Dynamic Facility Location in High Dimensional Euclidean Spaces

Sayan Bhattacharya [1] **Gramoz Goranci** [2] **Shaofeng H.-C. Jiang** [3] **Yi Qian** [3] **Yubo Zhang** [3]

Abstract

We consider the facility location problem in the dynamic setting, where the goal is to efficiently process an intermixed sequence of point insertions and deletions while maintaining a high-quality and stable solution. Although the problem has been studied in the context of general metrics and low-dimensional spaces, much remains unknown concerning dynamic facility location in high-dimensional spaces. In this work, we present the first fully dynamic algorithm for facility location in high-dimensional spaces $\mathbb{R}^d$. For any $c \geq 1$, our algorithm achieves $O(c)$-approximation, supports point updates in $\tilde{O}(\mathrm{poly}(d)n^{1/c+o(1)})$ amortized time and incurs $O(1)$ amortized recourse. More generally, our result shows that despite the linear-time lower bound on the update time for general metrics, it is possible to achieve sub-linear update times for metric spaces that admit dynamic nearest neighbour oracles. Experiments on real datasets confirm that our algorithm achieves high-quality solutions with low running time, and incurs minimal recourse.

1. Introduction

Clustering is one of the most important problems in unsupervised learning, finding a wide range of applications in social network analysis, image segmentation, anomaly detection, and grouping of search results, among others. A classical problem at the core of cluster analysis is the facility location problem: the input is a set of points (known as 'clients') in a metric space, and the goal is to find a subset of clients (known as 'facilities'), so as to minimize the cost of opening the facilities plus the cost of assigning clients to their closest facilities. Facility Location is an NP-complete problem whose approximability has been well-studied (Jain & Vazirani, 2001; Mettu & Plaxton, 2003). Because of its practical relevance and connections to other areas such as operations research, this problem has also been extensively studied under different computational paradigms, including the streaming model (Indyk, 2004; Lammersen & Sohler, 2008; Czumaj et al., 2013), massive parallel computation (MPC) model (Czumaj et al., 2024), online algorithms (Meyerson, 2001; Fotakis, 2008; Cygan et al., 2018), dynamic algorithms (Goranci et al., 2018; Cohen-Addad et al., 2019; Guo et al., 2020; Bhattachcharya et al., 2022), and many more.

In many modern applications of facility location, the underlying data evolves dynamically. While many efficient approximation algorithms have been developed for facility location, these algorithms usually operate under the assumption that input data is static. To tackle the evolving nature of data, there has been a recent and growing focus on facility location, and more broadly, clustering, in the *fully dynamic* setting. Here, the input data undergoes an intermixed sequence of point insertions and deletions, referred to as *updates*. The goal is to process these updates as *efficiently* as possible, ensuring that (i) the maintained clustering achieves a small approximation ratio and that (ii) the number of changes in the clustering between any two adjacent updates, known as the *recourse*, is minimal.

Much of the work on dynamic facility location has focused on general metrics and low-dimensional spaces. For the former, (Cohen-Addad et al., 2019) showed an algorithm that achieves $O(1)$ approximation, $O(n \log n)$ update time, and $O(1)$ amortized recourse. One drawback of considering general metric spaces is the natural linear-update time lower bound: inserting a point entails describing distances to other points, which requires $\Omega(n)$ time. In the low-dimensional regime, (Goranci et al., 2018) presented an $O(1)$-approximation algorithm with $\tilde{O}(1)$ update time. However, their update time grows exponentially with the dimension, which is prohibitive for high-dimensional Euclidean spaces.

In this work, we give the first fully dynamic algorithm

The authors are listed in alphabetical order. [1]University of Warwick, UK [2]Faculty of Computer Science, University of Vienna, Austria [3]Peking University, China. Correspondence to: Sayan Bhattacharya <S.Bhattacharya@warwick.ac.uk>, Gramoz Goranci <gramoz.goranci@univie.ac.at>, Shaofeng H.-C. Jiang <shaofeng.jiang@pku.edu.cn>, Yi Qian <qianyi@stu.pku.edu.cn>, Yubo Zhang <zhangyubo18@pku.edu.cn>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

for facility location in high-dimensional Euclidean spaces. Specifically, our algorithm achieves $O(1)$ approximation in *sub-linear* in n update time. Our main result and its exact guarantees are summarized in the theorem below.

Theorem 1.1. *For any $c \geq 1$, there is a fully dynamic algorithm that supports insertions and deletions of points from a d-dimensional Euclidean space $\mathbb{R}^d$, and whp, it maintains a $O(c)$-approximation to the facility location problem in $\tilde{O}(\mathrm{poly}(d)n^{1/c+o(1)})$ amortized update time and $O(1)$ amortized recourse.*

In fact, the above result is a corollary of a much stronger theorem, as outlined in Theorem 4.10. For any $\rho, \tau \geq 1$, and any metric space that admits a dynamic nearest neighbour (NN) oracle that is ρ-approximate and support updates and queries in $O(\tau)$ time, we obtain a fully dynamic algorithm for facility location that has $O(\rho^2)$ approximation ratio, $\tilde{O}(\tau)$ amortized update time and $O(1)$ amortized recourse. This black-box reduction combined with the implementation of NN oracle for Euclidean space via *locally sensitive hash functions* (LSH) (Har-Peled et al., 2012) leads to the trade-off presented in Theorem 1.1. Similar improved dynamic algorithms for facility location can be obtained for metric spaces that admit competitive LSH guarantees including ℓ_p spaces (Datar et al., 2004), the Hamming metric (Har-Peled et al., 2012), and the Jaccard metric (Broder, 1997). Finally, we remark that the reduction to NN oracle queries has also been used for obtaining sub-quadratic running times for approximate high-dimensional proximity problems in the static setting (Goel et al., 2001).

To complement our primary contribution (see Theorem 1.1), we perform an experimental evaluation of our algorithm on real datasets. The results obtained demonstrate that our algorithm consistently produces high-quality solutions, and the recourse of the maintained facilities is minimal.

1.1. Technical Overview

For ease of exposition, we assume w.l.o.g. that the opening cost $f = 1$, and that we have access to an exact dynamic nearest neighbor (NN) oracle. Only minor changes to the ideas described in this section are required to handle the error introduced by the approximate NN.

Technical Contribution: Near Neighbor Indicator Structure Before we discuss how our algorithm works, we first introduce a general data structure called near-neighbor indicator structure (see Section 3), which is our main technical contribution. This structure is crucially used in major steps of our algorithm. Due to the fundamental nature of this structure, we believe it may be of independent interest to the study of dynamic algorithms in $\mathbb{R}^d$.

This structure enhances the vanilla NN oracle, such that it not only answers the nearest neighbor for one point, but also explicitly maintains an (approximate) near neighbor for every point present in the data structure simultaneously. Specifically, given a threshold $\lambda > 0$, under point insertion and deletions, this structure maintains for each point $x \in P$ (where P is the current point set) an indicator $b_\lambda(x) \in \{0, 1\}$, such that $b_\lambda(x) = 1$ roughly means x has a neighbor within $O(\lambda)$ distance in P. Furthermore, on each point insertion or deletion, it only needs to make $O(1)$ vanilla NN queries (see Theorem 3.1).

A Basic Static Algorithm: Modified Mettu-Plaxton Our dynamic algorithm is based on a modified version of the Mettu-Plaxton algorithm (MP), (Czumaj et al., 2024) which returns a $O(1)$-approximate solution to the facility location problem (see Section 2.1). We start with a brief review of the original MP algorithm, which is suggested by (Mettu & Plaxton, 2003). For every data point x in the dataset P, the MP algorithm computes a value r_x, such that $\sum_x r_x = \Theta(1) \cdot \mathrm{OPT}$. Then it scans the data points in non-decreasing order of r_x, and opens the current point x as a facility if so far no facility is open in the metric ball $B(x, 2r_x)$.

The procedure for computing the set of open facilities in the original MP algorithm is quite global/sequential and hence is hard to dynamize, while the modified version, which our algorithm is based on, is more local. In particular, in addition to computing the r_x value for every data point x, it also assigns a random value $h(x) \in [0, 1]$ to each $x \in P$. Then this modified MP algorithm decides to open x if $h(x)$ is the smallest among the points in $B(x, r_x)$[1].

Our Dynamic Algorithm To make this modified MP algorithm dynamic, our strategy consists of two parts. The first part explicitly maintains r_x's, which are then taken as input in the second part to maintain the set of open facilities.

We start with the first part which maintains approximations to r_x's (see Section 4.1). As first observed in (Badoiu et al., 2005), to compute an $O(1)$-approximation to r_x, it suffices to know for $r' = 1/n, 2/n, \ldots, 1$, whether $B(x, r')$ contains at least $\Omega(1/r')$ data points. To test if $B(x, r')$ contains at least $\Omega(1/r')$ data points with $\Theta(1)$ success probability, it suffices to first do a sub-sampling with surviving rate r' to the data points, denoting the resultant dataset as $P_{r'}$, and test if the nearest neighbor of x in the survived points $P_{r'}$ is within distance r'. We follow exactly this idea, and the key step is to make use of the above-mentioned near neighbor indicator structure, to maintain for every data point x if there is a near enough point after the sub-sampling (see Corollary 3.2 and its proof in Section 3.2).

