K-median Algorithms: Theory in Practice

David Dohan, Stefani Karp, Brian Matejek

Submitted: January 16, 2015

1 Introduction

1.1 Background and Notation

The k-median problem has received significant attention for decades, primarily in computer science and operations research. The problem is generally formalized as follows:

Given a metric space X, a set of clients $C \in X$, and a set of facilities $F \in X$ such that $|X| = |F \cup C| = n$, open k facilities in F so as to minimize the sum of distances from each client to its nearest open facility.

We define the distance metric as d_{ij} for $i \in \{1, \ldots, n\}, j \in \{1, \ldots, n\}$, such that d_{ij} is the distance between points i and j in the metric space X. Kariv and Hakim [1] proved that finding such k medians in a network is an NP-hard problem by reducing the dominating set problem to it. A simple brute-force algorithm would examine every possible size-k subset in F, compute the closest facility in this set for every client, and return the best set overall. This brute-force algorithm would run in $\mathcal{O}\left(\binom{n_f}{k} n_c k\right)$ time, where $|F| = n_f, |C| = n_c$. Thus, academic research into this problem has focused primarily on producing good approximation algorithms. For a given algorithm, the approximation ratio is defined as the provably worst possible ratio between the cost of the algorithm's output and the optimal cost. However, for most problem instances, we do not know the actual optimal cost, and we thus compute the approximation ratio as the total cost returned by the algorithm divided by the optimal value of the relaxed linear program discussed in section 1.2. Jain et al. [2] proved that the k-median problem is $1 + \frac{2}{e} \approx$ "1.736"-hard to approximate in a metric space. We note that, throughout this paper, all of our distance metrics satisfy the properties of a metric space.

1.2 Relation to Other NP-Hard Problems

The k-median problem has many similarities to the facility location problem (FLP). In this problem, we are given a metric space X with clients C, facilities F, costs d_{ij} of client j using facility i, and costs f_i associated with opening facility i. There are two variants of the facility location problem: uncapacitated and capacitated. In the uncapacitated facility location problem (UFLP), any number of clients can use any given facility. In the capacitated facility location problem (CFLP), we define a set of variables $V(i)$ such that the number of clients that use facility i must be less than or equal to $V(i)$. For the rest of this paper, we concern ourselves only with the uncapacitated facility location problem. In 1966, Balinski [3] defined the following integer program for the UFLP. Here, y_i indicates if facility i is open and x_{ij} indicates if client j is connected to facility i.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in F, j \in C} d_{ij} x_{ij} + \sum_{i \in F} f_i y_i \\
\text{subject to} \quad & \forall j \in C : \sum_{i \in F} x_{ij} \geq 1 \\
& \forall i \in F, j \in C : x_{ij} \leq y_i \\
& \forall i \in F, j \in C : x_{ij} \in \{0, 1\} \\
& \forall i \in F : y_i \in \{0, 1\}
\end{aligned}
$$

This integer program can be relaxed by changing the integer constraints $x_{ij} \in \{0, 1\}$ and $y_i \in \{0, 1\}$ into $x_{ij} \geq 0$ and $y_i \geq 0$ respectively. Another variant of the UFLP restricts the number of open facilities to k (UKFLP). We can construct the integer (and resulting linear relaxation) program by adding the constraint that the sum of the y_i values must be less than or equal to k. We can easily connect the the UKFLP to an uncapacitated k-median problem by setting $f_i = 0$, $\forall i \in F$. The resulting integer program is defined below:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in F, j \in C} d_{ij} x_{ij} \\
\text{subject to} \quad & \forall j \in C : \sum_{i \in F} x_{ij} = 1 \\
& \forall i \in F, j \in C : x_{ij} \leq y_i \\
& \sum_{i \in F} y_i \leq k \\
& \forall i \in F, j \in C : x_{ij} \in \{0, 1\} \\
& \forall i \in F : y_i \in \{0, 1\}
\end{aligned}
$$

This integer program can be relaxed by changing the integer constraints $x_{ij} \in \{0, 1\}$ and $y_i \in \{0, 1\}$ into $x_{ij} \geq 0$ and $y_i \geq 0$ respectively. For the purposes of this paper, we shall refer to the uncapacitated k-median problem as simply the k-median problem. If $X = F = C$, a k-median algorithm simply clusters the points in the metric space.

1.3 Existing Theoretical Research

There is a large body of existing research on k-median approximation algorithms. A forward greedy algorithm iteratively open facilities by choosing the next unopened facility so as to minimize the cost function defined above. Chrobak et al. [4] proved that a reverse greedy algorithm (one that starts with all facilities open and closes facilities that least increase the cost function) produces an approximation ratio between $\Omega(\log n / \log \log n)$ and $\mathcal{O}(\log n)$. Another frequently used algorithm, partitioning around medoids (PAM), iteratively computes the cost of swapping medoids and chooses the swap that lowers the total cost the most (if any such swap still exists).[1] This is a local search algorithm. With respect to performance guarantees, in 2003 Arya et al. [5] proved that this local search with single swaps has a worst-case approximation ratio of 5. In the general case, in which at most p facilities can be swapped at once, the approximation ratio is $3 + 2/p$.

Charikar et al. [6] provided the first constant-factor approximation algorithm for the k-median problem, achieving a ratio of $6\frac{2}{3}$. This relied on a rounding scheme to the LP relaxation of the integer program presented in the previous section. Several other papers since have used linear programming techniques.

[1] In the Operations Research literature, the term "medoid" is often used instead of "median." However, its meaning is equivalent.

We briefly consider some of the papers here. Jain and Vazirani [7] obtained a 3-approximation primal-dual algorithm for the UFLP. This provided a 6-approximation algorithm for the k-median problem by expressing the UFL problem as a Lagrange relaxation of the k-median problem and using the 3-approximation UFL algorithm as a subroutine. Building on his existing paper, Jain et al. [8] improved this ratio to 2 for the UFLP problem and therefore 4 for the k-median problem. In 2012, Charikar and Li [9] developed a new LP relaxation-based algorithm that achieves an approximation ratio of $3.25(1+\delta)$. Although worse than the $3+2/p$ local search approximation mentioned above, the running time was reduced from $\mathcal{O}\left(n^8\right)$ to $\mathcal{O}\left(k^3n^2/\delta^2\right)$. In 2013, Li and Svensson [10] developed an approximation algorithm that achieves an approximation guarantee of $1+\sqrt{3}+\epsilon$, finally improving the $3+\epsilon$ ratio of Arya et al.

1.4 Existing Experimental Research

Various empirical studies of the k-median problem have been performed in both computer science and operations research. However, to our knowledge (and that of Charikar and Li as well), the Charikar 2012 LP algorithm has never before been implemented or studied empirically; this is the primary motivation of our paper. Below, we briefly discuss the most relevant prior empirical studies.

