An experimental evaluation of incremental and hierarchical k-median algorithms

Chandrashekhar Nagarajan* David P. Williamson†

January 21, 2011

Abstract

In this paper, we consider different incremental and hierarchical k-median algorithms with provable performance guarantees and compare their running times and quality of output solutions on different benchmark k-median datasets. We determine that the quality of solutions output by these algorithms for all the datasets is much better than their performance guarantees suggest. Since some of the incremental k-median algorithms require approximate solutions for the k-median problem, we also compare some of the existing k-median algorithms' running times and quality of solutions obtained on these datasets.

1 Introduction

A company is building facilities in order to supply its customers. Because of limited capital, it can only build a few at this time, but intends to expand in the future in order to improve its customer service. Its plan for expansion is a sequence of facilities that it will build in order as it has funds. Can it plan its future expansion in such a way that if it opens the first k facilities in its sequence, this solution is close in value to that of an optimal solution that opens any choice of k facilities? The company's problem is the *incremental* k-median problem, and was originally proposed by Mettu and Plaxton [10].

The standard k-median problem has been the object of intense study in the algorithms community in recent years. Given the locations of a set of facilities and a set of clients in a metric space, and a parameter k, the *k-median problem* asks to find a set of k facilities to *open* such that the sum of the distances of the clients to the nearest open facility is minimized. Since the metric k-median problem is NP-hard [8], many researchers have focused on obtaining approximation algorithms for it. An α-approximation algorithm for a minimization problem runs in polynomial time and outputs a solution whose cost is at most α times the cost of the optimal solution. The factor α is sometimes called the *approximation factor* or *performance guarantee* of the algorithm. A solution for which the cost is at most α times the optimal cost is sometimes called *α-approximate*. The best approximation algorithm known for this problem has a performance guarantee of $3 + \epsilon$ and is due to Arya, Garg, Khandekar, Meyerson, Munagala and Pandit [2]; it is based on a local search heuristic.

*Yahoo! Inc., Sunnyvale, CA, 94089. Email: cn54@yahoo-inc.com. Supported in part by NSF grant CCF 0514528.

†School of Operations Research and Information Engineering, Cornell University, Ithaca, NY, 14853. Email: dpw@cs.cornell.edu. Supported in part by NSF grant CCF 0514528.

In the incremental k-median problem, we are given the input of the k-median problem without the parameter k and must produce a sequence of the facilities. For each k, consider the ratio of the cost of opening the first k facilities in the ordering to the cost of an optimal k-median solution. The goal of the problem is to find an ordering that minimizes the maximum of this ratio over all values of k. An algorithm for the problem is said to be α-*competitive* if the maximum of the ratio over all k is no more than α. This value α is called the *competitive ratio* of the algorithm. Mettu and Plaxton [10] gave a 29.86-competitive algorithm for the incremental k-median problem. Later Lin, Nagarajan, Rajaraman and Williamson [9] gave deterministic 16-competitive and randomized 10.88-competitive algorithms for the incremental k-median problem[1]. Their algorithms use either a k-median approximation algorithm or a Lagrangean Multiplier Preserving (LMP) facility location algorithm as a black box.

We also consider algorithms for the hierarchical k-median problem. In hierarchical clustering, we give clusterings with k clusters for all values of k by starting with each point in its own cluster and repeatedly merging selected pairs of clusters until all points are in a single cluster. We also consider a variation of this problem in which each cluster has a point designated as its center, and when we merge two clusters together to form a single cluster, one of the two centers becomes the center of the new cluster. Given some objective function on a k-clustering, again we would like to ensure that for any k, the cost of our k-clustering obtained in this way is not too far away from the cost of an optimal k-clustering. For the hierarchical k-median problem, the objective function for the k-clustering is its k-median cost; that is, the sum of the distances of each point to its cluster center. Plaxton [11] gave a 238.88-competitive algorithm for the problem. Lin et al. [9] later gave deterministic 40.42-competitive and randomized 20.06-competitive algorithms for the problem. Their algorithms again use either a k-median approximation algorithm or a LMP facility location algorithm as a black box.

In this paper, we consider the performance of these incremental and hierarchical k-median algorithms on different k-median benchmark datasets and compare their running times and quality of output solutions. Since the algorithms of Lin et al. require a k-median approximation algorithm or a LMP facility location algorithm as a black box, we also compare the performance of some of the existing k-median and LMP facility location algorithms. In particular, we implement five different k-median and LMP facility location algorithms. The first one is the single swap local search algorithm by Arya et al. [2], which gives 5-approximate solutions. We also consider the linear program (LP) rounding algorithm of Charikar, Guha, Tardos and Shmoys [4] which rounds the LP optimum to get 8-approximate solutions. Jain, Mahdian, Markakis, Saberi and Vazirani [7] give a greedy dual-fitting Lagrangean Multiplier Preserving (LMP) Facility Location (FL) algorithm which gives 2-approximate k-median solutions for some values of k. We also consider the standard k-median linear program and solve it optimally using CPLEX. The optimal solution can be fractional but still gives a good lower bound for the k-median problem. We also solve the k-median integer program optimally using CPLEX even though the algorithm is not polynomial time. These linear and integer programs give us bounds on the quality of the solutions of the other algorithms.

Given these algorithms, we implement several variants of the Lin et al. algorithms for the incremental k-median problem. We implement their algorithm using the Arya et al. local search algorithm for k-median, the Charikar et al. LP rounding algorithm for k-median, and the Jain et al. greedy algorithm which is an LMP algorithm for facility location. Additionally, we implement the original algorithm of Mettu and Plaxton for the incremental k-median problem. We are able

[1]Some of the results of Lin et al. were obtained independently by Chrobak, Kenyon, Noga, and Young [5].

to use the linear and integer programming solutions to bound the quality of the results we obtain.

We also implement several variants of the Lin et al. algorithms for hierarchical k-median problem. Again, we implement their algorithm using the Arya et al. local search algorithm for k-median, the Charikar et al. LP rounding algorithm, and the Jain et al. greedy algorithm. Additionally, we implement Plaxton's algorithm for the hierarchical k-median problem. Plaxton's algorithm requires an incremental k-median algorithm as a black box, and originally used the algorithm of Mettu and Plaxton as a subroutine. We implement this variant of Plaxton's algorithm, and also a variant that uses Lin et al.'s algorithm given the Arya et al. local search algorithm.

We test our algorithms on 43 different k-median instances drawn from the literature. In particular, we use forty instances from the OR Library [3], two instances from Galvão and ReVelle [6], and one instance from Alp, Erkut, and Drezner [1].

From the results we obtained we determine that all these algorithms perform much better in terms of quality of solution than their respective competitive/approximation ratios suggest. In particular, while we know of no polynomial-time algorithm with a competitive ratio better than 10 for the incremental and hierarchical median k-median problems, we typically obtained results which were within 10% of the k-median LP relaxation for incremental problems and 20% of the k-median LP relaxation for hierarchical k-median problems. We find this quite surprising in view of the strong constraints required on the structure of solutions for the incremental and hierarchical problems.

The algorithms of Mettu and Plaxton for incremental k-median and Plaxton for hierarchical k-median produce solutions that are not as good as those of Lin et al.; however, our implementation of the Mettu-Plaxton algorithm is significantly faster than our implementations of the Lin et al. algorithms, at least in part because the Lin et al. algorithms require approximate solutions of the k-median problem for all values of k.

Our paper is structured as follows. In Section 2, we give the details of the various algorithms we implemented. In Section 3, we discuss the datasets we used. In Section 4, we give the experimental results we obtained. In Section 5, we give our conclusions as well as some open problems prompted by our work. For space reasons, some detailed statements of the algorithms and complete tables of results can be found in the Appendix.

2 Algorithms

In this section, we discuss the various algorithms we implemented for the k-median, incremental k-median, and hierarchical k-median problems respectively.

2.1 The k-median problem

In this section we consider five different algorithms for the k-median problem: the single swap local search algorithm by Arya et al. [2]; the linear program (LP) rounding algorithm of Charikar et al. [4] which rounds the LP optimum to get an integer solution which is no more than 8 times the cost of the optimal LP solution; the Jain et al. [7] greedy dual-fitting Lagrangean Multiplier Preserving (LMP) Facility Location (FL) algorithm, which gives 2-approximate k-median solutions for some values of k; the standard k-median linear program, which we solve optimally using CPLEX; and the k-median integer program, which we also solve optimally using CPLEX even though the algorithm is not polynomial time. The optimal solution to the linear program can be fractional but still gives a good lower bound for the k-median problem. We now discuss each of these algorithms in turn.