[1]There is another rule, which independently opens each x with probability r_x, but this is trivial to dynamize and we choose to ignore this step in the technical overview.

Now, for the second step, we need to follow the modified MP algorithm, and maintain for every data point $x \in P$, if it has the smallest h value in the ball $B(x, r_x)$ (recalling that for every point $x \in P$ we assign an independent uniform random value $h(x) \in [0, 1]$). In fact, this task is somewhat similar to that of maintaining r_x, and can be "reduced" to the near neighbor indicator data structure (see Section 4.2). In particular, we first "discretize" the h values by rounding them up to the next power of 2. After this step there are only $O(\log n)$ distinct h values. We manage to show that plugging this rounded version of h (which we call $h^\star$) into the modified MP algorithm, it still yields an $O(1)$-approximation (see Lemma 4.5). Now, to test if $h^\star(x)$ is the smallest among points within the ball $B(x, r_x)$, we make the following crucial observation: If $h^\star(x)$ indeed is the smallest in the ball, then with high probability (whp) there are very few, that is $\mathrm{poly}(\log n)$, points within $B(x, r_x)$ that achieve this minimum $h^\star$ value. Hence, it suffices to do a sub-sampling with survival rate $\mathrm{poly}(\log n)$ of the data points, and test if there is a survived point within distance r_x to x whose $h^\star$ value is larger than $h^\star(x)$. This procedure can again be done using near neighbor indicator structure, similar to the steps for maintaining the r_x's.

1.2. Related Work

Recently, there has been a surge of interest in designing dynamic algorithms for various clustering objectives. This line of work was initiated by (Charikar et al., 2004) who gave an insertions-only algorithm for approximate k-center clustering. Their result was extended to the fully dynamic setting by the work of (Chan et al., 2018a), who also re-emphasized the importance of dynamic clustering algorithms in the machine learning literature. Since then, several dynamic algorithms have been developed for clustering problems such as k-median/k-means (Cohen-Addad et al., 2019; Henzinger & Kale, 2020; Bhattachcharya et al., 2023) and k-center (Goranci et al., 2021; Bateni et al., 2023; Cruciani et al., 2024; Biabani et al., 2023; Lacki et al., 2024; Chan et al., 2024). In the high dimensional regime, the only prior work we are aware of is the dynamic k-center algorithm in (Bateni et al., 2023).

2. Preliminaries

Throughout this paper, we assume that we are dealing with "points" embedded in a specific metric space. For any two points x, y, let $\mathtt{dist}(x, y)$ denote the distance between them in the underlying metric. For any point-set S and any point x (not necessarily in S), we define $\mathtt{dist}(x, S) := \min_{y \in S} \mathtt{dist}(x, y)$.[2] In the FACILITY LOCATION problem, the input is a set of n points P, and a parameter $f > 0$. We have to "open" a subset $F \subseteq P$ of "facilities", so as to minimize $f \cdot |F| + \sum_{x \in P} \mathtt{dist}(x, F)$. Here, f denotes the "opening cost" of a facility, and $\mathtt{dist}(x, F)$ denotes the "connection cost" we have to pay for assigning point $x \in P$ to its nearest open facility. We let $\mathtt{Opt}$ denote the optimal objective value for the concerned input instance.

This paper is focused on the *dynamic setting*, where the set P changes via a sequence of "updates". Each update inserts/deletes a point in P. Throughout these updates, we have to explicitly maintain the subset $F \subseteq P$ of open facilities and an (approximate) estimate of the optimal objective value. The time taken to handle an update is called the "update time" of the concerned algorithm. Another important property of a dynamic algorithm is its "recourse", which is defined as the number changes (insertions/deletions of points) to the maintained solution F due to an update. We wish to ensure that our update time remains as small as possible, and ideally, obtain $O(1)$ recourse.

Notations For any point $x \in P$, any subset $S \subseteq P$, and any $r \geq 0$, we let $B_S(x, r) := \{y \in S : \mathtt{dist}(x, y) \leq r\}$ denote the "ball of radius r" around x (w.r.t. S). Throughout the rest of the paper, w.l.o.g. we assume that the facility opening cost $f = 1$.

Nearest-Neighbor Oracle We will assume that there is an oracle, with parameters $\rho, \tau \geq 1$, that can maintain a dynamic point-set S, and supports the following operations. (i) UPDATE: Insert/delete a point in S. (ii) QUERY: Given a point p (not necessarily in S), return a ρ-approximate nearest neighbor of p in S, which we denote by $\mathrm{NN}(p, S)$. Thus, $\mathrm{NN}(p, S) \in S \setminus \{p\}$,[3] and $\mathtt{dist}(p, \mathrm{NN}(p, S)) \leq \rho \cdot \mathtt{dist}(p, S \setminus \{p\})$. The oracle handles an update or query operation in $O(\tau)$ time. Our dynamic algorithm will invoke this oracle in a black-box manner, and its approximation ratio and update time will depend respectively on ρ and τ.

2.1. A Static Algorithm

The starting point of our work will be a well-known static algorithm due to (Mettu & Plaxton, 2003), with a slight twist that was discovered recently by (Czumaj et al., 2024).

Assuming that $f = 1$, this static algorithm defines a parameter r_p for every input point $p \in P$, as follows. For any $r \geq 0$, define $\mathtt{Vol}(p, r) = \sum_{x \in B_P(p,r)} (r - \mathtt{dist}(x, p))$. Note that $\mathtt{Vol}(p, r)$ is a monotonically increasing, continuous function of r, and that $\mathtt{Vol}(p, 0) = 0$. The parameter r_p is now defined to be the unique value of r at which $\mathtt{Vol}(p, r) = 1$. It is easy to verify that $r_p \in [0, 1]$.

The next two lemmas were derived by (Mettu & Plaxton, 2003). Lemma 2.1 states that $\sum_{p \in P} r_p$ gives us a $\Theta(1)$-approximation to $\mathtt{Opt}$. Lemma 2.2 essentially implies that

[2] For notational consistency, we set $\mathtt{dist}(p, S) = \infty$ if $S = \emptyset$.

[3] If $S \setminus \{p\} = \emptyset$, then we set $\mathrm{NN}(p, S) = \emptyset$.

if we find a parameter $\lambda \geq 1$ with $|B_P(p, \lambda^{-1})| = \Theta(1) \cdot \lambda$, then λ is a $\Theta(1)$-approximation to r_p.

Lemma 2.1. *We have:* $\texttt{Opt}/4 \leq \sum_{p \in P} r_p \leq 6 \cdot \texttt{Opt}$.

Lemma 2.2. $r_p \in \left[\frac{1}{|B_P(p,r_p)|}, \frac{2}{|B_P(p,r_p/2)|}\right]$ *for all* $p \in P$.

We next present a static algorithm for FACILITY LOCATION, due to (Czumaj et al., 2024).

Algorithm 1 (Czumaj et al., 2024) on input point-set P

1: For every $p \in P$, compute a value $\hat{r}_p$ that is $\Theta(1)$-approximation to r_p.
2: For each $p \in P$, sample a value $h(p) \in [0, 1]$ u.a.r.
3: For every $p \in P$, include p in a subset $F \subseteq P$ iff it is selected in either of the following (independent) steps.
   (S1) Select p if $h(p) \leq h(p')$ for all $p' \in B_P(p, \hat{r}_p)$.
   (S2) Select p with probability $\hat{r}_p$.
4: Return F as the set of open facilities.

Theorem 2.3. *(Czumaj et al., 2024) Algorithm 1 is a* $\Theta(1)$*-approximation algorithm for* FACILITY LOCATION.

3. A Key Building Block

In this section, we develop a data structure that will be crucially used by our dynamic facility location algorithm. Due to the space limit, we postpone proofs of technical claims in this section to Appendix A.

Theorem 3.1. *Given any parameter* $\lambda > 0$*, there is a data structure* $\mathcal{D}_\lambda(S)$ *for handling a dynamic point-set* S*. Specifically, it maintains a bit* $b_\lambda(p, S) \in \{0, 1\}$ *for each* $p \in S$*, and after every update (point insertion/deletion) in* S*, it reports all the points that change their* $b_\lambda(p, S)$ *values. The following properties are satisfied, for every* $p \in S$*.*

(i) If $\texttt{dist}(p, S \setminus \{p\}) \leq \lambda$*, then* $b_\lambda(p, S) = 1$*.*
(ii) If $\texttt{dist}(p, S \setminus \{p\}) > 2\rho\lambda$*, then* $b_\lambda(p, S) = 0$*.*

Each update in S *is handled in amortized* $\tilde{O}(\tau)$ *time.*[4]

In essence, the bit $b_\lambda(p, S)$ indicates whether or not the point $p \in S$ has a neighbor in S within distance λ (subject to a multiplicative slack of 2ρ). We refer to $\mathcal{D}_\lambda(S)$ as an approximate "nearest neighbor indicator data structure".

