Fully Dynamic k-Center Clustering*

T-H. Hubert Chan†
University of Hong Kong
Hong Kong
hubert@cs.hku.hk

Arnaud Guerquin
LTCI, Télécom ParisTech University
Paris, France
arnaud.guerquin@ens-paris-saclay.fr

Mauro Sozio‡
LTCI, Télécom ParisTech University
Paris, France
sozio@telecom-paristech.fr

ABSTRACT

Static and dynamic clustering algorithms are a fundamental tool in any machine learning library. Most of the efforts in developing dynamic machine learning and data mining algorithms have been focusing on the sliding window model (where at any given point in time only the most recent data items are retained) or more simplistic models. However, in many real-world applications one might need to deal with arbitrary deletions and insertions. For example, one might need to remove data items that are not necessarily the oldest ones, because they have beenfl agged as containing inappropriate content or due to privacy concerns. Clustering trajectory data might also require to deal with more general update operations.

We develop a $(2+\epsilon)$-approximation algorithm for the k-center clustering problem with "small" amortized cost under the fully dynamic adversarial model. In such a model, points can be added or removed arbitrarily, provided that the adversary does not have access to the random choices of our algorithm. The amortized cost of our algorithm is poly-logarithmic when the ratio between the maximum and minimum distance between any two points in input is bounded by a polynomial, while k and ϵ are constant. Our theoretical results are complemented with an extensive experimental evaluation on dynamic data from Twitter, Flickr, as well as trajectory data, demonstrating the effectiveness of our approach.

ACM Reference Format:
T-H. Hubert Chan, Arnaud Guerquin, and Mauro Sozio. 2018. Fully Dynamic k-Center Clustering. In *WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3178876.3186124

1 INTRODUCTION

With over 6000 tweets per second being posted on Twitter, Google processing over 40000 queries every second, and more than 400 hours worth of youtube videos uploaded every minute, there is an urgent need to develop dynamic machine learning and data mining algorithms. Most of the efforts in this direction have been focusing on the sliding window model of computation [6, 7], where at any given point in time only the most recent data items are retained, or more simplistic models.

However, in many real-world applications one might need to deal with arbitrary deletions and insertions. For example, one might need to remove data items that are not necessarily the oldest ones, because they have beenfl agged as containing inappropriate content or due to privacy concerns. The latter case has increasingly become commonplace, due to the 'right to be forgotten' principle. Clustering trajectory data might also require to deal with more general update operations than those modeled by the sliding window model.

Clustering algorithms provide a fundamental tool in any machine learning library. There have been increasing efforts in recent years to study clustering problems from both a theoretical and practical point of view [6, 8, 11, 12].

In our work, we consider a fully dynamic adversarial model, where points can be added or removed arbitrarily, provided that the adversary does not have access to the random choices of our algorithm. Moreover, our algorithm does not know the update operations in advance. We focus on the k-center clustering problem with a long-term goal of studying other machine learning and data mining problems in a fully dynamic environment. We develop a $(2+\epsilon)$-approximation algorithm for the k-center clustering problem, which requires "small" expected amortized cost under the fully dynamic adversarial model. In particular, the expected amortized cost is poly-logarithmic when the ratio between the maximum and minimum distance between any two points in input is bounded by a polynomial, while k and ϵ are constant. We also prove that the running time of our algorithm is concentrated around its expectation with high probability.

Our theoretical results are complemented with an extensive experimental evaluation on dynamic data from Twitter, Flickr, as well as trajectory data, demonstrating the effectiveness of our approach. We also evaluate our algorithm against the approach proposed in [6] for the same problem, under the sliding window model. Our experimental evaluation shows that our algorithm delivers clustering solutions with lower maximum radius than [6], although this comes at the price of a slightly worse average running time and more space. Another advantage of our algorithm is that it is efficient under a fully adversarial model, in contrast with [6] which works under the sliding window model.

The rest of the paper is organized as follows. We discuss the related work in Section 2, while Section 3 introduce the necessary definitions and notations. In Section 4 we present our main algorithm, while its theoretical analysis is provided in Section 5. Section 6 contains an experimental evaluation of our algorithm

*This research was partially supported by a grant from the PROCORE France-Hong Kong Joint Research Scheme sponsored by the Research Grants Council of Hong Kong and the Consulate General of France in Hong Kong under the project F-HKU702/16.
†This research was partially supported by the Hong Kong RGC under the grants 17217716.
‡This research was partially supported by French National Agency (ANR) under project FIELDS (ANR-15-CE23- 0006).

WWW 2018, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5639-8/18/04.
https://doi.org/10.1145/3178876.3186124

against the state-of-the-art approaches on real-world data. Finally, Section 7 contains our conclusion and future work.

2 RELATED WORK

There have been increasingly more efforts in studying clustering algorithms from both a practical and theoretical point of view, in recent years [1–3, 6, 8, 11–13]. One of thefi rst dynamic clustering algorithms has been presented in [4], where the authors developed an 8-approximation algorithm for the case with only insertions. A $(2+\epsilon)$-approximation algorithm was later developed by [14] building on the results of [4]. In the same work, the authors also studied the case with outliers (where a limited number of points can be deleted from the input) for which they developed a $(4+\epsilon)$-approximation algorithm.

The literature for clustering in the streaming model is rich. We mention the work in [10], where the authors give thefi rst single-pass constant approximation algorithm for k-median which has been improved later on in [5].

The work that is most relevant to ours is [6], where the authors studied the k-center clustering problem under the sliding window model. They developed a $(6+\epsilon)$-approximation algorithm requiring $O(k \cdot \frac{\log \delta}{\epsilon})$ time per update (on average), where δ is an upper bound on the ratio between the maximum and minimum distance between any two points in input. The algorithm requires $O(k \cdot \frac{\log \delta}{\epsilon})$ space. They studied their algorithm mainly from a theoretical point of view. One of the contributions of our work is an experimental evaluation of the algorithm in [6] on real-world data.

Observe that our algorithm requires $O(N \cdot \frac{\log(\delta)}{\epsilon})$ space where N is an upper bound on the maximum number of points occurring at any point in time, while the algorithm proposed [6] requires $O(k \cdot \frac{\log(\delta)}{\epsilon})$ space. However, in our case cluster membership can be tested in $O(1)$ time, while all points in a same cluster C of a given point can be produced in time $O(|C|)$. This is an appealing property when monitoring sensor data or trajectories evolving over time. Another advantage of our algorithm is its approximation guarantee which is a factor of $2+\epsilon$. Observe that any approximation ratio less than 2 would imply $P = NP$ [9]. Moreover, as proved in [6] any algorithm with an approximation ratio of less than 4 requires $\Omega(N^{\frac{1}{3}})$ space. Table 1 summarizes the two algorithms in terms of approximation guarantee, average running time, space requirement and query time.