2.1.1 Local Search Algorithm of Arya et al.

We consider the Arya et al.'s ([2]) single swap local search algorithm which computes a 5-approximate solution. The local search algorithm proceeds by starting with an arbitrary solution and repeatedly doing *valid swaps* on the current solution till no more valid swaps exist. A swap closes a facility in the current solution and opens a facility that was previously closed. A swap is considered valid if the cost of the new solution after swapping is less than the cost of the solution before swapping.

Arya et al. proved that the local search algorithm can be made to run in time polynomial in the input size by considering a swap as valid only if it improves the cost of the solution by a value that is an inverse polynomial in the input size. However, for simplicity, we consider any cost-improving swap as a valid swap. We run this local search algorithm for each cardinality k. After this procedure we have locally optimal solutions for each value of k.

We do not implement the multi-swap (swaps involving more than one facilities) local search algorithm by Arya et al. because of its high running time even though it gives better approximation guarantee of $3 + \epsilon$. We use the locally optimal solution of cardinality $k - 1$ as a starting solution for the local search iteration for cardinality k. Since this solution is already a good solution for cardinality k we reduce the running times of the subsequent iterations. On average this improves the running times of local search by about 40%.

2.1.2 LP rounding algorithm of Charikar et al.

We consider the LP rounding algorithm of Charikar et al. [4] which takes in the fractional optimal solution of the standard LP relaxation $(k - P)$ of the k-median problem and produces an integer solution that is no more than 8 times the cost of LP optimum. The LP relaxation is:

$$
\begin{aligned}
& \text{Min} && \sum_{i \in F, j \in C} c_{ij} x_{ij} && \\
& \text{subject to:} && && \\
& && \sum_{i \in F} x_{ij} = 1 && \forall j \in C \\
& && x_{ij} \leq y_i && \forall i \in F, \forall j \in C \\
(k - P) \qquad & && \sum_{i \in F} y_i \leq k && \\
& && x_{ij} \geq 0 && \forall i \in F, \forall j \in C \\
& && y_i \geq 0 && \forall i \in F.
\end{aligned}
$$

The algorithm is as follows. It starts with the optimal LP fractional solution for a particular value of k. First, the algorithm simplifies the problem instance by consolidating nearby locations and combining their demands such that the locations with nonzero demands are far from each other the resulting problem instance. It then simplifies the structure of the optimal fraction solution by consolidating nearby fractional centers. The resulting solution has nonzero fractional y value only on facilities with nonzero demands and their y values are no less than $\frac{1}{2}$. The algorithm then modifies this solution to a $\{0, \frac{1}{2}, 1\}$ solution where the y values take values of only 0, $\frac{1}{2}$ and 1. It then opens no more than k of these facilities with non-zero y values, selecting them based on their distance to other facilities with positive y values. This algorithm is explained in more detail in Algorithm 1, which is given in the Appendix.

2.1.3 Greedy LMP FL Algorithm of Jain et al.

Jain et al. [7] give a LMP greedy dual-fitting algorithm for the facility location problem. In this algorithm, we maintain a dual value v_j for every client which is its total contribution to getting connected to a open facility. Some part of this dual v_j pays for the j's connection cost and the remainder is paid toward facility opening costs. We increase the duals of the clients uniformly and open a facility when a facility has enough contribution from the clients to match the facility opening cost. We say a client is connected to a facility if the connection cost is paid for by its dual value. We stop increasing the dual for a client if it is connected to a open facility. The algorithm is explained in more detail in Algorithm 2, which is given in the Appendix.

Since this facility location algorithm is a LMP 2-approximation algorithm for the facility location (FL) problem, we can obtain something called a *bounded envelope* for the k-median problem as described in Lin et al. [9]. The bounded envelope gives 2-approximate solutions for the k-median problem for some values of k as well as a corresponding piecewise linear lower bound on the values of k-median solutions for all values of k, where the breakpoints of the lower bounds occur at values of k for which we have 2-approximate solutions. Lin et al. give a procedure for computing the bounded envelope given the LMP FL algorithm.

2.1.4 Solving Linear Program using CPLEX

We solve the linear programming relaxation $(k-P)$ of the standard k-median problem using the CPLEX solver.

To speed up the running time of the linear program solver, we tried to give the optimal solution of $(k-1)^{th}$ run as an initial starting solution to the iteration of cardinality k for all values of k. But there was no significant improvement of the running times of the linear programs on average.

2.1.5 Solving Integer Program using CPLEX

We solve the integer program $(k-IP)$ optimally the CPLEX solver; $(k-IP)$ is the same as $(k-P)$ except that we require $x_{ij} \in \{0,1\}$ and $y_i \in \{0,1\}$. The CPLEX solver provides a way to give a good initial guess to the solver so that it can prune many low quality solutions. We give the optimal integer solution with $k-1$ facilities as an initial guess for the CPLEX integer program iteration with cardinality k. As the optimal solution for the k-median problem for a smaller value of cardinality is a feasible solution for the k-median problem with larger cardinality, the initial guess is feasible. Even though this makes the solver find the optimal integral solution faster in some cases, it does not work in all cases and on average the improvement in running time is not significant.

2.2 Incremental k-median

In this section we briefly explain the Mettu and Plaxton's incremental k-median algorithm and Lin et al.'s incremental k-median algorithm.

2.2.1 Mettu and Plaxton's Algorithm

Mettu and Plaxton's [10] incremental k-median algorithm uses a *hierarchical greedy approach* to choose the next facility in the incremental order to be opened. The basic idea behind this approach is as follows. Rather than selecting the next point in the ordering based on a single greedy criterion,

they greedily choose a region and then recursively choose smaller regions till they arrive at a single facility which then becomes the next facility to open. Thus the choice of the next facility is influenced by a sequence of greedy criteria addressing successive finer levels of granularity. We give the details of the algorithm in the Appendix.

2.2.2 Lin et al.'s incremental k-median algorithm

We implement the incremental algorithm AltIncApprox of Lin et al. [9] for the incremental k-median problem on these datasets. We use Arya et al.'s local search algorithm with single swaps and the LP rounding technique of Charikar et al. to generate good k-median solutions for all possible k for each of these datasets. We bucket these solutions into buckets of geometrically increasing cost. We take the costliest solution from each bucket. We then consider each of these solutions in order of decreasing number of medians, and use each such solution to find another solution with the same number of medians that is contained with the next larger solution. This gives us a sequence of k-median solutions such that any smaller solution is a subset of any larger solution. This sequence of solutions gives a natural ordering of the facilities.

We also implement the incremental algorithm BoundedIncApprox of Lin et al. [9] using the k-median bounded envelope obtained by running the Jain et al. algorithm explained in Algorithm 2 on the datasets. By using the 2-approximate solutions obtained from this algorithm for some values of k, we can apply the procedure given above to obtain an ordering of the facilities.

More details of both algorithms are given in the Appendix.

2.3 Hierarchical k-median

We test the hierarchical k-median algorithms of Lin et al. [9] against the previously known hierarchical k-median algorithm by Plaxton [11].

2.3.1 Plaxton's Algorithm

Plaxton's algorithm takes in an incremental k-median solution as input and finds a *parent* function for each facility this incremental ordering. A hierarchical k-median solution obtained from an ordering can be considered as solutions obtained by repeatedly closing the last open facility in ordering and assigning its clients to an earlier facility. This mapping is exactly captured by the parent function in the Plaxton's algorithm. A parent function for an ordering maps every facility in the order to a facility that is earlier in the ordering. The parent of a facility is the facility that its clients will get assigned to when the facility is closed.

Plaxton's parent function is assigned as follows: Given an incremental k-median solution to the problem, a parent is assigned to every facility in the reverse order of the incremental solution. The parent of a facility f is determined by the earliest facility in the ordering that is either the closest facility or satisfies a certain equation. The equation essentially finds a facility whose distance to f is no more than the average distance of f's clients to f. See Algorithm 5 in the Appendix.

We run the Plaxton's parent function algorithm on the incremental k-median solutions given by running the Mettu and Plaxton's algorithm (Section 2.2.1) and AltIncApprox algorithm (Section 2.2.2) using Arya et al.'s local search solutions on the datasets.