The corollary below follows from Theorem 3.1. It allows us to maintain, for each point $p \in S$, whether there are sufficiently many other points "close" to it. We outline the main ideas behind the proofs of Theorem 3.1 and Corollary 3.2 in Section 3.1 and Section 3.2, respectively.

[4]Recall that ρ and τ respective denote the approximation ratio and update/query-time of the nearest neighbor oracle. The $\tilde{O}(.)$ notation hides polylogarithmic (in the the input-size) factors.

Corollary 3.2. *For all* $\lambda > 0, \kappa > 1$*, there is a data structure* $\mathcal{D}^\star_{\lambda,\kappa}(S)$ *for handling a dynamic point-set* S*. It maintains a bit* $b^\star_{\lambda,\kappa}(p, S) \in \{0, 1\}$ *for each* $p \in S$*, and after every update (point insertion/deletion) in* S*, it reports all the points that change their* $b^\star_{\lambda,\kappa}(p, S)$ *values. At every time-step, the following properties hold whp, for all* $p \in S$*.*

(i) If $|B_S(p, \lambda)| \geq 5\kappa$*, then* $b^\star_{\lambda,\kappa}(p, S) = 1$*.*
(ii) If $|B_S(p, 2\rho\lambda)| < \kappa$*, then* $b^\star_{\lambda,\kappa}(p, S) = 0$*.*

Whp, the algorithm has an amortized update time of $\tilde{O}(\tau)$*.*

3.1. Proof of Theorem 3.1

Throughout the sequence of updates, we maintain a partition of the point-set S into three subsets: $R \subseteq S$, $C \subseteq S \setminus R$, and $A = S \setminus (C \cup R)$. We refer to the points in R, C and A as "remote-points", "cluster-points" and "alive-points", respectively. The set C is further partitioned into subsets that we refer to as "clusters". These partitions can change over time (for example, a given point might move from A to R, due to an update in S). Each alive-point $p \in A$ maintains a "witness" $w(p) \in C$ (this necessarily implies that if $A \neq \emptyset$, then $C \neq \emptyset$). For each cluster-point $q \in C$, we let $w^{-1}(q) := \{p \in A : w(p) = q\}$ denote the set of alive-points for which q serves as a witness. We say that a point $q \in C$ is "responsible" for every point $p \in w^{-1}(q)$. We always satisfy the following two invariants.

Invariant 3.3. Every cluster-point is responsible for at most one alive-point, i.e., $|w^{-1}(q)| \leq 1$ for all $q \in C$. Furthermore, every cluster in C contains at least two points.

Invariant 3.4. Every point $p \in R$ has $\texttt{dist}(p, S \setminus \{p\}) > \lambda$. For every pair of points $p, q \in C$ belonging to the *same* cluster, we have $\texttt{dist}(p, q) \leq 2\rho\lambda$. Finally, for every point $p \in A$, we have $\texttt{dist}(p, w(p)) \leq \rho\lambda$.

Thus, for an alive-point $p \in A$, the existence of a witness guarantees that $\texttt{dist}(p, S \setminus \{p\}) \leq d(p, w(p)) \leq 2\rho\lambda$. Furthermore, for a cluster-point $p \in C$, the cluster it belongs to contains at least one other point $q \neq p$, and this guarantees that $\texttt{dist}(p, S \setminus \{p\}) \leq \texttt{dist}(p, q) \leq 2\rho\lambda$. In contrast, for a remote point $p \in R$, we have $\texttt{dist}(p, S \setminus \{p\}) > \lambda$. This implies that the set R (resp. $C \cup A$) corresponds to the set of points $p \in S$ with $b_\lambda(p, S) = 0$ (resp. $b_\lambda(p, S) = 1$). Since we explicitly maintain the partition of S into A, C and R, we are able to report the set of points that move in and out of R (i.e., change their $b_\lambda(p, S)$ values) after an update. Thus, Theorem 3.1 follows from Lemmas 3.5 and 3.6.

We use two auxiliary data structures, for nearest-neighbor oracles in R, and in $C \cup A$.

Well-behaved Update Sequence We say that a sequence of updates in S is "well-behaved" iff it never deletes a cluster-point $p \in C$ with $w^{-1}(p) \neq \emptyset$. W.l.o.g., we can assume that our algorithm deals with a well-behaved update

sequence, as follows. Whenever the algorithm is asked to delete a point $p \in C$ with $w^{-1}(p) \neq \emptyset$, we identify the unique point $q \in A$ such that $w(q) = p$, and then do the following. We first delete q, then delete p, and finally insert q back. We refer to the first and the last of these three updates (involving q) as "pseduo-updates", while the middle one (involving p) being a "real-update". It is easy to verify that none of the two invariants get violated due to first pseudo-update (which deletes q), and hence other than deleting q, nothing else changes due to the first pseudo-update. This implies that when the real-update arrives, the node $p \in C$ has $w^{-1}(p) = \emptyset$, and hence the resulting "pseduo-update sequence" is well-behaved. Using this technique, we can convert any real update-sequence of length σ into a well-behaved, and equivalent, pseudo-update sequence of length at most 3σ, and feed this pseudo-update sequence into our algorithm. Accordingly, from now on we will assume that our algorithm deals with only a well-behaved update sequence.

Insertion of a Point p in S We handle this by calling the subroutine INSERT(p), as described in Algorithm 2. Because of this insertion, some remote-points might now violate Invariant 3.4, if the point p is too close to them. This is addressed in lines 1–6 of Algorithm 2. If there were no such remote-points close to p, then we implement lines 7–17. This requires us to first make a call to the oracle NN$(p, C \cup A)$, and then update the structure (A, C, R) in a natural manner, based on the outcome of the call. Note that if we create a new cluster $\{p, q\}$ in line 11, then it satisfies Invariant 3.4 for cluster-points, because $\texttt{dist}(p, q) \leq \texttt{dist}(p, x) + \texttt{dist}(x, q) \leq \rho\lambda + \rho\lambda = 2\rho\lambda$. Similarly, if we create a new cluster $\{p, x\}$ in line 10, then it also satisfies Invariant 3.4, because $\texttt{dist}(p, x) \leq \rho\lambda$.

Deletion of a Point p from S If $p \in A \cup R$, then there is nothing else to be done as both the invariants continue to remain satisfied. The same is true if $p \in C$ and belongs to a cluster containing at least two other points (recall that since the sequence is well-behaved, if $p \in C$ then $w^{-1}(p) = \emptyset$). Thus, the only interesting case occurs when p and belongs to a cluster $C_0 = \{p, q\} \subseteq C$, where q is the only other node belonging to the same cluster as p. In this case, we first remove q from the set C (thereby destroying the cluster C_0), and then call the subroutine INSERT(p) (see Algorithm 2).

The next lemma follows from the preceding discussion.

Lemma 3.5. *The above algorithm always satisfies Invariant 3.3 and Invariant 3.4.*

Lemma 3.6. *The amortized update time of the above algorithm is $\tilde{O}(\tau)$.*

Proof. The time taken to handle an update (both insertion and deletion) is dominated by the time spent on calls to the INSERT(p) subroutine. To bound the latter, note that lines 7–17 of Algorithm 2 require at most one call to the nearest neighbor oracle, and hence they can be implemented in $\tilde{O}(\tau)$ time. Lines 1–8, on the other hand, takes $\tilde{O}(\tau \cdot (1 + |X|))$ time. Clearly, each point $p \in X$ moves into the set C in line 6. Say that a "critical event" occurs whenever a point moves "in or out" of the set C. Thus, the amortized update time of our algorithm is $\tilde{O}(\tau \cdot (1 + \gamma))$, where γ denotes the amortized number of critical events. To conclude the proof, we next show that $\gamma = \Theta(1)$.

Consider any sequence of σ updates (assuming $S = \emptyset$ in the beginning). Note that during this sequence, whenever a point p moves into C, it joins a cluster with at least one other point $q \neq p$ (see Invariant 3.3). Subsequently, the point p moves out of C only if either: (i) p gets deleted from S itself, or (ii) we end up in a scenario where p belongs to a cluster with exactly one other point q, and q gets deleted from S. Thus, in either case, we can "charge" this event (of p moving out of C) to a deletion in S. It follows that during the entire update sequence, the total number of times some point moves out of C is at most $O(\sigma)$. Hence, the total number of critical events is also at most $O(\sigma)$. Accordingly, the amortized number of critical events is $\gamma = \Theta(1)$. □

Algorithm 2 INSERT(p)

1: $X \leftarrow \emptyset$.
2: **while** $R \neq \emptyset$ and $\texttt{dist}(p, \text{NN}(p, R)) \leq \rho\lambda$ **do**
3: $X \leftarrow X \cup \{\text{NN}(p, R)\}$, and $R \leftarrow R \setminus \{\text{NN}(p, R)\}$.
4: **end while**
5: **if** $X \neq \emptyset$ **then**
6: Create a new cluster $X \cup \{p\}$, and add all these points to C (after removing all the points in X from R).
7: **else**
8: **if** $C \cup A \neq \emptyset \bigwedge \texttt{dist}(p, \text{NN}(p, C \cup A)) \leq \rho\lambda$ **then**
9: Let $x \leftarrow \text{NN}(p, C \cup A)$.
10: **if** $x \in C$ **then**
11: If $w^{-1}(x) = \emptyset$, then add p to A, and set $w(p) \leftarrow x$. Otherwise, let $w^{-1}(x) = \{q\}$, where $q \in A$. Then, move q from A to C and add p to C, by creating a new cluster $\{p, q\}$.
12: **else**
13: $x \in A$. In this case, move x from A to C and add p to C, by creating a new cluster $\{p, x\}$.
14: **end if**
15: **else**
16: Add p to the set R.
17: **end if**
18: **end if**

3.2. Proof of Corollary 3.2

Let n be an upper bound on the number of points in S, throughout the sequence of updates. Let $c > 0$ be a sufficiently large constant, and let $T := \lceil c\kappa \log n \rceil$. For each

$i \in [1, T]$, we maintain a subset $S_i \subseteq S$, constructed as follows: Each point $p \in S$ is sampled in S_i independently with probability $1/\kappa$. Invoking Theorem 3.1, we also maintain the data structure $\mathcal{D}_\lambda(S_i)$, for all $i \in [1, T]$.

