On the Facility Location Problem in Online and Dynamic Models

Xiangyu Guo
Department of Computer Science and Engineering, University at Buffalo, NY, USA
xiangyug@buffalo.edu

Janardhan Kulkarni
The Algorithms Group, Microsoft Research, Redmond, WA, USA
jakul@microsoft.com

Shi Li
Department of Computer Science and Engineering, University at Buffalo, NY, USA
shil@buffalo.edu

Jiayi Xian
Department of Computer Science and Engineering, University at Buffalo, NY, USA
jxian@buffalo.edu

Abstract

In this paper we study the facility location problem in the online with recourse and dynamic algorithm models. In the online with recourse model, clients arrive one by one and our algorithm needs to maintain good solutions at all time steps with only a few changes to the previously made decisions (called recourse). We show that the classic local search technique can lead to a $(1+\sqrt{2}+\epsilon)$-competitive online algorithm for facility location with only $O\left(\frac{\log n}{\epsilon}\log\frac{1}{\epsilon}\right)$ amortized facility and client recourse, where n is the total number of clients arrived during the process.

We then turn to the dynamic algorithm model for the problem, where the main goal is to design fast algorithms that maintain good solutions at all time steps. We show that the result for online facility location, combined with the randomized local search technique of Charikar and Guha [8], leads to a $(1+\sqrt{2}+\epsilon)$-approximation dynamic algorithm with total update time of $\tilde{O}(n^2)$ in the incremental setting against adaptive adversaries. The approximation factor of our algorithm matches the best offline analysis of the classic local search algorithm.

Finally, we study the fully dynamic model for facility location, where clients can both arrive and depart. Our main result is an $O(1)$-approximation algorithm in this model with $O(|F|)$ preprocessing time and $O(n\log^3 D)$ total update time for the HST metric spaces, where $|F|$ is the number of potential facility locations. Using the seminal results of Bartal [3] and Fakcharoenphol, Rao and Talwar [12], which show that any arbitrary N-point metric space can be embedded into a distribution over HSTs such that the expected distortion is at most $O(\log N)$, we obtain an $O(\log|F|)$ approximation with preprocessing time of $O(|F|^2\log|F|)$ and $O(n\log^3 D)$ total update time. The approximation guarantee holds in expectation for every time step of the algorithm, and the result holds in the oblivious adversary model.

2012 ACM Subject Classification Theory of computation → Facility location and clustering; Theory of computation → Online algorithms

Keywords and phrases Facility location, online algorithm, recourse

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2020.42

Category APPROX

Related Version Full version at https://arxiv.org/abs/2002.10658

Funding X. Guo, S. Li and J. Xian are partially supported by NSF grants CCF- 1717134 and CCF-1844890.

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020).
Editors: Jarosław Byrka and Raghu Meka; Article No. 42; pp. 42:1–42:23
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the *(uncapacitated) facility location problem*, we are given a metric space $(F \cup C, d)$, where F is the set of facility locations, C is the set of clients, and $d : (F \cup C) \times (F \cup C) \rightarrow \mathbb{R}_{\geq 0}$ is a distance function, which is non-negative, symmetric and satisfies triangle inequalities. For each location $i \in F$, there is a facility opening cost $f_i \geq 0$. The goal is open a subset $S \subseteq F$ of facilities so as to minimize cost of opening the facilities and the connection cost. The cost of connecting a client j to an open facility i is equal to $d(j, i)$. Hence, the objective function can be expressed concisely as $\min_{S \subseteq F} \Big(f(S) + \sum_{j \in C} d(j, S) \Big)$, where for a set $S \subseteq F$, $f(S) := \sum_{i \in S} f_i$ is the total facility cost of S and $d(j, S) := \min_{i \in S} d(j, i)$ denotes the distance of j to the nearest location in S. The facility location problem arises in countless applications: in the placement of servers in data centers, network design, wireless networking, data clustering, location analysis for placement of fire stations, medical centers, and so on. Hence, the problem has been studied extensively in many different communities: approximation algorithms, operations research, and computational geometry. In the approximation algorithms literature in particular, the problem occupies a prominent position as the development of every major technique in the field is tied to its application on the facility location problem. See the text book by Williamson and Shmoys [27] for more details. The problem is hard to approximate to a factor better than 1.463 [20]. The current best-known polynomial-time algorithm is given by the third author, and achieves 1.488-approximation [23].