2.3.2 Lin et al.'s hierarchical k-median algorithm

We run the generic algorithm AltIncApprox of Lin et al. [9] for the hierarchical k-median problem on the datasets using different k-median algorithms as black box. We use Arya et al.'s local search algorithm and Charikar et al.'s LP rounding algorithm to generate good k-median solutions. We also implement the incremental algorithm BoundedIncApprox of Lin et al. [9] using the k-median bounded envelope obtained by running Jain et al. algorithm on the datasets. As in the incremental k-median algorithm of Lin et al. we must find approximate solutions to the k-median problem, which we then put in buckets of geometrically increasing cost, then take the costliest solution from each bucket. We consider these solutions in order of decreasing size, and use each solution to find a k-clustering that is consistent with a hierarchical clustering on the larger solutions already considered.

3 Datasets

In our experiments we use these following sets of datasets for the comparison of k-median, incremental k-median and the hierarchical k-median algorithms.

1. *OR Library:* These 40 datasets of the uncapacitated k-median problems are part of the OR Library [3] which is a collection of test datasets for a variety of OR problems created by J.E.Beasley. These 40 test problems are named *pmed*1, *pmed*2, ..., *pmed*40 and their sizes range from $n = 100$ to 900. As noted in [3], we apply Floyd's algorithm on the adjacency cost matrix in order to obtain the complete cost matrix.

2. *Galvão:* This set of instances (*Galvão100* and *Galvão150*) is obtained from the work of Galvão and ReVelle [6]. Even though the sizes of these datasets are small ($n = 100$ and $n = 150$), the integrality gaps for some values of k (number of medians) are larger than traditional datasets.

3. *Alberta:* This dataset is generated from a 316-node network using all population centers in Alberta (see Alp, Erkut and Drezner [1]) where the distances are computed using the shortest path metric on the actual road network of Alberta.

4 Experimental Results

4.1 The k-median problem

In this section we compare the performance in terms of running times and quality of solutions of five different algorithms on the datasets described: CPLEX solver for the k-median linear program, CPLEX solver for k-median integer program, Arya et al.'s single swap local search algorithm, Charikar et al.'s LP rounding algorithm and then the bounded envelope of Jain et al.'s greedy algorithm. All experiments were done on machines with Intel Core 2 2.40GHz processor with 2 gigabytes of physical memory. The linear programs and integer programs on the data sets are solved using CPLEX Version 10.1.0. The Arya et al.'s single swap local search algorithm and Jain et al. algorithm are solved using MATLAB version 7.0. The tolerance for the bounded envelope that we use for the termination of binary search is 0.01 (see Lin et al. [9] for the bounded envelope procedure).

Table 1 in the Appendix shows the average and maximum ratios of the costs of the integer optimum solution (IP OPT), Arya et al.'s local optimum solutions (Local) and Charikar et al.'s LP rounding solutions (LPR) to the linear program optimum (LP OPT) over all values of k for each of the datasets. Even though the Arya et al.'s algorithm's performance guarantee is 5, in practice the local search algorithm performs much better than that. The local optimums are within 1% from the linear program optimum on average. Charikar's et al.'s LP rounding algorithm performs even better as most of the LP solutions are already integral or very close to being integral except for some small values of k. The $*$ in the tables represent unavailable data as the corresponding algorithmic procedure timed out.

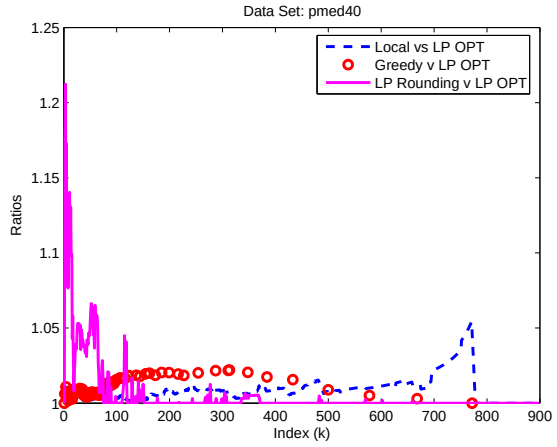

Figure 1: Quality of solutions of k-median algorithms (dataset *pmed*40)

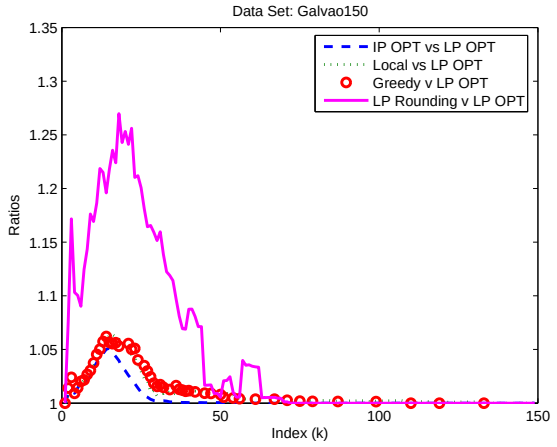

Figure 2: Quality of solutions of k-median algorithms (dataset *Galvão150*)

Figures 1 and 2 show how the costs of the k-median solutions from the integer optimum, Arya et al.'s local search algorithm, Charikar et al.'s LP rounding algorithm and the Jain et al.'s greedy algorithm compare to the linear program for different values of k for two sample datasets *pmed*40 and *Galvão150* (see also Figures 7 and 8 in the Appendix). Note that the Jain et al.'s greedy LMP FL algorithm gives only a bounded envelope and does not give k-median solutions for all values of k. Here we can see that the LP rounding algorithm and the local search algorithm perform better than Jain et al.'s algorithm.

Table 2 in the Appendix gives the times in seconds of the runs of each of the above mentioned algorithms on each of the data sets for all values of k. For the LP and IP columns, each time entry denotes the sum of the times taken for the CPLEX solves over all values of k for that particular dataset. Note that the LP solver runs faster than the local search and greedy algorithm for all datasets. Also note that the IP solver takes a lot more time to solve all the instances of k for bigger datasets.

4.2 Incremental k-median

In this section we compare the performances of four different incremental k-median algorithms on the selected datasets: Mettu and Plaxton's incremental k-median algorithm (MPInc), Lin et al.'s AltIncApprox algorithm with solutions from the Arya et al.'s single swap local search algorithm (LInc) and Charikar et al.'s LP rounding (LPR) and Lin et al.'s BoundedIncApprox algorithm

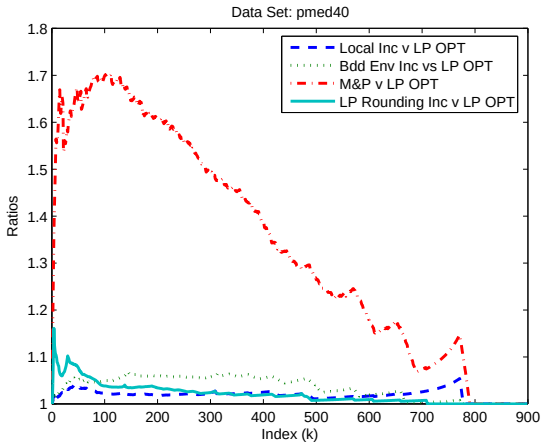

Figure 3: Quality of solutions of incremental k-median algorithms (dataset *pmed*40)

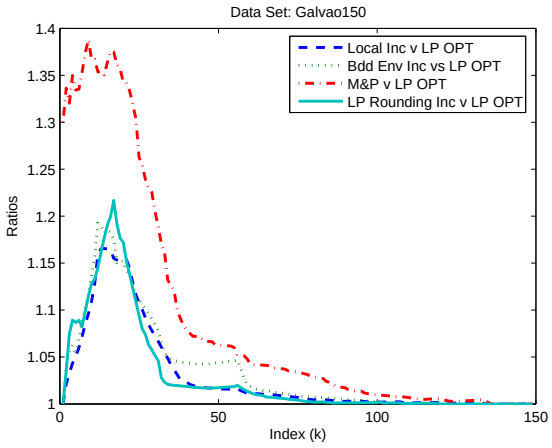

Figure 4: Quality of solutions of incremental k-median algorithms (dataset *Galvão150*)

with the bounded envelope obtained from the Jain et al.'s greedy LMP FL algorithm (GInc). We pick $\lambda = 1, \alpha = 3, \beta = 2, \gamma = 15$ for the Mettu-Plaxton algorithm so that they satisfy the equations in Section 6.3.