For all points $p \in S$ and indices $i \in [1, T]$, let $X_{p,i} \in \{0, 1\}$ be an indicator random variable that is set to 1 iff $p \in S_i$ and the data structure $\mathcal{D}_\lambda(S_i)$ sets $b_\lambda(p, S_i) = 1$. Finally, define $X_p := \sum_{i \in [1,T]} X_{p,i}$ for all $p \in P$.

We maintain the following output for each point $p \in P$: If $X_p \geq \frac{c}{2} \log n$, then $b^\star_{\lambda,\kappa}(p, S) = 1$, else $b^\star_{\lambda,\kappa}(p, S) = 0$.

Analysis For any point $p \in S$, define $\mathcal{S}_p := \{S_i : i \in [1, T], p \in S_i\}$ to be the collection of sampled subsets where p "survives". Since p is sampled with probability $1/\kappa$ in each S_i and since $T = \Theta(\kappa \log n)$, it is easy to verify that whp $|\mathcal{S}_p| = O(\log n)$. Thus, whp insertion/deletion of a point in S leads to $\tilde{O}(1)$ updates to the data structures $\{\mathcal{D}_\lambda(S)\}_i$ maintained by our algorithm. Since each of these data structures $\{\mathcal{D}_\lambda(S_i)\}_i$ has an amortized update time of $\tilde{O}(\tau)$, with some extra effort and book-keeping we can ensure that the amortized update time of our algorithm is also $\tilde{O}(\tau)$ whp. It now remains to show that the output maintained by our algorithm satisfies properties (i) and (ii) in the statement of Corollary 3.2. This follows from Claim 3.9 below (which, in turn, follows from Claim 3.7 and Claim 3.8).

Claim 3.7. *Consider any point $p \in S$ with $|B_S(p, \lambda)| \geq 5\kappa$. Then, we have $\mathbf{E}[X_{p,i}] \geq (1 - e^{-4})/\kappa$ for all $i \in [1, T]$.*

Claim 3.8. *Consider any $p \in S$ with $|B_S(p, 2\rho\lambda)| < \kappa$. Then, we have $\mathbf{E}[X_{p,i}] \leq e^{-2}/\kappa$ for all $i \in [1, T]$.*

Claim 3.9. *The following hold for every point $p \in S$.*

(i) If $|B_S(p, \lambda)| \geq 5\kappa$, then whp $b^\star_{\lambda,\kappa}(p, S) = 1$.
(ii) If $|B_S(p, 2\rho\lambda)| < \kappa$, then whp $b^\star_{\lambda,\kappa}(p, S) = 0$.

4. Our Dynamic Algorithm

Let $n \geq 1$ and $\delta \in (0, 1]$ respectively denote an upper bound on the size of the input point-set P and a lower bound on the minimum distance between any two distinct points in P, at all times. We assume that n and δ are known in advance, and that $1/\delta \in [1, O(\text{poly}(n))]$. We will maintain the output of a (slight) variant of Algorithm 1, under point insertions/deletions in P. In Section 4.1, we show how to maintain a $\Theta(\rho)$-approximate estimate $\hat{r}_p$ for each point $p \in P$. By Lemma 2.1, this already gives us a dynamic $\Theta(\rho)$-approximation algorithm for the *value* of the optimal facility location objective. In Section 4.2, we show how to explicitly maintain an $\Theta(\rho)$-approximate *solution* to our problem, which is defined by the set of open facilities. Finally, in Section 4.3, we explain how to ensure that the recourse of our algorithm remains at most $O(1)$, leading to our main result (summarized in Theorem 4.10).

4.1. Maintaining the $\hat{r}_p$ Value of Each Point $p \in P$

Below, we summarize the main result in this section. Due to the space limit, we postpone proofs of technical claims in this section to Appendix B.

Theorem 4.1. *Consider an input point-set P undergoing a sequence of updates (point insertions/deletions). There is a dynamic algorithm that explicitly maintains an estimate $\hat{r}_p \in [0, 1]$ for all $p \in P$. After each update in P, the algorithm also reports the set of points that change their $\{\hat{r}_p\}$ values. Whp, the algorithm has $\tilde{O}(\tau)$ amortized update time, and it ensures that $r_p \leq \hat{r}_p \leq \Theta(\rho) \cdot r_p$ for all $p \in P$.*[5]

We devote the rest of Section 4.1 to outlining the main ideas behind the proof of Theorem 4.1. Let $I := \lfloor \log_2(2\rho/\delta) \rfloor$, and for each $i \in [0, I]$, define the parameter $\lambda_i := \frac{\delta}{4\rho} \cdot 2^i$.

The Algorithm For each $i \in [0, I]$, we maintain the data structure $\mathcal{D}^\star_{\lambda_i, \lambda_i^{-1}}(P)$ by invoking Corollary 3.2. To ease notation, henceforth we use the symbol $b^\star_i(p)$ as a shorthand for $b^\star_{\lambda_i, \lambda_i^{-1}}(p, P)$ (see the statement of Corollary 3.2). For each point $p \in P$, we maintain the index $i_p := \max\{i \in [0, I] : b^\star_i(p) = 0\}$, and the value $\hat{r}_p := \min(1, 6\rho\lambda_{i_p})$. The index i_p is well-defined, due to Claim 4.2.

Claim 4.2. *Consider any $p \in P$. We have $b^\star_0(p) = 0$ whp.*

Analysis In Claim 4.3 below, we prove that these $\{\hat{r}_p\}$ values satisfy the property required by Theorem 4.1. Next, observe that an update in P leads to one update in each of the data structures $\{\mathcal{D}^\star_{\lambda_i, \lambda_i^{-1}}(P)\}_{i \in [0,T]}$. Further, we have $T = O(\log(\rho/\delta)) = \tilde{O}(1)$, and each of the data structures $\{\mathcal{D}^\star_{\lambda_i, \lambda_i^{-1}}(P)\}_i$ has an update time of $\tilde{O}(\tau)$ (see Corollary 3.2). Thus, with some additional book-keeping, we can ensure that the overall update time of our dynamic algorithm is $\tilde{O}(1) \cdot \tilde{O}(\tau) = \tilde{O}(\tau)$. This leads us to Theorem 4.1.

Claim 4.3. *For all $p \in P$, we have: $r_p \leq \hat{r}_p \leq 30\rho \cdot r_p$.*

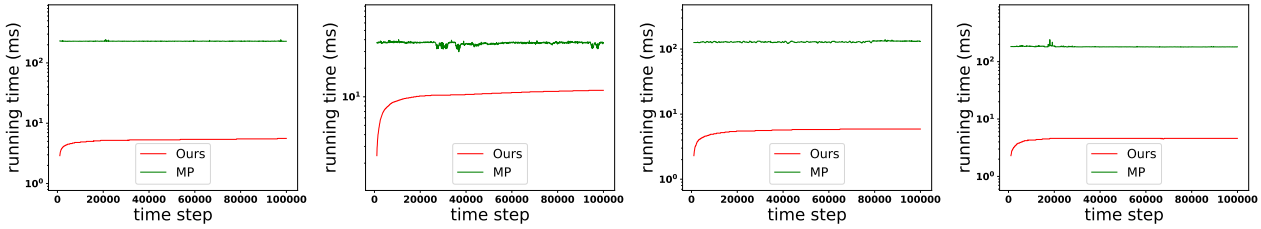
4.2. Maintaining the Set F of Open Facilities

Our main result is stated in the theorem below. We devote the rest of Section 4.2 towards proving Theorem 4.4. Due to the space limit, we postpone proofs of technical claims in this section to Appendix D.

