How to Solve Fair k-Center in Massive Data Models

Ashish Chiplunkar [*1] **Sagar Kale** [*2] **Sivaramakrishnan Natarajan Ramamoorthy** [*3]

Abstract

Fueled by massive data, important decision making is being automated with the help of algorithms, therefore, fairness in algorithms has become an especially important research topic. In this work, we design new streaming and distributed algorithms for the fair k-center problem that models fair data summarization. The streaming and distributed models of computation have an attractive feature of being able to handle massive data sets that do not fit into main memory. Our main contributions are: (a) the first distributed algorithm; which has provably constant approximation ratio and is extremely parallelizable, and (b) a two-pass streaming algorithm with a provable approximation guarantee matching the best known algorithm (which is not a streaming algorithm). Our algorithms have the advantages of being easy to implement in practice, being fast with linear running times, having very small working memory and communication, and outperforming existing algorithms on several real and synthetic data sets. To complement our distributed algorithm, we also give a hardness result for natural distributed algorithms, which holds for even the special case of k-center.

1. Introduction

Data summarization is a central problem in the area of machine learning, where we want to compute a small summary of the data. For example, if the input data is enormous, we do not want to run our machine learning algorithm on the whole input but on a small *representative* subset. How we select such a representative summary is quite important. It is well known that if the input is biased, then the machine learning algorithms trained on this data will exhibit the same bias. This is a classic example of *selection bias* but as exhibited by algorithms themselves. Currently used algorithms for data summarization have been shown to be biased with respect to attributes such as gender, race, and age (see, e.g., (Kay et al., 2015)), and this motivates the fair data summarization problem. Recently, the fair k-center problem was shown to be useful in computing fair summary (Kleindessner et al., 2019). In this paper, we continue the study of fair k-center and add to the series of works on fairness in machine learning algorithms. Our main results are streaming and distributed algorithms for fair k-center. These models are extremely suitable for handling massive datasets. The fact that the data summarization problem arises when the input is huge makes our work all the more relevant!

Suppose the input is a set of real vectors with a gender attribute and you want to compute a summary of k data points such that both[1] genders are represented equally. Say we are given a summary S. The cost we pay for not including a point in S is its distance from S. Then the cost of S is the largest cost of a point. We want to compute a summary with minimum cost that is also fair, i.e., contains $k/2$ women and $k/2$ men. In one sentence, we want to compute a fair summary such that the point that is farthest from this summary is not too far. Fair k-center models this task: let the number of points in the input be n, the number of *groups* be m, target summary size be k, and we want to select a summary S such that S contains k_j points belonging to Group j, where $\sum_j k_j = k$. And we want to minimize $\max_x d(x, S) = \max_x \min_{x' \in S} d(x, x')$, where d denotes the distance function. Note that each point belongs to exactly one of the m groups; for the case of gender, $m = 2$.

We call the special case where $m = 1$ and $k_1 = k$ as just k-center throughout this paper. For k-center, there are simple greedy algorithms with an approximation ratio of 2 (Gonzalez, 1985; Hochbaum & Shmoys, 1985), and getting better than 2-approximation is NP-hard (Hsu & Nemhauser, 1979). The NP-hardness result also applies to the more general fair k-center. The best algorithm known for fair k-center is a 3-approximation algorithm that runs in time $O(n^2 \log n)$ (Chen et al., 2016). A linear-time algorithm with approximation guarantee of $O(2^m)$, which is constant if m is, was given recently (Kleindessner et al., 2019). Both of these algorithms work only in the traditional random

[*]Equal contribution [1]IIT Delhi [2]University of Vienna, Faculty of Computer Science [3]Theorem LP. Correspondence to: Sagar Kale <SGRKL4@gmail.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

[1]sincere apologies to the people who identify with neither

access machine model, which is suitable only if the input is small enough to fit into fast memory. We give a two-pass streaming algorithm that achieves the approximation ratio arbitrarily close to 3. In the streaming setting, input is thought to arrive one point at a time, and the algorithm has to process the input quickly, using minimum amount of working memory—ideally linear in the size of a feasible solution, which is k for fair k-center. Our algorithm processes each incoming input point in $O(k)$ time and uses space $O(km)$, which is $O(k)$ if the number of groups m is very small. This improves the space usage of the existing streaming algorithm (Kale, 2019) almost quadratically, from $O(k^2)$, while also matching the best approximation ratio achieved by Chen et al. We also give the first distributed, constant approximation algorithm where the input is divided among multiple processors, each of which performs one round of computation and sends a message of size $O(km)$ to a central processor, which then computes the final solution. Both rounds of computation are linear time. All the approximation, communication, space usage, and running-time guarantees are provable. To complement our distributed algorithm, we prove that any distributed algorithm, even randomized, that works by each processor sending a subset of its input to a central processor which outputs the solution, needs to essentially communicate the whole input to achieve an approximation ratio of better than 4. This, in fact, applies for the special case of k-center showing that known 4-approximation algorithm (Malkomes et al., 2015) for k-center is optimal.

We perform experiments on real and synthetic datasets and show that our algorithms are as fast as the linear-time algorithm of Kleindessner et al., while achieving improved approximation ratio, which matches that of Chen et al. Note that this comparison is possible only for small datasets, since those algorithms do not work either in streaming or in distributed setting. We also run our algorithms on a really large synthetic dataset of size 100GB, and show that their running time is only one order of magnitude more than the time taken to just read the input dataset from secondary memory.

As a further contribution, we give faster implementations of existing algorithms—those of Kale and Chen et al.

Related work

Chen et al. gave the first polynomial-time algorithm that achieves 3-approximation. Kale achieves almost the same ratio using just two passes and also gives a one-pass $(17+\varepsilon)$-approximation algorithm, both using $O(k^2)$ space. All these algorithms are designed to work on a single machine.