In 2011, Nagarajan and Williamson [11] performed an empirical study of various k-median approximation algorithms, including Arya et al.'s local search algorithm, Charikar's 1999 LP rounding algorithm, and Jain et al.'s 4-approximation algorithm. However, this experimental study was prior to the development of Charikar's 2012 LP algorithm, which is the main motivation of our paper. Furthermore, Nagarajan and Williamson only analyzed local search with single swaps (which guarantees an approximation ratio of $3+2/p=5$) due to the high running times of the multi-swap version. Regarding data, they tested their algorithms on the 40 datasets of the OR-Library and three other datasets of sizes 100, 150, and 316 nodes. In all of these datasets, $F=C$. Nagarajan and Williamson concluded that local search and Charikar's 1999 LP rounding algorithm performed better than the Jain-Vazirani algorithm, and further that Charikar's algorithm generally performed better than local search. However, for most of the datasets used, the LP solutions were typically integral or very close to integral. From the results presented, it is not clear how these algorithms perform on datasets for which the LP does not return integral solutions.

In 2013, Bhattacharya et al. [12] empirically analyzed the *robust* k-median problem, which attempts to minimize the worst case connection cost over all clients (instead of the total cost). Although their paper focused specifically on the *robust* k-median problem (as opposed to the original k-median problem), its methods for generating random metric spaces of facilities and clients for testing can be extended to any variant of the k-median problem. These methods are discussed further in 4.2. Regarding LP analyses, in 2014, Awasthi et al. [13] performed a theoretical and empirical study of the integrality of LP relaxations for the k-means and k-median problems. They found that the LP solution is integral more frequently for the k-median problem than for the k-means problem. In fact, for cluster separation at least some constant c and any k, the k-median LP solution will be integral if n is large enough (though "large enough" is not precisely defined in terms of the problem's parameters). These conclusions motivate studying a large number of k-median problem instances with LP relaxation-based approximation algorithms.

Our research builds on Nagarajan and Williamson's analyses through an empirical study of various k-median algorithms using randomly-generated datasets as described in [12], which provide a much more comprehensive picture of the algorithms' performance. Furthermore, we focus much of our analysis on Charikar's 2012 algorithm, neither previously implemented nor studied empirically. Given that Charikar's 2012 algorithm has an approximation ratio of at most 3.25 (ideally below 3 with proper parameter adjustment), this algorithm is competitive with Arya et al.'s local search, which had provided the best-known approximation guarantee until Li-Svensson in 2013. Determining how Charikar's algorithm performs in practice is therefore of significant interest.

2 Implemented Algorithms

In this section, we will explore the algorithms which we implemented in more detail. In general, let C be the set of cities, F the set of possible facility locations, $n_f = |F|$, and $n_c = |C|$. We let $cost(S)$ be the sum of costs from each city to the nearest facility in S.

2.1 Greedy Algorithms

2.1.1 Forward Greedy

A forward greedy algorithm for the k-median problem maintains a set of S_t of medians at step t and sets $S_{t+1} = S \cup f$ to minimize $cost(S_{t+1})$ until $|S| = k$. This is a straightforward algorithm that performs quickly in practice, but it is at least an n-approximation algorithm for the problem [4].

Algorithm 1 Forward Greedy Algorithm

$S \leftarrow \{\}$
while $|S| < k$ **do**
  $f \leftarrow \arg\min_i(cost(S \cup \{i\}))$
  $S \leftarrow S \cup \{f\}$
end while
return S

2.1.2 Reverse Greedy

The reverse greedy algorithm begins by placing a facility on every location, repeatedly removing the facility from the set of medians that will increase the cost the least until k facilities remain. [4] proved that the reverse greedy algorithm has an approximation ratio between $\Omega(\log n / \log\log n)$ and $\mathcal{O}(\log n)$.

Algorithm 2 Reverse Greedy Algorithm

$S \leftarrow F$
while $|S| > k$ **do**
  $f \leftarrow \arg\min_i(cost(S - \{i\}))$
  $S \leftarrow S - \{f\}$
end while
return S

2.2 Local Search

In its simplest form, given some set of medians S, local search arrives at a solution for a k-median by repeatedly swapping a median from S for an element from $F - S$ that minimizes the new cost. In the more general case, local search features an adjustable parameter p that allows swapping sets of up to p medians at a time.

The initialization phase is generally done via an application of the forward greedy algorithm to the problem. In this case, the algorithm is fully deterministic.

Local search as presented in [5] is below, where $\mathcal{B}(S)$ is the set of possible sets reachable from S by making p swaps.

Algorithm 3 Local Search

$S \leftarrow ForwardGreedy(C, F, k)$
while $\exists S' \in \mathcal{B}(S)$ s.t. $cost(S') < cost(S)$ **do**
  $S \leftarrow S'$
end while
return S

[5] proved that local search with p swaps is a $3 + 2/p$ approximation algorithm, which gives an approximation ratio of 5 in the simple single swap case. Each swap in the algorithm runs in $n^{\mathcal{O}(p)}$ time. They also prove that as long as only swaps that provide a gain of at least some constant amount are accepted, the number of steps will be polynomial as well. There are many implementation tricks and data structures to speed up local search in practice, such as caching which cities are associated with each facility, but we do not explore these techniques in detail.

It is interesting to note that local search with single swaps is a long standing technique in the statistical and operations research (OR) communities, where it is referred to as Partitioning around Medoids (PAM) [14].

2.3 Jain Vazirani

Jain and Vazirani [7] developed a 3-approximation algorithm for the uncapacitated facility location (UFL) problem using the primal-dual schema. Furthermore, they showed that the UFL problem is a Langrangian relaxation of the k-median problem. Therefore, Jain and Vazirani were able to use their primal-dual approach as a subroutine in a 6-approximation algorithm for the k-median problem. Furthermore, their k-median algorithm has a very fast running time: $O(m \log m(L + \log(n)))$, where n and m are the numbers of vertices and edges in the bipartite graph of cities and facilities.

The intuition for the algorithm is as follows: Each city has a contributing and a connected state. As time t increases from 0, each city is willing to consider connecting to a facility within radius t of its location. Consider a facility at distance $r \leq t$ from a contributing city c. If f is open, then c connects to f and switches to the connected state for all future time. If f is not open, then c contributes $t - r$ toward opening it. A facility opens when the sum of contributions are greater than the facility opening cost. When a city connects, all its existing contributions remain, but do not increase any more. This is the first phase of the algorithm.

The second phase focuses on choosing a subset of the temporarily open cities to return as open. In [8], Jain et al. simplified the algorithm to remove the need for a cleanup phase. In the second phase, any edge i, j between city i and facility j is considered a "special" edge if $dist(i, j)$ is less than the time when i connected to a facility. The algorithm forms a graph T from these edges, finds T^2, which has an edge i, j if there is a path of length 1 or 2 between i and j in T, and finds H as the subgraph of T^2 induced on the set of temporarily open facilities (i.e., only keep vertices corresponding to temporarily open facilities from phase 1). This means that for any city c, all temporarily opened facilities that received positive contributions from c will form a clique in H. Finally, select a maximal independent subset from H and return it. This means that in the final set of open facilities, each city will have contributed a positive amount to at most 1 of them. While finding the maximal independent subset is an NP-hard problem in general, greedily adding facilities that don't conflict with the current independent set in the order they were opened turns out to produce a maximal independent set in this case [15].