Theorem 4.4. *Consider an input point-set P undergoing a sequence of updates (point insertions/deletions). There is a dynamic algorithm that explicitly maintains a subset $F \subseteq P$ of open facilities. Whp, the algorithm has an amortized update time of $\tilde{O}(\tau)$, and F is a $\Theta(\rho)$-approximate solution to the* FACILITY LOCATION *instance defined by P.*[6]

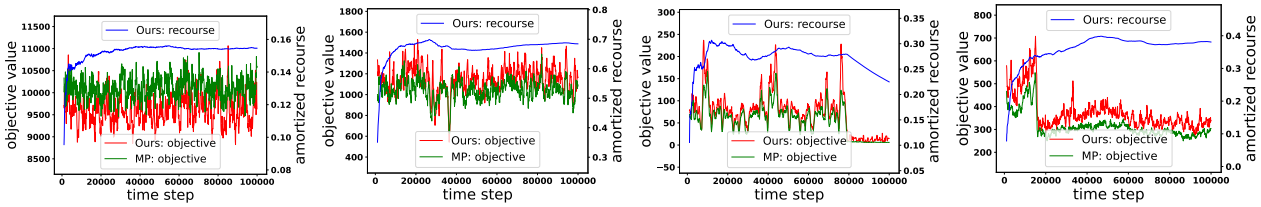
[5] Recall that ρ and τ respectively denote the approximation ratio and update/query-time of the nearest-neighbor oracle (Section 2).

[6] Recall that ρ and τ respectively denote the approximation ratio



Census1990 Twitter KDD-Cup Covertype

Figure 1: The running time comparison of our dynamic algorithm and MP algorithm (Mettu & Plaxton, 2003). Since MP algorithm is static, we re-run it from scratch to handle updates. However, since re-running every step takes prohibitively long time, we choose to re-run it only every 100 time steps (and report the running time of that specific invocation). To make a fair comparison, we report the average running time of our algorithm during these 100 time steps.



Census1990 Twitter KDD-Cup Covertype

Figure 2: The objective value and amortized recourse evaluated on the four datasets. The objective of the MP algorithm (Mettu & Plaxton, 2003), re-run after each update, is also reported for comparison.

We first propose a (slightly) modified version of Algorithm 1. For each point $p \in P$, let $h^\star(p)$ be obtained by rounding the value of $h(p)$ in powers of 2 (see line 2 of Algorithm 1): If $2^{-i} \le h(p) < 2^{-i+1}$ for an integer $i \ge 0$, then $h^\star(p) = 2^{-i}$. Further, we define $P_i := \{p \in P : h^\star(p) = 2^{-i}\}$. In the "new algorithm", we change step (S1) of Algorithm 1 to: "Select p if $h^\star(p) \le h^\star(p')$ for all $p' \in B_P(p, \hat{r}_p)$". Everything else remains the same as before. Thus, the new algorithm can be thought of a simple discretization of Algorithm 1, where we replace $h(p)$ by $h^\star(p)$ for all $p \in P$.

Lemma 4.5. *The new algorithm described above returns a* $\Theta(1)$*-approximate solution to* FACILITY LOCATION.

Proof. The proof can be found in Appendix C. □

We proceed towards highlighting a few key properties of this new (static) algorithm. Say that a point $p \in P$ is in "layer i" iff $p \in P_i$. Let $P_{\ge i} := \bigcup_{j \ge i} P_j$. The claim below guarantees that there are $O(\log n)$ non-empty layers, whp.

Claim 4.6. *Fix a sufficiently large constant* $\alpha > 0$ *and let* $L_\alpha := \lceil \alpha \log n \rceil$. *Then, whp, we have:* $P_{\ge L_\alpha} = \emptyset$.

Next, we prove that for all $p \in P$, whp the non-empty layer with the largest index (in $B_P(p, \hat{r}_p)$) has $O(\log n)$ points.

and update/query-time of the nearest-neighbor oracle (Section 2).

Claim 4.7. *Consider any* $p \in P$, *and let* $i_p := \max\{i : P_i \cap B_P(p, \hat{r}_p) \ne \emptyset\}$. *Then, whp:* $|P_{i_p} \cap B_P(p, \hat{r}_p)| \le 4\beta \log n$, *where* $\beta > 0$ *is a sufficiently large constant.*

Fix a parameter $L_\beta := \lceil 16\beta^2 \log^2 n \rceil$. For each point $p \in P$, choose an index $j_p \in [1, L_\beta]$ independently and u.a.r. For each $j \in [1, L_\beta]$, define the subset of points $Q_j := \{p \in P : j_p = j\}$. In addition, for each $i \in [1, L_\alpha]$ and each $j \in [1, L_\beta]$, define the subset of points $S_{i,j} := P_{\ge i} \cup Q_j$. Our dynamic algorithm will maintain these subsets, and will crucially rely upon the following property.

Claim 4.8. *Condition on the (high probability) events described in the statements of Claim 4.6 and Claim 4.7. Consider any point* $p \in P$. *Then, with constant probability, we have* $j_q \ne j_{q'}$ *for all* $q, q' \in P_{i_p} \cap B_P(p, \hat{r}_p)$ *with* $q \ne q'$.

Our Dynamic Algorithm Recall steps (S1) and (S2) in Algorithm 1. For each point $p \in P$, let $Z_p^{(2)} \in \{0, 1\}$ be an indicator random variable that is set to 1 iff a facility is opened at point p because of step (S2). Similarly, let $Z_p^{(1)} \in \{0, 1\}$ be an indicator random variable that is set to 1 iff a facility is opened at point p because of the modified step (S1) in the new algorithm (i.e., iff $h^\star(p) \le h^\star(p')$ for all $p' \in B_P(p, \hat{r}_p)$). In the dynamic setting, we will maintain the output of the new algorithm. Towards this end, it suffices to maintain the $\left\{Z_p^{(1)}, Z_p^{(2)}\right\}_{p \in P}$ values.

To begin with, it is relatively straightforward to keep track of the $\left\{Z_p^{(2)}\right\}_{p\in P}$ values: Whenever a point p is inserted into P, we sample a number $\gamma_p \in [0,1]$ independently and u.a.r. Subsequently, throughout the lifespan of point p (i.e., until it gets deleted), we set $Z_p^{(2)} = 1$ iff $\gamma_p \leq \hat{r}_p$. This ensures that $\Pr\left[Z_p^{(2)} = 1\right] = \hat{r}_p$. Invoking Thoerem 4.1, this can be implemented in $\tilde{O}(\tau)$ update time.

To explain how we maintain the $\left\{Z_p^{(1)}\right\}_{p\in P}$ values, henceforth we condition on the (high-probability) events described in the statements of Claim 4.6 and Claim 4.7. We also make the following simplifying assumptions: We have access to an "exact" nearest-neighbor oracle (i.e., $\rho = 1$), and a data structure for Theorem 3.1 that does *not* need any slack. In $\tilde{O}(\tau)$ update time, this idealized data structure maintains a bit $b_\lambda(p, S) \in \{0,1\}$ that is set to 1 iff $\texttt{dist}(p, S \setminus \{p\}) \leq \lambda$. Under these assumptions, we now outline how to get a dynamic algorithm for Theorem 4.4, with update time $\tilde{O}(\tau)$ and approximation ratio $\Theta(1)$. It is relatively straightforward to extend the ensuing discussion to the general case, where the simplifying assumptions do not hold, leading to a formal proof of Theorem 4.4.

Fix any point $p \in P$, and let $p \in P_i$, where $0 \leq i < L_\alpha$. Note that we should set $Z_p^{(1)} = 0$ iff $P_{\geq i+1} \cap B_P(p, \hat{r}_p) \neq \emptyset$. In our dynamic algorithm, we instead set $Z_p^{(1)} = 0$ iff the nearest neighbor of p in S_{i+1,j_p} is at most a distance $\hat{r}_p$ away (i.e., iff $\texttt{dist}(p, S_{i+1,j_p} \setminus \{p\}) \leq \hat{r}_p$). We can keep track of whether this latter condition is being satisfied or not, by invoking Theorem 3.1; specifically, by maintaining a nearest neighbor indicator (NNI) data structure $\mathcal{D}_\lambda(S_{i,j})$ for all $i \in [1, L_\alpha], j \in [1, L_\beta]$, where λ is an approximate "guess" (in powers of 2) on the value of $\hat{r}_p$. Note that there are only $\tilde{O}(1)$ such guesses, assuming the "aspect ratio" of the metric is polynomially bounded. Since $L_\alpha, L_\beta = \tilde{O}(1)$, we need to maintain only $\tilde{O}(1)$ such NNI data structures. Furthermore, it is easy to verify that each update in P leads to $\tilde{O}(1)$ updates in these NNI data structures. By Theorem 3.1, this gives us an amortized update time of $\tilde{O}(\tau)$.

We now explain why, in the scheme described above, the value of $Z_p^{(1)}$ is set correctly with constant probability. Towards this end, condition on the event described in Claim 4.8 (which occurs with constant probability). Now, there are two cases to consider. (i) $P_{\geq i+1} \cap B_P(p, \hat{r}_p) \neq \emptyset$. In this case, we clearly have $\texttt{dist}(p, S_{i+1,j_p} \setminus \{p\}) \leq \hat{r}_p$, because $S_{i+1,j_p} \supseteq P_{\geq i+1}$, and hence our dynamic algorithm maintains the correct value of $Z_p^{(1)}$. (ii) $P_{\geq i+1} \cap B_P(p, \hat{r}_p) = \emptyset$. In this case, we have $i = i_p$, and hence Claim 4.8 implies that $Q_{j_p} \cap B_P(p, \hat{r}_p) = \{p\}$. As $S_{i+1,j_p} = P_{\geq i+1} \cup Q_{j_p}$, it follows that $\texttt{dist}(p, S_{i+1,j_p} \setminus \{p\}) > \hat{r}_p$, and hence our dynamic algorithm sets the correct value of $Z_p^{(1)}$.