One (incomparable) way to compute a fair summary is using a determinantal measure of diversity (Celis et al., 2018). Fair clustering has been studied under another notion of fairness, where each cluster must be balanced with respect to all the groups (no over-or-under-representation of any group) (Chierichetti et al., 2017), and this line of work also has received a lot of attention in a short span of time (Bera et al., 2019; Ahmadian et al., 2019; Bandyapadhyay et al., 2019; Schmidt et al., 2020; Jia et al., 2020).

The k-median clustering problem with fairness constraints was first considered by (Hajiaghayi et al., 2010) and with more general matroid constraints was studied by (Krishnaswamy et al., 2011). The work of Chen et al. and Kale also actually applies for matroid constraints.

There has been a lot of work done on fairness, and due to space constraints, we refer the reader to overviews by (Kleindessner et al., 2019; Celis et al., 2018).

2. Preliminaries

The input to fair k-center is a set X of n points in a metric space given by a distance function d. We denote this metric space by (X, d). Each point belongs to one of m groups, say $\{1, \ldots, m\}$. Let $g : X \longrightarrow \{1, \ldots, m\}$ denote this group assignment function. Further, for each group j, we are given a capacity k_j. Let $k = \sum_{j=1}^{m} k_j$. We call a subset $S \subseteq X$ *feasible* if for every j, the set S contains at most k_j points from group j. (Readers familiar with matroids will recognize this as a partition matroid constraint.) The goal is to compute a feasible set of centers that (approximately) minimizes the clustering cost, formally defined as follows.

Definition 1. *Let $A, B \subseteq X$. Then the clustering cost of A for B is defined as* $\max_{b \in B} \min_{a \in A} d(a, b)$.

Note here that A need not be a subset of B. The following lemmas follow easily from the fact that the distance function d satisfies the triangle inequality.

Lemma 1. *Let $A, B, C \subseteq X$. The clustering cost of A for C is at most the clustering cost of A for B plus the clustering cost of B for C.*

Lemma 2. *Suppose for a set T of points there exists a set of k centers, not necessarily a subset of T, whose clustering cost for T is at most ρ. If $P \subseteq T$ is a set of points separated pairwise by distance more than 2ρ, then $|P| \leqslant k$.*

Proof. If $|P| > k$ then some two points in P must share one of the k centers, and must therefore be both within distance ρ from that common center. Then by the triangle inequality, they cannot be separated by distance more than 2ρ. □

We denote by S^* a feasible set which has the minimum clustering cost for X, and by OPT the minimum clustering cost. We assume that our algorithms have access to an estimate τ of OPT. When τ is at least OPT, our algorithms compute a solution of cost at most $\alpha\tau$ for a constant α. Thus, when $\tau \in [\text{OPT}, (1+\varepsilon)\text{OPT}]$, our algorithms compute a

$(1+\varepsilon)\alpha$-approximate solution. In Section 3.3 we describe how to efficiently compute such a τ.

3. Algorithms

Before stating algorithms, we describe some elementary procedures which will be used as subroutines in our algorithms.

getPivots(T, d, r) takes as input a set T of points with distance function d and a radius r. Starting with $P = \emptyset$, it performs a single pass over T. Whenever it finds a point q which is not within distance r from any point in P, it adds q to P. Finally, it returns P. Thus, P is a maximal subset of T of points separated pairwise by distance more than r. We call points in P *pivots*. By Lemma 2, if there is a set of k points whose clustering cost for T is at most $r/2$, then $|P| \leqslant k$. Moreover, due to maximality of P, its clustering cost for T is at most r. Note that getPivots() runs in time $O(|P| \cdot |T|)$.

getReps(T, d, g, P, r) takes as input a set T of points with distance function d, a group assignment function g, a subset $P \subseteq T$, and a radius r. For each $p \in P$, initializing $N(p) = \{p\}$, it includes in $N(p)$ one point, from each group, which is within distance r from p whenever such a point exists. Note that this is done while performing a single pass over T. This procedure runs in time $O(|P| \cdot |T|)$.

Informally, if P is a good but infeasible set of centers, then getReps() finds representatives $N(p)$ of the groups in the vicinity of each $p \in P$. This, while increasing the clustering cost by at most r, gives us enough flexibility to construct a feasible set of centers. The procedure HittingSet() that we describe next finds a feasible set from a collection of sets of representatives.

HittingSet$(\mathcal{N}, g, \overline{k})$ takes as input a collection $\mathcal{N} = \{N_1, \dots, N_K\}$ of pairwise disjoint sets of points, a group assignment function g, and a vector $\overline{k} = (k_1, \dots, k_m)$ of capacities of the m groups. It returns a feasible set S intersecting as many N_i's as possible. This reduces to finding a maximum cardinality matching in an appropriately constructed bipartite graph. It is important to note that this procedure does the post-processing: it doesn't make any pass over the input stream of points. This procedure runs in time $O(K^2 \cdot \max_i |N_i|)$.

For interested readers, the pseudocodes of these procedures, an explanation of HittingSet(), and the proof of its running time appear in the full version (Chiplunkar et al., 2020).

3.1. A Two-Pass Algorithm

Recall that τ is an upper bound on the minimum clustering cost. Our two-pass algorithm given by Algorithm 1 consists of three steps. First, the algorithm constructs a maximal subset $P \subseteq X$ of pivots separated pairwise by distance more than 2τ by executing one pass on the stream of points. In another pass, the algorithm computes a representative set $N(q)$ of each pivot $q \in P$ using the function getReps. By the property of getReps, points in the representative set of a pivot are within distance τ from the pivot. Due to the separation of 2τ between the pivots, these representative sets are pairwise disjoint. Finally, a feasible set S intersecting as many $N(q)$'s as possible is found and returned. (It will soon be clear that S intersects all the $N(q)$'s.)