Jain and Vazirani use binary search on facility opening costs to arrive at a solution to the k-median problem. If every facility is assigned an opening cost of 0, the above two phases will open n_f facilities. If the facility opening cost is arbitrarily high, one facility will open. The algorithm performs a binary search between these two opening costs to find the cost that opens k facilities. If no such cost exists, the algorithm takes two facility subsets of size k_1, k_2, such that $k_1 < k < k_2$. The algorithm provides a rounding scheme to find k facilities to open given these two subsets. The rounding scheme increases the cost by at most a factor of 2. For the purpose of this analysis, we did not implement the rounding phase;

instead, we compared performance only on instances for which an integral solution was successfully found via binary search.

2.4 Charikar 2012 Rounding

Charikar and Li developed a dependent LP-rounding approach in [9] that achieved a guaranteed $3.25(1+\delta)$ approximation in $\mathcal{O}\left(k^3n^2/\delta^2\right)$ running time, where $n = |F \cup C|$. The algorithm relies on Young's approximation for the k-median LP [16], which yields a $(1+\delta)k$ approximation. In our implementation, we do not use Young's approximation algorithm so the running time is slightly slower but the guaranteed bound does not depend on δ.

2.4.1 Preliminary Steps and Notation

First, the algorithm initializes the linear program stated in section 1.3 and generates the fractional solution. In the following explanation we refer to y_i as the fractional component of facility i that is open in the LP solution and x_{ij} as the "amount" of connection open between facility i and client j. For each client j in C define F_j as the set of facilities where $x_{ij} > 0$. These are the facilities that j is connected to in the fractional solution. The connection cost of j in the fractional solution is thus $\sum_{i \in F_j} y_i d_{ij}$. This is the cost for a client j in the LP relaxation. Define $d_{av}(j)$ as this cost for each client j. Also, define the function $B(j, r)$ that denotes the set of facilities that have a distance strictly smaller than r to j. All facilities i where $y_i = 0$ are removed from F, since they are not at all fractionally open. The rounding scheme proceeds in four steps:

2.4.2 Filtering Phase

The algorithm reduces the number of considered clients by selecting a subset $C' \subseteq C$ that has two properties: (1) the clients in C' are far away from each other and (2) a client in $C \backslash C'$ is close to some client in C'. More specifically, for all clients j, j' in C', $d_{ij} > 4 \max\{d_{av}(j), d_{av}(j')\}$ and for every j in $C \backslash C'$ there is a client j' in C' such that $d_{jj'} \leq 4 d_{av}(j)$.

2.4.3 Bundling Phase

The bundling phase attempts to group possible facility locations to clients in C'. Since these clients are far apart from each other, each bundle has a large probability of generating an open facility in the final solution. Call the bundle for client j, U_j and define R_j to be half the distance to the next closest client in C'. For each client j define $F'_j = F_j \cap B(j, 1.5R_j)$.[2] Every facility that appears in at least one F'_j is placed into a bundle. Facilities that appear in multiple F'_j are assigned to the client that is closest.

2.4.4 Matching Phase

The matching phase groups two bundles U_j and $U_{j'}$ so that they can be sampled from a joint distribution. Matched bundles should be close together so a greedy algorithm is used where clients j and j' are matched in increasing order of distance. Each client in j is only matched once. If there are an odd number of clients in C', the client furthest from all others is not matched.

2.4.5 Sampling Phase

Define the function $vol(F') = \sum_{i \in F'} y_i$. The sampling phase concludes as discussed in algorithm 4 below.

After the sampling phase, return the k open facilities. Since this algorithm only returns k facilities in expectation, we run the sampling phase until k facilities are open. Alternatively, one could implement a complicated tree structure that samples the distributions dependently.

[2] The choice of 1.5 allows for an easier mathematical analysis. This choice will be studied empircally in section 5.2.

Algorithm 4 Sampling Phase

```
for each pair (j, j') ∈ M do
    random_double = rand();
    if random_double < 1 − vol(U_{j'}) then
        open U_j
    else if random_double < 2 − vol(U_j) − vol(U_{j'}) then
        open U_{j'}
    else
        open both U_j and U_{j'}
    end if
end for
if ∃j ∈ C' and not matched in M then
    open J_j with probability vol(U_j)
end if
for each facility i not in any bundle U_j do
    open i with probability y_i
end for
```

2.4.6 Practical Improvements and Variants

Multiple Iteration Sampling

The only randomized portion of this algorithm occurs in the sampling phase. Consider the case where the linear program does not return the integer solution. There are multiple possible rounding schemes that can transform the fractional solution into an integer solution. These rounding schemes may have different costs. Rather than accept the first, in this paper we propose an algorithm to run the rounding schemes a thousand times and return the set of facilities that produces the lowest cost. Although sampling multiple times does not improve the guaranteed approximation ratio, in practice it produces results at least as good as the traditional Charikar algorithm. We analyze this variant of the algorithm in section 5.1. For the duration of this paper, we refer to the traditional Charikar and Li algorithm documented above as "Charikar 2012" and the multiple iterative sampling algorithm as "Charikar Multi Sample".

Changing Parameters

In section 2.4.3, the constant of 1.5 was introduced. The constant of 1.5 provides an approximation ratio of 3.25. In the paper, it is briefly mentioned that the analysis is simpler if a constant of 1.0 is used. However, the guaranteed approximation ratio reduces to a factor of 4. In a discussion with Professor Charikar, he suggested that a value of ∞ would produce the best approximation ratio, perhaps beating the (at the time) decade long $3 + \epsilon$ barrier. If that constant were equal to ∞, the bundles would be comprised entirely of the facilities where $x_{ij} > 0$. In section 5.2, we explore the results of varying this constant; specifically, we study the values 1, 1.5, 10, 100 and ∞.

3 Implementation Details

We chose to implement the algorithms in the Julia language. Julia is a high level language designed for high-performance scientific computing. It shares many syntactic characteristics with Python and Matlab (in addition to many other languages) while remaining performant. Our primary reason for using Julia is the following: while it is faster to develop than the C family, it is also significantly faster with respect to runtime than a language such as Python or Matlab (i.e., Julia is often within a small factor of C or Fortran). While we did not focus primarily on performance (such as by implementing complicated data structures to help in local search), this speed was helpful in allowing our algorithms to run within reasonale time spans. Julia also provides a powerful interface to LP solvers (we used the

COIN-OR solver), which was necessary for the rounding scheme algorithm.

In order to test our algorithms on our large number of datasets (see 4), we distributed computation across the Ionic cluster in Princeton University's Computer Science department. We ran on the order of 50,000 experiments using each of the algorithms.

Each algorithm implementation expects a parameter k and an $n_f \times n_c$ cost matrix representing pairwise distances between facilities and cities. The algorithm returns a list of IDs corresponding to a set of facilities to open.