Motivated by its applications in network design and data clustering, Meyerson [24] initiated the study of facility location problem in the online setting. Here, clients arrive online one-by-one, the algorithm has to assign the newly arriving client to an already opened facility or needs to open a new facility to serve the request. The decisions made by the algorithm are *irrevocable*, in the sense that a facility that is opened cannot be closed and the clients cannot be reassigned. In the online setting, Meyerson [24] designed a very elegant randomized algorithm that achieves an $O(\log n)$ competitive ratio, and also showed that no online algorithm can obtain $O(1)$ competitive ratio. This result was later extended by Fotakis [15] to obtain an *asymptotically optimal* $O(\log n/\log\log n)$-competitive algorithm. Both the algorithms and analysis techniques in [15, 24] were influential, and found many applications in other models such as streaming [17]. The lowerbound in Fotakis [15] holds even in very special metric spaces such as HSTs or the real line. Since then, several online algorithms have been designed achieving the same competitive ratio with more desirable properties such as deterministic [1], primal-dual [14], or having a small memory footprint [16]. We refer to a beautifully written survey by Fotakis [17] for more details.

The main reason to assume that decisions made by an algorithm are irrevocable is because the cost of changing the solution is expensive in some applications. However, if one examines these above applications closely, say for example connecting clients to servers in data centers, it is more natural to assume that decisions need not be irrevocable but the algorithm *should not change the solution too much*. This is even more true in modern data centers where topologies can be reconfigured; see [18] for more details. There are two standard ways of enforcing that an algorithm does not make too many changes to the solution:

- **Recourse.** The recourse per step of an online algorithm is the *number of changes* it makes to the solution. Recourse captures the *minimal* amount of changes an online algorithm *has to make* to maintain a desired competitive ratio due to the *information theoretic* limits. For the facility location problem, depending on the application, the recourse can correspond to: 1) the number of changes made to the opened facilities (called *facility* *recourse*) 2) the number of reconnections made to the clients (called *client recourse*). Notice that we can assume for every facility we open/close, we have to connect/disconnect at least one client. Thus the client recourse is at least the facility recourse.

- **Update time.** While having a small recourse is enough in some applications, it is not enough in some others. Take wireless networks as a concrete example. Here, the set of clients (mobile devices) keeps changing over time, and it is necessary to *update* the assignment of clients to facilities as *quickly* as possible so to minimize the service disruption. This motivates the *dynamic facility location problem*, which is similar to the one in online setting except that at each time step either *a new client arrives or an existing client departs.* The goal is to always maintain a solution that is a constant factor approximation to the optimal solution, while minimizing *the total time spent in updating the solution.* We emphasize that we require our dynamic algorithms to maintain *an actual assignment of clients to facilities*, not just the set of open facilities and an estimate of connection cost. This is important for applications mentioned above.

