k-median problem; we assume that U is nonempty. A ***configuration*** is a nonempty subset of U. An ***m-configuration*** is a configuration of size at most m. For any points x and y in U, let $w(x)$ denote the nonnegative weight of x, and let $d(x, y)$ denote the distance between x and y. The ***cost*** of any configuration X, denoted $cost\,(X)$, is defined as $\sum_{x \in U} d(x, X) \cdot w(x)$. We denote the minimum cost of any m-configuration by OPT_m. For brevity, we say that an m-configuration with cost at most $a \cdot OPT_k$ is an ***(m, a)-configuration***.

A k-median algorithm is ***(m, a)-approximate*** if it produces an (m, a)-configuration. A k-median algorithm is ***a-approximate*** if it is (k, a)-approximate. Let R_d denote the ratio of the diameter of U (i.e., the maximum distance between any pair of points in U) to the minimum distance between any pair of distinct points in U. Let R_w denote the ratio of the maximum weight of any point in U to the minimum nonzero weight of any point in U. (Remark: We can assume without loss of generality that at least one point in U has nonzero weight since the problem is trivial otherwise.) Let $r_d = 1 + \lfloor \log R_d \rfloor$ and $r_w = 1 + \lfloor \log R_w \rfloor$.

Our main result is a randomized $O(1)$-approximate k-median algorithm that runs in $O(nk)$ time subject to the constraints $k = \Omega(\log n)$, $kr_w^2 = O(n)$, and $r_d r_w \log(\frac{n}{kr_w}) = O(n)$. The algorithm succeeds *with high probability*, that is, for any positive constant ξ, we can adjust constant factors in the definition of the algorithm to achieve a failure probability less than $n^{-\xi}$. We establish a matching $\Omega(nk)$ lower bound on the running time of any randomized $O(1)$-approximate k-median algorithm with a nonnegligible success probability (e.g., at least $\frac{1}{100}$), subject to the requirement that R_d exceeds n/k by a sufficiently large constant factor relative to the desired approximation ratio. To obtain tight bounds for the clustering variant, we also prove an $\Omega(nk)$ time lower bound for any $O(1)$-approximate algorithm, but we only require that R_d be a sufficiently large constant relative to the desired approximation ratio. Additionally, our lower bounds assume only that $R_w = O(1)$.

Our main technical result is a successive sampling technique that we use in all of our algorithms. The basic idea behind the technique is to take a random sample of the points, set aside a constant fraction of the n points that are "close" to the sample, and recurse on the remaining points. We show that this technique rapidly produces a configuration whose cost is within a constant factor of optimal. Specifically, for the case of uniform weights, our successive sampling algorithm yields a $(k \log (n/k), O(1))$-configuration with high probability in $O(n \max\{k, \log n\})$ time.

In addition to this sampling result, our algorithms rely on an extraction technique due to Guha *et al.* [5] that uses a black box $O(1)$-approximate k-median algorithm to compute a $(k, O(1))$-configuration from any $(m, O(1))$-assignment. The black box algorithm that we use is the linear-time deterministic online median algorithm of Mettu and Plaxton [10].

In developing our randomized algorithm for the k-median problem we first consider the special case of uniform weights, that is, where $R_w = r_w = 1$. For this special case we provide a randomized algorithm running in $O(n \max\{k, \log n\})$ time subject to the constraint $r_d \log \frac{n}{k} = O(n)$. The uniform-weights algorithm is based directly on the two building blocks discussed above: We apply the successive sampling algorithm to obtain $(k \log (n/k), O(1))$-configuration and then use the extraction technique to obtain a $(k, O(1))$-configuration. We then use this algorithm to develop a k-median algorithm for the case of arbitrary weights. Our algorithm begins by partitioning the n points into r_w power-of-2 weight classes and applying the uniform-weights algorithm within each weight class (i.e., we ignore the differences between weights belonging to the same weight class, which are less than a factor of 2 apart). The union of the r_w k-configurations thus obtained is an $(r_w k, O(1))$-configuration. We then make use of our extraction technique to obtain a $(k, O(1))$-configuration from this $(r_w k, O(1))$-configuration.

1.1 Problem Definitions

Without loss of generality, throughout this paper we consider a fixed set of n points, U, with an associated distance function $d : U \times U \rightarrow \mathbb{R}$ and an associated nonnegative demand function $w : U \rightarrow \mathbb{R}$. We assume that d is a metric, that is, d is nonnegative, symmetric, satisfies the triangle inequality, and $d(x, y) = 0$ iff $x = y$. For a configuration X and a set of points Y, we let $cost\,(X, Y) = \sum_{x \in Y} d(x, X) \cdot w(x)$. For any set of points X, we let $w(X)$ denote $\Sigma_{x \in X} w(x)$.

We define an ***assignment*** as a function from U to U. For any assignment τ, we let $\tau(U)$ denote the set $\{\tau(x) \mid x \in U\}$. We refer to an assignment τ with $|\tau(U)| \leq m$ as a m-***assignment***. Given an assignment τ, we define the cost of τ, denoted $c\,(\tau)$, as $\sum_{x \in U} d(x, \tau(x)) \cdot w(x)$. It is straighforward to see that for any assignment τ, $cost\,(\tau(U)) \leq c\,(\tau)$. For brevity, we say that an assignment τ with $|\tau(U)| \leq m$ and cost at most $a \cdot OPT_k$ is an (m, a)-***assignment***. For an assignment τ and a set of points X, we let $c\,(\tau, X) = \sum_{x \in X} d(x, \tau(x)) \cdot w(x)$.

The input to the k-median problem is (U, d, w) and an integer k, $0 < k \leq n$. Since our goal is to obtain a $(k, O(1))$-configuration, we can assume without loss of generality that all input points have nonzero weight. We note that for all m, $0 < m \leq n$, removing zero weight points from an m-configuration at most doubles its cost. To see this, consider an m-configuration X; we can obtain an m-configuration X' by replacing each zero weight point with its closest nonzero weight point. Using the triangle inequality, it is straightforward to see that $cost\,(X') \leq 2cost\,(X)$. This argument can be used to show that any minimum-cost set of size m contained in the set of nonzero weight input points has cost at most twice OPT_m. We also assume that the input weights are scaled such that the smallest weight is 1; thus the input weights lie in the range $[1, R_w]$. For output, the k-median problem requires us to compute a minimum-cost k-configuration. The ***uniform weights*** k-median problem is the special case in which $w(x)$ is a fixed real for all points x. The output is also a minimum-cost k-configuration.