Algorithm 1 Two-pass algorithm

```
Input: Metric space $(X, d)$, group assignment function
$g$, capacity vector $\overline{k}$.
/* Pass 1: Compute pivots. */
$P \leftarrow$ getPivots$(X, d, 2\tau)$.
/* Pass 2: Compute representatives. */
$\{N(q) : q \in P\} \leftarrow$ getReps$(X, d, g, P, \tau)$.
/* Compute solution. */
$S \leftarrow$ HittingSet$(\{N(q) : q \in P\}, g, \overline{k})$.
Output $S$.
```

The algorithm needs working space only to store the pivots and their representative sets. By substituting $S = S^*$ in Lemma 2, the number of pivots is at most k, that is, $|P| \leqslant k$. Since $N(q)$ contains at most one point from any group, it has at most $m-1$ points other than q. Thus,

Observation 1. *The two-pass algorithm needs just enough working space to store km points.*

The calls to getPivots and getReps both take time $O(|P| \cdot |X|) = O(kn)$, with $O(|P|) = O(k)$ update time per point. The call to HittingSet takes time $O(|P|^2 \cdot \max_{q \in P} |N(q)|) = O(mk^2)$. Thus,

Observation 2. *The two-pass algorithm runs in time $O(kn + mk^2)$, which is $O(kn)$ when m, the number of groups, is constant.*

We now prove the approximation guarantee.

Theorem 1. *The two-pass algorithm returns a feasible set whose clustering cost is at most 3τ. This is a $3(1+\varepsilon)$-approximation when $\tau \in [OPT, (1+\varepsilon)OPT]$.*

Proof. Recall that S^* is a feasible set having clustering cost at most τ. For each $q \in P$ let $c_q \in S^*$ denote a point such that $d(q, c_q) \leqslant \tau$. Since the points in P are separated by distance more than 2τ, the points c_q are all distinct. Recall that $N(q)$, the output of getReps(), contains one point from every group which has a point within distance τ from q. Therefore, $N(q)$ contains a point, say b_q, from the same group as c_q such that $d(q, b_q) \leqslant \tau$. Consider the set $B = \{b_q : q \in P\}$. This set intersects $N(q)$ for each q.

Algorithm 2 Summary computation by the i'th processor

```
Input: Set $X_i$, metric $d$ restricted to $X_i$, group assignment function $g$ restricted to $X_i$.
/* Compute local pivots. */
$p^i_1 \leftarrow$ an arbitrary point in $X_i$.
for $j = 2$ to $k + 1$ do
    $p^i_j \leftarrow \arg\max_{p \in X_i} \min_{j':1 \leqslant j' < j} d(p, p^i_{j'})$.
end for
$P_i \leftarrow \{p^i_1, \ldots, p^i_k\}$.
$r_i \leftarrow \min_{j':1 \leqslant j' \leqslant k} d(p^i_{k+1}, p^i_{j'})/2$.
/* Compute local representative sets. */
$\{L(p) : p \in P_i\} \leftarrow$ getReps($X_i, d, g, P_i, 2r_i$).
$L_i \leftarrow \bigcup_{p \in P_i} L(p)$.
/* Send message to coordinator. */
Send $(P_i, L_i)$ to the coordinator.
```

Furthermore, B contains exactly as many points from any group as $\{c_q : q \in P\} \subseteq S^*$, and therefore, B is feasible. Thus, there exists a feasible set, namely B, intersecting all the pairwise disjoint $N(q)$'s. Recall that S, the output of `HittingSet()`, is a feasible set intersecting as many $N(q)$'s as possible. Thus, S also intersects all the $N(q)$'s.

Now, the clustering cost of S for P is at most τ, because S intersects $N(q)$ for each $q \in P$. The clustering cost of P for X is at most 2τ by the maximality of the set returned by `getPivots()`. These facts and Lemma 1 together imply that the clustering cost of S, the output of the algorithm, for X is at most 3τ. □

3.2. A Distributed Algorithm

In the distributed model of computation, the set X of points to be clustered is distributed equally among ℓ processors. Each processor is allowed a restricted access to the metric d: it may compute the distance between only its own points. Each processor performs some computation on its set of points and sends a summary of small size to a coordinator. From the summaries, the coordinator then computes a feasible set S of points which covers all the n points in X within a small radius. Let X_i denote the set of points distributed to processor i.

The algorithm executed by each processor i is given by Algorithm 2, which consists of two main steps. In the first step, the processor uses Gonzalez's farthest point heuristic to find $k + 1$ points. The first k of those constitute the set P_i, which we will call the set of *local pivots*. The point p_{k+1} is the farthest point from the set of local pivots, and r_i is defined to be half the distance of p_{k+1} from the set of local pivots. Thus, every point X_i is within distance $2r_i$ from the set of local pivots. This means,

Observation 3. *The clustering cost of P_i for X_i is $2r_i$.*

In the second step, for each local pivot $p \in P_i$, the processor computes a set $L(p)$ of local representatives in the vicinity of p using the function `getReps`. Finally, the set P_i of local pivots and the union $L_i = \bigcup_{p \in P_i} L(p)$ of local representative sets is sent to the coordinator. By the property of `getReps`, $L(p)$ contains at most one point from any group. Therefore, it has at most $m - 1$ points other than p. Since $|P_i| = k$ we have the following observation.

Observation 4. *Each processor sends at most km points to the coordinator.*

Moreover, the separation between the local pivots is bounded as follows.

Lemma 3. *For every processor i, we have $r_i \leqslant$ OPT $\leqslant \tau$.*

Proof. Suppose $r_i >$ OPT. Then $\{p^i_1, \ldots, p^i_{k+1}\} \subseteq X_i$ is a set of $k + 1$ points separated pairwise by distance more than 2OPT. But S^* is a set of at most k points whose clustering cost for X_i is OPT $\leqslant \tau$. This contradicts Lemma 2. □

Observation 3 allows us to define a covering function cov from X, the input set of points, to $\bigcup_{i=1}^{\ell} P_i$, the set of local pivots, as follows.

Definition 2. *Let p be an arbitrary point in X. Suppose p is processed by processor i, that is, $p \in X_i$. Then cov(p) is an arbitrary local pivot in P_i within distance $2r_i$ from p.*

Since the processors send only a small number of points to the coordinator, it is very well possible that the optimal set S^* of centers is lost in this process. In the next lemma, we claim that the set of points received by the coordinator contains a good and feasible set of centers nevertheless.