Recently, these applications have motivated the study of the facility location problem, and its closely related k-clustering problems, in the framework of recourse and dynamic algorithms [13, 10, 19, 9, 22, 21, 7]. The models for facility location problem considered in [13, 10] are different from what we study here, so we discuss them in Section 1.3; we also defer the discussion of k-clustering results [22, 21, 7] to Section 1.3. Goranci *et al* [19] showed that for metric spaces with *doubling dimension* κ, there is a deterministic *fully* dynamic algorithm with $\tilde{O}(2^{\kappa^2})$ update time, which maintains a constant approximation. Unfortunately, this result does not imply any non-trivial bound for the general metric spaces. More recently, Cohen-Addad *et al* [9] studied the facility location problem in general metric spaces but when the cost of opening facilities are same; that is, when $f_i = f, \forall i \in F$. They showed that there exists a *fully* dynamic algorithm that computes a $O(1)$ approximation with update time of $O(n \log n)$ and facility recourse of $O(1)$ per step. They also showed matching lowerbounds for both these results. However, their positive results do not imply any non-trivial bounds to the case when the cost of opening facilities are *non-uniform.* This is an important consideration in network design problems, and is also the form studied in Goranci *et al* [19] and in classic offline literature [27]. Further, to our understanding, none of the existing results bound the *client recourse*, which is an important consideration in real-world problems. As a concrete example, consider the problem of connecting clients to servers in datacenters, which was the original motivation for Meyerson [24] to initiate the study of online facility location problem. Here, it is important that one does not reconnect clients to servers too many times, as such changes can incur significant costs both in terms of disruption of service and the labor cost. Hence a natural question that arises is:

Are there online and dynamic algorithms for facility location problem in arbitrary metric spaces when the facility opening costs are not uniform, which achieve small client recourse and update times?

1.1 Our Contributions

In the following theorems, we use n to denote the total number of facility locations and all clients that ever arrived, $|F|$ the number of facilities, and D to denote the diameter of the metric d (assuming all distances are integers).

Our first result concerns online algorithms with small recourse in the incremental setting; that is, when clients only arrive and not depart.

▶ **Theorem 1.** *There is a deterministic online algorithm for the facility location problem that achieves a competitive ratio of* $(1+\sqrt{2}+\epsilon)$ *with* $O\left(\frac{\log n}{\epsilon}\log\frac{1}{\epsilon}\right)$ *amortized facility and client recourse against an adaptive adversary.*

Our next result is in the online dynamic model with the goal to achieve small update time. First we consider incremental setting.

▶ **Theorem 2.** *In the incremental setting against an adaptive adversary, there is a randomized dynamic algorithm for the facility location problem that, with probability at least* $1-1/n^2$*, maintains an approximation factor of* $(1+\sqrt{2}+\epsilon)$ *with amortized update time of* $O(\frac{n}{\epsilon^2}\log^3 n\log\frac{1}{\epsilon})$*. Further, there exists an* $O(1)$*-approximate dynamic algorithm with* $O(\frac{|F|}{\epsilon^2}\log^3 n\log\frac{1}{\epsilon})$*-amortized update time.*

Note that it takes $\Theta(n|F|)$ space to specify the input in our model (see Section 2.1). Our algorithm for the above theorem also simultaneously achieves an amortized *logarithmic* client recourse per step; see the remark at the end of Section 5.

Our algorithms to show the above two theorems differ from the previous approaches used in the context of online variants of facility location problem, and is based on *local search.* The local search algorithm is one of the most widely used algorithms for the facility location problem in practice, and is known to achieve an approximation factor of $(1+\sqrt{2})$ in the offline setting. See the influential paper by Arya *et al* [2] and a survey by Munagala [25]. Thus our result matches the best known approximation ratio for offline facility location using local search. Further, our result shows that the local search algorithm augmented with some small modifications is *inherently stable* as it does not make too many changes to the solutions even if clients are added in an online fashion. This gives further justification for its popularity among practitioners.

Finally, we study the fully dynamic setting, where the clients arrive and depart. Here, we first consider an important class of metric spaces called hierarchically well separated tree (HST) metrics [3]; see Definition 13 for the formal definition, and Section 2.1 for more details about how the input sequence is given. For HST metric spaces, we show the following result.