Table 1: Specifications of datasets and parameters

dataset	size	dimension	opening cost
Twitter	21040936	2	1
Census1990	2458285	68	10
Covertype	581012	52	1
KDD-Cup	4898431	28	0.5

To summarize, our dynamic algorithm maintains the correct value $Z_p^{(1)} \in \{0,1\}$ with constant probability. We can now boost this success probability using standard techniques, by keeping $O(\log n)$ independent copies of the above data structure. This ensures that our dynamic algorithm works correctly whp, at the cost of increasing its update time by only a $O(\log n)$ factor.

4.3. Consistent Facility Location: A General Reduction

Finally, we show how to ensure that our algorithm achieves constant (amortized) recourse. Towards this end, we make use of the following general reduction which was implicitly shown in (Cohen-Addad et al., 2019). It turns a generic dynamic algorithm for facility location (without any recourse bound) to a "consistent" one (i.e., with $O(1)$ recourse), with negligible additional overhead.

Lemma 4.9 (Cohen-Addad et al. (2019))**.** *Let there be a dynamic algorithm $\mathcal{A}$ that maintains an α-approximate solution for facility location, with amortized update time T. Then there is another dynamic facility location algorithm that maintains a $O(\rho\alpha)$-approximate solution, with $O(1)$ amortized recourse and $\tilde{O}(T + \tau)$ amortized update time.*[7]

Combining Lemma 4.9 with Theorem 4.4, we obtain the main result of this paper, which is summarized below.

Theorem 4.10. *There is a randomized dynamic algorithm for facility location that has $O(\rho^2)$-approximation ratio, $O(1)$ amortized recourse, and $\tilde{O}(\tau)$ amortized update time.*

5. Experimental Evaluation

We implement our algorithm in C++ language and run it on real datasets. We aim to validate that our algorithm consistently generates accurate solutions, and that the recourse of the open facility set is very small.

Datasets and Parameters Our experiment is done on Twitter (Chan et al., 2018b), Census1990 (Meek et al.), Covertype (Blackard, 1998) and KDD-Cup (Stolfo et al., 1999) datasets. These datasets have been used in various previous works for evaluating clustering and facility location algorithms (Chan et al., 2018a; Cohen-Addad et al.,

[7] Recall that ρ and τ respectively denote the approximation ratio and update/query time of the nearest neighbor oracle (Section 2).

2019; Bhattachcharya et al., 2022). We take the numerical attributes to generate the input points in $\mathbb{R}^d$, and normalize each attribute/dimension to prevent a single attribute from dominating the vector. We consider a sliding window of size $\ell = 1000$, and our update operations on the datasets are generated by this sliding window. The opening cost is set such that a moderate number of facilities would be open in a (near-)optimal solution. We summarize the specifications and the opening cost for the four datasets in Table 1.

Implementation Details Our implementation has an important difference to the one listed in the proof, is that we decide *not* to apply Lemma 4.9 which is a generic step to reduce the amortized recourse to $O(1)$. In fact, even without this step, our algorithm can still achieve $O(\text{poly} \log n)$ amortized recourse, and it turns out to still achieve good recourse in the experiments. We make use of standard locality-sensitive hashing (LSH) techniques (see e.g. Har-Peled et al. (2012)) to implement the nearest neighbor oracle. In our experiments, we use 15 random hash functions and we find this setup already produces decent accuracy when combining with our algorithm. All the experiments are conducted on an Apple computer with M1 Pro CPU and 16GB memory.

Experiment Setup We run our algorithm on the four datasets, and we report the objective value of the approximate solution, the recourse and running time after processing each query in Figure 2. The reported recourse is amortized, i.e., it is the total recourse divided by the number of time steps, which may be viewed as a running average. We also report the objective and running time achieved by the Mettu-Plaxton (MP) algorithm (Mettu & Plaxton, 2003). This is a static algorithm and is re-run after each update. Due to the fact that our dynamic algorithm is designed based on a modified version of MP, and that this algorithm is shown to be 3-approximation (which achieves the same ratio as another well-known algorithm by Jain & Vazirani (2001)), it serves as a baseline for evaluating our algorithm.

Experiment Results We can see from Figure 1 that our algorithm typically runs roughly two magnitudes faster than MP. This justifies the efficiency of our dynamic algorithm. Moreover, this speedup is achieved without sacrificing the accuracy. As depicted in Figure 2, on all datasets, our algorithm maintains a solution whose objective value is very similar to that of MP algorithm. This is much better than the worst-case theoretical bound in which our ratio is worse than that of MP by a (large) constant, due to the additional error introduced by the approximate nearest-neighbor oracle and that in the modified MP (Theorem 2.3). In addition, the solution is maintained with very small amortized recourse, which is even less than 1, and this is again much better than the worst-case bound. The fact that the amortized recourse is less than 1 may be caused by the nature of the sliding window, since the removal of a single point may not be enough to force the change of a facility.

Acknowledgments

The authors thank the anonymous referees for valuable comments and suggestions. Research of SJ was supported in part by a national key R&D program of China No. 2021YFA1000900 and a startup fund from Peking University.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Badoiu, M., Czumaj, A., Indyk, P., and Sohler, C. Facility location in sublinear time. In *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pp. 866–877. Springer, 2005.

Bateni, M., Esfandiari, H., Fichtenberger, H., Henzinger, M., Jayaram, R., Mirrokni, V., and Wiese, A. Optimal fully dynamic *k*-center clustering for adaptive and oblivious adversaries. In *SODA*, pp. 2677–2727. SIAM, 2023.

Bhattachcharya, S., Lattanzi, S., and Parotsidis, N. Efficient and stable fully dynamic facility location. In *NeurIPS*, 2022.

Bhattachcharya, S., Costa, M., Lattanzi, S., and Parotsidis, N. Fully dynamic k-clustering in $\tilde{O}$(k) update time. In *NeurIPS*, 2023.

Biabani, L., Hennes, A., Monemizadeh, M., and Schmidt, M. Faster query times for fully dynamic k-center clustering with outliers. In *NeurIPS*, 2023.

Blackard, J. Covertype. UCI Machine Learning Repository, 1998. DOI: https://doi.org/10.24432/C50K5N.

Broder, A. Z. On the resemblance and containment of documents. In *SEQUENCES*, pp. 21–29. IEEE, 1997.

Chan, T. H., Guerquin, A., and Sozio, M. Fully dynamic *k*-center clustering. In *WWW*, pp. 579–587. ACM, 2018a.

Chan, T. H., Lattanzi, S., Sozio, M., and Wang, B. Fully dynamic k-center clustering with outliers. *Algorithmica*, 86(1):171–193, 2024.

Chan, T.-H. H., Guerquin, A., and Sozio, M. Twitter data set, 2018b. `https://github.com/fe6Bc5R4JvLkFkSeExHM/k-center`.

Charikar, M., Chekuri, C., Feder, T., and Motwani, R. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004.

Cohen-Addad, V., Hjuler, N., Parotsidis, N., Saulpic, D., and Schwiegelshohn, C. Fully dynamic consistent facility location. In *NeurIPS*, pp. 3250–3260, 2019.

Cruciani, E., Forster, S., Goranci, G., Nazari, Y., and Skarlatos, A. Dynamic algorithms for k-center on graphs. In *SODA*, pp. 3441–3462. SIAM, 2024.

Cygan, M., Czumaj, A., Mucha, M., and Sankowski, P. Online facility location with deletions. In *ESA*, volume 112 of *LIPIcs*, pp. 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

Czumaj, A., Lammersen, C., Monemizadeh, M., and Sohler, C. $(1+\varepsilon)$-approximation for facility location in data streams. In *SODA*, pp. 1710–1728. SIAM, 2013.

Czumaj, A., Gao, G., Jiang, S. H., Krauthgamer, R., and Veselý, P. Fully scalable MPC algorithms for clustering in high dimension. In *ICALP*, 2024.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*, pp. 253–262. ACM, 2004.

Fotakis, D. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008.

Goel, A., Indyk, P., and Varadarajan, K. R. Reductions among high dimensional proximity problems. In *SODA*, pp. 769–778. ACM/SIAM, 2001.

Goranci, G., Henzinger, M., and Leniowski, D. A tree structure for dynamic facility location. In *ESA*, volume 112 of *LIPIcs*, pp. 39:1–39:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

Goranci, G., Henzinger, M., Leniowski, D., Schulz, C., and Svozil, A. Fully dynamic k-center clustering in low dimensional metrics. In *ALENEX*, pp. 143–153. SIAM, 2021.