Lemma 4. *The set $L = \bigcup_{i=1}^{\ell} L_i$ contains a feasible set, say B, whose clustering cost for $\bigcup_{i=1}^{\ell} P_i$ is at most 5τ.*

Proof. Consider any $c \in S^*$, and suppose it is processed by processor i. Then $d(c, \text{cov}(c)) \leqslant 2r_i$ by Definition 2. Recall that $L(\text{cov}(c))$, the output of `getReps()`, contains one point from every group which has a point within distance $2r_i$ from cov(c). Therefore, $L(\text{cov(c)}) \subseteq L_i$ contains some point, say c', from the same group as c (possibly c itself), such that $d(c', \text{cov}(c)) \leqslant 2r_i$. Then $d(c, c') \leqslant 4r_i \leqslant 4\tau$ by the triangle inequality and Lemma 3. Let $B = \{c' : c \in S^*\}$. Clearly, $B \subseteq \bigcup_{i=1}^{\ell} L_i$. Since B has exactly as many points from any group as S^*, B is feasible. The clustering cost of B for S^* is at most 4τ. The clustering cost of S^* for $\bigcup_{i=1}^{\ell} P_i$ is at most τ, because $\bigcup_{i=1}^{\ell} P_i \subseteq X$. By Lemma 1, the clustering cost of B for $\bigcup_{i=1}^{\ell} P_i$ is at most 5τ, as required. □

The algorithm executed by the coordinator is given by Algorithm 3. The coordinator constructs a maximal subset P of

Algorithm 3 Coordinator's algorithm

```
$X' \leftarrow \emptyset$, $L \leftarrow \emptyset$.
/* Receive messages from processors. */
for $i = 1$ to $\ell$ do
  Receive $(P_i, L_i)$ from processor $i$.
  $X' \leftarrow X' \cup P_i$, $L \leftarrow L \cup L_i$.
end for
/* Coordinator now has access to $d$ and $g$ restricted to $X' \cup L$, and capacity vector $\overline{k} = (k_1, \dots, k_m)$. */
/* Compute global pivots. */
$P \leftarrow \texttt{getPivots}(X', d, 10\tau)$.
/* Compute global representative sets. */
$\{N(q) : q \in P\} \leftarrow \texttt{getReps}(L, d, g, P, 5\tau)$.
/* Compute solution. */
$S \leftarrow \texttt{HittingSet}(\{N(q) : q \in P\}, g, \overline{k})$.
Output $S$.
```

the set of pivots $X' = \bigcup_{i=1}^{\ell} P_i$ returned by the processors such that points in P are pairwise separated by distance more than 10τ. P is called the set of global pivots. For each global pivot $q \in P$, using the function getReps, the coordinator computes a set $N(q) \subseteq L = \bigcup_{i=1}^{\ell} L_i$ of its global representatives, all of which are within distance 5τ from q. Due to the separation between points in P, the sets $N(q)$ are pairwise disjoint. Finally, a feasible set S intersecting as many $N(q)$'s as possible is found and returned. (As before, it will be clear that S intersects all the $N(q)$'s.)

Theorem 2. *The coordinator returns a feasible set whose clustering cost is at most 17τ. This is a $17(1+\varepsilon)$-approximation when $\tau \in [OPT, (1+\varepsilon)OPT]$.*

Proof. By Lemma 4, L contains a feasible set, say B, whose clustering cost for X' is at most 5τ. For each $q \in P \subseteq X'$, let b_q denote a point in B that is within distance 5τ from q. Since the points in X' are separated pairwise by distance more than 10τ, b_q's are all distinct. By the property of getReps(), the set $N(q)$ returned by it contains a point, say b'_q, from the same group as b_q. Let $B' = \{b'_q : q \in P\}$. This set B' intersects $N(q)$ for each $q \in P$. Since b'_q and b_q are from the same group and b_q's are all distinct, B' contains at most as many points from any group as B does. Since B is feasible, so is B'. To summarize, there exists a feasible set, namely B', intersecting all the $N(q)$'s. Recall that S, the output of HittingSet(), is a feasible set intersecting as many $N(q)$'s as possible. Thus, S also intersects all the $N(q)$'s.

Now, the clustering cost of S for P is at most 5τ, because S intersects $N(q)$ for each $q \in P$. The clustering cost of P for X' is at most 10τ by the maximality of the set returned by getPivots(). The clustering cost of $X' = \bigcup_{i=1}^{\ell} P_i$ for $X = \bigcup_i X_i$ is at most 2τ because the clustering cost of each P_i for X_i is at most $2r_i \leqslant 2\tau$. These facts and Lemma 1 together imply that the clustering cost of S, the output of the coordinator, for X is at most 17τ. □

We note here that even though our distributed algorithm has the same approximation guarantee as Kale's one-pass algorithm, it is inherently a different algorithm. Ours is extremely parallel whereas Kale's is extremely sequential. We now prove a bound on the running time.

Theorem 3. *The running time of the distributed algorithm is $O(kn/\ell + mk^2\ell)$, which can be made $O(m^{1/2}k^{3/2}n^{1/2})$ by an appropriate choice of ℓ, the number of processors.*

Proof. For each processor i, computing local pivots as well as the call to getReps() takes $O(|P_i| \cdot |X_i|) = O(kn/\ell)$ time each. For the coordinator, the separation between the global pivots and Lemma 2 together enforce $|P| \leqslant k$. Observation 4 implies $|L| \leqslant m \cdot \max_i |L_i| \leqslant mk\ell$. Therefore, getPivots() takes time $O(|P| \cdot |X'|) = O(k^2\ell)$ and getReps() takes time $O(|P| \cdot |L|) = O(mk^2\ell)$. The call to HittingSet() takes time $O(k^2 \max_q |N(q)|) = O(mk^2)$, thus limiting the coordinator's running time to $O(mk^2\ell)$. Choosing $\ell = \Theta(\sqrt{n/(mk)})$ minimizes the total running time to $O(m^{1/2}k^{3/2}n^{1/2})$. □

3.3. Handling the Guesses

Given an arbitrarily small parameter ε, a lower bound $L \leqslant \mathrm{OPT}$, and an upper bound $U \geqslant \mathrm{OPT}$, we run our algorithms for guess $\tau \in \{L, L(1+\varepsilon), L(1+\varepsilon)^2, \dots, U\}$, which means at most $\log_{1+\varepsilon}(U/L)$ guesses. Call this method of guesses as geometric guessing starting at L until U. For the $\tau \in [\mathrm{OPT}, \mathrm{OPT}(1+\varepsilon)]$, our algorithms will compute a solution successfully.