▶ **Theorem 3.** *In the fully dynamic setting against adaptive adversaries, there is a deterministic algorithm for the facility location problem that achieves an* $O(1)$ *approximation factor with* $O(|F|)$ *preprocessing time and* $O(\log^3 D)$ *amortized update time for the HST metric spaces.*

A seminal result by Bartal [3], which was later tightened by Fakcharoenphol, Rao and Talwar [12], shows that any arbitrary N-point metric space can be embedded into a distribution over HSTs such that the expected distortion is at most $O(\log N)$, which is also tight. Moreover, such a probabilistic embedding can also be computed in $O(N^2\log N)$ time; see recent results by Blelloch, Gu and Sun for details [5]. These results immediately imply the following theorem, provided the input is specified as in Section 2.1.

▶ **Theorem 4.** *In the fully dynamic setting against oblivious adversary, there is a randomized algorithm for the facility location problem that maintains an approximation factor of* $O(\log|F|)$ *with preprocessing time of* $O(|F|^2\log|F|)$ *and* $O(\log^3 D)$ *amortized update time. The approximation guarantee holds only in expectation for every time step of the algorithm.*

Observe that unlike the incremental setting, the above theorem holds only in the oblivious adversary model, as probabilistic embedding techniques preserve distances only in expectation as can be seen by taking a cycle on n points. Our result also shows that probabilistic tree embeddings using HSTs can be a very useful technique in the design of dynamic algorithms, similar to its role in online algorithms [3, 4, 26, 6].

Our algorithms in Theorems 3 and 4 in the fully dynamic setting also have the property that amortized client and facility *recourse* is $O(\log^3 D)$ (in fact, we can achieve a slight better bound of $O(\log^2 D)$ as can be seen from the analysis). This holds as our dynamic algorithms maintain the entire assignment of clients to facilities *explicitly* in memory at every time step. Thus, the amortized client reconnections is at most the amortized update time.

1.2 Our Techniques

Our main algorithmic technique for proving Theorems 1 and 2 is local search, which is one of the powerful algorithm design paradigms. Indeed, for both results, the competitive (approximation) ratio we achieve is $1 + \sqrt{2} + \epsilon$, which matches the best approximation ratio for offline facility location obtained using local search [2]. Both of our results are based on the following key lemma. Suppose we maintain local optimum solutions at every time step in our algorithm. When a new client j_t comes at time t, we add it to our solution using a simple operation, and let Δ_t be the increase of our cost due to the arrival of j_t. The key lemma states that the sum of Δ_t values in the first T' time steps can be bounded in terms the optimum cost at time T'. With a simple modification to the local search algorithm, in which we require each local operation decreases enough cost for every client it reconnects, one can bound the total client recourse.

The straightforward way to implement the local search algorithm takes time $\Omega(n^3)$. To derive a better running time, we leverage the randomized local search idea of Charikar and Guha [8]. At every iteration, we randomly choose a facility i or a closing operation, and then perform the best operation that opens or swaps in i, or closes a facility if that is what we choose. By restricting the facility i and with the help of the heap data structure, an iteration of the algorithm can be implemented in time $O(|C| \log |F|)$. As in [8] we can also show that each iteration can make a reasonable progress in expectation, leading to a bound of $\tilde{O}(|F|)$ on the number of iterations for the success of the algorithm with high probability. We remark that the algorithm in [8] used a different local search framework. Therefore, our result shows that the classic algorithm of [2] can also be made fast.

However, directly replacing the randomized local search procedure with a deterministic one does not work: The solution at the end of each time might not be a local optimum as we did not enumerate all possible local operations. Thus the key lemma does not hold any more. Nevertheless we show that applying a few local operations around j_t upon its arrival can address the issue. With the key lemma, one can bound the number of times we perform the iterative randomized local search procedure, and thus the overall running time.