1.2 Comparison to Previous Work

The first $O(1)$-approximate k-median algorithm was given by Charikar *et al.* [3]. Subsequently, there have been several improvements to the approximation ratio (see, e.g., [2] for results and citations). In this section, we focus on the results that are most relevant to the present paper; we compare our results with other recent sublinear-time algorithms for the k-median problem. The first of these results is due to Indyk, who gives a randomized $(O(k), O(1))$-approximate algorithm for the uniform weights k-median problem [6]. Indyk's algorithm combines random sampling of the input points with a black-box $(\alpha k, \beta)$-approximate k-median algorithm to achieve a $((1+\delta)(6+3\alpha)k, 2\beta)$-approximate algorithm, where δ is the desired success probability. Given an $\tilde{O}(n^2)$-time[1] black-box k-median algorithm, Indyk's algorithm runs in $\tilde{O}(nk/\delta^2)$ time. (The polylogarithmic factor in the running time is $\Omega(\log^2 k)$.) Indyk's algorithm takes $O(\sqrt{nk \log k})$ sample points and then runs the black-box k-median algorithm on those points to obtain a configuration X. The black-box algorithm is then run again on a set of points that are distant from points in X to produce another configuration Y. The final output is the union of X and Y, which is shown to be an $(O(k), O(1))$-configuration.

Thorup [13] gives a randomized $O(1)$-approximate algorithms for the k-median, k-center, and

[1]The $\tilde{O}$-notation omits polylogarithmic factors in n and k.

facility location problems in a graph. For these problems, we are not given a metric distance function but rather a graph on the input points with m positively weighted edges from which the distances must be computed; all of the algorithms in [13] run in $\tilde{O}(m)$ time. Thorup [13] also gives an $\tilde{O}(nk)$ time randomized constant-factor approximation algorithm for the k-median problem that we consider. (The polylogarithmic factor in the running time is $\Omega(\log^4 n)$.) As part of this k-median algorithm, Thorup gives a successive sampling technique that also consists of a series of sampling steps but produces an $(O((k \log^2 n)/\varepsilon),\ 2 + \varepsilon)$-configuration for any positive real ε with $0 < \varepsilon < 0.4$ with probability $1/2$.

Our successive sampling technique is similar in spirit to both of the above algorithms, but we take a total of $O(\log (n/k))$ samples, each of size $O(k)$, and construct an $(O(k \log (n/k)), O(1))$-assignment from the union of the samples. Overall, our sample size is much smaller than in Indyk's algorithm ($O(k \log (n/k))$ points versus $O(\sqrt{nk \log k})$ points) and smaller than the sample size in Thorup's algorithm by a logarithmic factor. However, our algorithm produces an $(O(k \log (n/k)), O(1))$-assignment whereas Indyk's algorithm produces an $(O(k), O(1))$-configuration. Additionally, the algorithms of Indyk and Thorup both succeed with a constant probability, while our sampling algorithm is guaranteed to succeed with high probability.

Guha *et al.* [5] give k-median algorithms for the data stream model of computation. Under the data stream model of computation, input data is processed sequentially, and the performance of an algorithm is measured by how many passes it makes over the input and by its space requirements. Guha *et al.* [5] give a single-pass $O(1)$-approximate algorithm for the k-median problem that runs in $\tilde{O}(nk)$ time and requires $O(n^{\varepsilon})$ space for a positive constant ε. (Their algorithm uses Indyk's k-median algorithm as a black box and hence the polylogarithmic factor in the running time is also $\Omega(\log^2 k)$.)

Mishra *et al.* [11] show that in order to find a $(k, O(1))$-configuration, it is enough to take a sufficiently large sample of the input points and use it as input to a black-box $O(1)$-approximate k-median algorithm. To compute a $(k, O(1))$-configuration with an arbitrarily high constant probability, the required sample size is $\tilde{O}(R_d^2 k)$. The running time of this technique depends on the black-box algorithm used. In the general case, the size of the sample may be as large as n, but depending on the diameter of the input metric space, this technique can yield running times of $o(n^2)$ (e.g., if the diameter is $o(n^2/k)$).

As noted earlier, we also make use of a technique due to Guha *et al.* [5] that takes an $(m, O(1))$-configuration and extracts a $(k, O(1))$-configuration; they use this technique in isolation in a divide-and-conquer fashion to develop their k-median algorithms. We view the extraction technique as a postprocessing step that yields a $(k, O(1))$-approximate k-median algorithm given an $(m, O(1))$-approximate k-median algorithm. In our algorithms, we take advantage of the fact that this postprocessing step can be performed rapidly. For example, if $m = O(k)$ and the black-box algorithm requires $O(n^2)$ time, the time required for postprocessing is just $O(k^2)$.

Guha *et al.* [5] establish a lower bound of $\Omega(nk)$ for deterministic $O(1)$-approximate k-median algorithms. We note that they work with a slightly different definition of the k-median problem in which the distance between two distinct points is allowed to be 0. We adopt the view that points at distance zero are represented by a single point with commensurately higher weight; this view avoids having an infinite value for R_d. For the proof of the lower bound, Guha *et al.* [5] construct a problem instance for which optimal solution has cost 0 and reduce the problem to a graph k-partitioning problem [7]. The intuition is that any algorithm producing a k-configuration with

nonzero cost is not $O(1)$-approximate. Although their problem instance contains distinct points at distance 0 (i.e., an infinite R_d), with a slight modification their proof only requires that R_d exceed n by a sufficiently large constant factor relative to the desired approximation ratio. Intuitively, with such a large setting of R_d, a deterministic k-median algorithm taking $o(nk)$ time and making just one "mistake" has fails to achieve the desired approximation ratio. Our lower bounds are stronger in the sense that we focus on constructing problem instances that have small values of R_d, and then show that any randomized k-median algorithms running in $o(nk)$ time is likely to make many "mistakes" on these instances.

1.3 Outline

The rest of this paper is organized as follows. In Section 2, we present and analyze our successive sampling algorithm. In Section 3, we make use of our sampling algorithm, in conjunction with an extraction result, to develop an $O(1)$-approximate uniform weights k-median algorithm. Then, in Section 4, we use the uniform weights algorithm as a subroutine to develop an $O(1)$-approximate k-median algorithm for the case of arbitrary weights. We present our lower bounds for the k-median problem and its clustering variant in Appendix A.

2 Approximate Clustering via Successive Sampling

Our first result is a successive sampling algorithm that constructs an assignment that has cost $O(OPT_k)$ with high probability. We make use of this algorithm to develop our uniform weights k-median algorithm. (Remark: We assume arbitrary weights for our proofs since the arguments generalize easily to the weighted case; furthermore, the weighted result may be of independent interest.) Informally speaking, the algorithm works in sampling steps. In each step we take a small sample of the points, set aside a constant fraction the weight whose constituent points are each close to the sample, and recurse on the remaining points. Since we eliminate a constant fraction of the weight at each sampling step, the number of samples taken is logarithmic in the total weight. We are able to show that using the samples taken, it is possible to construct an assignment whose cost is within a constant factor of optimal with high probability. For the uniform weights k-median problem, our sampling algorithm runs in $O(n \max\{k, \log n\})$ time. (We give a k-median algorithm for the case of arbitrary weights in Section 4.)