In the distributed algorithm, by Lemma 3, for each processor, $r_i \leqslant \mathrm{OPT}$. Therefore, $\max_i r_i \leqslant \mathrm{OPT}$. We then run Algorithm 3 with geometric guessing starting at $\max_i r_i$ until it successfully finds a solution.

For the two-pass algorithm, let P be the set of first $k+1$ points; then $L = \min_{x_1, x_2 \in P} d(x_1, x_2)/2$ is a lower bound (call this the simple lower bound). Note that no passes need to be spent to compute the simple lower bound. We also need an upper bound $U \geqslant \mathrm{OPT}$. One can compute an arbitrary solution and its cost—which will be an upper bound—by spending two more passes (call this the simple upper bound). This results in a four-pass algorithm. To obtain a truly two pass algorithm and space usage $O(km\log(1/\varepsilon)/\varepsilon)$, one can use Guha's trick (Guha, 2009), which is essentially starting $O(\log(1/\varepsilon)/\varepsilon)$ guesses and if a run with guess τ fails, then continuing the run with guess τ/ε and treating the old summary as the initial stream for this guess; see also (Kale, 2019) for details. But obtaining and using an upper bound is convenient and easy to implement in practice.

4. Distributed k-Center Lower Bound

(Malkomes et al., 2015) generalized the greedy algorithm of (Gonzalez, 1985) to obtain a 4-approximation algorithm for the k-center problem in the distributed setting. Here we prove a lower bound for the 3-center problem with 9 processors for a special class of distributed algorithms: If each processor communicates less than $2/9$ fraction of their input points, then with a constant probability, the output of the coordinator will be no better than a 4-approximation to the optimum. Figure 1 shows a graph metric with $9n' + 7$ points for which lower bound holds, where the point x is not a part of the metric but is only used to define the distances. Note that $|S_1| = |S_2| = |S_3| = 3n'$ and x is at distance of 1 from each point in $S_1 \cup S_2 \cup S_3$.

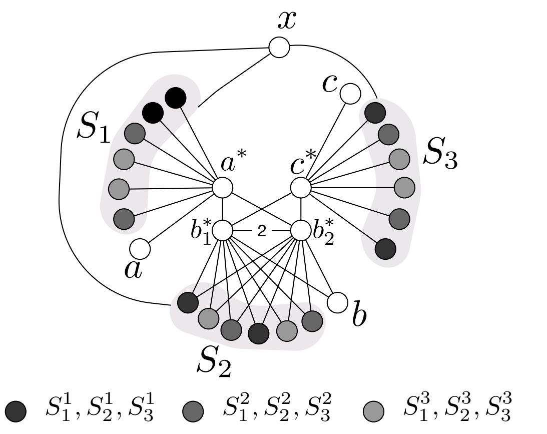


Figure 1. The underlying metric for $n' = 2$

For $i \in \{1, 2, 3\}$, let S_i^1, S_i^2, S_i^3 denote an arbitrary equipartition of S_i. There are 9 processors, whose inputs are given by $Y_1^j = \{b_1^*, b_2^*, a\} \cup S_1^j$, $Y_2^j = \{a^*, c^*, b\} \cup S_2^j$ and $Y_3^j = \{b_1^*, b_2^*, c\} \cup S_3^j$, for $j \in \{1, 2, 3\}$. The goal is to solve the 3-center problem on the union of their inputs. (Observe that the optimum solution is $\{a^*, c^*, b_1^*\}$ with distance 1.) Each processor is allowed to send a subset of their input points to the coordinator, who outputs three of the received points. For this class of algorithms, we show that if each processor communicates less than $2n'/9$ points, then the output of the coordinator is no better than a 4-approximation to the optimum with probability at least $2 \cdot 10^{-6}$. Using standard amplification arguments, we can generate a metric instance for the (3α)-center problem on which with probability at least $1 - \varepsilon$, the algorithm outputs no better than 4-approximation ($\alpha \approx \log(1/\varepsilon)$).

We discuss the intuition behind the proof, and the full proof is deferred to the full version (Chiplunkar et al., 2020). The key observation is that all points in each Y_i^j are pairwise equidistant. Therefore, sending a uniformly random subset of the inputs is the best strategy for each processor. Since each processor communicates only a small fraction of its input points, the probability that the coordinator receives any of the points in $\{a^*, b_1^*, b_2^*, c^*, a, b, c\}$ is negligible. Conditioned on the coordinator not receiving these points, all the received points are a subset of $S_1 \cup S_2 \cup S_3$. As all points in $S_1 \cup S_2 \cup S_3$ are pairwise equidistant, the best strategy for the coordinator is to output 3 points at random. Hence, with constant probability, all the points in the output belong to S_1 or all of them belong to S_3. This being the case, the output has cost 4, whereas the optimum cost is 1.

5. Experiments

All experiments are run on HP EliteBook 840 G6 with Intel® Core™ i7-8565U CPU 1.80GHz having 4 cores and 15.5 GiB of RAM, running Ubuntu 18.04 and Anaconda. We make our code available on GitHub[2].