Our proof for Theorem 3 is based on a generalization of the greedy algorithm for facility location on HST metrics, which was developed in [11] in the context of *differential privacy* but only for the case of *uniform* facility cost. The intuition of the algorithm is as follows: If for some vertex v of the HST T, the number of clients in the tree T_v (the sub-tree of T rooted at v) times the length of parent edge of v is big compared to the cost of the cheapest facility in T_v, then we should open that facility. Otherwise, we should not open it and let the clients in T_v be connected to outside T_v through the parent edge. This intuition can be made formal: We mark v in the former case; then simply opening the cheapest facility in T_v for all *lowest marked* vertices v leads to a constant approximation for facility location.

The above offline algorithm leads to a *dynamic data structure* that maintains $O(1)$-approximate solutions, supports insertion and deletion of clients, and reports the connecting facility of a client in $O(\log D)$ time. This is the case since each time a client arrives or departs, only its ancestors will be affected. However, in a dynamic algorithm setting, we need to maintain the assignment vector in memory, so that when the connecting facility of a client changes, it needs to be notified. This requires that the number of reconnections made by our algorithm to be small. To achieve the goal, we impose two constants for each v when deciding whether v should be marked and the cheapest facility in T_v should be open. When a vertex v changes its marking/opening status, we update the constants in such a way that it becomes hard for the status to be changed back.

1.3 More Related Work

Lattanzi and Vassilvitskii [22] first studied the role of recourse (which they call "consistency") in online k-clustering problem when clients only arrive – a setting similar to our Theorem 1. Lattanzi and Vassilvitskii gave a constant approximation algorithm with $O(k^2 \log^4 n)$ total recourse (reclusterings). They also showed an $\Omega(k \log n)$ recourse lowerbound. There are some subsequent works studying the k-clustering problems in the context of recourse and dynamic algorithms; we refer the readers to [22, 21, 7, 9] for more details.

A fully dynamic online model of facility location problem, where clients arrive and depart was recently studied by Cygan *et al.* [10], but with different assumption on recourse. In this model, the set of facility locations is the same as the set of clients and thus they are also changing dynamically. When a client arrives, the algorithm has to assign it to an open facility immediately; while upon departure of a client, if a facility was opened at the same location, then the clients that were assigned to that location should be reassigned immediately and irrevocably. They showed that a delicate extension of Meyerson's [24] algorithm obtains asymptotically tight competitive ratio of $O(\log n/\log\log n)$. However, the result holds only for the *uniform facility costs* and Cygan *et al* [10] even showed an *unbounded* lower bound for the non-uniform facility cost case in their model. Moreover, in their model reconnections of clients are assumed to be "automatic" and do not count towards the client recourse; it is not clear how many client reconnections their algorithm will make. Hence, our result in Theorem 4 also implies that one can circumvent the unbounded lowerbound for the general problem with small recourse.

As we noted earlier, a strictly online algorithm for facility location problem cannot achieve a competitive ratio better than $O(\log n/\log\log n)$. To overcome this, the problem has been studied in the so-called *incremental model*, which allows two open facilities to be merged at any given time. In this model [13, 17] showed that one can obtain $O(1)$ approximation for case when facility opening costs are uniform. However, the recourse of such an algorithm may be arbitrary.

2 Preliminaries

Throughout the paper, we use F to denote the set of potential facilities for all the problems and models; we assume F is given upfront. C is the dynamic set of clients we need to connect by our algorithm. This is not necessarily the set of clients that are present: In the algorithms for online facility location with recourse and dynamic facility location in the incremental setting, we fix the connections of some clients as the algorithms proceed. These clients are said to be "frozen" and excluded from C. We shall always use d to denote the hosting metric containing F and all potential clients. For any point j and subset V of points in the metric, we define $d(j, V) = \min_{v \in V} d(j, v)$ to be the minimum distance from j to a point in V. We assume all distances are integers, the minimum non-zero distance between two points is 1. We define D, the diameter or the aspect ratio of a metric space, as the largest distance between two points in it. Let n be $|F|$ plus the total number of clients arrived during the whole process. The algorithms do not need to know the exact value of

n in advance, except that in the dynamic algorithm for facility location in the incremental setting (the problem in Theorem 2), to achieve the $1 - 1/n^2$ success probability, a sufficiently large $\Gamma = \text{poly}(n, \log D, \frac{1}{\epsilon})$ needs to be given.[1]