The third and fourth column of Table 3 in the Appendix shows the average and maximum ratios of the costs of the incremental k-median solution obtained from the AltIncApprox algorithm using Arya et al.'s local search k-median solutions (LInc) to the linear program optimum (LP OPT) for each of the datasets. The fifth and sixth columns give the corresponding average and maximum value ratios for the incremental k-median solution costs of the BoundedIncApprox using bounded envelope obtained by running Jain et al.'s algorithm. The next two columns give the corresponding average and maximum ratios for the Mettu and Plaxton's incremental k-median algorithm and the last two columns give the ratios for Charikar et al.'s LP rounding algorithm. From the Table we infer that Lin et al.'s algorithms perform much better than the Mettu and Plaxton's algorithm on the datasets. This inference is reinforced by Figures 3 and 4 which show that the ratios of the costs of solutions obtained from Lin et al.'s incremental algorithms to the LP optimum are always better than the corresponding ratios of Mettu and Plaxton's algorithm for a sample of two datasets (*pmed*40 and *Galvão*; see also Figures 9 and 10 in the Appendix).

Tables 4 in the Appendix gives the times in seconds of the runs of each of the above mentioned algorithms on each of the data sets for all values of k. Note that the Mettu-Plaxton algorithm runs much faster than Lin et al.'s algorithms; these use a k-median algorithm or a bounded envelope algorithm as a blackbox, which make them very slow. However the quality of the incremental solutions obtained from Lin et al.'s algorithm is much better than that of the Mettu-Plaxton algorithm.

4.3 Hierarchical k-median

In this section we compare the performance of Plaxton's hierarchical k-median algorithm against Lin et al.'s AltIncApprox hierarchical k-median algorithm on the datasets. Note that Plaxton's algorithm takes in any incremental k-median solution as input and outputs a parent function which defines the hierarchical solution. We give the incremental k-median solutions from our runs

of AltIncApprox and the Mettu-Plaxton algorithms as input to the Plaxton's hierarchical algorithm (PHLI and PHMP) and compare them against Lin et al.'s hierarchical k-median algorithms' solutions (HL, HG and LPRH) for different datasets.

Figures 5 and 6 show how the costs of the hierarchical k-median solutions for different algorithms compare against the optimal linear program solutions for different values of k for two sample datasets *pmed*40 and *Galvão150* (see also Figures 11 and 12 in the Appendix). The algorithms we consider are AltIncApprox algorithm (using Arya et al.'s local search k-median solutions (HL) and Charikar et al.'s LP rounding solutions (LPRH)), BoundedIncApprox algorithm (using bounded envelope from Jain et al.'s greedy algorithm) (HG), Plaxton's hierarchical k-median algorithm on the incremental solutions of AltIncApprox algorithm (PHLI) and Plaxton's algorithm on Mettu and Plaxton's incremental k-median solutions (PHMP).

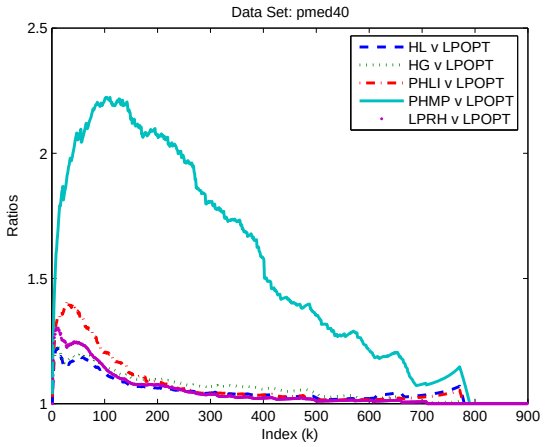

Figure 5: Quality of solutions of hierarchical k-median algorithms (dataset *pmed*40)

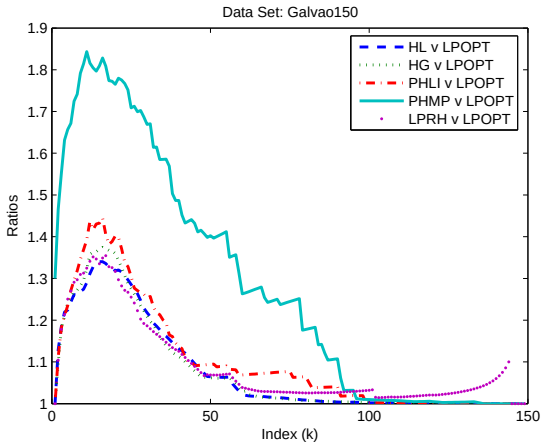

Figure 6: Quality of solutions of hierarchical k-median algorithms (dataset *Galvão150*)

Table 5 in the Appendix shows the average and maximum ratios of the costs of the hierarchical k-median solutions of different hierarchical algorithms (HL, HG, PHLI, PHMP) to the cost of linear program optimum (LP OPT) for each of the datasets.

We can see clearly that the hierarchical solutions obtained by AltIncApprox algorithms are better than other algorithms. Note that the ratios for the PHMP algorithm are not as good as for the other algorithms since PHMP uses the incremental k-median solutions of Mettu and Plaxton as input which are not as good as other incremental algorithms in terms of quality (See Table 3). Lin et al.'s hierarchical algorithm (HL) which computes hierarchical solutions directly from k-median solutions performs better than the Plaxton's hierarchical algorithm even when the incremental solutions from AltIncApprox are given as input. We do not provide a table with the running times for different hierarchical algorithms as the major chunk of the running times is contributed by the underlying incremental k-median or k-median algorithms.

5 Conclusions

We evaluate different k-median, incremental k-median and hierarchical k-median algorithms on different datasets and show our results here. For the k-median problem, Charikar et al.'s LP

rounding algorithm performs better and faster on average than other k-median algorithms like Arya et al.'s local search algorithm. We also notice that in many real-life datasets the optimal LP solution for the k-median problems for most values of k are integers which also makes the LP rounding techniques much better in terms of the quality of the solutions.

The quality of incremental solutions, when AltIncApprox algorithm is run on the k-median solutions of Arya et al.'s local search algorithm and Charikar et al.'s LP rounding algorithm, are much better than the incremental solutions of Mettu and Plaxton's algorithm. Even though the LP rounding algorithm performs poorly for some small values of k, Lin et al.'s incremental and hierarchical algorithms skips many of these poor solutions while bucketing the solutions geometrically and this makes the corresponding incremental solutions comparable in quality to the incremental solutions obtained from Arya et al.'s local search k-median solutions.

Mettu and Plaxton's incremental k-median algorithm is much faster than the other incremental k-median algorithms we implement. However one important point to note here is that we find good k-median solutions for all values of k both in Arya et al.'s local search algorithm and Charikar et al.'s LP rounding algorithm. Most of these solutions are not used at all by the Lin et al. algorithms since it uses only one solution from each of the geometrically increasing buckets. It would be useful if we would somehow be able to find a sequence of k-median solutions that are geometrically increasing in cost in a faster way; this could lead to significant improvements in the running times of the Lin et al. algorithms.

References

[1] O. Alp, E. Erkut, and D. Drezner. An efficient genetic algorithm for the p-median problem. *Annals of Operations Research*, 122:21–42(22), 2003.

[2] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33:544–562, 2004.

[3] J. E. Beasley. A note on solving large p-median problems. *European Journal of Operational Research*, 21:270–273, 1985.

[4] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem (extended abstract). In *Proceedings of the 31th Annual ACM Symposium on Theory of Computing*, pages 1–10, 1999.

[5] M. Chrobak, C. Kenyon, J. Noga, and N. E. Young. Incremental medians via online bidding. *Algorithmica*, 50:455–478, 2008.

[6] R. D. Galvao and C. ReVelle. A Lagrangean heuristic for the maximal covering location problem. *European Journal of Operational Research*, 88(1):114–123, 1996.

[7] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50:795–824, 2003.

[8] O. Kariv and S. Hakimi. An algorithm approach to network location problems ii. the p-medians. *SIAM Journal of Applied Mathematics*, 37:539–560, 1979.

[9] G. Lin, C. Nagarajan, R. Rajaraman, and D. P. Williamson. A general approach for incremental approximation and hierarchical clustering. *SIAM Journal on Computing*, 39:3633–3669, 2010.

[10] R. R. Mettu and C. G. Plaxton. The online median problem. *SIAM Journal on Computing*, 32:816–832, 2003.

[11] C. G. Plaxton. Approximation algorithms for hierarchical location problems. *Journal of Computer and System Sciences*, 72:425–443, 2006.

6 Appendix

6.1 Charikar et al.'s LP rounding algorithm

Algorithm 1 LP rounding of Charikar et al. [4]

1. Let $(\overline{x}, \overline{y})$ be the optimal LP solution to $(k-P)$. Let the demands be 1 for all locations.
2. Step 1: Let the clients be indexed in the increasing order of their objective function contribution i.e. $\overline{C}_1 \leq \overline{C}_2 \leq \ldots \leq \overline{C}_n$ where $\overline{C}_j = \sum_{i \in N} c_{ij}\overline{x}_{ij}$.