Throughout this section, we use the symbols α, β, and k' to denote real numbers appearing in the definition and analysis of our successive sampling algorithm. The value of α and k' should be chosen to ensure that the failure probability of the algorithm meets the desired threshold. (See the paragraph preceding Lemma 2.3 for discussion of the choice of α and k'.) The asymptotic bounds established in this paper are valid for any choice of β such that $0 < \beta < 1$.

We also make use of the following definitions:

- A ***ball*** A is a pair (x, r), where the ***center*** x of A belongs to U, and the ***radius*** r of A is a nonnegative real.
- Given a ball $A = (x, r)$, we let $Points(A)$ denote the set $\{y \in U \mid d(x, y) \leq r\}$. However, for the sake of brevity, we tend to write A instead of $Points(A)$. For example, we write "$x \in A$" and "$A \cup B$" instead of "$x \in Points(A)$" and "$Points(A) \cup Points(B)$", respectively.

- For any set X and nonnegative real r, we define $Balls(X, r)$ as the set $\cup_{x \in X} A_x$ where $A_x = (x, r)$.

2.1 Algorithm

The following algorithm takes as input an instance of the k-median problem and produces an assignment σ such that with high probability, $c(\sigma) = O(cost(X))$ for any k-configuration X.

Let $U_0 = U$, and let $S_0 = \emptyset$. While $|U_i| > \alpha k'$:

- Construct a set of points S_i by sampling (with replacement) $\lfloor \alpha k' \rfloor$ times from U_i, where at each sampling step the probability of selecting a given point is proportional to its weight.
- For each point in U_i, compute the distance to the nearest point in S_i.
- Using linear-time selection on the distances computed in the previous step, compute the smallest real ν_i such that $w(Balls(S_i, \nu_i)) \geq \beta w(U_i)$. Let $C_i = Balls(S_i, \nu_i)$.
- For each x in C_i, choose a point y in S_i such that $d(x, y) \leq \nu_i$ and let $\sigma(x) = y$.
- Let $U_{i+1} = U_i \setminus C_i$.

Note that the loop terminates since $w(U_i) < w(U_{i+1})$ for all $i \geq 0$. Let t be the total number of iterations of the loop. Let $C_t = S_t = U_t$. By the choice of C_i in each iteration and the loop termination condition, t is $O(\log(w(U)/k'))$. For the uniform demands k-median problem, t is simply $O(\log(n/k'))$. From the first step it follows that $|\sigma(U)|$ is $O(tk')$.

The first step of the algorithm can be performed in $O(nk')$ time over all iterations. In each iteration the second and third steps can be performed in time $O(|U_i|\, k')$ by using a (weighted) linear time selection algorithm. For the uniform demands k-median problem, this computation requires $O(nk')$ time over all iterations. The running times of the third and fourth steps are negligible. Thus, for the uniform demands k-median problem, the total running time of the above algorithm is $O(nk')$.

2.2 Approximation Bound

The goal of this section is to establish Theorem 1. The proof of the theorem makes use of Lemmas 2.3, 2.5, and 2.11, which are established below. We remark that Theorem 1 is used in Sections 3 and 4.

Theorem 1 *With high probability, $c(\sigma) = O(cost(X))$ for any k-configuration X.*

Proof: The claim of Lemma 2.3 holds with high probability if we set $k' = \max\{k, \log n\}$ and α and β appropriately large. The theorem then follows from Lemmas 2.3, 2.5, and 2.11. ∎

The proof of Lemma 2.3 below relies on bounding the failure probability of a certain family of random experiments. We begin by bounding the failure probability of a simpler family of random experiments related to the well-known coupon collector problem. For any positive integer m and any nonnegative reals a and b, let us define $f(m, a, b)$ as the probability that more than am bins remain empty after $\lceil b \rceil$ balls are thrown at random (uniformly and independently) into m bins.

Techniques for analyzing the coupon collector problem (see. e.g., [12]) can be used to obtain sharp estimates on $f(m, a, b)$. However, the following simple upper bound is sufficient for our purposes.

Lemma 2.1 *For any positive real ε, there exists a positive real λ such that for all positive integers m and any real $b \geq m$, we have $f(m, \varepsilon, \lambda b) \leq e^{-b}$.*

Proof: Note that a crude upper bound on $f(m, \varepsilon, \lambda b)$ is given by the probability of obtaining at most $(1 - \varepsilon)m$ successes in $\lceil \lambda b \rceil$ Bernoulli trials, each of which has success probability ε. The claim then follows by choosing λ sufficiently large and applying a standard Chernoff bound. (We have in mind the following tail bound: If X is a random variable drawn from a Bernoulli distribution with n trials and each trial has success probability p, then for all δ such that $0 \leq \delta \leq 1$, $\Pr\{X \leq (1 - \delta)np\} \leq e^{-\delta^2 np/2}$; see [1, Appendix A] for a derivation.) ∎

We now develop a weighted generalization of the preceding lemma. For any positive integer m, nonnegative reals a and b, and m-vector $v = (r_0, \ldots, r_{m-1})$ of nonnegative reals r_i, we define define $g(m, a, b, v)$ as follows. Consider a set of m bins numbered from 0 to $m - 1$ where bin i has associated weight r_i. Let R denote the total weight of the bins. Assume that each of $\lceil b \rceil$ balls is thrown independently at random into one of the m bins, where bin i is chosen with probability r_i/R, $0 \leq i < m$. We define $g(m, a, b, v)$ as the probability that the total weight of the empty bins after all of the balls have been thrown is more than aR.

Lemma 2.2 *For any positive real ε there exists a positive real λ such that for all positive integers m and any real $b \geq m$, we have $g(m, \varepsilon, \lambda b, v) \leq e^{-b}$ for all m-vectors v of nonnegative reals.*

Proof: Fix ε, b, m, and v. We will use Lemma 2.1 to deduce the existence of a suitable choice of λ that depends only on ε. Our strategy for reducing the claim to its unweighted counterpart will be to partition almost all of the weight associated with the m weighted bins into $\Theta(m)$ "sub-bins" of equal weight. Specifically, we let s denote $\frac{\varepsilon R}{2m}$ and for each i we partition the weight r_i associated with bin i into $\left\lfloor \frac{r_i}{s} \right\rfloor$ *complete* sub-bins of weight s and one *incomplete* sub-bin of weight less than s. Furthermore, when a ball is thrown into a particular bin, we imagine that the throw is further refined to a particular sub-bin of that bin, where the probability that a particular sub-bin is chosen is proportional to its weight.

Note that the total weight of the incomplete sub-bins is less than $\varepsilon R/2$. Furthermore, we can assume without loss of generality that $\varepsilon \leq 1$, since the claim holds vacuously for $\varepsilon > 1$. It follows that less than half of the total weight R lies in incomplete sub-bins. Thus, by a standard Chernoff bound argument, for any positive real λ' we can choose λ sufficiently large to ensure that the following claim holds with probability of failure at most $e^{-b}/2$ (i.e., half the desired failure threshold appearing in the statement of the lemma): At least $\lambda' b$ of the $\lceil \lambda b \rceil$ balls are thrown into complete sub-bins.