We perform our experiments on a massive synthetic dataset, several real datasets, and small synthetic datasets. The same implementation is used for the large synthetic dataset and the real datasets, but a slightly different implementation is used for small synthetic datasets. Before presenting the experiments, we first discuss the implementation details that are common to all three experiments. Specific details are mentioned along with the corresponding experimental setup. For all our algorithms if the solution size is less than k, then we extend the solution using an arbitrary solution of size k (which also certifies the simple upper bound). In the case of the distributed algorithm, an arbitrary solution is computed using only the points received by the coordinator. Also, one extra pass is spent into computing solution cost. In the processors' algorithm, we return r_i along with (P_i, L_i). No randomness is used for any optimization, making our algorithms completely deterministic. Access to distance between two points is via a method `get_distance()`, whose implementation depends on the dataset.

We use the code shared by Kleindessner et al. for their algorithm on github[3], exactly as is, for all datasets. In their code, the distance is assumed to be stored in an $n \times n$ distance matrix.

As mentioned in the introduction, we give new implementations for existing algorithms—those of Chen et al. and Kale (we choose to implement Kale's two-pass algorithm only, because it is the better of his two). Instead of using a matroid intersection subroutine, which can have running time of super quadratic in n, we reduce the postprocessing steps of these algorithms to finding a maximum matching in an appropriately constructed graph (for details, see `HittingSet()` in the full version (Chiplunkar et al., 2020)). We further reduce maximum matching to max-

[2] https://github.com/sagark4/fair_k_center
[3] https://github.com/matthklein/fair_k_center_clustering

flow which is computed using Python package NetworkX. This results in a postprocessing time of $O(k^2 n)$ for Chen et al. and $O(k^3)$ for Kale. This step itself makes Chen et al.'s algorithm practical for much larger n than what is observed by Kleindessner et al.

Handling the guesses For all algorithms (except Kleindessner et al.'s), we use $\varepsilon = 0.1$. For Chen et al.'s algorithm, we use geometric guessing starting with the lower bound given by the farthest point heuristic (call this Gonzalez's lower bound). For our two-pass algorithm and Kale's algorithm, we use geometric guessing starting with the simple lower bound until the upper bound given by an arbitrary solution. The values for the guesses τ in the coordinator's algorithm are scaled down by a factor of 5.1. Concretely, let r_1 be the maximum among the r_i's. Then the guesses take values in $\frac{r_1}{5.1}, \frac{1.1 \cdot r_1}{5.1}, \frac{(1.1)^2 \cdot r_1}{5.1}, \ldots$, until a feasible solution is found. The factor of 5.1 ensures that when `getPivots()` is run with the parameter $10\tau < 2r_1$, we end up picking at least k pivots from X'.

We now proceed to present our experiments. To show the effectiveness of our algorithms on massive datasets, we run them on a 100 GB synthetic dataset which is a collection of 4,000,000 points in 1000 dimensional Euclidean space, where each coordinate is a uniformly random real in $(0, 10000)$. Each point is assigned one of the four groups uniformly at random, and capacity of each group is set to 2. Just reading this data file takes more than four minutes. Our two-pass algorithm takes 1.95 hours and our distributed algorithm takes 1.07 hours; both compute a solution of almost the same cost, even though their theoretical guarantees are different. Here, we use block size of 10000 in the distributed algorithm, i.e., the number of processors $\ell = 400$.

For the above dataset and the real datasets: The input is read from the input file and attributes are read from the attribute file, one data point at a time, and fed to the algorithms. This is done in order to be able to handle the 100 GB dataset. Using Python's multiprocessing library, we are able to use four cores of the processor [4].

5.1. Real Datasets

We use three real world datasets: Celeb-A (Liu et al., 2015), Sushi (sus), and Adult (Kohavi & Becker), with $n = 1000$ by selecting the first 1000 data points (see Table 1).

Celeb-A dataset is a set of 202,599 images of human faces with attributes including male/female and young/not-young, which we use. We use Keras to extract features from each image (fea) via the pretrained neural network VGG16, which returns a 15360 dimensional real vector for each image. We use the ℓ_1 distance as the metric and two settings of groups: male/female with capacity of 2 each (denoted by $[2, 2]$ in Table 1), and $\{\text{male, female}\} \times \{\text{young, not-young}\}$ with capacity of 2 each (denoted by $[2] * 4$ in Table 1).

Sushi dataset is about preferences for different types of Sushis by 5000 individuals with attributes of male/female and six possible age-groups. In SushiB, the preference is given by a score whereas in SushiA, the preference is given by an order. For SushiB, we use the ℓ_1 distance whereas for SushiA, we use the number of inversions, i.e., the distance between two Sushi rankings is the number of doubletons $\{i, j\}$ such that Sushi i is preferred over Sushi j by one ranking and not the other. For both SushiA and SushiB, we use three different group settings: with gender only, with age group only, and combination of gender and age group. This results in 2, 6, and 12 groups, respectively, and the capacities appear as $[2, 2]$, $[2] * 6$, and $[2] * 12$, respectively, in Table 1.

Motivated by Kleindessner et al., we consider the adult dataset (Kohavi & Becker), which is extracted from US census data and contains male/female attribute and six numerical attributes that we use as features. We normalize this dataset to have zero mean and standard deviation of one and use the ℓ_1 distance as the metric. There are two attributes that can be used to generate groups: gender and race (Black, White, Asian Pacific Islander, American Indian Eskimo, and Other). Individually and in combination, this results in 2, 5, and 10 groups, respectively.

Based on a reviewer's suggestion, we also ran our experiments on the Celeb-A, Sushi, and Adult datasets for capacities other than 2 for all groups. Results appear in Table 2.

On a majority of settings, our two-pass algorithm outputs a solution with cost smaller than the rest (see Table 1 and Table 2). We reiterate for emphasis that in addition to being at least as good as the best in terms of solution quality, our algorithms can handle massive datasets.

For the distributed algorithm, we use block size of 25, i.e., the number of processors are $1000/25 = 40$: theoretically, using $\approx \sqrt{n}$ processor gives maximum speedup.