   For $j = 1$ to n

   If there is another location $i < j$ such that the demand $d_i > 0$ and $c_{ij} \leq 4\overline{C}_j$ and set $d_i \leftarrow d_i + d_j$ and $d_j \leftarrow 0$.

   Let $N = \{r : d_r > 0\}$.
3. Step 2: Set $y'_j = \overline{y}_j$ for all locations j.
   For each location i such that $y'_i > 0$ and $i \notin N$ let $j \in N$ be its closest location in N. Set $y'_j \leftarrow \min(1, y'_i + y'_j)$ and $y'_j \leftarrow 0$.
   Let $s(j)$ be the closest location to j in N (other than j and ties broken with smallest index). Let $n' = |N|$. Sort the locations in N in the decreasing order of $d_j c_{s(j)j}$. Set $\hat{y} = 1$ for the first $2k - n'$ locations and $\hat{y} = \frac{1}{2}$ for the remaining $2(n' - k)$ locations.
4. We build a collection of trees as follows: For each node $i \in N$ with $\hat{y}_i = \frac{1}{2}$ draw a directed edge from i to $s(i)$. Delete one arbitrary edge in every directed (2-)cycle in this graph. This graph is now a collection of rooted trees. Define the level of any node to be the number of edges on the path from the node to the root.
   We select the nodes with $\hat{y} = 1$ in the k-median solution. We partition the nodes $\{i \in N | \hat{y} = \frac{1}{2}\}$ into two subsets corresponding to odd and even levels and include the smaller of the two subsets in the k-median solution. This ensures that there are no more than k facilities in the solution.

6.2 Jain et al.'s greedy facility location algorithm

6.3 Mettu and Plaxton's incremental k-median algorithm

Throughout this section, let λ, α, β and γ denote real numbers satisfying the following inequalities.

$$\begin{aligned} \lambda &\geq 1 \\ \alpha &> 1 + \lambda \\ \beta &\geq \frac{\lambda(\alpha - 1)}{\alpha - 1 - \lambda} \\ \gamma &\geq \left(\frac{\alpha^2\beta + \alpha\beta}{\alpha - 1} + \alpha \right) \lambda \end{aligned}$$

The algorithm is listed in Algorithm 3. It uses the following additional definitions.

- A ball A is a pair (x, r) where x is the center of the ball A. We let $center(A) = x$ denote the center of the ball A and $radius(A) = r$ denote the radius of the ball A.

Algorithm 2 Jain et al.'s Greedy Facility Location Algorithm

1. There is a notion of time. The algorithm starts at time 0. At this time, each client is defined to be unconnected (U:=C), all facilities are unopened, and v_j is set to 0 for every client j. At every moment, each client j offers some money from its contribution to each unopened facility i. The amount of this offer is computed as follows: If j is unconnected, the offer is equal to $max(v_j - c_{ij}, 0)$ (i.e., if the contribution of j is more than the cost that it has to pay to get connected to i, it offers to pay this extra amount to i); If j is already connected to some other facility i', then its offer to facility i is equal to $max(c_{i'j} - c_{ij}, 0)$ (i.e., the amount that j offers to pay to i is equal to the amount j would save by switching its facility from i' to i).
2. While $U \neq \emptyset$, increase the time, and simultaneously, for every client $j \in U$, increase the parameter v_j at the same rate, until one of the following events occurs (if two events occur at the same time, we process them in an arbitrary order).
   (a) For some unopened facility i, the total offers that it receives from clients is equal to the cost of opening i. In this case, we open facility i, and for every client j (connected or unconnected) which has a non-zero offer to i, we connect j to i. The amount that j had offered to i is now called the contribution of j toward i, and j is no longer allowed to decrease this contribution.
   (b) For some unconnected client j, and some open facility i, $v_j = c_{ij}$. In this case, connect client j to facility i and remove j from U.

- Here $d(x, y)$ denotes the distance between points x and y. $d(x, Y)$ denotes the minimum distance between the point x and one of the points in the set Y.
- The value of ball $A = (x, r)$ is $\sum_{y \in A}(r - d(x, y))$ where the sum is taken over all the points within the ball A. A child of a ball (x, r) is any ball $(y, \frac{r}{\alpha})$ where $d(x, y) \leq \beta r$.
- For any point x, let $isolated(x, \emptyset)$ denote the ball $(x, max_{y \in F} d(x, y))$.
- For any point x and and set of facilities X, let isolated(x, X) denote the ball $(x, d(x, X)/\gamma)$.
- For any non empty sequence ρ let $head(\rho)$ (resp. $tail(\rho)$) denote the first (resp. last) element of ρ.

Algorithm 3 Mettu and Plaxton's incremental k-median algorithm

1. Let $Z_0 = \emptyset$. For $i = 0$ to $n - 1$, execute the following steps
   - Let σ_i denote the singleton sequence $< A >$ where A is a maximum value ball in $\{\text{isolated}(x, Z_i) | x \in F \setminus Z_i\}$
   - While the ball $tail(\sigma_i)$ has more than one child, append a maximum value child of $tail(\sigma_i)$ to σ_i.
   - Let $Z_{i+1} = Z_i \cup \{center(tail(\sigma_i))\}$
2. The output is the collection of facility sets Z_i such that $|Z_i| = i$, $0 \leq i \leq n$ and $Z_i \subseteq Z_{i+1}$, $0 \leq i < n$.

6.4 Lin et al.'s incremental k-median algorithm

Lin et al. [9] show a way to take two arbitrary k-median solutions of k_1 and k_2 medians ($k_1 < k_2$) and find a k_1-median solution that is a subset of the k_2-median solution without increasing the

cost by too much. They used this nesting technique to nest selected k-median solutions and get an incremental k-median ordering.

Given two k-median solutions S_1 and S_2 with $|S_1| < |S_2|$, let $Nest(S_1, S_2)$ finds $S \subset S_2$ such that $|S| = |S_1|$ and $c(S) \leq 2c(S_1) + c(S_2)$. The algorithm AltIncApprox is described here.

Algorithm 4 AltIncApprox(α)

1. Initialization: $i = 1$.
2. Use an α-approximation algorithm to compute approximate solutions $V_1, V_2, \ldots V_n$ for cardinalities $1, 2, \ldots, n$ respectively where n is the number of facilities.
3. Bucketing: Order these solutions according to their cost into buckets of form $[0, 0]$, $(1, 2]$, $(2, 2^2]$, $\ldots$, $(2^{k-1}, 2^k], \ldots$
4. Pick the solution with lowest cardinality from each of these non-empty buckets. Let these solutions be $\overline{V}_1, \overline{V}_2, \ldots, \overline{V}_r$ respectively with $\overline{V}_1 = 1$. Let $S_r = \overline{V}_r$
5. Iteration i: $S_{r-i} = Nest(\overline{V}_{r-i}, S_{r-i+1})$,
6. Termination: If $i = r - 1$, return sequence $S_1, \cdots, S_r$, Otherwise, $i \leftarrow i + 1$, go to step 5.

The final ordering of the facilities can be obtained by interpolating these nested solutions by using an interpolating procedure.

The algorithm BoundedIncApprox given by Lin et al. [9] is essentially the AltIncApprox algorithm with a small modification. It uses approximate solutions from an LMP facility location algorithm instead of the k-median approximate solutions in Step 3 of the Algorithm 4 for bucketing which gives the required competitive ratio for the incremental k-median algorithm.

6.5 Plaxton's hierarhical k-median algorithm

We use $c = 3$ for the Plaxton's parent function calculation.

Algorithm 5 Plaxton's hierarchical k-median algorithm: Parent function calculation

1. Let $0, 1, 2..., n - 1$ be the ordered incremental solution.
2. Let T_i^p be the set of clients that are assigned to i directly or to the descendants of i i.e. $T_i^p = i \cup \{T_j^p | p(j) = i\}$.
3. $p(i)$ is calculated starting from $n - 1$ down to 1 as follows: $p(i)$ is set to the minimum j in $0, 1, ..., i - 1$ such that $d(i, j) = \min_{k \in \{0,1,\ldots i-1\}} d(i, k)$ or $d(i, j).|T_i^p| \leq c \cdot \sum_{k \in T_i^p} d(k, i)$ for some constant c.