Guo, X., Kulkarni, J., Li, S., and Xian, J. On the facility location problem in online and dynamic models. In *APPROX-RANDOM*, volume 176 of *LIPIcs*, pp. 42:1–42:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

Har-Peled, S., Indyk, P., and Motwani, R. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory Comput.*, 8(1):321–350, 2012.

Henzinger, M. and Kale, S. Fully-dynamic coresets. In *ESA*, volume 173 of *LIPIcs*, pp. 57:1–57:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

Indyk, P. Algorithms for dynamic geometric problems over data streams. In *STOC*, pp. 373–380. ACM, 2004.

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

Lacki, J., Haeupler, B., Grunau, C., Jayaram, R., and Rozhon, V. Fully dynamic consistent k-center clustering. In *SODA*, pp. 3463–3484. SIAM, 2024.

Lammersen, C. and Sohler, C. Facility location in dynamic geometric data streams. In *ESA*, volume 5193 of *Lecture Notes in Computer Science*, pp. 660–671. Springer, 2008.

Meek, C., Thiesson, B., and Heckerman, D. US Census Data (1990). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5VP42.

Mettu, R. R. and Plaxton, C. G. The online median problem. *SIAM J. Comput.*, 32(3):816–832, 2003.

Meyerson, A. Online facility location. In *FOCS*, pp. 426–431. IEEE Computer Society, 2001.

Stolfo, S., Fan, W., Lee, W., Prodromidis, A., and Chan, P. KDD Cup 1999 Data. UCI Machine Learning Repository, 1999. DOI: https://doi.org/10.24432/C51C7N.

A. Missing Proofs in Section 3

Claim 3.7. *Consider any point $p \in S$ with $|B_S(p,\lambda)| \geq 5\kappa$. Then, we have $\mathbf{E}[X_{p,i}] \geq (1-e^{-4})/\kappa$ for all $i \in [1,T]$.*

Proof. Fix any index $i \in [1,T]$, and let $\mathcal{E}_{p,i}$ be the event that at least one point $q \in B_S(p,\lambda) \setminus \{p\}$ is sampled in S_i. It follows that $\Pr[\mathcal{E}_{p,i}] = 1-(1-1/\kappa)^{|B_S(p,\lambda)|-1} \geq 1-(1-1/\kappa)^{4\kappa} \geq 1-e^{-4}$. Observe that if $p \in S_i$ and the event $\mathcal{E}_{p,i}$ occurs, then $X_{p,i}=1$. To see why this is true, consider any point $q \in (B_S(p,\lambda)\setminus\{p\}) \cap S_i$ (such a point must exist under the event $\mathcal{E}_{p,i}$). Then, we have $\texttt{dist}(p, S_i \setminus \{p\}) \leq \texttt{dist}(p,q) \leq \lambda$ and $p \in S_i$, and hence Theorem 3.1 implies that $b_\lambda(p,S_i) = 1 = X_{p,i}$.

To conclude, since the events $p \in S_i$ and $\mathcal{E}_{p,i}$ are independent, we get: $\mathbb{E}[X_{p,i}] = \Pr[X_{p,i}=1] = \Pr[p \in S_i] \cdot \Pr[\mathcal{E}_{p,i}] \geq (1-e^{-4})/\kappa$. □

Claim 3.8. *Consider any $p \in S$ with $|B_S(p,2\rho\lambda)| < \kappa$. Then, we have $\mathbf{E}[X_{p,i}] \leq e^{-2}/\kappa$ for all $i \in [1,T]$.*

Proof. Fix any index $i \in [1,T]$, and let $\mathcal{E}^\star_{p,i}$ be the event that no point $q \in B_S(p,2\rho\lambda)\setminus\{p\}$ is sampled in S_i. Thus, we have $\Pr[\mathcal{E}^\star_{p,i}] = (1-1/\kappa)^{|B_S(p,2\rho\lambda)|-1} > (1-1/\kappa)^{\kappa} \geq e^{-2}$. Observe that if $p \in S_i$ and the event $\mathcal{E}^\star_{p,i}$ occurs, then $X_{p,i}=0$. To see why this is true, note that under this event $\texttt{dist}(p, S_i\setminus\{p\}) > 2\rho\lambda$. Thus, Theorem 3.1 implies that $b_\lambda(p,S_i)=0=X_{p,i}$.

To conclude, since the events $p \in S_i$ and $\mathcal{E}^\star_{p,i}$ are independent, we get: $\mathbb{E}[X_{p,i}] = 1-\Pr[X_{p,i}=0] \leq 1-\Pr[(p \in S_i) \wedge \mathcal{E}^\star_{p,i}] = 1-\Pr[p\in S_i]\cdot\Pr[\mathcal{E}^\star_{p,i}] \leq e^{-2}/\kappa$. □

Claim 3.9. *The following hold for every point $p \in S$.*

(i) If $|B_S(p,\lambda)| \geq 5\kappa$, then whp $b^\star_{\lambda,\kappa}(p,S)=1$.
(ii) If $|B_S(p,2\rho\lambda)| < \kappa$, then whp $b^\star_{\lambda,\kappa}(p,S)=0$.

Proof. Throughout the proof, fix any point $p \in S$.

Part (i): We have $\mathbb{E}[X_p] = \sum_{i\in[1,T]} \mathbb{E}[X_{p,i}]$. Since $|B_S(p,\lambda)| \geq 5\kappa$, Claim 3.7 implies that $\mathbb{E}[X_p] \geq (1-e^{-4})\kappa^{-1}T = (1-e^{-4})\cdot c\log n \geq \frac{3c}{4}\log n$. Since $\{X_{p,i}\}_i$ are mutually independent $0/1$ random variable, $X_p = \sum_i X_{p,i}$, and c is a sufficiently large constant, applying a Chernoff bound we get: whp $X_p \geq \frac{c}{2}\log n$ (which is equivalent to claiming that $b^\star_{\lambda,\kappa}(p,S)=1$).

Part (ii): We have $\mathbb{E}[X_p] = \sum_{i\in[1,T]} \mathbb{E}[X_{p,i}]$. Since $|B_S(p,2\rho\lambda)| < \kappa$, Claim 3.8 implies that $\mathbb{E}[X_p] \leq e^{-2}\kappa^{-1}T = e^{-2}c\log n \leq \frac{c}{4}\log n$. Applying a Chernoff bound as in part (i) above, we get: whp $X_p < \frac{c}{2}\log n$ (which is equivalent to claiming that $b^\star_{\lambda,\kappa}(p,S)=0$). □

B. Missing Proofs in Section 4.1

Claim 4.2. *Consider any $p \in P$. We have $b^\star_0(p)=0$ whp.*

Proof. Note that $\lambda_0 = \delta/(4\rho)$. Since δ is a lower bound on the minimum distance between any two distinct points in P, we have $B_P(p,2\rho\lambda_0) = B_P(p,\delta/2) = \{p\}$. This implies that $|B_P(p,2\rho\lambda_0)| = 1 < 4\rho/\delta = \lambda_0^{-1}$. Thus, by Corollary 3.2, we get: $b^\star_0(p)=0$ whp. □

Claim 4.3. *For all $p \in P$, we have: $r_p \leq \hat{r}_p \leq 30\rho\cdot r_p$.*

We first prove the following Claim B.1 as it is used in the proof of Claim 4.3.

Claim B.1. *Consider any point $p \in P$ and any index $i \in [0,I]$. Then the following two properties hold whp.*

(i) If $b^\star_i(p)=1$, then $r_p \leq \min(1, 3\rho\lambda_i)$.
(ii) If $b^\star_i(p)=0$, then $r_p > \lambda_i/5$.

Proof. For part (i), let $b^\star_i(p)=1$. Then, by Corollary 3.2, we have $|B_P(p,2\rho\lambda_i)| \geq \lambda_i^{-1}$. Thus, we infer that $\texttt{Vol}(p,3\rho\lambda_i) := \sum_{x\in B_P(p,3\rho\lambda_i)}(3\rho\lambda_i - \texttt{dist}(x,p)) \geq \rho\lambda_i \cdot |B_P(p,2\rho\lambda_i)| \geq 1$, and hence $r_p \leq \min(1,3\rho\lambda_i)$.

For part (ii), let $b^\star_i(p) = 0$. Then, by Corollary 3.2, we have $|B_P(p, \lambda_i)| < 5\lambda_i^{-1}$. Thus, we infer that $\texttt{Vol}(p, \lambda_i/5) := \sum_{x \in B_P(p,\lambda_i/5)}(\lambda_i/5 - \texttt{dist}(x, p)) \leq |B_P(p, \lambda_i/5)| \cdot (\lambda_i/5) \leq |B_P(p, \lambda_i)| \cdot (\lambda_i/5) < 1$, and hence $r_p > \lambda_i/5$.

This concludes the proof of the claim. □

Proof of Claim 4.3. Fix any point $p \in P$, and consider two cases.