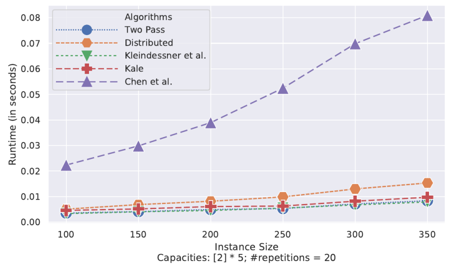
5.2. Synthetic Datasets

Motivated by the experiments in Kleindessner et al., we use the Erdős-Rényi graph metric to compare the running time and cost of our algorithms with existing algorithms. For a fixed natural number n, a random metric on n points is generated as follows. First, a random undirected graph on n vertices is sampled in which each edge is independently picked with probability $2 \log n / n$. Second, every edge is assigned a uniformly random weight in $(0, 1000)$. The points in the metric correspond to the vertices of the graph, and the pairwise distances between the points are given by the

[4] https://www.praetorian.com/blog/multi-core-and-distributed-programming-in-python

Table 1. Comparison of solution quality of algorithms for fair k-center on real datasets. Each column after the third corresponds to an algorithm and shows ratio of its cost and Gonzalez's lower bound. Note that this is not the approximation ratio. Our two-pass algorithm is the best for majority of the settings. Dark shaded cell shows the best-cost algorithm and lightly shaded cell shows the second best.

Dataset	Capacities	Gonzalez's Lower Bound	Chen et al.	Kale	Kleindessner et al.	Two pass	Distributed
CelebA	[2, 2]	30142.4	1.9	1.9	1.85	1.76	1.76
CelebA	[2, 2, 2, 2]	28247.3	2.0	2.0	1.9	1.88	1.88
SushiA	[2, 2]	11.0	2.18	2.18	2.27	2.0	2.09
SushiA	[2] * 6	8.5	2.35	2.35	2.24	2.35	2.24
SushiA	[2] * 12	7.5	2.13	2.13	2.0	2.4	2.4
SushiB	[2, 2]	35.0	1.8	1.8	2.11	1.8	1.86
SushiB	[2] * 6	32.5	1.94	1.88	2.22	1.82	1.88
SushiB	[2] * 12	30.5	1.93	2.0	2.07	2.0	2.0
Adult	[2, 2]	4.9	2.04	2.13	2.44	1.9	2.02
Adult	[2] * 5	3.92	2.66	2.66	2.02	2.36	2.35
Adult	[2] * 10	2.76	2.75	2.41	2.48	2.48	2.75



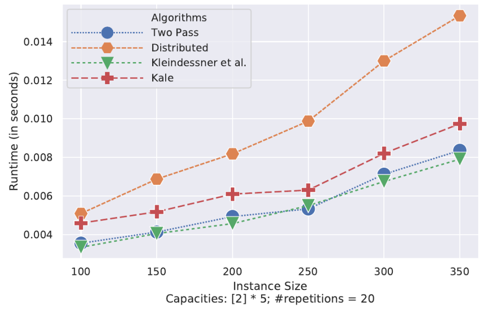


Figure 2. Comparing Running Times

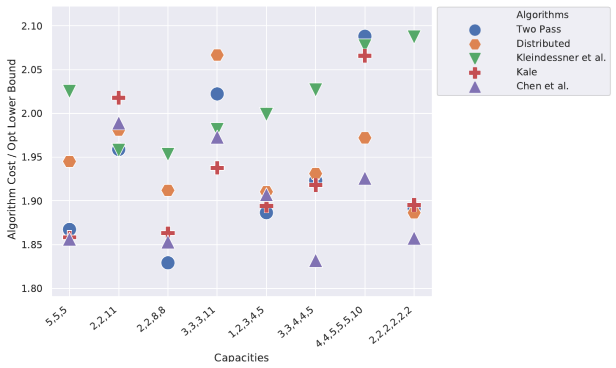


Figure 3. Comparing Approximation Ratios

shortest path distance. In addition, if m is the number of groups, then each point in the metric is assigned a group in $\{1, 2, \ldots, m\}$ uniformly and independently at random.

Figure 2 shows the plots between the running time and instance size n; the bottom one is a zoom-in of the top one to the lower four plots. In this experiment, n takes values in $\{100, 150, 200, \ldots, 350\}$. The number of groups is fixed to 5 and the capacity of each group is 2. For each fixing of n, we run the five algorithms on 20 independent random metric instances of size n to compute the average running time. Our two pass algorithm and Kleindessner et al.'s algorithm are the fastest. Our distributed algorithm is faster than Chen et al.'s algorithm, but slower than Kale's.

Figure 3 shows the ratios of the cost of various algorithms to Gonzalez's lower bound. For this comparison, the instance size is fixed to 500 and capacities are $[5, 5, 5], [2, 2, 11], [2, 2, 8, 8], [3, 3, 3, 11], [1, 2, 3, 4, 5], [3, 3, 4, 4, 5], [4, 4, 5, 5, 5, 10], [2, 2, 2, 2, 2, 2]$. Here again, for every fixing of the capacities, the algorithm is run on 20 independent random metric instances to compute the average costs. Chen et al.'s algorithm achieves the least cost for almost all settings, and Kleindessner et al.'s algorithm gives the highest cost on majority (5 out of 8) of settings. Our two-pass algorithm and Kale's algorithm perform similar to each other and are quite close to Chen et al.'s. Our distributed algorithm is somewhere in between Chen et al.'s and Kleindessner et al.'s. Note that the ratios of the costs between any two algorithms is at most 1.167.

In the implementation of our two pass algorithm, we use geometric guessing starting with the simple lower bound until the algorithm returns a success instead of running all guesses. This is done for a fair comparison in terms of running time.

Table 2. Extension of Table 1 for capacities other than 2. See the description of Table 1.