3 NOTATION AND DEFINITIONS

We study the k-center clustering problem, which is formally defined as follows.

Definition 3.1 (k-Center Clustering). We are given a set S of points equipped with some metric d and an integer $k > 0$. We wish tofind a set $C = \{c_1, \ldots, c_k\}$ of k points (centers) so as to minimize the quantity $\max_{x \in S} d(x, C)$, where $d(x, C) = \min_{c \in C} d(x, c)$.

Observe that a set of centers defines a partition of S (clustering) into k sets. We consider the following adversarial model of computation.

Adversarial Model. We assume the adversaryfi rstfi xes a (possibly countably infinite) sequence O of operations, where for each $t \in \mathbb{N}$, the operation $o_t \in X \times \{-, +\}$ consists of a point $x_t \in X$ in the metric space and afl ag to indicate whether it is an insertion (+) or deletion (−). By naturally extending the metric space to $X \times \mathbb{N}$, we can assume without loss of generality that at most one copy of a point is inserted in the sequence. We also assume that any point to be removed has been inserted earlier on. The algorithm does not know the sequence O in advance. However, we assume that any randomness used by the algorithm is generated after the adversary fixes the sequence. We refer to this adversarial model as the *fully dynamic model.*

4 ALGORITHM

To illustrate the main ideas of our algorithm, we start describing a simple $(2+\epsilon)$-approximation for k-center, $\epsilon > 0$. Let β be a guess for the value of an optimum solution. We pick one point c_1 arbitrarily from X and we create a cluster C_1 containing c_1 as well as all points in X_1 being within distance 2β from c_1. The ith cluster, $1 < i \leq k$ is built as follows. If $X \setminus \cup_{j=1}^{i-1} C_j$ is empty we let C_i be the empty set and we stop. Otherwise, we pick one point x_i from $X \setminus \cup_{j=1}^{i-1} C_j$, arbitrarily. We then create a new cluster C_i with x_i as center and containing all the points in $X \setminus \cup_{i=1}^{t-1} C_j$ within distance 2β from x_i.

It can be shown that if β is equal to the value of an optimum solution, then such an algorithm gives a 2-approximation. This follows from the fact that if k clusters have been formed and $X \setminus \cup_{j=1}^{k} C_j$ is not empty, then there are $k+1$ points whose pairwise distance is greater than 2β, which implies that k clusters with radius at most β cannot be formed. This would contradict our assumption on β. If the value of an optimum solution is not known, we would run the previous algorithm for any β in $\Gamma = \{(1+\epsilon)^i : d_{\min} \leq (1+\epsilon)^i \leq (1+\epsilon) \cdot d_{\max}, i \in \mathbb{N}\}$, where $d_{\max}$ and $d_{\min}$ denote the max and min distance between any two points in X, respectively. Observe that if β is too small, we might not be able to cluster all points, which might result in a set of unclustered points U. The smallest β which allows to partition all points in X (i.e. $\cup_{j=1}^{k} C_j = X$) gives a $(2+\epsilon)$-approximation.

A simple (but inefficient) incremental algorithm can be derived from the previous algorithm as follows. At any point in time, for each β in Γ, we maintain the following invariant. We either maintain $k+1$ points (k of which are centers) whose pairwise distance is greater than 2β or a clustering of all the points with maximum radius 2β. The former property guarantees that there cannot be any clustering with maximum radius β. For each β in Γ, we wish to maintain the same set of clusters that would be computed using the algorithm discussed above. Whenever a new point x is inserted, we proceed as follows. Let β in Γ and let $C_1, \ldots, C_k, U$ be the corresponding clusters with centers $c_1, \ldots, c_k$, respectively. We insert x in cluster C_i if it is within distance 2β from c_i. Otherwise, if there is no such a center, we insert x in U. This is repeated for each β in Γ, which ensures that the invariant is maintained. Such an algorithm gives a $(2+\epsilon)$-approximation with amortized cost $O(k \cdot \frac{1}{\epsilon} \cdot \log \frac{d_{\max}}{d_{\min}})$ and $O(|X| \cdot \frac{1}{\epsilon} \cdot \log \frac{d_{\max}}{d_{\min}})$ space.

	approximation	avg. run. time	space	model	$x \in C$?	list all y s.t. $x, y \in C$
FD	$2(1+\epsilon)$	$k^2 \cdot \frac{\log(\delta)}{\epsilon}$	$N \cdot \frac{\log(\delta)}{\epsilon}$	Fully Dynamic	$O(1)$	$\|C\|$
SW	$6(1+\epsilon)$	$k \cdot \frac{\log(\delta)}{\epsilon}$	$k \cdot \frac{\log(\delta)}{\epsilon}$	Sliding Window	$O(k)$	$k \cdot N$

Table 1: Summary of our algorithm (FD) and the one proposed in [6] (SW).

Now, suppose that points are deleted uniformly at random. If none of the k centers are deleted, the invariant is not violated. Otherwise, we might have to re-cluster all the points in order to maintain the invariant. However, the probability to remove any such a point is $\frac{k}{n}$ while the cost of reclustering is at most $k \cdot n$, where n is number of points. Therefore, the expected amortized cost of such a randomized algorithm is $O(k^2 \cdot \frac{1}{\epsilon} \cdot \log \frac{d_{\max}}{d_{\min}})$. Unfortunately, such an algorithm might not be efficient if deletions are not random.

To overcome this problem we equip the algorithm with some randomness, so that it would be efficient even in the case with adversarial deletions. For each β in Γ, we create and maintain a set of clusters $C_1, \ldots, C_k$ as follows. Let X be the current set of points. Thefi rst center c_1 is chosen uniformly at random from X and a cluster C_1 is built, as in the previous algorithm. For $1 < i \leq k$, c_i is chosen uniformly at random from $X \setminus \cup_{j=1}^{i-1} C_j$. When a center c_i is deleted, we re-cluster only the points in $A = \cup_{j=i}^{k} C_j \cup U$, which requires at most $k \cdot |A|$ operations. Observe that the probability that c_i is selected as center is at most $\frac{1}{|A|}$, as each of the points in A were possible candidates when c_i was selected. In other words, the probability of selecting c_i as center is inversely proportional to the cost of handling the deletion of c_i (times k). Let $o_1, \ldots, o_m$ be a set of update operations which arefi xed by the adversary before the execution of the algorithm, where $o_i = (x, +)$ if x is added at step i or $o_i = (x, -)$ if x is removed. Consider the operation $o_i = (x, -)$. The algorithm maintains the following invariant: the probability of x being a center, when o_i is executed, is inversely proportional to the cost of handling the deletion of c_i. This suggests that the expected amortized cost is somehow limited. The analysis of the expected amortized cost of the algorithm is non-trivial, in that, deletions and insertions might intertwine arbitrarily. More efforts are needed to show that the amortized cost is concentrated around its expected value. A theoretical analysis of the algorithm is presented in Section 5. A formal description of the algorithm together with its pseudocode is given in Sections 4.1-4.4. In particular, Algorithms1 and 2 show the pseudocode of the procedures handling deletions.