In all the algorithms, we maintain a set S of open facilities, and a connection $\sigma \in S^C$ of clients in C to facilities in S. We do not require that σ connects clients to their respective nearest open facilities. For any solution $(S' \subseteq F, \sigma' \in S'^C)$, we use $\mathsf{cc}(\sigma') = \sum_{j \in C} d(j, \sigma_j)$ to denote the connection cost of the solution. For facility location, we use $\mathsf{cost}(S', \sigma') = f(S') + \mathsf{cc}(\sigma')$ to denote the total cost of the solution (S', σ'), where $f(S') := \sum_{i \in S'} f_i$. Notice that σ and the definitions of cc and cost functions depend on the dynamic set C.

Throughout the paper, we distinguish between a "moment", a "time" and a "step". A moment refers to a specific time point during the execution of our algorithm. A time corresponds to an arrival or a departure event: At each time, exactly one client arrives or departs, and time t refers to the period from the moment the t-th event happens until the moment the $(t+1)$-th event happens (or the end of the algorithm). One step refers to one statement in our pseudo-codes indexed by a number.

2.1 Specifying Input Sequence

In this section we specify how the input sequence is given. For the online and dynamic facility location problem, we assume the facility locations F, their costs $(f_i)_{i \in F}$, and the metric d restricted to F are given upfront, and they take $O(|F|^2)$ space. Whenever a client $j \in C$ arrives, it specifies its distance to every facility $i \in F$ (notice that the connection cost of an assignment $\sigma \in S^C$ does not depend on distances between two clients and thus they do not need to be given). Thus the whole input contains $O(n|F|)$ words.

For Theorems 3 and 4, as we do not try to optimize the constants, we *do not* need that a client specifies its distance to every facility. By losing a multiplicative factor of 2 and an additive factor of 1 in the approximation ratio, we can assume that every client j is collocated with its nearest facility in F Thus, we only require that when a client j comes, it reports the position of its nearest facility. For Theorem 3, the HST T over F is given at the beginning using $O(|F|)$ words. For Theorem 4, the metric d over F is given at the beginning using $O(|F|^2)$ words. Then, we use an efficient algorithm [5] to sample a HST T.

2.2 Local Search for facility location

The local-search technique has been used to obtain the classic $(1 + \sqrt{2})$-approximation offline algorithm for facility location [2]. We now give an overview of the algorithm, which will be the baseline of our online and dynamic algorithms for facility location. One can obtain a (tight) 3-approximation for facility location without scaling facility costs. Scaling the facility costs by a factor of $\lambda := \sqrt{2}$ when deciding whether an operation can decrease the cost, we can achieve a better approximation ratio of $\alpha_{\mathsf{FL}} := 1 + \sqrt{2}$. Throughout, we fix the constants $\lambda = \sqrt{2}$ and $\alpha_{\mathsf{FL}} = 1 + \sqrt{2}$. For a solution (S', σ') to a facility location instance, we use $\mathsf{cost}_\lambda(S', \sigma') = \lambda f(S') + \mathsf{cc}(\sigma')$ to denote the cost of the solution (S', σ') with facility costs scaled by $\lambda = \sqrt{2}$. We call $\mathsf{cost}_\lambda(S', \sigma')$ the *scaled cost* of (S', σ').

Given the current solution (S, σ) for a facility location instance defined by F, C, d and $(f_i)_{i \in F}$, we can apply a *local operation* that changes the solution (S, σ). A valid local operation is one of the following.

[1] For an algorithm that might fail, we need to have some information about n to obtain a failure probability that depends on n.