4 Datasets

4.1 OR-Library

The OR-Library was originally described in [17]. It is a collection of test datasets for various Operations Research problems, including p-median. Among the OR-Library, there are 40 uncapacitated p-median data files with known optimal values. In theses instances, $X = F = C$ and the number of clients and facilities range from 100 to 800. Each instance is presented as an edge list representing an undirected graph. We convert it to an $n \times n$ cost matrix by applying an all-pairs shortest path algorithm to this graph.

4.2 Randomly Generated Distributions

We generated random data sets based on the methods described in [12]. The data generated produces clients and facilities in $\mathbb{R}^2$ and the distance function is the Euclidean distance. The facilities are randomly chosen based on a uniform distribution between $0 \leq x \leq 100$ and $0 \leq y \leq 100$. The clients are generated in groups. The number of clients in a group is either constant or chosen from a normal distribution or exponential distribution. The clients themselves are either distributed based on a uniform or gaussian distribution. In the following sections, we use the notation $gauss(constant)$ to indicate that a constant number of clients belonged to a group, with locations determined by a gaussian distributions. Note that the optimal k median solution might group clients into the originally generated groups since the distributions overlap in $\mathbb{R}^2$. Using this client/facility generation method, we created over 13,000 instances. For each instance, we ran the algorithms above for $k = 7, 100, 200, 300$ and 400.

5 Charikar Analysis

We begin the analysis with a proof about the Charikar 2012 rounding scheme that will be used later. Denote OPT_{LP} as the optimal value of the linear program relaxation, OPT_{IP} as the optimal value of the integer program (and thus the true minimum of the k-median problem), and $CHAR$ as the value returned by the Charikar 2012 algorithm.

Theorem 1. *$OPT_{LP} = OPT_{IP}$ if and only if $OPT_{LP} = CHAR$.*

Proof. Linear programs either return 0, 1, or an infinite number of solutions (that is all linear programs are either infeasible, unbounded, or unique). The integer solution to the integer program is a feasible solution to the linear program. Thus, the optimal value of the integer program is considered when determining the optimal value of the linear program. If the linear program and the integer program have the same optimal value, the linear program has returned the integer solution to the integer program since the solution is unique. The Charikar 2012 algorithm starts by removing all facilities from F that have $y_i = 0$. If the LP returns an integer solution, all but k of these facilities will have $y_i = 0$. Thus, the Charikar algorithm will proceed through the rest of the steps with only k facilities. No other facilities are removed after this first step and the algorithm returns k facilities. Thus these k facilities will be returned and $OPT_{LP} = CHAR$.

Type	Worst C2012 Ratio	Worst CMS Ratio	Largest Percent Increase CMS v. C2012
gauss_constant	23.1%	1.52%	22.0%
gauss_expo	22.4%	1.20%	22.4%
gauss_normal	17.8%	0.97%	17.7%
uniform_constant	36.2%	1.34%	35.7%
uniform_expo	35.7%	1.90%	34.7%
uniform_normal	33.5%	1.47%	33.4%

Table 1: Performance of the Charikar 2012 algorithm versus the Charikar Multi Sample algorithm

In the reverse direction, the Charikar 2012 algorithm returns a feasible solution to the integer program. Thus $OPT_{IP} \leq CHAR$ and $CHAR = OPT_{LP}$ so $OPT_{IP} \leq OPT_{LP}$. However, the LP was defined such that $OPT_{LP} \leq OPT_{IP}$ so $OPT_{LP} = OPT_{IP}$. □

Corollary 2. *The Charikar 2012 algorithm will return the optimal linear program solution if and only if the Charikar Multi Sample algorithm returns the optimal linear program solution.*

Proof. In the forward direction, if the Charikar 2012 algorithm returns the optimal linear program solution then only k facilities make it to the final phase and thus these k facilities will be returned regardless of the number of iterations. Thus, the Charikar Multi Sample algorithm will return the optimal solution. In the reverse direction, if the Charikar Multi Sample algorithm returns the optimal linear program solution, then the LP returned the optimal integer program solution and thus Charikar 2012 returned the optimal linear program solution. □

Note it is possible that the Charikar Multi Sample algorithm can return OPT_{IP} if Charikar 2012 does not if $OPT_{IP} \neq OPT_{LP}$. With this theorem, we can determine the number of times that $OPT_{LP} = OPT_{IP}$ which will provide information on how well the local and greedy searches perform with respect to the optimal integer program solution.

5.1 Charikar Multi Sample versus Charikar 2012

As discussed in section 2.4.6, we produced a variant of the Charikar 2012 algorithm that randomly samples multiple times to produce a better approximation to the k-median problem. Although the guaranteed approximation will not decrease, in practice we can improve on the cost by sampling many times and returning the best set of k facilities. We can imagine a random variable Y representing the cost of the Charikar 2012 rounding scheme. This variable has some mean μ and variance σ^2. By randomly sampling multiple times and choosing the best result we increase the probability of selecting the optimal rounding scheme. Table 1 display the percent increase from the LP solution. Specifically, if LP_{OPT} is the fractional LP solution and C is the objective function value returned by the algorithm, then the value in the table is $\frac{C - LP_{OPT}}{LP_{OPT}} * 100\%$. The final column shows over all instances the largest percent increase achieved by the Charikar Multi Sample variant over the Charikar 2012 algorithm. This value is calculated by $\frac{C2012 - CMS}{CMS} * 100\%$.

5.2 Empirical Study of Constants in Charikar 2012

As discussed in section 2.4.6, the value of 1.5 was chosen in the bundling phase to make the mathematical analysis easier. During our discussions with Charikar, he suggested that the approximation ratio might be improved if the algorithm were to use a value of ∞ (so that each bundle U_j would actually just include the facilities i where $x_{ij} > 0$). The analysis is actually much simpler if a value of 1 is used, although the guaranteed approximation ratio falls to 4. Obviously an empirical analysis on the values of 1.5 versus ∞ cannot prove anything about the guaranteed approximation ratio. However, we believe it is still interesting to see how the Charikar algorithm performs while varying this constant. In Figure

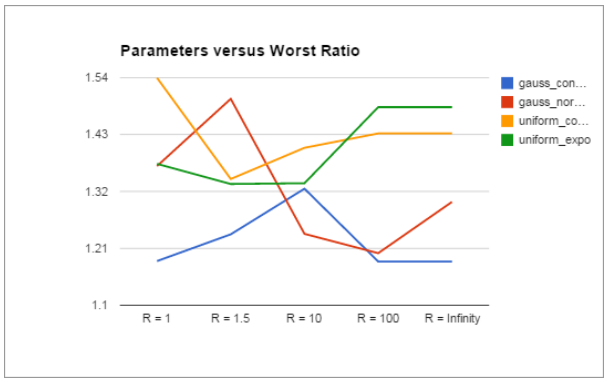

Figure 1: This graph shows the worst ratio achieved on an instance with varying values of R_{max}, where R_{max} is the constant presented earlier.