Observe that the variant where we re-cluster all the points whenever a center is deleted might not be efficient. In particular, the adversary might be able to force a given point to become center and then repeatedly remove and insert back such a point. Each such an update operation would then incur in a total cost of $k \cdot n$, where n is the current number of points.

4.1 Data Structure for Dynamic Clustering

We shall denote with $d_{\min}$ and $d_{\max}$ the minimum and maximum distance between any two points that are ever inserted, respectively. We allow the set of update operations to be countably infinite, however, we assume that $d_{\min}$ and $d_{\max}$ always provide a lower or upper bound on the minimum and maximum distance between any two points, respectively.

We start describing the algorithm assuming that $d_{\min}$ and $d_{\max}$ are known in advance, while discussing the more general case in Section 4.4. Let $\Gamma := \{(1+\epsilon)^i : d_{\min} \leq (1+\epsilon)^i \leq d_{\max}, i \in \mathbb{N}\}$, and denote $\gamma := |\Gamma| = O(\frac{1}{\epsilon} \cdot \log \frac{d_{max}}{d_{\min}})$. For each $\beta \in \Gamma$, with respect to the current set X of points, we maintain a data structure $\mathcal{L}_\beta$ consisting of the following components and invariants:

- A list $S_\beta = \{c_1, c_2, \ldots, c_\kappa\}$ of $\kappa \leq k$ centers such that for all $x \neq y \in S_\beta$, $d(x, y) > 2\beta$.
- A collection $\mathcal{C}_\beta = \{C_1, C_2, \ldots, C_\kappa\}$ of disjoint clusters such that for all $i \in [\kappa]$, for all $x \in C_i$, $d(x, c_i) \leq 2\beta$.
- A set $U_\beta = X \setminus (\cup_{i \in [\kappa]} C_i)$ of unclustered points such that for all $i \in [\kappa]$, for all $x \in U_\beta$, $d(c_i, x) > 2\beta$. Moreover, we require that $U \neq \emptyset$ implies that $\kappa = k$.

Observe that we can store the data structure $\mathcal{L}_\beta$ using $O(|X|)$ space such that for any $x \in X$, it takes $O(1)$ time to return the cluster containing x and decide whether x is one of the centers in S_β.

4.2 Arbitrary Insertions

Inserting a new point x is straightforward. For each $\beta \in \Gamma$ consider the data structure $\mathcal{L}_\beta = (S_\beta, \mathcal{C}_\beta, U_\beta)$, where $\kappa = |S_\beta| = |\mathcal{C}_\beta|$. First check whether there is $c_i \in S_\beta$ such that $d(x, c_i) \leq 2\beta$. If this is the case insert x into the cluster C_i. If no such c_i is found and $\kappa < k$, then set $c_{\kappa+1} := x$ and $C_{\kappa+1} := \{x\}$. Otherwise, if there is no such c_i and $\kappa = k$ insert x into U_β.

4.3 Arbitrary Deletions

Maintaining $\mathcal{L}_\beta$ to support deletion of some point x is slightly trickier. The easy case is when $x \notin S_\beta$ is not one of the centers, and so the point x can simply be removed from its cluster in $O(1)$ time. However, when $x = c_i$ for some i, then we need to rebuild the data structure for the remaining points in $(\cup_{j \geq i} C_j) \cup U_\beta$.

4.4 All Pieces Together and Practical Aspects

Thefi rst step of the algorithm consists of initializing the $\mathcal{L}_\beta$'s, for each β in Γ, so that $S_\beta = \mathcal{C}_\beta = U_\beta \leftarrow \emptyset$. Then, the algorithm waits for an update operation o. If $o = (x, +)$ then the insertion procedure is executed, otherwise the deletion procedure depicted in Algorithm 1 and Algorithm 2 are executed. Observe, that the invariants discussed in Section 4.1 are always maintained. Therefore, a $(2+\epsilon)$-approximation for the k-center clustering problem is given by the clustering C_β with $\beta \in \Gamma$ being smallest such that U_β is the empty set.

For ease of presentation, we made the assumption that $d_{\min}$ and $d_{\max}$ are known in advance. In practice, this might not always be the case. If they are not known, one could compute $d_{\min}$ and $d_{\max}$

Algorithm 1 Point Deletion

procedure Delete($\mathcal{L}_\beta = (S_\beta, C_\beta, U_\beta), x$) ▷ Removes x and modifies $\mathcal{L}_\beta$ accordingly.
  if $x \notin S_\beta$ **then**
    Remove x from the cluster in C_β or U_β.
    return
  $\kappa \leftarrow |S_\beta|$
  Let $\{c_1, c_2, \ldots, c_\kappa\} \leftarrow S_\beta$.
  Let $\{C_1, C_2, \ldots, C_\kappa\} \leftarrow C_\beta$.
  Let $i \in [\kappa]$ such that $x = c_i$.
  $\widehat{X} \leftarrow (\cup_{j \geq i} C_j) \cup U_\beta$
  $(\widehat{S}, \widehat{C}, \widehat{U}) \leftarrow$ RandReclust$(\widehat{X}, \beta, k - i + 1)$
  $S_\beta \leftarrow \{c_1, c_2, \ldots, c_{i-1}\} \oplus \widehat{S}$ ▷ $\oplus$ does list concatenation
  $C_\beta \leftarrow \{C_1, C_2, \ldots, C_{i-1}\} \oplus \widehat{C}$
  $U_\beta \leftarrow \widehat{U}$
  return

Algorithm 2 Random Re-clustering

procedure RandReclust(X,β,k) ▷ Produces $\leq k$ centers (picked randomly) with pairwise distance $> 2\beta$, the corresponding clusters with radius $\leq 2\beta$, and a set U of unclustered points.
  $U \leftarrow X$
  $\kappa \leftarrow 0$
  while $U \neq \emptyset$ and $\kappa < k$ **do**
    $\kappa \leftarrow \kappa + 1$
    Pick c_κ from U, uniformly at random.
    $C_\kappa \leftarrow \{x \in U : d(x, c_\kappa) \leq 2\beta\}$
    $U \leftarrow U \setminus C_\kappa$
  return $(S \leftarrow \{c_1, c_2, \ldots, c_\kappa\}, C \leftarrow \{C_1, C_2, \ldots, C_\kappa\}, U)$ ▷ If $U \neq \emptyset$, then the optimal cluster radius is larger than β.

for thefirst k inserted points, while updating them throughout the execution of the algorithm. Observe that whenever $d_{\min}$ or $d_{\max}$ change, one would need to compute S_β, C_β, and U_β for each β in Γ that is missing. Removing such an assumption makes the algorithm more efficient in practice, without affecting the theoretical guarantees on the expected amortized cost.