Dataset	Capacities	Gonzalez's Lower Bound	Chen et al.	Kale	Kleindessner et al.	Two pass	Distributed
CelebA	[1, 3]	30142.45	1.95	1.95	1.93	1.95	2.02
CelebA	[3, 1]	30142.45	1.9	1.9	2.19	1.76	1.76
SushiA	[1, 3]	11.0	2.18	2.18	2.0	2.18	2.09
SushiA	[3, 1]	11.0	2.18	2.18	2.0	2.09	2.45
SushiB	[1, 3]	35.0	1.74	1.74	2.11	1.74	1.86
SushiB	[3, 1]	35.0	1.8	1.8	2.0	1.8	1.86
Adult	[1, 3]	4.9	2.35	2.29	2.04	1.9	1.9
Adult	[3, 1]	4.9	2.04	2.13	2.29	2.53	2.17
CelebA	[1, 1, 3, 3]	28247.28	2.0	2.0	1.76	2.0	1.96
CelebA	[3, 3, 1, 1]	28247.28	1.99	1.99	1.87	1.88	1.88
SushiA	[1, 1, 1, 3, 3, 3]	8.5	2.35	2.47	2.35	2.35	2.35
SushiA	[3, 3, 3, 1, 1, 1]	8.5	2.24	2.24	2.35	2.24	2.24
SushiA	[3, 3, 3, 1, 1, 1, 3, 3, 3, 1, 1, 1]	7.5	2.27	2.27	2.4	2.13	2.13
SushiB	[1, 1, 1, 3, 3, 3]	32.5	1.91	1.91	2.06	1.82	1.88
SushiB	[3, 3, 3, 1, 1, 1]	32.5	1.88	2.03	2.12	1.97	1.82
SushiB	[3, 3, 3, 1, 1, 1, 3, 3, 3, 1, 1, 1]	30.5	1.93	1.93	2.0	1.93	2.03
Adult	[1, 1, 3, 3, 3]	3.57	2.72	2.72	2.42	2.72	2.64
Adult	[3, 3, 1, 1, 1]	3.92	2.43	1.95	2.2	1.95	1.95

6. Research Directions

One research direction is to improve the theoretical bounds, e.g., get a better approximation ratio in the distributed setting or prove a better hardness result. Another interesting direction is to use fair k-center for fair rank aggregation using the number of inversions between two rankings as the metric.

Acknowledgments

Ashish Chiplunkar is supported by Pankaj Gupta Young Faculty fellowship. Sagar Kale was fully supported by Vienna Science and Technology Fund (WWTF) through project ICT15-003. This work was done when Sivaramakrishnan Natarajan Ramamoorthy was a graduate student at the University of Washington.

References

Keras: Extract features with vgg16. `https://keras.io/applications/#extract-features-with-vgg16`. Accessed: 2020-01-26.

Sushi preference data sets. `http://www.kamishima.net/sushi/`. Accessed: 2020-01-26.

Ahmadian, S., Epasto, A., Kumar, R., and Mahdian, M. Clustering without over-representation. In *KDD*, pp. 267–275, 2019.

Bandyapadhyay, S., Inamdar, T., Pai, S., and Varadarajan, K. A Constant Approximation for Colorful k-Center. In *ESA*, volume 144, pp. 12:1–12:14, 2019.

Bera, S. K., Chakrabarty, D., Flores, N., and Negahbani, M. Fair algorithms for clustering. In *NIPS*, pp. 4955–4966, 2019.

Celis, E., Keswani, V., Straszak, D., Deshpande, A., Kathuria, T., and Vishnoi, N. Fair and diverse DPP-based data summarization. In *ICML*, volume 80, pp. 716–725, 2018.

Chen, D. Z., Li, J., Liang, H., and Wang, H. Matroid and knapsack center problems. *Algorithmica*, 75(1):27–52, 2016.

Chierichetti, F., Kumar, R., Lattanzi, S., and Vassilvitskii, S. Fair clustering through fairlets. In *NIPS*, pp. 5029–5037. 2017.

Chiplunkar, A., Kale, S., and Ramamoorthy, S. N. How to solve fair k-center in massive data models. *CoRR*, abs/2002.07682, 2020. URL `https://arxiv.org/abs/2002.07682`.

Gonzalez, T. F. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38: 293–306, 1985.

Guha, S. Tight results for clustering and summarizing data streams. In *ICDT*, pp. 268–275, 2009.

Hajiaghayi, M., Khandekar, R., and Kortsarz, G. Budgeted red-blue median and its generalizations. In *ESA*, pp. 314–325, 2010.

Hochbaum, D. S. and Shmoys, D. B. A best possible heuristic for the k-center problem. *Math. Oper. Res.*, 10(2): 180–184, 1985.

Hsu, W.-L. and Nemhauser, G. L. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3): 209 – 215, 1979.

Jia, X., Sheth, K., and Svensson, O. Fair colorful k-center clustering. In *IPCO*, volume 12125, pp. 209–222, 2020.

Kale, S. Small Space Stream Summary for Matroid Center. In *APPROX/RANDOM*, volume 145, pp. 20:1–20:22, 2019.

Kay, M., Matuszek, C., and Munson, S. A. Unequal representation and gender stereotypes in image search results for occupations. In *CHI*, pp. 3819–3828, 2015.

Kleindessner, M., Awasthi, P., and Morgenstern, J. Fair k-center clustering for data summarization. In *ICML*, volume 97, pp. 3448–3457, 2019.

Kohavi, R. and Becker, B. Adult data set. `https://archive.ics.uci.edu/ml/datasets/Adult`. Accessed: 2020-01-26.

Krishnaswamy, R., Kumar, A., Nagarajan, V., Sabharwal, Y., and Saha, B. The matroid median problem. In *SODA*, pp. 1117–1130, 2011.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *ICCV*, December 2015.

Malkomes, G., Kusner, M. J., Chen, W., Weinberger, K. Q., and Moseley, B. Fast distributed k-center clustering with outliers on massive data. In *NIPS*, pp. 1063–1071. 2015.

Schmidt, M., Schwiegelshohn, C., and Sohler, C. Fair coresets and streaming algorithms for fair k-means. In *Approximation and Online Algorithms*, pp. 232–251, 2020.