1, we explore the worst ratio achieved for each of the six datasets for values of $R = 1, 1.5, 10, 100, \infty$. Interestingly, the value of ∞ does better on gaussian inputs than on uniform inputs.

6 General Analysis

The following tables display the percent deviation from the LP solution. Specifically, if LP_{OPT} is the fractional LP solution and C is the objective function value returned by the algorithm, then the value in the table is $\frac{C-LP_{OPT}}{LP_{OPT}} \times 100\%$.

When the number of facilities increases from 10 to 110, PAM with multiple swaps generally fails to converge in the allowed time. Therefore, PAM with multiple swaps is analyzed in tables 5, 6, and 7 below. If we are interested solely in the feasibility of these approximation algorithms, we quickly see that PAM with multiple swaps does not scale well with the number of facilities. In fact, the Charikar 2012 algorithm has a worse approximation guarantee than PAM with $p = 8$ swaps. However, such a PAM algorithm would run in $\mathcal{O}\left(n^8\right)$ time, which is infeasible for all but small n.

We throw out all instances for which any algorithm fails to converge in the time allowed. We note that this eliminates most instances with 310 and 410 facilities and favors instances with fewer facilities (i.e., PAM with single swaps generally fails to terminate in the allotted time when the dataset includes more than 210 facilities). Tables 2, 3, 4 show the average, median, and largest percent deviations respectively for $k = 7$. (For other values of k, see the Appendix). We believe the three statistics shown (average ratio, median ratio, and worst ratio) are of significant interest. First off, the papers originally presenting these algorithms only produce worst case guarantees for the k-median problem. However, some of the papers can only produce a loose upper bound because of complexities in the analysis (e.g. Charikar 2012 [9]). By tracking the worst ratio over several thousand diverse randomly generated problem instances, we can examine the likelihood that the guaranteed bound is reached. An algorithm with a $3 + \epsilon$ bound that is frequently tight is worse in practice than the 6 approximation that is almost never reached. The average and median ratios are also of interest because they reflect how well the algorithms will do on typical instances. Further, since the median is robust to outliers, it does not allow the few worst case problem instances to skew the results.

As can be seen in Table 2, Charikar 2012 performs better on average than PAM with single swaps for gauss_constant and gauss_expo. For gauss_normal, Charikar 2012 and PAM perform approximately the same, and for uniform_constant, uniform_expo, and uniform_normal, Charikar 2012 performs slightly worse than PAM. However, Charikar Multi Sample performs significantly better than any of the other algorithms on average, achieving a percent deviation very close to 0. The greedy algorithms and the Jain-Vazirani algorithm generally perform worse on average than Charikar 2012 and PAM. Table 3 shows that the median percent deviations for Charikar 2012, Charikar Multi Sample, and PAM with single

Type	Charikar 2012	Charikar Multi	Forward Greedy	JV	Reverse Greedy	PAM
gauss_constant	0.13	0.01	3.12	2.91	1.89	0.24
gauss_expo	0.08	0.0	3.98	2.64	1.26	0.16
gauss_normal	0.28	0.01	3.99	3.21	1.95	0.26
uniform_constant	0.94	0.03	4.53	4.1	3.87	0.66
uniform_expo	0.93	0.03	4.17	4.18	3.76	0.58
uniform_normal	0.78	0.02	4.78	4.22	3.83	0.66

Table 2: Average Performance for $k = 7$

Type	Charikar 2012	Charikar Multi	Forward Greedy	JV	Reverse Greedy	PAM
gauss_constant	0.0	0.0	0.0	1.78	0.0	0.0
gauss_expo	0.0	0.0	0.54	1.42	0.0	0.0
gauss_normal	0.0	0.0	0.29	2.16	0.0	0.0
uniform_constant	0.0	0.0	3.23	3.73	4.17	0.0
uniform_expo	0.0	0.0	3.24	3.69	4.0	0.0
uniform_normal	0.0	0.0	3.44	3.91	4.13	0.0

Table 3: Median Performance for $k = 7$

swaps are all 0. Reverse greedy has a median value of 0 for the gaussian distributions, and forward greedy also performs better on the gaussian distributions. The Jain-Vazirani algortihm has a high median in general.

Worst-case analyses in Table 4 indicate that forward greedy performs very poorly in the worst case, but Charikar with multiple iterations performs very well in the worst case (a percent deviation of no more than 1.5). The other algorithms' worst-case values are generally between forward greedy and Charikar with multiple iterations. The results of the worst case provide insight into optimal input structures for the various algorithms. We note how, for all algorithms, in general the uniform distributions caused more problems than the gaussian distributions.

These results give us fundamental insight into the structure of inputs that succeed in these approximation algorithms. Recall from section 4.2 that a certain number of group centers are generated in each instance. Clients generated around these group centers either follow a uniform or gaussian distribution. If the clients follow a gaussian distribution, they will generally be more concetrated around the group centers. The facilites are uniformally distributed across the plane of possible locations. Since the clients are more clustered, a single facility that is closer to the group center can better service the clients in that cluster. When the clusters are uniformally distributed, three facilities might fall in the range of this client cluster but none has a clear advantage over the others. Figure 2 further illustrates this point. If there are multiple groups such as these across the metric space, with some of the distributions overlapping, it becomes easy to see why the algorithms perform better on the gaussian distributions. We can look back at the algorithms themselves to further justify the improvements. Recall section 2.4.2 where the filtering phase of the Charikar 2012 algorithm is explained. Clusters of clients that are close together are grouped to form client centers in C'. In a gaussian distribution, these client centers are much more apparent.

In the forward greedy algorithm, facilities closest to the gaussian centers will open first. In a uniform distribution, the algorithm cannot exploit the structure and must choose between several similar cost facilities. In the gaussian distributions, a facility that is close to a cluster center with a small standard deviation will open quickly, serving all of those clients. Such structure is not as exploitable in uniform distributions.

In the reverse greedy algorithm, every facilitiy is initially opened. The facilities are closed in increasing order of cost penalty. In a uniform client distribution, more facilities will have similar penalties for

Type	Charikar 2012	Charikar Multi	Forward Greedy	JV	Reverse Greedy	PAM
gauss_constant	18.1	0.52	29.45	17.3	14.3	6.77
gauss_expo	13.0	0.66	30.35	23.76	11.03	5.25
gauss_normal	49.53	0.97	43.41	21.13	14.9	6.86
uniform_constant	35.4	1.29	57.85	22.51	19.81	15.64
uniform_expo	29.53	1.9	80.5	18.01	14.1	7.7
uniform_normal	41.53	1.47	82.68	21.64	17.5	19.52

Table 4: Worst Case Performance for $k = 7$

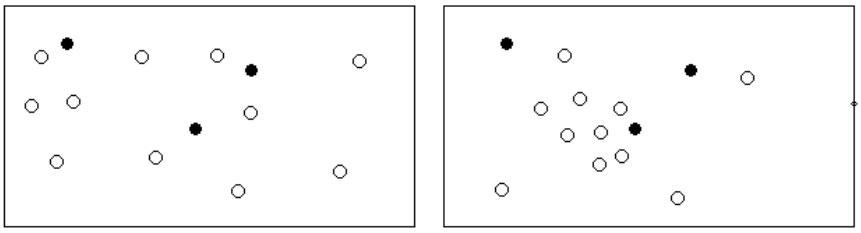

Figure 2: The black circles represent possible facility locations and the clear circle clients. The clients are distributed by a uniform distribution on the left and a gaussian distribution on the right.

closing. In a gaussian distributions, the facilities closer to the cluster centers will remain open longer until it becomes time to reduce the number of open facilities beyond the number of client groups. However, by this point the algorithm can remove facilities servicing a group of clients close to another group of clients. Thus, the overall structure can be better exploited.