We shall refer to our algorithm as FullyDynClust. In the next section, we shall prove strong theoretical guarantees on the expected amortized cost of our algorithm, while showing that its amortized cost is close to its expected value with high probability.

5 AMORTIZED ANALYSIS

We analyze the expected amortized cost of the deletion operation for the data structure $\mathcal{L}_\beta$. In Section 5.1, we perform a warmup analysis for the case in which a deletion operation removes a random point uniformly at random. In Section 5.2, we extend the analysis to the case of arbitrary insertions and deletions.

5.1 Warmup: Random Deletions

Observe that a deletion is costly only when one of at most k centers is removed. Hence, we readily have the following lemma.

Lemma5.1. *For the removal of a uniformly random point, the* Delete *operation in Algorithm 1 has expected cost* $O(k^2)$.

Proof. Observe that the cost is $O(nk)$ when a center in S_β is removed, where $n = |X|$ is the number of current points stored in the data structure; otherwise, the cost is $O(1)$. Since the probability that a center is removed is at most $\frac{k}{n}$, it follows that the expected cost is $O(k^2)$. □

5.2 Arbitrary Insertions and Deletions

For $t \in \mathbb{N}$, we use the superscript t to indicate the state of the data structure at the end of the t-th step. For instance, we use X^t to denote the set of points that are currently stored in the data structure $\mathcal{L}_\beta$ after step t, and we let $n^t := |X^t|$.

Intuition of Charging Scheme. Each Insert operation on the data structure $\mathcal{L}_\beta$ has cost $O(k)$ for every β in Γ. However, in order to pay for the cost of the Delete operations, we shall charge an extra cost of k^2 for each Insert operation that will be stored as a credit for future Delete operations. When a Delete operation is called, its cost will be paid for with (i) some of the credits stored (denoted by F^t), and (ii) possibly some extra cost (denoted by Z^t). Observe that a Delete operation is expensive only if one of the centers in S_β is removed. The following formal description defines F^t and Z^t explicitly.

Formal Description of Charging Scheme. We describe our charging scheme in details as follows. Suppose at step t, we have either of the following operations:

(1) Insert($\mathcal{L}_\beta, x$). A credit of k^2 is stored at the point x that is inserted. In this case, define $Z^t = F^t := 0$.
(2) Delete($\mathcal{L}_\beta, x$). If the point x to be deleted is not one of the centers, then the cost is $O(1)$, and we set $Z^t = F^t := 0$. Otherwise, suppose the point x to be deleted is the center c_i out of the current κ centers. Then, in this case, the rebuild cost is $O(k \cdot |\widehat{X}|)$, where $\widehat{X} := ((\cup_{j \geq i} C_j) \cup U_\beta) \setminus \{x\}$ are the points that need to be reclustered. Our charging scheme pays a cost of k for each point $u \in \widehat{X}$. Specifically, we decompose the reclustering cost $k \cdot |\widehat{X}| = F^t + Z^t$ in the following way, and we will analyze F^t and Z^t separately.
  (a) Denote $X_{\text{new}} := \{u \in \widehat{X} :$ when c_i was chosen as the center, the point u is not yet inserted$\}$. Define $F^t := k \cdot |X_{\text{new}}|$. In this case, for each $u \in X_{\text{new}}$, we will use k of the credits stored at u to pay for the cost; hence, this cost will not contribute towards Z^t. We shall prove in Lemma 5.2 that we will always have enough credits stored at the points in X_{new} to pay this way.
  (b) Denote $X_{\text{old}} := \{u \in \widehat{X} :$ when c_i was chosen as the center, the point u is already inserted$\}$. Define $Z^t := k \cdot |X_{\text{old}}|$. Hence, for each such point $u \in X_{\text{old}}$, we will incur a cost k that contributes towards Z^t.
  Observe that such a point u can be reclustered many times after its insertion. Apart from the initial times that can be paid in case 2(a), for each subsequent reclustering in some step τ, its reclustering cost will be counted towards the corresponding Z^τ.

Lemma 5.2 (Credits forR eclusteringN ew Points). *Fix $t \in \mathbb{N}$. Suppose a_t is the number of insertions from the beginning up to (and including) step t. Then, with probability 1, we have $\sum_{\tau=1}^{t} F^{\tau} \leq a_t \cdot k^2$.*

Proof. Observe that $a_t \cdot k^2$ is the total number of credits received by the points inserted up to step t. In order to show the required inequality, it suffices to show that the k^2 credits stored at each inserted point u will be enough to pay for its reclustering cost under case 2(a) in the charging scheme.

Suppose at the moment when u was inserted, the centers were $\{c'_1, c'_2, \ldots, c'_k\}$ and the clusters were $\{C'_1, \ldots, C'_k\}$ together with the unclustered set U'. Suppose $r \in [k]$ is the largest index such that $u \in (\cup_{j \geq r} C'_j) \cup U'$.

Then, the credits stored at u will be consumed only when the points in $W := \{c'_1, \ldots, c'_r\}$ are removed in some subsequent steps. Since there are at most k such centers in W, and k credits are consumed from u whenever such a center in W is removed, we can conclude that the k^2 credits stored at u will be enough. This completes the proof of the lemma. □

For each $t \in \mathbb{N}$, we also write $Z^t := \sum_{i=1}^{k} Z_i^t$, where Z_i^t is the extra cost incurred when the center c_i is deleted (at the beginning of step t). Observe that for afi xed t, there is at most one $i \in [k]$ such that Z_i^t is non-zero.

Lemma 5.3 (BoundingE xpectation). *For each $t \in \mathbb{N}$, $\mathbb{E}[Z^t] \leq k^2$.*

Proof. Recall that $Z^t := \sum_{i=1}^{k} Z_i^t$, where Z_i^t is the extra cost incurred when center c_i is deleted. Hence, it suffices to prove that $E[Z_i^t] \leq k$. Observe that c_i is a random object.

Observe that the center c_i was chosen in line 6 of some invocation of Algorithm 2. We use U_i to denote the set U in line 6 at the moment when c_i was chosen. Again, observe that U_i is a random object.