Let m' denote the number of complete sub-bins. Since at least half of the total weight R belongs to complete sub-bins, we have $m/\varepsilon \leq m' \leq 2m/\varepsilon$. Accordingly, by a suitable application of Lemma 2.1, we can establish the existence of a positive real λ' (depending only on ε) such that, after at least $\lambda' b$ balls have landed in complete sub-bins, the probability that the number of empty complete sub-bins exceeds $\varepsilon m'/2$ is at most $e^{-b}/2$.

From the claims of the two preceding paragraphs, we can conclude that there exists a λ (depending only on ε) such that the following statement holds with probability of failure at most e^{-b}:

The number of empty complete sub-bins is at most $\varepsilon m'/2$. Note that the total weight of the complete sub-bins is at most $s \cdot \frac{\varepsilon}{2} \cdot \frac{2t}{\varepsilon} = \varepsilon R/2$. As argued earlier, the total weight of the incomplete sub-bins is also at most $\varepsilon R/2$. Thus, there exists a positive real λ such that after $\lceil \lambda b \rceil$ ball tosses, the probability that the total weight of the empty bins is more than εR is at most e^{-b}. ∎

For the remainder of this section, we fix a positive real γ such that $\beta < \gamma < 1$. We also let μ_i denote the minimum real such that there exists a k-configuration X with the property that $w(Balls(X, \mu_i)) \geq \gamma w(U_i)$. Lemma 2.3 below establishes the main probabilistic claim used in our analysis of the algorithm of Section 2.1. We note that the lemma holds with high probability by taking $k' = \max\{k, \lceil \log n \rceil\}$ and α and β appropriately large.

Lemma 2.3 *For any positive real ξ, there exists a sufficiently large choice of α such that $\nu_i \leq 2\mu_i$ for all i, $0 \leq i \leq t$, with probability of failure at most $e^{-\xi k'}$.*

Proof: Fix i and let X denote a k-configuration such that $w(Balls(X, \mu_i)) \geq \gamma w(U_i)$. Let us define each point y in U_i to be *good* if it belongs to $Balls(X, \mu_i)$, and *bad* otherwise. Let G denote the set of good points. We associate each good point y with its closest point in X, breaking ties arbitrarily. For each point x in X, let A_x denote the set of good points associated with x; note that the sets A_x form a partition of G. Recall that S_i denotes the ith set of sample points chosen by the algorithm. For any x in X, we say that S_i *covers* A_x iff $S_i \cap A_x$ is nonempty. For any point y, we say that S_i *covers* y iff there exists an x in X such that y belongs to A_x and S_i covers A_x. Let G' denote the set of points covered by S_i; note that $G' \subseteq G$.

We will establish the lemma by proving the following claim: For any positive reals ε and ξ, there exists a sufficiently large choice of α such that $w(G') \geq (1 - \varepsilon)w(G)$ with probability of failure at most $e^{-\xi k'}$. This claim then implies the lemma because β (the factor appearing in the definition of ν_i) is less than γ (the factor appearing in the definition of μ_i) and for all points y covered by S_i, $d(y, S_i) \leq 2\mu_i$.

It remains to prove the preceding claim. First, note that the definition of μ_i implies that at least a γ fraction of the total weight is associated with good points. Thus, a standard Chernoff bound argument implies that for any positive reals λ and ξ, there exists a sufficiently large choice of α such that at least $\lambda k'$ of the $\lfloor \alpha k' \rfloor$ samples associated with the construction of S_i are good with probability of failure at most $e^{-\xi k'}/2$.

To ensure that $w(G')$ is at least $(1 - \varepsilon)w(G)$ with failure probability $e^{-\xi k'}/2$, we can apply Lemma 2.2 by viewing each sample associated with a good point in S_i as a ball toss and each set A_x as a bin with weight $w(A_x)$. The claim then follows. ∎

Lemma 2.4 *For all i such that $0 \leq i \leq t$, $c(\sigma, C_i) \leq \nu_i w(C_i)$.*

Proof: Observe that

$$\begin{aligned} c(\sigma, C_i) &= \sum_{x \in C_i} d(x, \sigma(x)) \cdot w(x) \\ &\leq \sum_{x \in C_i} \nu_i \cdot w(x) \\ &= \nu_i w(C_i), \end{aligned}$$

where the second step follows from the definition of C_i and the construction of $\sigma(x)$. ∎

Lemma 2.5

$$c(\sigma) \leq \sum_{0 \leq i \leq t} \nu_i w(C_i).$$

Proof: Observe that

$$\begin{aligned} c(\sigma) &= \sum_{0 \leq i \leq t} c(\sigma, C_i) \\ &\leq \sum_{0 \leq i \leq t} \nu_i w(C_i). \end{aligned}$$

The first step follows since the sets C_i, $0 \leq i \leq t$, form a partition of U. The second step follows from Lemma 2.4. ∎

Throughout the remainder of this section we fix an arbitrary k-configuration X. For all i such that $0 \leq i \leq t$, we let F_i denote the set $\{x \in U_i \mid d(x, X) \geq \mu_i\}$, and for any integer $m > 0$, we let F_i^m denote $F_i \setminus (\cup_{j>0} F_{i+jm})$.

Lemma 2.6 *Let i, j, and m be integers such that $0 \leq i \leq t$, $0 \leq j \leq t$, $m > 0$, and $(i - j) \bmod m = 0$. Then $F_i^m \cap F_j^m = \emptyset$.*

Proof: Without loss of generality, assume that $i < j$. Then, by definition, $F_i^m = F_i \setminus (\cup_{s>0} F_{i+sm})$. Since $F_j^m \subseteq F_j$ and $(i - j) \bmod m = 0$, it follows that F_i^m and F_j^m do not intersect. ∎

Lemma 2.7 *Let i be an integer such that $0 \leq i \leq t$ and let Y be a subset of F_i. Then $w(F_i) \geq (1 - \gamma) w(U_i)$ and $cost(X, Y) \geq \mu_i w(Y)$.*

Proof: First, note that by the definition of μ_i, $w(F_i)$ is at least $(1 - \gamma) w(U_i)$. By the definition of F_i, $d(y, X) \geq \mu_i$ for any y in F_i. Thus $cost(X, Y) = \sum_{y \in Y} d(y, X) \cdot w(y) \geq \mu_i w(Y)$. ∎

Lemma 2.8 *For all i, j, and m such that $0 \leq i \leq t$, $0 \leq j \leq t$, and $m > 0$,*

$$cost\left(X, \cup_{(i-j) \bmod m = 0} F_i^m\right) \geq \sum_{(i-j) \bmod m = 0} \mu_i w(F_i^m).$$

Proof: By Lemma 2.6, for all i, j, and m such that $0 \leq i \leq t$,$0 \leq j \leq t$, and $m > 0$,

$$cost\left(X, \cup_{(i-j) \bmod m = 0} F_i^m\right) = \sum_{(i-j) \bmod m = 0} cost(X, F_i^m).$$

By Lemma 2.7, $cost(X, F_i^m) \geq \mu_i w(F_i^m)$, and the claim follows. ∎

For the remainder of the section, let $r = \lceil \log_{(1-\beta)}((1 - \gamma)/3) \rceil$.