6.6 Results

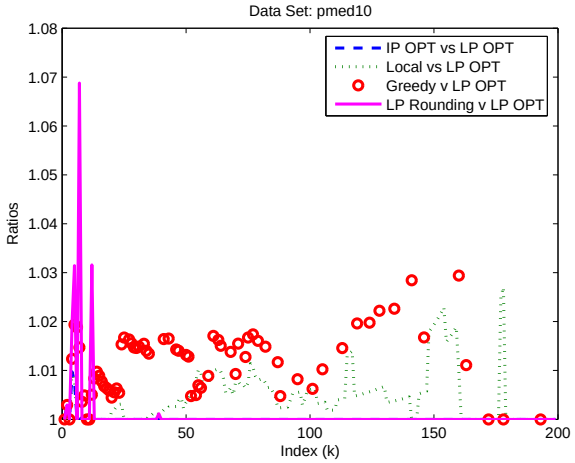

Figure 7: Quality of solutions of k-median algorithms (dataset *pmed*10)

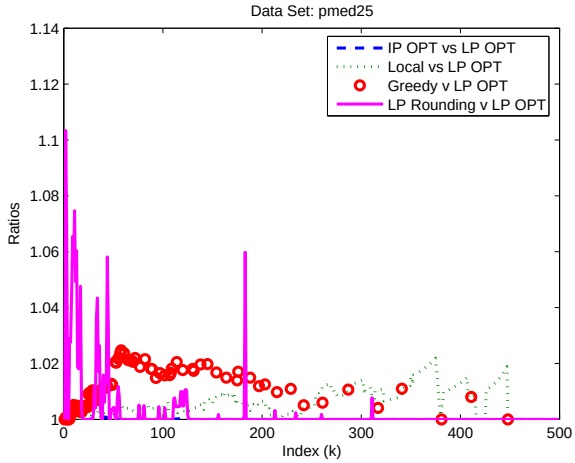

Figure 8: Quality of solutions of k-median algorithms (dataset *pmed*25)

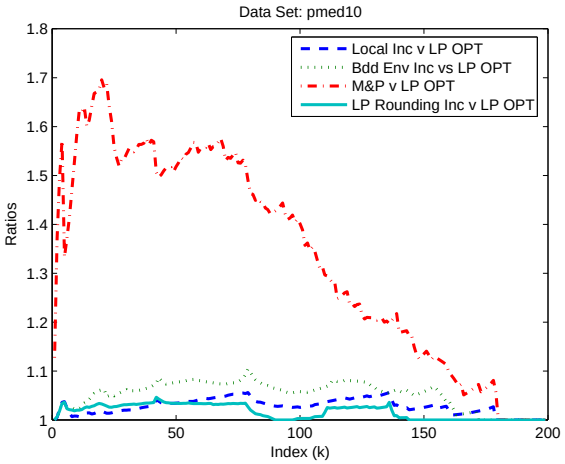

Figure 9: Quality of solutions of incremental k-median algorithms (dataset *pmed*10)

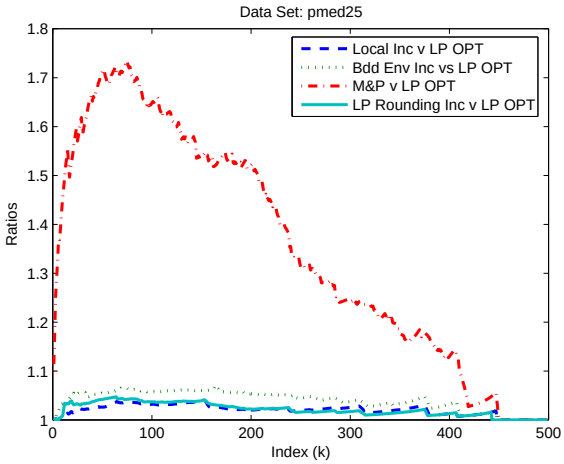

Figure 10: Quality of solutions of incremental k-median algorithms (dataset *pmed*25)

Table 1: Performance of k-median algorithms

		IP OPT/LP OPT		Local /LP OPT		LPR/LP OPT	
Dataset	n	Mean	Max	Mean	Max	Mean	Max
pmed1	100	1.0001	1.01	1.0098	1.1143	1.0007	1.0532
pmed2	100	1.0001	1.0017	1.0097	1.0794	1.0035	1.1024
pmed3	100	1.0001	1.005	1.0086	1.1111	1.0029	1.0885
pmed4	100	1	1.0015	1.0022	1.0157	1.0014	1.0391
pmed5	100	1	1.002	1.0042	1.0177	1.0026	1.1311
pmed6	200	1.0004	1.0203	1.0028	1.0303	1.004	1.181
pmed7	200	1	1.0021	1.006	1.0714	1.0011	1.066
pmed8	200	1.0001	1.0084	1.0046	1.0209	1.004	1.1792
pmed9	200	1.0002	1.0071	1.0046	1.0238	1.005	1.1516
pmed10	200	1.0001	1.0097	1.0046	1.027	1.0008	1.0688
pmed11	300	1.0001	1.0039	1.0062	1.04	1.0025	1.1555
pmed12	300	1.0001	1.0149	1.0044	1.0196	1.0026	1.1285
pmed13	300	1.0001	1.0038	1.0085	1.0373	1.0026	1.0805
pmed14	300	1.0003	1.0092	1.0078	1.0253	1.0054	1.1748
pmed15	300	1	1.0024	1.007	1.0476	1.0011	1.0472
pmed16	400	1.0001	1.0087	1.0071	1.0206	1.0025	1.1701
pmed17	400	1.0001	1.0091	1.0059	1.018	1.0024	1.0977
pmed18	400	1.0002	1.0075	1.0062	1.0213	1.0037	1.1504
pmed19	400	1.0002	1.0111	1.004	1.0145	1.004	1.1506
pmed20	400	1.0001	1.0121	1.0073	1.0299	1.0022	1.1071
pmed21	500	1.0001	1.0031	1.0073	1.0227	1.004	1.1206
pmed22	500	1.0003	1.0104	1.0101	1.0789	1.0057	1.222
pmed23	500	1.0001	1.0081	1.008	1.0417	1.0044	1.1589
pmed24	500	1.0001	1.0073	1.0052	1.0238	1.0029	1.1359
pmed25	500	1.0001	1.0035	1.0062	1.0218	1.0024	1.1034
pmed26	600	1.0002	1.0074	1.0086	1.0448	1.005	1.1093
pmed27	600	1.0002	1.0054	1.0073	1.0435	1.0037	1.1133
pmed28	600	*	*	1.0084	1.038	1.0046	1.1578
pmed29	600	*	*	1.0079	1.0311	1.0051	1.188
pmed30	600	*	*	1.0068	1.0189	1.0036	1.1529
pmed31	700	*	*	1.0085	1.0306	1.004	1.1141
pmed32	700	*	*	1.007	1.0239	1.0048	1.2603
pmed33	700	*	*	1.0122	1.0625	1.0077	1.1875
pmed34	700	*	*	1.006	1.0205	1.0061	1.2068
pmed35	800	*	*	1.0079	1.0352	1.0053	1.2359
pmed36	800	*	*	1.0077	1.027	1.0061	1.1944
pmed37	800	*	*	1.0089	1.0323	1.0059	1.3148
pmed38	900	*	*	1.0081	1.0392	1.0042	1.2426
pmed39	900	*	*	1.0098	1.0449	1.0041	1.2423
pmed40	900	*	*	1.009	1.0543	1.0053	1.2123
Galvão100	100	1.0029	1.045	1.0041	1.0459	1.0352	1.2615
Galvão150	150	1.0048	1.0512	1.0096	1.0642	1.0479	1.2698
Alberta	316	1.0002	1.002	1.0132	1.0299	1.0026	1.0866