We next analyze $\mathbb{E}[Z_i^t | U_i]$ using the randomness used in line 6. Observe that when the point to be deleted is c_i only points in U_i contribute towards Z_i^t. However, some points in U_i could have been deleted by time t. Let $\widehat{U} \subseteq U_i$ denote the points that still remain at the beginning of step t.

Therefore, Z_i^t is non-zero only if the point $x_t \in \widehat{U}$ to be deleted at step t was chosen as the center in line 6, which happens with probability at most $\frac{1}{|\widehat{U}|}$ (conditioning on U_i).

Therefore, it follows that $\mathbb{E}[Z_i^t | U_i] \leq \frac{1}{|\widehat{U}|} \cdot k \cdot |\widehat{U}| = k$.

At this point, we would like to remind the reader that Z^t only accounts for the reclustering cost in case 2(b) of the description. For points that are inserted after c_i was chosen as the center, their reclustering costs are paid using the credits stored in themselves as in case 2(a) of the formal description of the charging scheme.

Taking expectation of the random variable $\mathbb{E}[Z_i^t | U_i]$ gives $\mathbb{E}[Z_i^t] \leq k$, as required. □

Theorem5.4. *For any $\epsilon > 0$, for any sequence of $T \in \mathbb{N}$ insert/delete operations, the FullyDynClust algorithm maintains a $(2+\epsilon)$-approximation solution for the k-center problem, while requiring $O(\frac{\log \delta}{\epsilon} \cdot k^2 T)$ expected time and $O(\frac{\log \delta}{\epsilon} \cdot |N|)$ space under the fully dynamic model, where $N := \max_{t \in [T]} |X^t|$. Clustering membership can be tested in $O(1)$ time, while producing in output all points in the cluster C of a given point requires $O(k + |C|)$ time.*

Proof. The algorithm always maintains all the invariants discussed in Section 4.1. Therefore, the clustering C_β where β is smallest such that U_β is the empty set, gives a $(2+\epsilon)$-approximation. For any given β in Γ, the total cost due to deletion operations in $\mathcal{L}_\beta$ is $\sum_{t=1}^{T} O(F^t + X^t)$. Lemmas 5.2 and 5.3 imply that its expectation is at most $O(k^2 T)$. We conclude the proof by recalling that there are $|\Gamma| = O(\frac{\log \delta}{\epsilon})$ values of β. □

High Probability Statements. Lemma 5.3 implies that any sequence of T insert/delete operations on a data structure has expected cost $O(k^2 T)$. We next show that with high probability, the cost is also $O(k^2 T)$. As seen in the proof of Lemma 5.3, our analysis is based on the randomness used in line 6 of Algorithm 2.

Suppose N is an upper bound on the number of points stored by the data structure at any time. For $n \leq N$, consider the random variable ζ_n that takes value n with probability $\frac{1}{n}$ and value 0 with probability $1 - \frac{1}{n}$. As in the proof of the well-known Chernoff bound, we analyze the moment generating function of ζ_n to prove measure concentration results.

Lemma 5.5 (MomentG eneratingF unction of ζ). *For $n \leq N$ and $0 \leq \theta \leq \frac{1}{N}$, $\mathbb{E}[e^{\theta \zeta_n}] \leq e^{\theta + \frac{3}{4} \cdot \theta^2 N}$.*

Proof. For $0 \leq \theta \leq \frac{1}{N}$, we have

$$\begin{aligned} \mathbb{E}[e^{\theta \zeta_n}] &= (1 - \frac{1}{n}) + \frac{1}{n} \cdot e^{\theta n} && (1) \\ &\leq \exp(\frac{e^{\theta n} - 1}{n}) && (2) \\ &\leq e^{\theta + \frac{3}{4} \cdot \theta^2 n} && (3) \\ &\leq e^{\theta + \frac{3}{4} \cdot \theta^2 N}, && (4) \end{aligned}$$

where (2) comes from $1 + x \leq e^x$, and (3) comes from the inequality $e^x \leq 1 + x + \frac{3}{4} \cdot x^2$ for $0 \leq x \leq 1$. □

Lemma5.6. *Suppose N is an upper bound on the number of points stored by the data structure $\mathcal{L}_\beta$ at any time, andfi x any $i \in [k]$. For any $0 \leq \theta \leq \frac{1}{kN}$ and $T \in \mathbb{N}$, we have $\mathbb{E}[e^{\theta \sum_{t \in [T]} Z_i^t}] \leq e^{T(\theta k + \frac{3}{4} \cdot \theta^2 k^2 N)}$.*

Proof. We prove the statement by induction on T. The base case $T = 0$ holds trivially. We next consider step $T + 1$. Let $\mathcal{F}$ be the sigma-algebra generated by the random variables $(Z_i^t : t \in [T])$ together with the indicator variables $(I_j^t : j \in [k], t \in [T])$, where $I_j^t = 1$ *iff* a new center c_j is picked in step t.

We next define a random object U_{T+1}. At the beginning of step $T + 1$, consider the center c_i and the moment when it was picked in some previous step. Observe that c_i was picked in line 6 of some invocation of Algorithm 2. Define U_{T+1} to be the set U in line 6 at that moment. Observe that some points in U_{T+1} could be deleted from the time c_i was picked as the center to the beginning of step $T + 1$. Denote $\widehat{U} \subseteq U_{T+1}$ as the points that still remain at the beginning of step $T + 1$.

Similar to the proof of Lemma 5.3, we show that conditioning on the history ($\mathcal{F}$ and U_{T+1}), the random variable Z_i^{T+1} is stochastically dominated[1] by $k \cdot \zeta_{|\widehat{U}|}$; note that this trivially holds if the $(T+1)$-st operation is an insertion, because Z_i^{T+1} equals 0 in this case.

We next argue why this is also true when the $(T+1)$-st operation is a deletion. Observe that conditioning on the history $\mathcal{F}$ and U_{T+1}, we know exactly in which step the current center c_i was picked, namely, the largest $i \leq t$ such that $I_i^t = 1$. Therefore, conditioning on $\mathcal{F}$ and U_{T+1}, the current center c_i is a uniformly random point in $\widehat{U}$. Therefore, it follows that the center c_i is deleted in step $T+1$ with probability at most $\frac{1}{\widehat{U}}$, which incurs a cost of $k \cdot |\widehat{U}|$ when this event happens. Hence, conditioning on $\mathcal{F}$ and U_{T+1}, Z_i^{T+1} is stochastically dominated by $k \cdot \zeta_{|\widehat{U}|}$, as required. Hence, from Lemma 5.5, we have for $0 \leq \theta \leq \frac{1}{N}$, $\mathbb{E}[e^{\theta Z_i^{T+1}} | \mathcal{F}, U_{T+1}] \leq e^{\theta k + \frac{3}{4} \cdot \theta^2 k^2 N}$.