Case 1: $0 \leq i_p < I$. Thus, $b^\star_{i_p}(p) = 0$ and $b^\star_{i_p+1}(p) = 1$, and Claim B.1 implies that $\lambda_{i_p}/5 < r_p \leq \min(1, 3\rho\lambda_{i_p+1}) = \min(1, 6\rho\lambda_{i_p}) = \hat{r}_p$. Hence, we get: $r_p \leq \hat{r}_p \leq 30\rho \cdot r_p$.

Case 2: $i_p = I$. As before, Claim B.1 implies that $\lambda_{i_p}/5 < r_p$. We also have $r_p \leq 1$. This leads us to conclude that $\lambda_{i_p}/5 < r_p \leq 1 = \min(1, 6\rho\lambda_{i_p}) = \hat{r}_p$ (the second last inequality holds because $\lambda_{i_p} = \lambda_I \geq 1/4$ and $\rho \geq 1$). Thus, we get: $r_p \leq \hat{r}_p \leq 30\rho \cdot r_p$. □

C. Proof of Lemma 4.5

Conditioned on the $\{h(p)\}_{p\in P}$ values, the set of facilities opened by the new algorithm is a superset of the set of facilities opened by Algorithm 1. Since each point $p \in P$ gets connected to its nearest open facility, this implies that the total connection cost of the new algorithm is at most the total connection cost of Algorithm 1. The latter quantity, in turn, is at most $\Theta(1)$ times the optimal objective (see Theorem 2.3). Thus, it now remains to upper bound the total facility opening cost of the new algorithm.

This is achieved in Lemma C.1 below. Specifically, Lemma C.1 implies that the expected contribution of a point $p \in P$ towards the total facility opening cost is $O(\hat{r}_p)$. Hence, by linearity of expectation, the total expected facility opening cost is $\sum_{p\in P} O(\hat{r}_p) = \sum_{p\in P} O(r_p) = O(1) \cdot \texttt{Opt}$, where $\texttt{Opt}$ denotes the optimal objective. The first equality holds due to line 1 of Algorithm 1, whereas the last equality holds because of Lemma 2.1.

Lemma C.1. *In the new algorithm, for all $p \in P$ we have:* $\Pr[p \in F] = O(\hat{r}_p)$.

Proof. Fix any point $p \in P$. Clearly, the point p is opened as a facility due to step (S2) (see Algorithm 1) with probability $\hat{r}_p$. Thus, it remains to upper bound the probability that the point p is opened as a facility due to the modified step (S1) in the new algorithm. Specifically, let $\mathcal{E}^\star_p$ denote the event that $h^\star(p) \leq h^\star(p')$ for all $p' \in B_P(p, \hat{r}_p)$. We want to show that $\Pr[\mathcal{E}^\star_p] = O(\hat{r}_p)$.

Let $m_p := |B_P(p, \hat{r}_p)|$. For any integer $i \geq 0$, let $\mathcal{E}^\star_p(i)$ denote the event that (i) $p \in P_i$ and (ii) $p' \notin P_{\geq i+1}$ for all $p' \in B_P(p, \hat{r}_p) \setminus \{p\}$. Clearly, we have $\mathcal{E}^\star_p := \bigcup_{i\geq 0} \mathcal{E}^\star_p(i)$. We will first upper bound $\Pr[\mathcal{E}^\star_p(i)]$, for a fixed i. Observe that $\Pr[p \in P_i] = 2^{-i+1} - 2^{-i} = 2^{-i}$, and that $\Pr[q \in P_{\geq i+1}] = 2^{-i}$ for all $q \in B_P(p, \hat{r}_p)$. Thus, we infer that: $\Pr[\mathcal{E}^\star_p(i)] = 2^{-i} \cdot (1 - 2^{-i})^{m_p} \leq 2^{-i} \cdot \exp(-m_p/2^i) \leq 2^{-i-m_p/2^i}$. Now, via a union bound over all $i \geq 0$, we get:

$$\begin{aligned}\Pr\left[\mathcal{E}^\star_p\right] &\leq \sum_{i\geq 0} 2^{-i-m_p/2^i} \\ &= \sum_{i:2^i\leq m_p} 2^{-i-m_p/2^i} + \sum_{i:2^i>m_p} 2^{-i-m_p/2^i}. \end{aligned} \tag{1}$$

Let Λ_1 and Λ_2 respectively denote the first and the second term on the RHS of (1). To bound Λ_1, we change the summation variable to $j = \log m_p - i$, so that $j \in \{0, \ldots, \log m_p\}$, and get: $\Lambda_1 = \sum_{0\leq j\leq \log m_p} 2^{-\log m_p + j - m_p/2^{\log m_p - j}} = (1/m_p) \cdot \sum_{0\leq j\leq \log m_p} 2^{j-2^j} = O(1/m_p)$. Next, to bound Λ_2, we observe that: $\Lambda_2 = \sum_{i:2^i>m_p} 2^{-i-m_p/2^i} \leq \sum_{i:2^i>m_p} 2^{-i} = O(1/m_p)$. To conclude, we infer that:

$$\Pr[\mathcal{E}^\star_p] \leq \Lambda_1 + \Lambda_2 = O(1/m_p). \tag{2}$$

Finally, since $r_p \leq \hat{r}_p$ (see Theorem 4.1), we have: $1 \leq \texttt{Vol}(p, \hat{r}_p) = \sum_{q\in B_P(p,\hat{r}_p)}(\hat{r}_p - \texttt{dist}(p, q)) \leq \hat{r}_p \cdot |B_P(p, \hat{r}_p)| = \hat{r}_p \cdot m_p$, and hence: $1/m_p \leq \hat{r}_p$. This last inequality, in conjunction with (2), gives us: $\Pr[\mathcal{E}^\star_p] = O(\hat{r}_p)$, which concludes the proof of Lemma C.1. □

D. Other Missing Proofs in Section 4.2

Claim 4.6. *Fix a sufficiently large constant $\alpha > 0$ and let $L_\alpha := \lceil \alpha \log n \rceil$. Then, whp, we have: $P_{\geq L_\alpha} = \emptyset$.*

Proof. Consider any point $p \in P$. Note that $\Pr[p \in P_{\geq L_\alpha}] \leq \Pr[h(p) < 2^{-L_\alpha+1}] = 2^{-L_\alpha+1} \leq 2/n^\alpha$. Since there are at most n points in P, the claim now follows from a union bound over all $p \in P$. □

Claim 4.7. *Consider any $p \in P$, and let $i_p := \max\{i : P_i \cap B_P(p, \hat{r}_p) \neq \emptyset\}$. Then, whp: $|P_{i_p} \cap B_P(p, \hat{r}_p)| \leq 4\beta \log n$, where $\beta > 0$ is a sufficiently large constant.*

Proof. Define $m_p := |B_P(p, \hat{r}_p)|$. Let $\mathcal{E}_p$ denote the event that $h(q) \leq (\beta \log n)/m_p$ for some $q \in B_P(p, \hat{r}_p)$. Note that: $\Pr[\mathcal{E}_p] = 1 - \Pr[\overline{\mathcal{E}_p}] = 1 - (1 - \beta \log n/m_p)^{m_p} \geq 1 - 1/n^\beta$. So, the event $\mathcal{E}_p$ occurs whp.

For each point $q \in B_P(p, \hat{r}_p)$, let $X_p(q) \in \{0, 1\}$ be an indicator random variable that is set to 1 iff $h(q) < (2\beta \log n)/m_p$. Further, let $X_p := \sum_{q \in B_P(p, \hat{r}_p)} X_p(q)$.

Since $h(q)$ is chosen independently and u.a.r. from $[0, 1]$, we have $\mathbb{E}[X_p(q)] = \Pr[X_p(q) = 1] = (2\beta \log n)/m_p$ for all $q \in B_p(p, \hat{r}_p)$. Thus, linearity of expectation implies that: $\mathbb{E}[X_p] = m_p \cdot (2\beta \log n)/m_p = 2\beta \log n$. At this stage, applying a Chernoff bound, we get: $X_p \leq 4\beta \log n$ whp.

Now, by a union bound, whp the following two events occur simultaneously: (i) $\mathcal{E}_p$ and (ii) $X_p \leq 4\beta \log n$.

Conditioned on event (i), it is easy to verify that $2^{-i_p} \leq (\beta \log n)/m_p$, and hence every point $q \in P_{i_p} \cap B_P(p, \hat{r}_p)$ has $X_p(q) = 1$. This, in turn, implies that $|P_{i_p} \cap B_P(p, \hat{r}_p)| \leq X_p \leq 4\beta \log n$. The last inequality holds conditioned on event (ii). This concludes the proof. □

Claim 4.8. *Condition on the (high probability) events described in the statements of Claim 4.6 and Claim 4.7. Consider any point $p \in P$. Then, with constant probability, we have $j_q \neq j_{q'}$ for all $q, q' \in P_{i_p} \cap B_P(p, \hat{r}_p)$ with $q \neq q'$.*

Proof. Since $|P_{i_p} \cap B_P(p, \hat{r}_p)| \leq 4\beta \log n$ and each j_q is chosen independently and u.a.r. from $[1, L_\beta]$ where $L_\beta \geq 16\beta^2 \log^2 n$, the claim follows from a balls-into-bins argument (specifically, from Birthday paradox). □