Table 2: Running times of k-median algorithms

		Time in seconds to finish for all n				
Dataset	n	LP	IP	Local	Greedy	LPR
pmed1	100	11.656	42.734	35.141	233.19	11.827
pmed2	100	11.547	42.609	34.734	234.92	11.735
pmed3	100	12.188	43.938	34.703	205.34	12.36
pmed4	100	11.922	39.844	34.688	197.39	12.109
pmed5	100	11.484	39.078	34.656	170.03	11.641
pmed6	200	73.203	817.52	315.94	937.38	74.374
pmed7	200	59.5	487.05	318.66	852.48	60.375
pmed8	200	64.234	603.52	317.33	979.06	65.172
pmed9	200	71.859	695.3	316.41	918.13	72.843
pmed10	200	66.859	531.39	316.89	818.47	67.75
pmed11	300	222.25	4133.9	1301.1	1927.5	225.72
pmed12	300	251.34	4158.3	1295.7	2326.8	254.47
pmed13	300	223.16	3368.8	1311.9	2397.8	226.27
pmed14	300	316.45	4979.5	1306.3	2072.1	320.31
pmed15	300	204.39	2991.9	1302.8	2001.9	207.63
pmed16	400	633.36	11117	3609.6	4033.8	641.33
pmed17	400	633.98	11853	3619	4052.7	641.27
pmed18	400	770.3	14411	3627.4	4295.1	778.7
pmed19	400	773.53	12823	3600.5	4008.1	781.13
pmed20	400	669.33	11851	3620	4186.6	677.13
pmed21	500	1427.1	34725	8122.2	5783.1	1442.7
pmed22	500	2021.1	46425	8137.7	6508.6	2037.7
pmed23	500	1494.8	34739	8128.2	6188.4	1510.1
pmed24	500	1777.1	28528	8215.2	6071.9	1791.8
pmed25	500	1223.1	29149	8372.3	6419.6	1237.8
pmed26	600	4191.2	*	19636	10777	4219
pmed27	600	3732.3	*	16712	9733.8	3757.2
pmed28	600	3672.3	*	19768	10017	3698
pmed29	600	4020.8	*	20380	9166.4	4046.2
pmed30	600	4141.1	*	16890	9997.3	4166
pmed31	700	6859.7	*	30450	14909	6899.5
pmed32	700	7443	*	30512	15803	7481.7
pmed33	700	7786.6	*	30353	13893	7834.6
pmed34	700	*	*	30605	15411	*
pmed35	800	*	*	51954	33600	*
pmed36	800	*	*	52061	34322	*
pmed37	800	*	*	52054	33445	*
pmed38	900	*	*	*	43930	*
pmed39	900	*	*	*	49183	*
pmed40	900	*	*	*	50414	*
Galvão100	100	12.641	1208.2	35.219	90.797	13.25
Galvão150	150	40.453	40839	129.7	256.13	41.875
Alberta	316	306.59	5248.5	1642.8	3791.1	311.11

Table 3: Performance of incremental k-median algorithms

		LInc/LP OPT		GInc/LP OPT		MPInc/LP OPT		LPRInc/LP OPT	
Dataset	n	Mean	Max	Mean	Max	Mean	Max	Mean	Max
pmed1	100	1.0397	1.1143	1.0808	1.1298	1.2388	1.5449	1.0548	1.1018
pmed2	100	1.0253	1.0587	1.0356	1.0869	1.2596	1.4596	1.0069	1.0332
pmed3	100	1.0297	1.1379	1.0583	1.1339	1.1989	1.3753	1.0248	1.0726
pmed4	100	1.0044	1.0333	1.0419	1.0978	1.2093	1.6107	1.0099	1.0333
pmed5	100	1.0078	1.03	1.0619	1.1593	1.1812	1.4791	1.0127	1.0938
pmed6	200	1.0115	1.0603	1.0273	1.0603	1.2287	1.4456	1.0103	1.0603
pmed7	200	1.0211	1.0714	1.0526	1.0985	1.2742	1.4944	1.0152	1.0482
pmed8	200	1.0142	1.0519	1.0241	1.0554	1.2502	1.5199	1.0089	1.0398
pmed9	200	1.0171	1.0476	1.0308	1.0479	1.3135	1.6001	1.0073	1.0392
pmed10	200	1.0276	1.0562	1.0521	1.1043	1.3347	1.6957	1.0168	1.0467
pmed11	300	1.0205	1.04	1.0522	1.0998	1.3153	1.6443	1.0143	1.035
pmed12	300	1.0163	1.0408	1.0521	1.1121	1.3018	1.5773	1.0133	1.0432
pmed13	300	1.0236	1.0554	1.0402	1.0729	1.3329	1.6403	1.0213	1.0612
pmed14	300	1.0216	1.0458	1.0572	1.1061	1.2594	1.6084	1.0163	1.1177
pmed15	300	1.0214	1.0544	1.0404	1.0721	1.3826	1.7325	1.018	1.0306
pmed16	400	1.0255	1.0469	1.0399	1.0755	1.3904	1.7797	1.0228	1.1024
pmed17	400	1.015	1.0488	1.0574	1.0891	1.3286	1.6403	1.0267	1.0575
pmed18	400	1.0227	1.0597	1.0486	1.0864	1.3195	1.5553	1.0225	1.1216
pmed19	400	1.0102	1.0462	1.0384	1.0585	1.3051	1.5939	1.0222	1.0727
pmed20	400	1.0273	1.0489	1.0385	1.0757	1.2429	1.4529	1.0256	1.0691
pmed21	500	1.023	1.0439	1.0425	1.0843	1.2958	1.6379	1.0235	1.0587
pmed22	500	1.0254	1.0789	1.0533	1.0973	1.3233	1.6869	1.0165	1.0573
pmed23	500	1.0249	1.0417	1.0354	1.0714	1.3228	1.5863	1.0236	1.0482
pmed24	500	1.0165	1.0441	1.0356	1.0687	1.3197	1.57	1.0193	1.0595
pmed25	500	1.0204	1.0395	1.0399	1.071	1.3522	1.7349	1.0215	1.0478
pmed26	600	1.0224	1.0496	1.0479	1.087	1.3655	1.6944	1.0197	1.0627
pmed27	600	1.0207	1.0475	1.0378	1.0781	1.3266	1.6875	1.0198	1.0533
pmed28	600	1.0302	1.0549	1.0398	1.0703	1.3821	1.8071	1.021	1.07
pmed29	600	1.0226	1.0365	1.0401	1.0774	1.3452	1.7871	1.0209	1.1158
pmed30	600	1.0179	1.0466	1.0362	1.0722	1.3233	1.8021	1.0192	1.0639
pmed31	700	1.0228	1.0441	1.0383	1.0817	1.3608	1.7558	1.0242	1.0588
pmed32	700	1.0178	1.0424	1.0362	1.0704	1.373	1.769	1.0154	1.0771
pmed33	700	1.0365	1.0627	1.049	1.0826	1.3381	1.7314	1.0237	1.0691
pmed34	700	1.0202	1.049	1.0374	1.07	1.3768	1.7976	1.0172	1.0836
pmed35	800	1.0231	1.1336	1.0367	1.0708	1.4248	1.9212	1.0163	1.1336
pmed36	800	1.0242	1.0411	1.035	1.0645	1.3849	1.8291	1.0229	1.0665
pmed37	800	1.0204	1.041	1.0356	1.0709	1.3405	1.7722	1.0189	1.0853
pmed38	900	1.0208	1.0392	1.0394	1.0688	1.3901	1.7921	1.0237	1.0656
pmed39	900	1.0245	1.0449	1.0451	1.0842	1.384	1.828	1.0172	1.0449
pmed40	900	1.0194	1.0543	1.035	1.0689	1.3397	1.7058	1.0208	1.161
Galvão100	100	1.0209	1.1601	1.024	1.1601	1.0711	1.3496	1.036	1.2205
Galvão150	150	1.0283	1.1655	1.0349	1.1925	1.0908	1.388	1.0285	1.2168
Albert a	316	1.0409	1.1967	1.0499	1.1432	1.157	1.4322	1.027	1.1223