Finally, the inductive step is completed by considering the following conditional expectation:

$$\begin{aligned} \mathbb{E}[e^{\theta \sum_{t \in [T+1]} Z_i^t} | \mathcal{F}, U_{T+1}] &= e^{\theta \sum_{t \in [T]} Z_i^t} \cdot \mathbb{E}[e^{\theta Z_i^{T+1}} | \mathcal{F}, U_{T+1}] \\ &\leq e^{\theta \sum_{t \in [T]} Z_i^t} \cdot e^{\theta k + \frac{3}{4} \cdot \theta^2 k^2 N}. \end{aligned}$$

Then, taking expectation and using the induction hypothesis completes the induction proof. □

Lemma5.7. *Suppose N is an upper bound on the number of points stored by the data structure $\mathcal{L}_\beta$ at any time, andfi x any $i \in [k]$. For any $\lambda \geq 2$, $\Pr[\sum_{t \in [T]} Z_i^t \geq \lambda k T] \leq e^{-\Omega(\frac{\lambda T}{N})}$.*

Proof. Using the standard method of moment generating function as in the proof of the Chernoff bound, we choose $\theta = \frac{1}{kN}$ in the following:

$$\begin{aligned} \Pr[\sum_{t \in [T]} Z_i^t \geq \lambda k T] &= \Pr[e^{\theta \sum_{t \in [T]} Z_i^t} \geq e^{\theta \lambda k T}] && (5) \\ &\leq \mathbb{E}[e^{\theta \sum_{t \in [T]} Z_i^t}] \cdot e^{-\theta \lambda k T} && (6) \\ &\leq e^{-\frac{(\lambda - \frac{7}{4})T}{N}}, && (7) \end{aligned}$$

where (6) follows from Markov's inequality and (7) follows from Lemma 5.6 and the choice of $\theta = \frac{1}{kN}$. □

The following theorem follows from Theorem 5.4 and Lemma 5.7.

Theorem5.8. *Let N be an upper bound on the number of points at any point in time ($|X^t| \leq N$, for all t). For any $\epsilon > 0$, for any sequence of $T \in \mathbb{N}$ insert/delete operations the FullyDynClust algorithm maintains a $(2+\epsilon)$-approximation solution for the k-center problem.*

Moreover, the probability that the total time exceeds $\Omega(\frac{\log \delta}{\epsilon} \cdot \lambda k^2 T)$ is at most $\frac{\log \delta}{\epsilon} \cdot k e^{-\Omega(\frac{\lambda T}{N})}$, for any $\lambda \geq 2$, under the fully dynamic model. The algorithm requires $O(\frac{\log \delta}{\epsilon} \cdot N)$ space.

[1] A random variable Z is stochastically dominated by a random variable Y if for all real x, $\Pr[Z \geq x] \leq \Pr[Y \geq x]$.

6 EXPERIMENTS

All our experiments have been carried on a machine equipped with two Intel(R) Xeon(R) CPU E5-2660 v3 running at 2.60GHz and with 250Go of DDR3 SDRAM. The Twitter dataset we used as well as our implementation in C of our algorithm have been made publicly available [2]. Observe that for each tweet we retain only its timestamp and its GPS coordinates, due to privacy concerns. We evaluate the algorithms under a wide range of values for k, ϵ and the length of the sliding window W. In all experiments the default values are $k = 20$ and $\epsilon = 0.1$, unless otherwise specified. For our randomized algorithm, each result is the average among ten runs, unless otherwise specified.

6.1 Datasets

We consider some datasets that are publicly available, as well as a dataset that we collected from Twitter.

Twitter. We collected geotagged tweets by means of the Twitter API. We were able to collect 21M tweets between 9/09/2017 and 20/10/2017. Each tweet is associated with GPS coordinates, such as longitude and latitude, as well as, a timestamp.

Flickr. The Yahoo Flickr Creative Commons 100 Million (YFCC100m) dataset [3] is a dataset containing the metadata of 100 Million picture posted on Flickr under the creative common licence between 2011 and 2015. Each picture is associated with GPS coordinate and a timestamp. We used the metadata of 47M pictures that possessed both a valid timestamp and GPS coordinate.

Trajectories. This dataset contains trajectories performed by all the 442 taxis running in the city of Porto, in Portugal [4]. Each trajectory consists of a set of two-dimensional points, each one being associated with a timestamp.Trajectories are updated every 15 seconds, with most of the trajectories consisting of at most 500 points. The dataset contains 83M two-dimensional points, in total leading to 83M updates.

Table 2 summarizes the statistics of the datasets considered in our experimental evaluation.

6.2 Distance Measures

Depending on the dataset at hand, we consider different distance measures. In the case of Twitter and Flickr, we compute the great circle distance between two GPS coordinates [15]. For the trajectories we use the symmetric Hausdorff distance [16], which is defined as follows. Let P and Q be two sets of points in the Euclidean space. $H(P,Q) := \max_{p \in P} \min_{q \in Q} d(p,q)$ where d is the Euclidean distance. The symmetric Hausdorff distance between two trajectories P and Q is defined as $\widehat{H}(P,Q) = \max(H(P,Q), H(Q,P))$. All the distance measures considered in our experiments are metrics.

6.3 Dynamic Model of Computation

In the case of Twitter and Filckr, we evaluated the algorithms under the sliding window model [6, 7]. In this case, if a point is inserted at time t, it will be removed at time $t + W$, where W is the duration of

[2] https://github.com/fe6Bc5R4JvLkFkSeExHM/k-center

[3] http://yfcc100m.appspot.com/

[4] https://archive.ics.uci.edu/ml/datasets/Taxi+Service+Trajectory+-+Prediction+Challenge,+ECML+PKDD+2015

Name	Source	Type	#Updates
Twitter	twitter.com	2D points	42M
Flickr	yfcc100m.appspot.com	2D points	96M
Porto taxi trajectories	archive.ics.uci.edu	Trajectories	83M

Table 2: Dataset statistics.

the sliding window. As a result, at each point in time t, new points might be added to the current dataset in which case any point with timestamp $t + W$ will be removed (if any).

We then focus on the task of maintaining a clustering of all trajectories available up to any given point. At any point in time, either a new point is added to one of the available trajectories at that time or a new trajectory is created. Since trajectories might be updated in any arbitrary order, the sliding window model turns out to be over-simplistic in this setting. Therefore the approach in [6] cannot be used.

6.4 Flickr and Twitter

We evaluate the algorithms under a wide range of values for k, ϵ and the length of the sliding window W. In all experiments the default values are $k = 20$, $eps = 0.1$. For Twitter, the length of the sliding window is two hours, while for Flickr it is six hours. Both those values result in a sliding window containing 60K points.