- An open operation, in which we open some facility $i \in F$ and reconnect a subset $C' \subseteq C$ of clients to i. We allow i to be already in S, in which case we simply reconnect C' to i. This needs to be allowed since our σ does not connect clients to their nearest open facilities.
- A close operation, we close some facility $i' \in S$ and reconnect the clients in $\sigma^{-1}(i')$ to facilities in $S \setminus \{i'\}$.
- In a swap operation, we open some facility $i \notin S$ and close some facility $i' \in S$, reconnect the clients in $\sigma^{-1}(i')$ to facilities in $S \setminus \{i'\} \cup \{i\}$, and possibly some other clients to i. We say i is *swapped in* and i' is *swapped out* by the operation.

Thus, in any valid operation, we can open and/or close at most one facility. A client can be reconnected if it is currently connected to the facility that will be closed, or it will be connected to the new open facility. After we apply a local operation, S and σ will be updated accordingly so that (S, σ) is always the current solution.

For the online algorithm with recourse model, since we need to bound the number of reconnections, we apply a local operation only if the *scaled* cost it decreases is large compared to the number of reconnections it makes. This motivates the following definition:

▶ **Definition 5** (Efficient operations for facility location). *Given a $\phi \geq 0$, we say a local operation on a solution (S, σ) for a facility location instance is ϕ-efficient, if it decreases* $\mathsf{cost}_\lambda(S, \sigma)$ *by more than ϕ times the number of clients it reconnects.*

The following theorem can be derived from the analysis for the local search algorithms for facility location.

▶ **Theorem 6.** *Consider a facility location instance with cost of the optimum solution being* opt *(using the original cost function). Let (S, σ) be the current solution in our algorithm and $\phi \geq 0$ be a real number. If there are no ϕ-efficient local operations on (S, σ), then we have*

$$\mathsf{cost}(S, \sigma) \leq \alpha_{\mathsf{FL}}\big(\mathsf{opt} + |C|\phi\big).$$

In particular, if we apply the theorem with $\phi = 0$, then we obtain that (S, σ) is an $(\alpha_{\mathsf{FL}} = 1 + \sqrt{2})$-approximation for the instance.

2.3 A Useful Lemma

The following lemma will be used repeatedly in our analysis. It's not hard to prove, but due to space constraints, we only include the proof in the full version of this paper.

▶ **Lemma 7.** *Let $b \in \mathbb{R}^T_{\geq 0}$ for some integer $T \geq 1$. Let $B_{T'} = \sum_{t=1}^{T'} b_t$ for every $T' = 0, 1, \cdots, T$. Let $0 < a_1 \leq a_2 \leq \cdots \leq a_T$ be a sequence of real numbers and $\alpha > 0$ such that $B_t \leq \alpha a_t$ for every $t \in [T]$. Then we have*

$$\sum_{t=1}^{T} \frac{b_t}{a_t} \leq \alpha \left(\ln \frac{a_T}{a_1} + 1 \right).$$

Organization. The rest of the paper is organized as follows. In Section 3, we prove Theorem 1 by giving our online algorithm for facility location with recourse. Section 4 gives the randomized local search procedure, that will be used in the proof of Theorem 2 in Section 5. We give some open problems and future directions in Section 6. Finally, the proof of Theorem 4 is deferred to Appendix A, where we give the fully dynamic algorithm for facility location in HST metrics. Due to space constraint, some proofs are not included in this paper, and we refer interested readers to the full version.

3 $(1+\sqrt{2}+\epsilon)$-Competitive Online Algorithm with Recourse

In this section, we prove Theorem 1 by giving the algorithm for online facility location with recourse.