Tables 6, 5, 7 show how PAM with single swaps and PAM with 2-way swaps perform in comparison with Charikar Multi. We used the time in which Charikar Multi terminated as the time limit for both PAM algorithms (i.e., when the time limit was exceeded, we simply terminated the local search and returned the best value found thus far). This allowed the three algorithms to be compared more accurately in practice. When these restrictions are placed on PAM, we find that PAM and PAM Multiswap perform worse on average than Charikar Multi, although PAM performs better in the worst case. Interestingly, when we limit the amount of time that both PAM and PAM Multiswap can run, PAM ofter performs better when $p = 1$. This leads us to believe that it is better to run PAM alone when time is the major constraint.

We included the tables for $k = 100, 200, 300$, and 400 in the appendix. Most noteworthy is that all of the algorithms tend to do better for larger values of k. This is generally expected since there are fewer decisions that need to be made (i.e. fewer facilities that need to be closed). For exceptionally large k with respect to n_f, the Charikar algorithm always returns the optimal solution.

Type	CharikarMulti	PAM	PAM Multiswap
gauss_constant	0.1	0.42	2.54
gauss_expo	0.1	0.44	3.61
gauss_normal	0.19	0.46	3.64
uniform_constant	0.99	1.05	5.78
uniform_expo	0.93	0.97	4.84
uniform_normal	0.81	1.07	5.91

Table 5: Average Performance for $k = 7$

Type	CharikarMulti	PAM	PAM Multiswap
gauss_constant	0.0	0.0	0.0
gauss_expo	0.0	0.0	0.0
gauss_normal	0.0	0.0	0.0
uniform_constant	0.0	0.48	4.16
uniform_expo	0.0	0.34	3.98
uniform_normal	0.0	0.5	4.27

Table 6: Median Performance for $k = 7$

Type	CharikarMulti	PAM	PAM Multiswap
gauss_constant	16.4	8.31	29.45
gauss_expo	21.69	11.81	30.77
gauss_normal	24.96	9.96	43.41
uniform_constant	40.67	17.04	68.78
uniform_expo	43.79	20.52	80.5
uniform_normal	33.43	19.52	140.21

Table 7: Worst Case Performance for $k = 7$

6.1 OR-Library

	PAM	Charikar	CharikarMulti	Forward Greedy	Reverse Greedy
Average	0.2415	2.4074	0.0816	1.5742	4.1480
Median	0.1011	0.0000	0.0000	1.6173	4.2005
Worst Case	1.0558	15.1897	0.9325	3.8840	7.3854

We also ran the algorithms on instances from the OR-Library. Per problem results are located in the Appendix. Charikar Multi performed better than PAM on every instance except for 6 out of the 37 problems, where it still achieves a smaller than 1% increase over optimal.

7 Timing Analysis

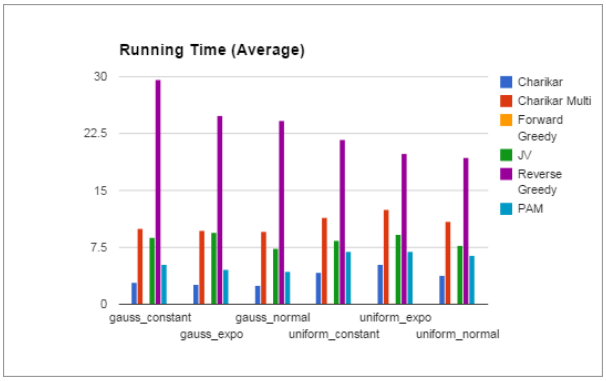

Figure 3: Average running times for the various implemented algorithms

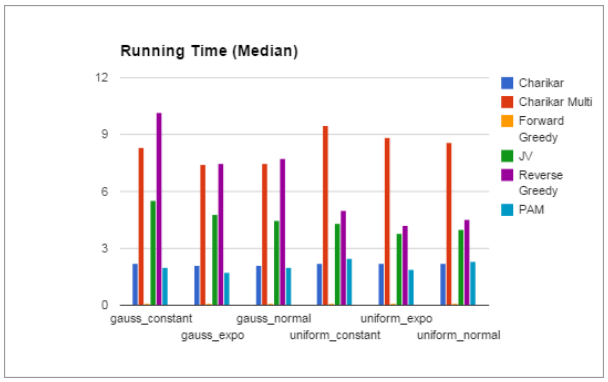

Figure 4: Median running times for the various implemented algorithms

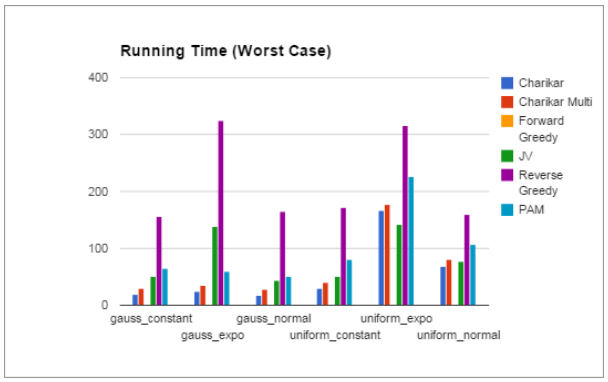

Figure 5: Worst case running times for the various implemented algorithms

Figures 3, 4, and 5 show that the forward greedy algorithm runs significantly faster on average than the others. However, the total cost of the solution it returns (as seen in Table 2) is significantly higher than that of Charikar 2012 and PAM.

After forward greedy, Charikar 2012 is the fastest on average. However, on the uniform datasets, this gain in speed comes at a slight loss in optimality of the returned solution. We note that the values for PAM are only for small datasets which it completed in a reasonable time. Charikar Multi Sample is slower than PAM in the average case on these datasets; however, it is not significantly slower.

In practice, the runtime of Charikar's rounding scheme is dominated by the time required to solve the LP. Once the LP is solved, running the rounding steps 1000 times takes only a few seconds in every instance. This characteristic makes the multiple rounding approach especially desirable since it can be done for a very low cost. The paper cites the fast algorithm for the LP presented in [16], but the algorithm does not appear to work in practice, as the number of loop iterations is prohibitively long if we wish to achieve only a small ϵ over the optimal LP value.