We start by evaluating the accuracy of the two algorithms. In particular, we measure the ratio between the maximum radius of the clustering produced by each algorithm and a lower bound on the optimum radius. At any point in time, we compute such a lower bound as follows. Let β be largest such that the set U_β computed by our algorithm is not the empty set. Our lower bound is computed as half the minimum pairwise distance between the $k + 1$ points (cluster centers) in S_β. We observe that in practice it is important to have small maximum radius, in that, this might lead to more "meaningful" clusters.

Figure 1 (left) measures the approximation ratio as a function of k, while Figure 1 (right) measures the approximation ratio as a function of ϵ in Twitter. For any given value of k and ϵ we report the median, thefi rst and third quartiles, as well as maximum and minimum approximation ratio obtained by the algorithm when executed on the whole dataset. The two algorithms have been tested for the same value of k and ϵ, however, some additional space between the corresponding values in the plot has been introduced to improve the presentation.

We can observe that our algorithm (FD) consistently delivers better approximation ratio then the algorithm presented in [6] (SW). In particular, we can observe that while the approximation ratio of our algorithm is always close to 2 when $\epsilon = 0.1$, it can be as large as 6 for SW with its median being as large as 4.5. This is expected, as our algorithm comes with stronger guarantees on the maximum radius. We can also see that on average FD performs better than its worst-case analysis suggests.

This is even more apparent in Flickr (Figure 3), where our algorithm produces results being closer to an optimum solution. In particular the median for the case when $k = 100$ and $\epsilon = 0.1$ ((Figure 3) (left)) is close to 1.5, while in a few cases a near-optimal

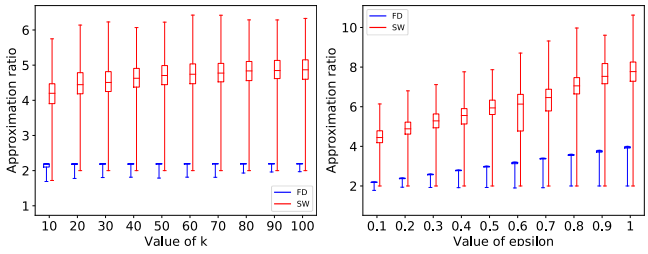

Figure 1: Ratio between the maximum radius and a lower bound on the optimum radius as a function of k (left) and ϵ (right) on Twitter. The two algorithms have been tested for the same value of k and ϵ, however, some additional space between the corresponding values in the plot has been introduced to improve the presentation.

solution is computed. We can also observe that the maximum radius for SW drops as k increases, while it improves again when k is large enough (Figure 3) (left)). This might be due to the way the deletion of a center is handled in SW. If a center is removed, the cluster is not reconstructed as in our case but a new point (called *orphan*) might be elected as representative of the cluster. Although this might be more efficient in practice, it might affect the quality of the results, in that, orphans might not be as good representatives of the clusters as the original centers. As a result, points might be added to the "wrong" clusters. As k increases, it becomes more likely that there is at least one "bad" orphan, which explains the drop in quality of the results. For even larger values of k, the task of clustering becomes "easier" with the easiest case being when k approaches the total number of points.

Figure 1 (right) and Figure 3 (right) show the approximation ratio as a function of ϵ for Twitter and Flickr, respectively. Those figures confirm that our algorithm produces better results in practice. Moreover, we can see that the approximation ratio worsens as ϵ increases, which is expected.

We then move to evaluate the average running time of the two algorithms, which is defined as the total running time of the algorithms divided by the total number of update operations. The expected running time of our algorithm has been studied in Theorem 5.4. Figure 2 shows the average running time of the two algorithms as a function of k (left) and as a function of ϵ (right) in Twitter, while Figure 4 shows the corresponding plots for Flickr. We can see that SW is consistently more efficient than our algorithm, which is expected. However, both algorithm are very efficient with the average running time being at most $5 * 10^{-4}$ seconds and as small as $2 * 10^{-5}$ seconds in some cases.

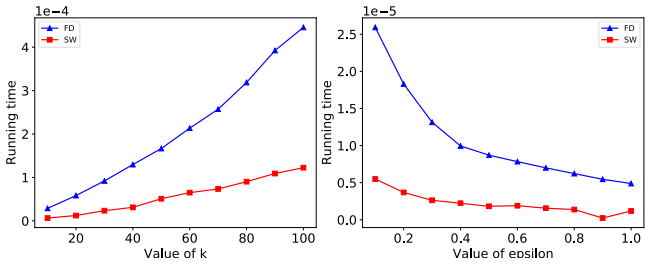

Figure 2: Running time of the algorithms in second as a function of k (left) and ϵ (right) on Twitter.

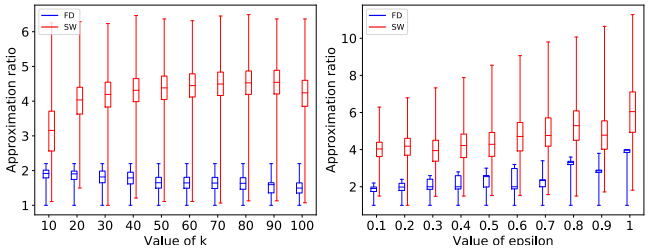

Figure 3: Ratio between the maximum radius and a lower bound on the optimum radius as a function of k (left) and ϵ (right) on Flickr. The two algorithms have been tested for the same value of k and ϵ, however, some additional space between the corresponding values in the plot has been introduced to improve the presentation.

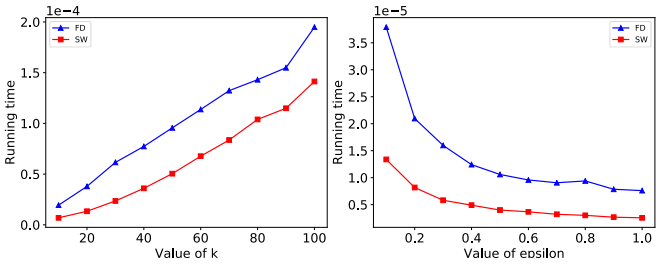

Figure 4: Running time of the algorithms in second as a function of k (left) and ϵ (right) on Flickr.

6.5 Trajectory Data and Multicore Implementation

In this section, we consider the task of clustering trajectory data. We consider the scenario where at each point in time, either a new point is added to one of the existing trajectories or a new trajectory consisting of one single point is created. Each update can therefore be seen as either an insertion of a new trajectory or a deletion of an existing trajectory followed by an insertion of the updated trajectory. Since trajectories can be modified in any arbitrary order, the sliding window model appears to be simplistic in this setting. Therefore, the algorithm in [6] cannot be used. The distance between any two trajectories is computed as the symmetric Hausdorff distance (see Section 6.2), which is a metric.