Lemma 2.9 *For all i such that $0 \leq i \leq t$, $w(F_{i+r}) \leq \frac{1}{3} w(F_i)$.*

Proof: Note that $w(F_{i+r}) \leq w(U_{i+r}) \leq (1 - \beta)^r w(U_i) \leq \frac{(1-\beta)^r}{1-\gamma} w(F_i)$, where the last step follows from Lemma 2.7. The claim then follows by the definition of r. ∎

Lemma 2.10 *For all i such that $0 \leq i \leq t$, $w(F_i^r) \geq \frac{w(F_i)}{2}$.*

Proof: Observe that

$$\begin{aligned} w(F_i^r) &= w(F_i \setminus \cup_{j>0} F_{i+jr}) \\ &\geq w(F_i) - \sum_{j>0} \frac{w(F_i)}{3^j} \\ &\geq \frac{w(F_i)}{2}, \end{aligned}$$

where the second step follows from Lemma 2.9. ∎

Lemma 2.11 *For any k-configuration X,*

$$\textit{cost}\,(X) \geq \frac{1-\gamma}{2r} \sum_{0 \leq i \leq t} \mu_i w(C_i).$$

Proof: Let $j = \arg\max_{0 \leq j < r} \{\sum_{(i-j) \bmod r = 0} w(F_i^r)\}$ and fix a k-configuration X. Then $cost\,(X)$ is at least

$$\begin{aligned} \textit{cost}\left(X, \cup_{(i-j) \bmod r = 0} F_i^r\right) &\geq \sum_{(i-j) \bmod r = 0} \mu_i w(F_i^r) \\ &\geq \frac{1}{r} \sum_{0 \leq i \leq t} \mu_i w(F_i^r) \\ &\geq \frac{1}{2r} \sum_{0 \leq i \leq t} \mu_i w(F_i) \\ &\geq \frac{1-\gamma}{2r} \sum_{0 \leq i \leq t} \mu_i w(U_i) \\ &\geq \frac{1-\gamma}{2r} \sum_{0 \leq i \leq t} \mu_i w(C_i), \end{aligned}$$

where the first step follows from Lemma 2.8, the second step follows from averaging and the choice of j, the third step follows from Lemma 2.10, the fourth step follows from Lemma 2.7, and the last step follows since $C_i \subseteq U$. ∎

3 An Efficient Algorithm for the Case of Uniform Weights

Theorem 2.4 of Guha *et al.* [5] implies that for any given $(m, O(1))$-configuration X, we can compute a $(k, O(1))$-configuration by simply running an $O(1)$-approximate k-median algorithm on the modified problem instance obtained by redistributing the point weights as follows: The weight of any given point x is moved to a point y in X such that $d(x, y) = d(x, X)$. (This result follows from the analysis of algorithm Small-Space of Guha *et al.* [5], since it corresponds to the case in which $\ell = 1$ in step 1 and the $(m, O(1))$-configuration is the output in step 2. It should be

remarked that although algorithm Small-Space is presented in a manner that assumes the output of step 2 to be ℓ $(O(k), O(1))$-configurations, the analysis of Small-Space given in [5] is easily seen to hold for the more general case in which the output of step 2 is a collection of ℓ $(m, O(1))$-configurations.)

By Theorem 1, the output of our sampling algorithm is an $(m, O(1))$-assignment with high probability, where $m = O(\max\{k, \log n\} \log(n/k))$ in the case of uniform weights. Thus $\sigma(U)$ is an $(m, O(1))$-configuration with high probability, and we can directly apply the Guha *et al.* [5] technique to extract a $(k, O(1))$-configuration from $\sigma(U)$. The only trouble with this approach is that a direct application of their technique expends $\Theta(mn) = \omega(nk)$ time in computing the closest point in $\sigma(U)$ to each point in U. Fortunately, it is straightforward to verify that the following variation of the Guha *et al.* [5] technique is also valid. Given an $(m, O(1))$-assignment, we can redistribute the weight of each point x to $\sigma(x)$ and then run an $O(1)$-approximate k-median algorithm on the modified problem instance. (Remark: In [5, Section 2], the point i' is defined to be the median closest to the point i. For the purposes of our variation, the point i' should instead be defined as $\sigma(i)$.)

We now analyze the running time of the above algorithm. To compute the assignment σ, we use our sampling algorithm with the parameter k' set to $O(\max\{k, \log n\})$. The time required to compute σ is then $O(n \max\{k, \log n\})$. We note that the required weight function can be computed during the execution of the sampling algorithm without increasing its running time. The deterministic online median algorithm of Mettu and Plaxton [10] can then be used to complete the extraction step in $O(|\sigma(U)|^2 + |\sigma(U)|\, r_d)$ time. The total time taken by the algorithm is therefore

$$\begin{aligned} O(nk' + |\sigma(U)|^2 + |\sigma(U)|\, r_d) &= O(nk' + k'^2 \log^2(n/k) + r_d k' \log(n/k)) \\ &= O(nk' + r_d k' \log(n/k)), \end{aligned}$$

where the first step follows from the analysis of our sampling algorithm for the case of uniform weights. By the choice of k', the overall running time is $O((n + r_d \log(n/k)) \max\{k, \log n\})$. Note that if $k = \Omega(\log n)$ and $kr_w^2 = O(n)$, this time bound simplifies to $O(nk)$.

4 An Efficient Algorithm for the Case of Arbitrary Weights

The algorithm developed in Sections 2 and 3 is $O(1)$-approximate for the k-median problem with arbitrary weights. However, the time bound established for the case of uniform weights does not apply to the case of arbitrary weights because the running time of the successive sampling procedure is slightly higher in the latter case. (More precisely, the running time of the sampling algorithm of Section 2 is $O(nk' \log \frac{w(U)}{k'})$ for the case of arbitrary weights.) In this section, we use the uniform-weight algorithm developed in Sections 2 and 3 to develop a k-median algorithm for the case of arbitrary weights that is time optimal for a certain range of k.

We first give an informal description of the algorithm, which consists of three main steps. First, we partition the input points according to weight into r_w sets. Next, we run our uniform weights k-median algorithm on each of the resulting sets, and show that the union of the resulting outputs is an $(O(kr_w), O(1))$-configuration. We then obtain a $(k, O(1))$-configuration by creating a problem instance from the $(O(kr_w), O(1))$-configuration computed in the previous step and then feeding this problem instance as input to an $O(1)$-approximate k-median algorithm.