8 Conclusions and Future Work

In conclusion, the forward greedy algorithm runs significantly faster than the other algorithms; however, its objective function value is much higher on average than that of Charikar and PAM. Thus, we recommend using the forward greedy algorithm to provide a very fast approximation when the optimality of the solution is not a high priority.

In general, though, running the Charikar 2012 algorithm multiple times (i.e., solving the LP once, with multiple iterations of rounding) seems to be the best approach. The algorithm scales very well in practice, and the values returned are very close to optimal. Although we cannot prove a definitive upper bound on the approximation ratio when changing parameters in the Charikar algorithm, preliminary results show that a better bound is achievable.

Regarding future work, given that the Charikar algorithm performs so well in practice, we would like to see how this algorithm performs in comparison with the Li-Svensson algorithm [10], which finally broke the 3-approximation ratio in 2013. The Li-Svensson algorithm, however, is very complex with numerous implementation challenges; thus, in practice, the Charikar algorithm (even with its complexities) is more feasible for users to implement.

9 Acknowledgements

We would like to thank Professor Moses Charikar of Princeton University for all of his help regarding the 2012 LP rounding algorithm and Shi Li of Toyota Technological Institute at Chicago for his email correspondence.
We would also like to thank Professor Sanjeev Arora for all of his time and guidance throughout the process.

References

[1] Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. ii: The p-medians. *SIAM Journal on Applied Mathematics*, 37(3):539–560, 1979.

[2] Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 731–740. ACM, 2002.

[3] M.L. Balinski. On finding integer solutions to linear programs.

[4] Marek Chrobak, Claire Kenyon, and Neal E. Young. The reverse greedy algorithm for the metric k-median problem. In *Proceedings of the 11th Annual International Conference on Computing and Combinatorics*, COCOON'05, pages 654–660, Berlin, Heidelberg, 2005. Springer-Verlag.

[5] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[6] Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem (extended abstract). In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 1–10, New York, NY, USA, 1999. ACM.

[7] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, March 2001.

[8] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *J. ACM*, 50(6):795–824, November 2003.

[9] Moses Charikar and Shi Li. A dependent lp-rounding approach for the k-median problem. In *Automata, Languages, and Programming*, pages 194–205. Springer, 2012.

[10] Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 901–910, New York, NY, USA, 2013. ACM.

[11] Chandrashekhar Nagarajan and David P Williamson. An experimental evaluation of incremental and hierarchical k-median algorithms. In *Experimental Algorithms*, pages 169–180. Springer, 2011.

[12] Sayan Bhattacharya, Parinya Chalermsook, Kurt Mehlhorn, and Adrian Neumann. New approximability results for the robust k-median problem. *arXiv preprint arXiv:1309.4602*, 2013.

[13] Pranjal Awasthi, Afonso S Bandeira, Moses Charikar, Ravishankar Krishnaswamy, Soledad Villar, and Rachel Ward. Relax, no need to round: integrality of clustering formulations. *arXiv preprint arXiv:1408.4045*, 2014.

[14] Leonard Kaufman and Peter J. Rousseeuw. *Partitioning Around Medoids (Program PAM)*, pages 68–125. John Wiley & Sons, Inc., 2008.

[15] Lecture 5: Primal-dual algorithms and facility location. `https://www.cs.cmu.edu/~anupamg/adv-approx/lecture5.pdf`. Accessed: 2015-01-12.

[16] Neal E Young. K-medians, facility location, and the chernoff-wald bound. *arXiv preprint cs/0205047*, 2002.

[17] John E Beasley. Or-library: distributing test problems by electronic mail. *Journal of the operational research society*, pages 1069–1072, 1990.

Appendix

See Github link *https://github.com/bmatejek525/521final* for Julia implementations.

Further tables below.
(Note, a value of N/A occurs when the algorithm cannot complete in the time allotted.)

Instance #	Opt Value	Pam	Charikar	Charikar Multi	Forward Greedy	Reverse Greedy	JV
1	5819.0000	0.0000	0.0000	0.0000	1.2373	0.1375	1.3576
2	4093.0000	0.2932	6.2057	0.0000	0.6108	1.1972	6.9387
3	4250.0000	0.0000	9.4824	0.0000	3.5059	3.8353	7.4824
4	3034.0000	0.3955	0.0000	0.0000	1.7798	4.5814	9.0310
5	1355.0000	0.0000	0.0000	0.0000	1.7712	2.8782	5.7565
6	7824.0000	0.0000	0.6646	0.0000	2.5946	2.2367	4.6907
7	5631.0000	0.2486	0.0000	0.0000	0.2664	2.6105	5.2921
8	4445.0000	0.2700	0.0000	0.0000	0.6074	3.5321	5.7368
9	2734.0000	0.6950	0.0000	0.0000	3.8040	6.8764	4.1697
10	1255.0000	0.6375	0.0000	0.0000	2.3108	5.0996	6.6135
11	7696.0000	0.0000	15.1897	0.0000	0.3248	3.4433	5.8342
12	6634.0000	0.0000	0.8592	0.0603	0.2563	5.4417	4.6277
13	4374.0000	0.0000	0.0000	0.0000	2.1719	5.4412	6.1043
14	2968.0000	0.1011	1.7857	0.0000	1.6173	4.4474	6.0310
15	1729.0000	0.5205	0.0000	0.0000	1.9086	5.4367	NA
16	8162.0000	0.0000	0.8454	0.0000	0.8576	1.5192	5.8197
17	6999.0000	0.0000	6.7152	0.0286	0.2858	4.3149	NA
18	4809.0000	0.0416	0.2911	0.0000	1.4764	4.2005	7.5276
19	2845.0000	0.4921	0.0000	0.0000	2.0035	6.3620	9.0685
20	1789.0000	0.8385	0.0000	0.0000	3.5774	6.1487	5.9810
21	9138.0000	0.0000	0.0000	0.0000	0.0000	3.7645	5.8875
22	8579.0000	1.0491	14.6054	0.9325	1.0607	1.9116	11.5748
23	4619.0000	0.0000	0.0000	0.0000	1.4722	4.1567	7.7506
24	2961.0000	0.2026	0.0000	0.0000	1.8913	6.6869	5.9102
25	1828.0000	0.6565	0.0000	0.0000	3.8840	6.7834	444.1466
26	9917.0000	0.0000	1.5428	0.0202	1.7747	2.3192	NA
27	8307.0000	0.0000	8.0173	0.6982	0.7343	2.3474	3.9364
28	4498.0000	0.3335	0.0000	0.0000	1.8008	5.5358	4.8466
29	3033.0000	0.1319	0.0000	0.0000	2.3739	7.3854	NA
30	1989.0000	1.0558	0.0000	0.0000	2.7652	6.6365	NA
31	10086.0000	0.0000	11.5308	0.0000	0.0000	2.6472	3.5296
32	9297.0000	0.0430	2.1405	0.3550	0.3657	2.6675	6.8732
33	4700.0000	0.4468	0.0000	0.0000	2.1277	4.5532	NA
34	3013.0000	0.3651	0.3983	0.0000	3.2858	5.6422	NA
35	10400.0000	0.0000	2.5000	0.0096	0.0577	3.4519	NA
36	9934.0000	0.0000	6.2211	0.9160	0.2013	2.1240	5.2949
37	5057.0000	0.1186	0.0791	0.0000	1.4831	5.1216	5.7939