Table 4: Running times of incremental k-median algorithms

		Running time in secs			
Dataset	Size	IncL	IncG	MP	IncLPR
pmed1	100	35.266	233.31	0.593	11.999
pmed2	100	34.859	235.06	0.594	11.907
pmed3	100	34.844	205.47	0.578	12.532
pmed4	100	34.829	197.52	0.61	12.281
pmed5	100	34.766	170.17	0.578	11.797
pmed6	200	316.83	938.33	2.375	75.327
pmed7	200	319.55	853.36	2.359	61.313
pmed8	200	318.22	979.92	2.36	66.063
pmed9	200	317.28	919	2.343	73.796
pmed10	200	317.8	819.36	2.375	68.672
pmed11	300	1304.7	1930.9	5.672	229.03
pmed12	300	1299	2330.2	5.938	257.84
pmed13	300	1315.4	2401.1	5.766	229.55
pmed14	300	1309.6	2075.4	5.625	323.61
pmed15	300	1306.1	2005.2	5.797	210.95
pmed16	400	3617.8	4042	11.218	649.55
pmed17	400	3627.4	4061	10.844	649.7
pmed18	400	3636.1	4303.4	11.844	786.97
pmed19	400	3608.9	4016.3	12.125	789.36
pmed20	400	3628.8	4194.8	11.031	685.36
pmed21	500	8139.7	5800.6	18.438	1460.2
pmed22	500	8155.5	6526.2	18.547	2055.4
pmed23	500	8146.3	6206.6	18.485	1527.6
pmed24	500	8233.5	6089.8	18.578	1809.4
pmed25	500	8390.4	6437.8	18.813	1255.2
pmed26	600	19667	10807	30	4249.8
pmed27	600	16743	9764.5	29.969	3788.9
pmed28	600	19799	10048	29.86	3730
pmed29	600	20411	9197.1	30.157	4077
pmed30	600	16921	10029	30.297	4196.9
pmed31	700	30499	14958	44.891	6949.1
pmed32	700	30561	15854	43.657	7532.4
pmed33	700	30404	13943	45.422	7884
pmed34	700	30655	15461	43.391	*
pmed35	800	52037	33683	61.454	*
pmed36	800	52141	34402	62.376	*
pmed37	800	52134	33525	61.735	*
pmed38	900	*	44037	82.595	*
pmed39	900	*	49290	78.954	*
pmed40	900	*	50522	81.673	*
Galvão100	100	35.36	90.937	0.563	13.422
Galvão150	150	130.05	256.45	1.281	42.265
Alberta	316	1646.6	3794.9	6.375	315.03

Table 5: Performance of hierarchical k-median algorithms

		HL/LP		HG/LP		PHLI/LP		PHMP/LP		LPRH/LP	
Dataset	n	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
pmed1	100	1.05	1.11	1.1	1.17	1.07	1.3	1.35	1.83	1.07	1.14
pmed2	100	1.04	1.13	1.05	1.13	1.04	1.17	1.31	1.56	1.02	1.13
pmed3	100	1.04	1.16	1.07	1.14	1.06	1.24	1.31	1.57	1.04	1.18
pmed4	100	1.02	1.1	1.06	1.14	1.04	1.24	1.3	1.85	1.02	1.1
pmed5	100	1.02	1.12	1.07	1.16	1.01	1.11	1.27	1.78	1.02	1.18
pmed6	200	1.03	1.17	1.04	1.14	1.04	1.23	1.34	1.74	1.03	1.16
pmed7	200	1.04	1.14	1.08	1.17	1.05	1.23	1.39	1.78	1.04	1.16
pmed8	200	1.03	1.12	1.04	1.15	1.06	1.29	1.33	1.71	1.03	1.11
pmed9	200	1.04	1.16	1.05	1.16	1.05	1.17	1.45	2.04	1.03	1.14
pmed10	200	1.04	1.14	1.07	1.13	1.05	1.17	1.45	1.99	1.03	1.15
pmed11	300	1.04	1.11	1.07	1.14	1.06	1.23	1.45	1.88	1.03	1.13
pmed12	300	1.04	1.17	1.07	1.18	1.06	1.31	1.46	1.95	1.03	1.15
pmed13	300	1.04	1.17	1.06	1.14	1.06	1.27	1.48	2.03	1.04	1.17
pmed14	300	1.04	1.18	1.08	1.18	1.06	1.39	1.42	1.99	1.04	1.25
pmed15	300	1.04	1.15	1.06	1.19	1.05	1.19	1.58	2.22	1.04	1.16
pmed16	400	1.05	1.18	1.06	1.2	1.07	1.3	1.62	2.4	1.04	1.19
pmed17	400	1.04	1.23	1.08	1.21	1.06	1.28	1.48	1.96	1.05	1.19
pmed18	400	1.05	1.21	1.07	1.18	1.08	1.35	1.49	1.96	1.05	1.22
pmed19	400	1.03	1.17	1.06	1.19	1.05	1.29	1.44	1.93	1.04	1.2
pmed20	400	1.05	1.17	1.07	1.2	1.07	1.3	1.35	1.74	1.05	1.17
pmed21	500	1.05	1.16	1.06	1.16	1.06	1.26	1.46	2	1.05	1.16
pmed22	500	1.05	1.2	1.08	1.2	1.07	1.35	1.57	2.32	1.04	1.26
pmed23	500	1.05	1.17	1.06	1.17	1.06	1.27	1.49	2.03	1.05	1.2
pmed24	500	1.04	1.18	1.06	1.19	1.06	1.34	1.48	1.96	1.04	1.2
pmed25	500	1.04	1.14	1.06	1.18	1.06	1.26	1.54	2.11	1.04	1.15
pmed26	600	1.05	1.18	1.07	1.18	1.07	1.3	1.54	2.2	1.04	1.18
pmed27	600	1.05	1.18	1.06	1.18	1.06	1.33	1.49	2.08	1.05	1.18
pmed28	600	1.05	1.19	1.06	1.19	1.08	1.32	1.57	2.34	1.05	1.22
pmed29	600	1.05	1.19	1.06	1.19	1.06	1.35	1.53	2.28	1.05	1.24
pmed30	600	1.04	1.19	1.06	1.18	1.06	1.32	1.54	2.3	1.04	1.25
pmed31	700	1.04	1.19	1.06	1.19	1.06	1.37	1.56	2.31	1.04	1.2
pmed32	700	1.05	1.22	1.07	1.24	1.06	1.33	1.54	2.27	1.04	1.23
pmed33	700	1.06	1.19	1.07	1.22	1.08	1.39	1.54	2.36	1.05	1.22
pmed34	700	1.05	1.19	1.06	1.18	1.07	1.31	1.58	2.23	1.04	1.29
pmed35	800	1.05	1.29	1.06	1.17	1.07	1.39	1.64	2.52	1.04	1.31
pmed36	800	1.05	1.24	1.06	1.25	1.08	1.38	1.6	2.38	1.05	1.22
pmed37	800	1.05	1.24	1.06	1.21	1.07	1.42	1.52	2.26	1.05	1.24
pmed38	900	1.05	1.19	1.06	1.2	1.06	1.36	1.62	2.44	1.05	1.21
pmed39	900	1.05	1.21	1.07	1.23	1.07	1.35	1.55	2.27	1.05	1.22
pmed40	900	1.05	1.22	1.06	1.23	1.07	1.4	1.53	2.22	1.05	1.3
Galvão100	100	1.05	1.27	1.05	1.36	1.07	1.37	1.24	1.81	1.09	1.43
Galvão150	150	1.08	1.34	1.08	1.37	1.1	1.44	1.28	1.84	1.09	1.35
Alberta	316	1.06	1.27	1.07	1.25	1.08	1.28	1.27	1.72	1.05	1.22

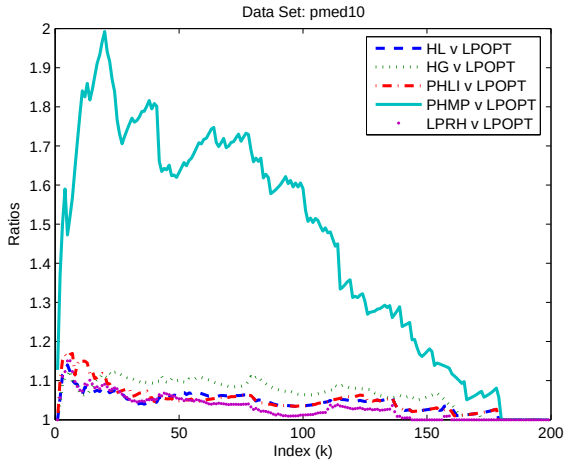

Figure 11: Quality of solutions of hierarchical k-median algorithms (dataset *pmed*10)

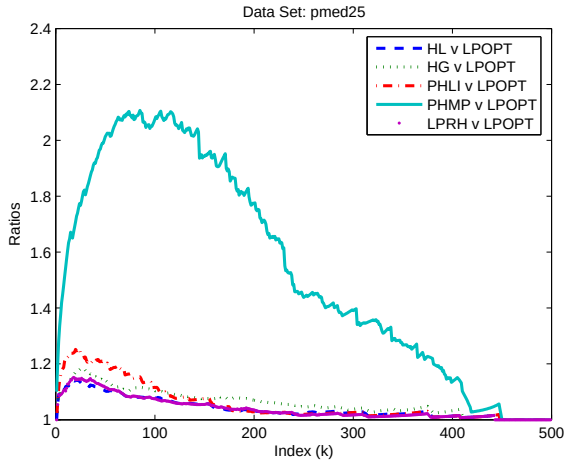

Figure 12: Quality of solutions of hierarchical k-median algorithms (dataset *pmed*25)