We now give a more precise description of our k-median algorithm. Let $\mathcal{A}$ be the uniform weights k-median algorithm of Sections 2 and 3, and let $\mathcal{B}$ be an $O(1)$-approximate k-median algorithm.

- Compute sets B_i for $0 \leq i < r_w$ such that for all $x \in B_i$, $2^i \leq w(x) \leq 2^{i+1}$.
- For $i = 0, 1 \ldots r_w - 1$: Run $\mathcal{A}$ with B_i as the set of input points, d as the distance function, 2^{i+1} as the fixed weight, and the parameter $k' = \max\{k, \lceil \log n \rceil\}$; let Z_i denote the output. Let ϕ_i denote the assignment induced by Z_i, that is, $\phi_i(x) = y$ iff y is in Z_i and $d(x, Z_i) = d(x, y)$. For a point x, if $x \in Z_i$, let $\tilde{w}_{\phi_i}(x) = w(\phi_i^{-1}(x))$, otherwise let $\tilde{w}_{\phi_i}(x) = 0$.
- Let ϕ be the assignment corresponding to the union of the assignments ϕ_i defined in the previous step, and let $\tilde{w}_\phi$ denote the weight function corresponding to the union of the weight functions $\tilde{w}_{\phi_i}$. Run $\mathcal{B}$ with $\phi(U)$ as the set of input points, d as the distance function, and $\tilde{w}_\phi$ as the weight function. Output the resulting k-configuration.

Note that in the second step, k' is defined in terms of n (i.e., $|U|$) and not $|B_i|$. Thus, the argument of the proof of Theorem 1 implies that $\mathcal{A}$ succeeds with high probability in terms of n. Assuming that r_w is polynomially bounded in n, with high probability we have that every invocation of $\mathcal{A}$ is successful.

We now observe that the above algorithm corresponds to the special case of algorithm Small-Space of [5] in which the parameter ℓ is set to r_w, the uniform weights algorithm of Section 3 is used in step 2 of Small-Space, and the online median algorithm of [10] is used in step 4 of Small-Space. Thus, [5, Theorem 2.4] implies that the output of $\mathcal{B}$ is a $(k, O(1))$-configuration with high probability.

We now discuss the running time of the above algorithm. It is straightforward to compute the sets B_i in $O(n)$ time. Our uniform weights k-median algorithm requires $O((|B_i| + r_d \log \frac{|B_i|}{k})k')$ time to compute Z_i, so the time required for all invocations of $\mathcal{A}$ is

$$\begin{aligned} O\left(\sum_{0 \leq i < r_w} \left(|B_i| + r_d \log\left(|B_i|/k\right)\right) k'\right) &= O\left(r_w\left(\frac{nk'}{r_w} + r_d k' \log\left(\frac{n}{kr_w}\right)\right)\right) \\ &= O\left(\left(n + r_d r_w \log \frac{n}{kr_w}\right) k'\right). \end{aligned}$$

(The first step follows from the fact that the sum is maximized when $|B_i| = n/r_w$.) Note that each weight function $\tilde{w}_{\phi_i}$ can be computed in $O(|B_i|\, k)$ time; it follows that $\tilde{w}_\phi$ can be computed in $O(nk)$ time. We employ the online median algorithm of [10] as the black-box k-median algorithm $\mathcal{B}$. Since $|\phi(U)|$ is at most kr_w, the time required for the invocation of $\mathcal{B}$ is $O((kr_w)^2 + kr_w r_d)$. The overall time required for our k-median algorithm is therefore $O((n + r_d r_w \log(\frac{n}{kr_w}))k' + (kr_w)^2 + kr_w r_d)$. Note that if $k = \Omega(\log n)$, $kr_w^2 = O(n)$, and $r_d r_w \log(\frac{n}{kr_w}) = O(n)$, this time bound simplifies to $O(nk)$.

References

[1] N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley, New York, NY, 1991.

[2] M. Charikar and S. Guha. Improved combinatorial algorithms for facility location and k-median problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 378–388, October 1999.

[3] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 1–10, May 1999.

[4] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York, 1973.

[5] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 359–366, November 2000.

[6] P. Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 428–434, May 1999. See also the revised version at http://theory.stanford.edu/˜indyk.

[7] L. Kavraki, J-C. Latombe, and P. Raghavan. Randomized query processing in robot path planning. *Journal of Computer and System Sciences*, 57:50–60, 1998.

[8] P. D. Mackenzie. Lower bounds for randomized exclusive write PRAMs. In *Proceeding of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 254–263, July 1995.

[9] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, 1999.

[10] R. R. Mettu and C. G. Plaxton. The online median problem. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 339–348, November 2000.

[11] N. Mishra, D. Oblinger, and L. Pitt. Sublinear time approximate clustering. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 439–447, January 2001.

[12] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 1995.

[13] M. Thorup. Quick k-medians, k-center, and facility location for sparse graphs. In *Proceedings of the 28th International Colloquium on Automata, Languages, and Programming*, July 2001. To appear.

[14] A. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1977.

A Lower Bounds

In this section, we give lower bounds for the k-median problem and its clustering variant. Throughout the section, we refer to the clustering variant as the *k-clustering problem*. Recall that the k-clustering problem asks us to partition the input points such that the sum, over all sets in the partition, of the weight of a point times the distance to the median of its set, is minimized. Since any k-median solution can be converted into a solution for the k-clustering problem in $O(nk)$ time, in developing our upper bounds it was sufficient to consider only the k-median problem. Unfortunately this reduction is not useful for the present purpose of establishing $\Omega(nk)$ lower bounds; accordingly, in this section we consider the problems separately.

For both the k-clustering problem and the k-median problem, we establish a lower bound of $\Omega(nk)$ time on any randomized algorithm that is $O(1)$-approximate with even a negligible probability. Since the overall objective of this paper is to study the complexity of approximate clustering in terms of the four parameters n, k, R_d, and R_w, it is desirable for the metric spaces associated with our lower bound arguments to have small values for both R_d and R_w. In terms of R_w, we achieve this goal completely, since all of the input distributions that we consider below have uniform weights, that is, $R_w = 1$. For the k-clustering problem, our lower bounds are established with R_d equal to a constant (sufficiently large relative to the desired approximation ratio); this is clearly best possible up to a constant factor. For the k-median problem, our lower bound requires R_d to exceed n/k by a sufficiently large constant factor relative to the desired approximation ratio.

In our proofs, we assume an oracle model of computation in which the algorithm is charged only for asking the oracle the distance between a pair of points. We refer to each call to the oracle as a *probe*. By a generalization of Yao's technique [14] due to Mackenzie [8], we can establish a lower bound of p on the success probability of a randomized algorithm by exhibiting an input distribution for which every deterministic algorithm has a success probability of at most p. (The intuition underlying this reduction is that the success probability of a randomized algorithm is just a convex combination of the success probabilities of a number of deterministic algorithms.) Thus in what follows, we restrict our attention to exhibiting "hard" distributions for determinstic algorithms. All of the problems considered in this section take the same input as the k-median problem. Our lower bounds also hold for the non-uniform case since for each choice of n and k, we exhibit a probability distribution over the set of n-point metric spaces on which no deterministic algorithm making a sufficiently small number of probes can achieve more than a negligible probability of success.