Table 8: Algorithm performances on specific OR-Lib instances as a percentage increase over the optimal value (first column)

Type	Charikar	Charikar Multi	F. Greedy	JV	R. Greedy	PAM	PAM Multi
gauss_constant	0.0039	0.0002	0.1988	2.2167	0.0015	0.0007	N/A
gauss_expo	0.0008	0.0000	0.0999	1.4488	0.0001	0.0002	N/A
gauss_normal	0.0042	0.0001	0.1797	1.6822	0.0005	0.0002	N/A
uniform_constant	0.0487	0.0021	1.6038	2.4441	0.0278	0.0150	N/A
uniform_expo	0.0387	0.0020	1.3466	2.4828	0.0240	0.0122	N/A
uniform_normal	0.0383	0.0016	1.5176	2.5010	0.0203	0.0105	N/A

Table 9: Average for $k = 100$

Type	Charikar	Charikar Multi	F. Greedy	JV	R. Greedy	PAM	PAM Multi
gauss_constant	0.0000	0.0000	0.0000	2.0783	0.0000	0.0000	N/A
gauss_expo	0.0000	0.0000	0.0000	1.2668	0.0000	0.0000	N/A
gauss_normal	0.0000	0.0000	0.0000	1.5163	0.0000	0.0000	N/A
uniform_constant	0.0000	0.0000	1.0489	2.5898	0.0000	0.0000	N/A
uniform_expo	0.0000	0.0000	0.0087	2.5640	0.0000	0.0000	N/A
uniform_normal	0.0000	0.0000	0.5887	2.5582	0.0000	0.0000	N/A

Table 10: Median for $k = 100$

Type	Charikar	Charikar Multi	F. Greedy	JV	R. Greedy	PAM	PAM Multi
gauss_constant	1.2833	0.0744	4.2600	5.8823	0.2132	0.0991	N/A
gauss_expo	0.5179	0.0207	2.9500	6.1905	0.0271	0.0833	N/A
gauss_normal	2.1163	0.0419	4.5890	6.5022	0.0784	0.0600	N/A
uniform_constant	2.6670	0.1295	7.1931	7.0728	0.3823	0.3803	N/A
uniform_expo	3.0713	0.1680	8.4991	7.8906	0.4326	0.3390	N/A
uniform_normal	3.1713	0.1509	7.4081	6.9296	0.3807	0.3190	N/A

Table 11: Worst case for $k = 100$

Type	Charikar	Charikar Multi	F. Greedy	JV	R. Greedy	PAM	PAM Multi
gauss_constant	0.0000	0.0000	0.0033	0.3966	0.0000	0.0000	N/A
gauss_expo	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	N/A
gauss_normal	0.0000	0.0000	0.0000	N/A	0.0000	0.0000	N/A
uniform_constant	0.0008	0.0000	0.1264	0.7617	0.0000	0.0000	N/A
uniform_expo	0.0012	0.0001	0.1138	1.2554	0.0000	0.0000	N/A
uniform_normal	0.0003	0.0000	0.1003	0.8804	0.0000	0.0000	N/A

Table 12: Average for $k = 200$

Type	Charikar	Charikar Multi	F. Greedy	JV	R. Greedy	PAM	PAM Multi
gauss_constant	0.0000	0.0000	0.0000	0.2486	0.0000	0.0000	N/A
gauss_expo	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	N/A
gauss_normal	0.0000	0.0000	0.0000	N/A	0.0000	0.0000	N/A
uniform_constant	0.0000	0.0000	0.0000	0.5697	0.0000	0.0000	N/A
uniform_expo	0.0000	0.0000	0.0000	1.1451	0.0000	0.0000	N/A
uniform_normal	0.0000	0.0000	0.0000	0.6042	0.0000	0.0000	

Table 13: Median for $k = 200$

Type	Charikar	Charikar Multi	F. Greedy	JV	R. Greedy	PAM	PAM Multi
gauss_constant	0.0000	0.0000	0.8734	2.8229	0.0000	0.0000	N/A
gauss_expo	0.0000	0.0000	0.0477	0.0000	0.0000	0.0000	N/A
gauss_normal	0.0000	0.0000	0.0235	N/A	0.0000	0.0000	N/A
uniform_constant	0.6170	0.0109	1.8969	3.1352	0.0000	0.0000	N/A
uniform_expo	0.3748	0.0398	2.1153	3.9932	0.0000	0.0000	N/A
uniform_normal	0.2204	0.0065	2.0389	3.4834	0.0000	0.0000	N/A

Table 14: Worst case for $k = 200$

Type	Charikar	Charikar Multi	F. Greedy	JV	R. Greedy	PAM	PAM Multi
gauss_constant	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
gauss_expo	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
gauss_normal	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_constant	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_expo	0.0000	0.0000	0.0030	0.3637	N/A	N/A	N/A
uniform_normal	0.0000	0.0000	0.0007	0.8213	N/A	N/A	N/A

Table 15: Average for $k = 300$

Type	Charikar	Charikar Multi	F. Greedy	JV	R. Greedy	PAM	PAM Multi
gauss_constant	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
gauss_expo	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
gauss_normal	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_constant	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_expo	0.0000	0.0000	0.0000	0.3315	N/A	N/A	N/A
uniform_normal	0.0000	0.0000	0.0000	0.6219	N/A	N/A	N/A

Table 16: Median for $k = 300$

Type	Charikar	Charikar Multi	F. Greedy	JV	R. Greedy	PAM	PAM Multi
gauss_constant	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
gauss_expo	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
gauss_normal	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_constant	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_expo	0.0000	0.0000	0.2742	0.8516	N/A	N/A	N/A
uniform_normal	0.0000	0.0000	0.3489	1.4005	N/A	N/A	N/A

Table 17: Worst case for $k = 300$

Type	Charikar	Charikar Multi	F. Greedy	JV	R. Greedy	PAM	PAM Multi
gauss_constant	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
gauss_expo	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
gauss_normal	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_constant	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_expo	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_normal	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A

Table 18: Average for $k = 400$

Type	Charikar	Charikar Multi	F. Greedy	JV	R. Greedy	PAM	PAM Multi
gauss_constant	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
gauss_expo	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
gauss_normal	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_constant	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_expo	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_normal	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A

Table 19: Median for $k = 400$

Type	Charikar	Charikar Multi	F. Greedy	JV	R. Greedy	PAM	PAM Multi
gauss_constant	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
gauss_expo	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
gauss_normal	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_constant	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_expo	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A
uniform_normal	0.0000	0.0000	0.0000	N/A	N/A	N/A	N/A

Table 20: Worst case for $k = 400$