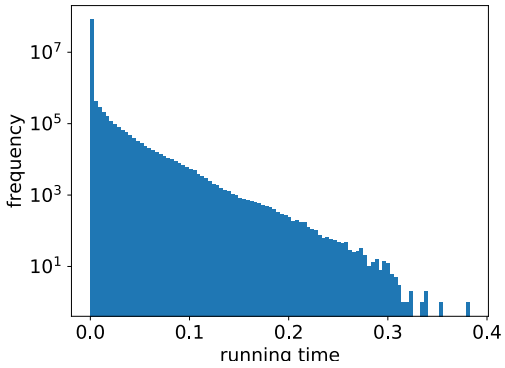

Figure 5: Histogram of the running time of each operation during a single run on trajectories of our algorithm

Computing the distance between two trajectories requires $\Omega(r)$ operations, where r is the maximum number of points in the two trajectories. In our dataset, most trajectory contain up to 500 points with the largest one containing nearly 4000 points. This makes the problem of clustering trajectory data more computationally extensive than in the case of two-dimensional euclidean space. Therefore, in order to be able to conduct an extensive evaluation of the algorithm, we consider a multicore implementation. Since the clusterings C_β's can be processed independently, a multicore implementation is straightforward: the C_β's are partitioned across the threads, with each thread taking care of all deletions and insertions in the clusterings associated to such a thread. All our experiments discussed in this section use 30 threads.

Wefi rst evaluate the running time of our algorithm. Figure 5 shows the histogram for the running time of each update operation (in seconds). In particular, for each running time value, the number of update operations requiring that running time is reported. We can see that approximately 80M operations require less than $4*10^{-3}$ seconds, while the maximum running time per operation is 0.38 seconds. Figure 6 shows the average running time as a function of the number of update operations. At any point in time, the average running time is computed as the total running time divided by the total number of operations up to that point. The running time is measured in seconds. We can see that the average running time is always less than one millisecond, while it becomes stable after approximately 18 million update operations. This is consistent with Theorem 5.8, which roughly speaking states that the probability of significantly deviating from the expected running time becomes very low when the number of update operations is large.

As for the approximation ratio, using the default value of k and ϵ, we obtain a median of 2.05, while thefi rst and third quartile are respectively equal to 2.03 and 2.13.

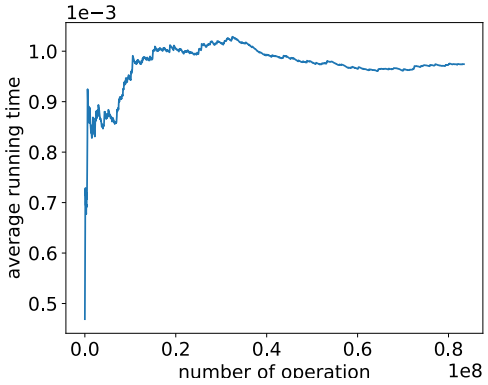

Figure 6: Evolution of the average running time per operation during a single run

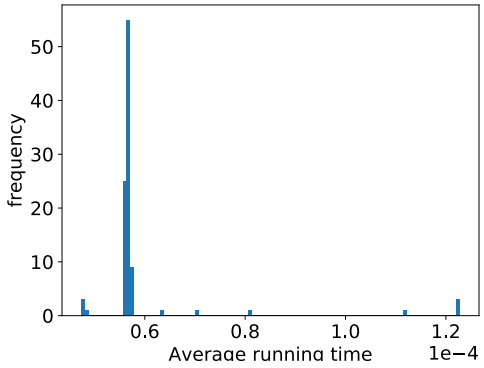

Figure 7: concentration of measure experiments

6.6 Concentration around the Expectation

Our last experiment aims to study the concentration around the expected running time (i.e. Theorem 5.8) from an experimental point of view. We consider 100 runs of our algorithm on the Twitter dataset using the default values and we show the histogram of the average running time in Figure 7. We can see that almost all runs require $5.8 * 10^{-4}$ seconds on average, while in very few cases the average running time is more than $12 * 10^{-4}$ seconds, which is consistent with Theorem 5.8.

7 CONCLUSION AND FUTURE WORK

In our work, we developed a $(2 + \epsilon)$-approximation algorithm for k-center clustering, under a fully dynamic adversarial model of computation. In such a model, points can be inserted or deleted arbitrarily, provided that the adversary does not have access to the random choices of our algorithm. Our theoretical analysis, shows that the expected amortized cost of our algorithm is poly-logarithmic (in some cases of interest) while we show concentration around its expected value. Our work is thefi rst work with strong theoretical guarantees for fully dynamic k-center clustering, to the best of our knowledge.

We then conducted an extensive evaluation of our algorithm on dynamic data from Twitter, Flickr, as well as trajectory data. Our experiments show that our algorithm produces clustering solutions with smaller maximum radius in comparison with state-of-the-art approaches, although this comes at the price of slightly worse average running time and more space.

For future work, we are planning to study other data mining and machine learning algorithms under the fully dynamic model of computation.

REFERENCES

[1] R. P. L. A. Epasto, S. Lattanzi. Ego-splitting framework: from non-overalapping to overalapping clusters. In *KDD 2017*.
[2] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 1027–1035, 2007.
[3] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++. *PVLDB*, 5(7):622–633, 2012.
[4] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 626–635. ACM, 1997.
[5] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 30–39, 2003.
[6] V. Cohen-Addad, C. Schwiegelshohn, and C. Sohler. Diameter and k-center in sliding windows. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 19:1–19:12, 2016.
[7] A. Epasto, S. Lattanzi, and M. Sozio. Efficient densest subgraph computation in evolving graphs. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pages 300–310, 2015.
[8] S. L. F. Chierichetti, R. Kumar and S. Vassilvitskii. Fair clustering through fairlets. In *NIPS*, 2017.
[9] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293 – 306, 1985.
[10] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 359–366, 2000.
[11] S. Gupta, R. Kumar, K. Lu, B. Moseley, and S. Vassilvitskii. Local search methods for k-means with outliers. *PVLDB*, 10(7):757–768, 2017.
[12] S. Lattanzi and S. Vassilvitskii. Consistent k-clustering. In *ICML*, 2017.
[13] M. D. M. H. S. L. M. Bateni, S. Behnezhad and V. Mirrokni. On distributed hierarchical clustering. In *NIPS 2017*.
[14] R. Matthew Mccutchen and S. Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. APPROX '08 / RANDOM '08, pages 165–178, 2008.
[15] H. Steinhaus. *Mathematical Snapshots*. 3rd ed. New York, 1999.
[16] A. A. Taha and A. Hanbury. An efficient algorithm for calculating the exact hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015.