For any positive real $\ell > 1$, it is convenient to define a metric space to be *ℓ-simple* if the following conditions hold: (1) all of the points have unit weight; (2) the points of the metric space can be partitioned into equivalence classes such that the distance between any pair of distinct points is 1 if the points belong to the same equivalence class, and ℓ otherwise. Thus, any ℓ-simple metric space has $R_d = \ell$ and $R_w = 1$. Our lower bounds are all based on ℓ-simple input distributions for some appropriately chosen value of ℓ.

In order to establish a lower bound for the k-clustering problem, we find it convenient to introduce a problem that we call the *k-matching problem*. The input to the k-matching problem is the same as the input to the k-clustering problem. The output is a partition of the n input points into a collection of disjoint pairs and singletons, subject to the constraint that there are at most k singletons. We refer to such an output as a *k-matching*. The *cost* of a k-matching is defined

as the sum, over all output pairs of points (x, y), of $d(x, y) \cdot \min\{w(x), w(y)\}$. The goal of the k-matching problem is to compute a minimum-cost k-matching.

Given an algorithm for the k-clustering problem, consider the associated k-matching algorithm defined as follows: (1) run the k-clustering algorithm to partition the n input points into at most k clusters; (2) arbitrarily partition each even-sized cluster into a number of pairs; (3) arbitrarily partition each odd-sized cluster into a singleton and a number of pairs; (4) return the k-matching formed by the singletons and pairs computed in the previous two steps. Using the triangle inequality, it is straightforward to prove that the cost of the k-matching produced by this algorithm is at most the cost of the k-clustering computed in step (1) (i.e., the sum over all points x of the weight of x multiplied by the distance from x to the medoid of its cluster). Furthermore, this k-matching algorithm uses exactly the same number of probes as the associated k-clustering algorithm. Below we will exhibit an input distribution with respect to which any deterministic k-matching algorithm making a sufficiently small number of probes has only a negligible probability of computing a k-matching with cost within a constant factor of the cost of the optimal clustering. By the foregoing reduction from the k-matching problem to the k-clustering problem, such a result implies that any deterministic k-clustering algorithm running on the same input distribution and making the same small number of probes has only the same negligible probability of computing a k-clustering with cost within a constant factor of optimal.

In order to state and prove our lower bounds it is convenient to introduce a shorthand notation for expressing certain kinds of statements. In particular, for any statement S, we define an associated statement, which we refer to as the *P-claim S*, as follows: For all positive reals ε and c, there exist positive reals δ and γ and positive integers n_0 and a such that for all positive integers n and k for which $n \geq n_0$ and $1 < k < n$, there exists a probability distribution D over the set of ℓ-simple n-point metric spaces where $\ell = \gamma$ such that any deterministic k-matching algorithm $\mathcal{A}$ making at most δnk probes on an input drawn uniformly at random from D, the statement S holds with probability at least $1 - \varepsilon$. (We remark that a given P-claim S need not contain the parameter c. We also remark that if the P-claims S and T hold, then the P-claim $S \wedge T$ holds.)

We define a P'-claim in the same way as a P-claim except that the restriction on k is strengthened to $1 < k < \frac{n}{2}$. Similarly, a P''-claim is a variant of a P-claim in which the restriction on k is $\frac{n}{2} \leq k < n$. Note that for any statement S, the P'-claim S and the P''-claim S imply the P-claim S.

Finally, for addressing the k-median problem we define Q-, Q'-, and Q''-claims in an analogous manner, where the algorithm $\mathcal{A}$ is assumed to be a k-median algorithm rather than a k-matching algorithm, and ℓ is defined to be $\frac{\gamma n}{k}$ instead of γ.

The rest of this section is devoted to proving the following two theorems.

Theorem 2 *The P-claim "the cost of the k-matching solution computed by $\mathcal{A}$ is more than c times the cost of an optimal k-clustering solution" holds.*

Theorem 3 *The Q-claim "the cost of the k-median solution computed by $\mathcal{A}$ is more than c times the cost of an optimal k-median solution" holds.*

The proof of the first theorem follows from Lemmas A.1 and A.2 below. The proof of the second theorem follows from Lemmas A.3 and A.4.

Lemma A.1 *The P'-claim "the cost of the k-matching solution computed by $\mathcal{A}$ is more than c times the cost of an optimal k-clustering solution" holds.*

Proof Sketch: Let D denote the distribution of ℓ-simple n-point metric spaces where each point is independently placed into one of k equivalence classes uniformly at random. Given an input instance drawn from D, the cost of an optimal k-clustering solution is easily seen to be $n - k$.

Let us define a point x to be *clean* with respect to an execution of algorithm $\mathcal{A}$ if the following two conditions are satisfied: (1) there is no point y such that $d(x, y) = 1$ and $\mathcal{A}$ has probed $d(x, y)$; (2) $\mathcal{A}$ has probed the distance between x and at most εk other points.

It is not difficult to establish the following P'-claim: "At least $(1 - \varepsilon)n$ points are clean". Since $\mathcal{A}$ is a k-matching algorithm it outputs at least $n - k \geq n/2$ pairs. This observation, together with the preceding P'-claim, implies the P'-claim "At least $n/3$ of the pairs produced by $\mathcal{A}$ consist of two clean points." Note that each such output pair of clean points independently contributes a cost of ℓ to the cost of the k-matching produced by $\mathcal{A}$ with probability at least $1 - \frac{1}{k(1-\varepsilon)}$, since a clean point is equally likely to belong to any of the at least $k(1 - \varepsilon)$ equivalence classes (those for which $\mathcal{A}$ has not probed a distance between the given clean point and some point in the equivalence class). The claim of the lemma now follows by choosing constants appropriately (i.e., by setting δ, γ, and n_0 to appropriate functions of ε and c) and applying a standard Chernoff bound argument. ∎

Lemma A.2 *The P''-claim "the cost of the k-matching solution computed by $\mathcal{A}$ is more than c times the cost of an optimal k-clustering solution" holds.*

Proof Sketch: The proof of the preceding lemma does not readily extend to large values of k, so we employ a somewhat different approach. In this case we define the input distribution D by randomly partitioning the n points into k clusters (i.e., equivalence classes), $n - k$ of which are pairs, and $2k - n$ of which are singletons. As in the proof of Lemma A.1, the cost of an optimal k-clustering solution is $n - k$.

Let us assume for the sake of simplicity that n is a multiple of $2a$. (Remark: It is not difficult to modify our argument to handle general n.) For the sake of the analysis, it is useful to think of sampling from the input distribution D via the following three-stage process: (1) randomly partition the n points into $\frac{n}{2a}$ *supergroups* of size $2a$; (2) randomly partition each supergroup into a pairs; (3) pick a random set of $k - \frac{n}{2}$ pairs and split them to obtain $2k - n$ singletons. In what follows we refer to these pairs and singletons as *input-pairs* and *input-singletons*, in order to avoid confusion with the pairs and singletons computed by algorithm $\mathcal{A}$, which we refer to as *output-pairs* and *output-singletons*.

We define a supergroup to be *interesting* if it contains at least one input-pair. Note that there are at least $\frac{n-k}{a}$ interesting supergroups. Let us define a supergroup to be *red* if it contains at least one output-pair; otherwise, it is *blue*.

If there are i blue supergroups then at least i output-pairs either span distinct supergroups or contain at least one input-singleton; it follows that the cost of the k-matching produced by $\mathcal{A}$ is at least $i\ell$. If at least half (say) of the interesting supergroups are blue, this argument is sufficient to establish the lemma. Thus, in what follows, we may assume that at least half of the interesting supergroups are red.

Let us define a supergroup to be *clean* with respect to an execution of algorithm $\mathcal{A}$ if $\mathcal{A}$ does not probe the distance between any two points in the supergroup. It is not difficult to establish

the following P''-claim: "At least a $1-\varepsilon$ fraction of the interesting supergroups are clean." By this P''-claim and the assumption of the previous paragraph, we establish the P''-claim "at least one-third of the interesting supergroups are clean and red".

Let G denote a clean interesting red supergroup and let (x, y) denote an output-pair that belongs to G (such a pair exists since G is red). If x is an input-singleton then the cost of pair (x, y) is ℓ, and we can attribute this cost to G. Otherwise, x belongs to some input-pair (x, z), and algorithm $\mathcal{A}$ pays ℓ for the pair (x, y) unless $y = z$. But the probability that $y = z$ is $\frac{1}{2a-1}$ since G is clean. Furthermore, the event that $y = z$ is independent of the analogous events defined for other clean interesting red supergroups. Thus each clean interesting red supergroup independently contributes, with probability at least $1 - \frac{1}{2a-1}$, a cost of at least ℓ to the total cost of the k-matching produced by $\mathcal{A}$. The claim of the lemma now follows by choosing constants appropriately and applying a standard Chernoff bound argument. ∎

Lemma A.3 *The Q'-claim "the cost of the k-median solution computed by $\mathcal{A}$ is more than c times the cost of an optimal k-median solution" holds.*

Proof Sketch: Let D denote the distribution of ℓ-simple n-point metric spaces associated with the following partitioning scheme: (1) independently place each of the n points into one of $\lfloor k/2 \rfloor$ *tentative equivalence classes* uniformly at random; (2) randomly select $\lceil k/2 \rceil$ *special* points and move each of these special points into a singleton equivalence class. Note that for any such instance, the cost of an optimal k-median solution is $n - k$.

We define a point x to be *clean* with respect to an execution of algorithm $\mathcal{A}$ if there is no point y belonging to the same tentative equivalence class as x for which $\mathcal{A}$ has probed $d(x, y)$.

It is not difficult to establish the following pair of Q'-claims: (1) at least $(1-\varepsilon)n$ points are clean; (2) at least $(1-\varepsilon)\lceil k/2 \rceil$ of the special points are clean.

Let X denote the random variable corresponding to the set of clean points, and let Y denote the remaining points. Let Z denote the random variable corresponding to the set of special clean points. We now argue that the conditional distribution of Z given X and $|Z|$ has a simple structure, namely, Z is a uniformly random subset of X of size $|Z|$. This claim holds because the definition of a clean point implies that the behavior of algorithm $\mathcal{A}$ is the same no matter which size-$|Z|$ subset of X is equal to Z. Combining this claim with the results of the preceding paragraph, it is straightforward to establish the Q'-claim "$\mathcal{A}$ fails to output $\frac{1}{4}$ (say) of the clean special points."

Note that each special point that does not appear in the output of $\mathcal{A}$ contributes ℓ to the cost of the k-median solution computed by $\mathcal{A}$. Thus we obtain the Q'-claim "the cost of the solution computed by $\mathcal{A}$ is at least $(1-\varepsilon)k\ell/8$". Choosing γ sufficiently large (depending on c), the claim of the lemma then follows since $\ell = \gamma n/k$. ∎

Lemma A.4 *The Q''-claim "the cost of the k-median solution computed by $\mathcal{A}$ is more than c times optimal" holds.*

Proof Sketch: This proof is similar to that of Lemma A.2 above. We define the input distribution D in the same manner, as well as the following terms: supergroup, clean supergroup, interesting supergroup, input-pair, input-singleton. As before, note that at least $\frac{n-k}{a}$ of the supergroups are interesting.

We define the *input-weight* of a supergroup as the number of input-pairs and input-singletons that it contains. We define the *output-weight* of a supergroup as the size of its intersection with the k-median solution computed by $\mathcal{A}$. We define the *discrepancy* of a supergroup as its input-weight minus its output-weight. Note that the sum of the discrepancies of all supergroups is zero since the total input-weight and the total output-weight are both equal to k. A supergroup is *balanced* if it has discrepancy 0.

If the total discrepancy of the supergroups with positive discrepancy is s then it is straightforward to prove that the cost of the k-median solution computed by $\mathcal{A}$ is at least $s\ell$. If s is at least one-quarter of the number of interesting supergroups then this argument is sufficient to establish the claim of the lemma. Thus in what follows we may assume that s is less than one-quarter of the number of interesting supergroups. Under this assumption, at least half of the interesting supergroups are balanced (since at most one-quarter of them can have negative discrepancy).

It is not difficult to establish the following Q''-claim: "At least a $1-\varepsilon$ fraction of the interesting supergroups are clean." Combining this with the conclusion of the preceding paragraph we obtain the Q''-claim "at least one-third of the interesting supergroups are clean and balanced".

Let G denote a clean interesting balanced supergroup with i input-pairs and j input-singletons. Thus the input-weight and output-weight of G is $i+j$ (since G is balanced), and $i > 0$ (since G is interesting). In order to avoid paying a cost of ℓ for servicing any of the points in supergroup G, the subset of G of size $i+j$ contained in the output of $\mathcal{A}$ has to include exactly one point out of each of the i input-pairs, and all of the j input-singletons. Since G is clean, the probability that $\mathcal{A}$ produces such an output is 2^i divided by $\binom{2a}{i}$. Given the constraints on i, namely, $1 \leq i \leq a$, this probability is at most $1/a$. Furthermore, the event that $\mathcal{A}$ produces such an output is independent the analogous events defined for other clean interesting balanced supergroups. Thus each clean interesting balanced supergroup independently contributes, with probability at least $1/a$, a cost of at least ℓ to the total cost of the k-median solution produced by $\mathcal{A}$. The claim of the lemma now follows by choosing constants appropriately and applying a standard Chernoff bound argument. ■