3.1 The Algorithm

For any $\epsilon > 0$, let $\epsilon' = \Theta(\epsilon)$ be a parameter that is sufficiently small so that the approximation ratio $\alpha_{\mathsf{FL}} + O(\epsilon') = 1+\sqrt{2}+O(\epsilon')$ achieved by our algorithm is at most $\alpha_{\mathsf{FL}}+\epsilon$. Our algorithm for online facility location is easy to describe. Whenever the client j_t comes at time t, we use a simple rule to connect j_t, as defined in the procedure initial-connect in Algorithm 1: either connecting j_t to the nearest facility in S, or opening and connecting j_t to its nearest facility in $F \setminus S$, whichever incurs the smaller cost. Then we repeatedly perform ϕ-efficient operations (Definition 5), until no such operations can be found, for $\phi = \frac{\epsilon' \cdot \mathsf{cost}(S,\sigma)}{\alpha_{\mathsf{FL}}|C|}$.[2]

Algorithm 1 initial-connect(j).

```
if $\min_{i \in F \setminus S}(f_i + d(i,j)) < d(j,S)$ then
    let $i^* = \arg\min_{i \in F \setminus S}(f_i + d(i,j))$, $S \leftarrow S \cup \{i^*\}, \sigma_j \leftarrow i^*$
else $\sigma_j \leftarrow \arg\min_{i \in S} d(j,i)$
```

We can show that the algorithm gives an $(\alpha_{\mathsf{FL}}+\epsilon)$-approximation with amortized recourse $O(\log D \log n)$; recall that D is the aspect ratio of the metric. To remove the dependence on D, we divide the algorithm into stages, and *freeze* the connections of clients that arrived in early stages. The final algorithm is described in Algorithm 3, and Algorithm 2 gives one stage of the algorithm.

Algorithm 2 One Stage of Online Algorithm for Facility Location.

```
Input:  - $C$: initial set of clients
        - $(S,\sigma)$: a solution for $C$ which is $O(1)$-approximate
        - Clients $j_1, j_2, \cdots$ arrive from time to time
Output: Guaranteeing that $(S,\sigma)$ at the end of each time $t$ is $\frac{\alpha_{\mathsf{FL}}}{1-\epsilon'}$-approximate
init $\leftarrow$ cost$(S,\sigma)$
for $t \leftarrow 1,2,\cdots$, terminating if no more clients will arrive do
    $C \leftarrow C \cup \{j_t\}$, and call initial-connect$(j_t)$
    while there exists an $\frac{\epsilon' \cdot \mathsf{cost}(S,\sigma)}{\alpha_{\mathsf{FL}}|C|}$-efficient local operation do perform the operation
    if cost$(S,\sigma) >$ init$/\epsilon'$ then terminate the stage
```

In Algorithm 2, we do as described above, with two modifications. First, we are given an initial set C of clients and a solution (S,σ) for C which is $O(1)$-approximate. Second, the stage will terminate if the cost of our solution increases by a factor of more than $1/\epsilon'$. The main algorithm (Algorithm 3) is broken into many stages. Since we shall focus on one stage of the algorithm for most part of our analysis, we simply redefine the time so that every stage starts with time 1. The improved recourse comes from the *freezing* operation: at the

[2] There are exponential number of possible operations, but we can check if there is a ϕ-efficient one efficiently. close operations can be handled easily. To check if we can open a facility i, it suffices to check if $\sum_{j \in C: d(j,i)+\phi < d(j,\sigma_j)}(d(j,\sigma_j) - d(j,i) - \phi) > \lambda f_i \cdot 1_{i \notin S}$. swap operations are more complicated but can be handled similarly.

Algorithm 3 Online Algorithm for Facility Location.

```
$C \leftarrow \emptyset, S \leftarrow \emptyset, \sigma = ()$
repeat
    $C^\circ \leftarrow C, (S^\circ, \sigma^\circ) \leftarrow (S, \sigma)$
    redefine the next time to be time 1 and run one stage as defined in Algorithm 2
    permanently open one copy of each facility in $S^\circ$, and permanently connect clients in $C^\circ$ according to $\sigma^\circ$ (we call the operation *freezing* $S^\circ$ and $C^\circ$)
    $C \leftarrow C \setminus C^\circ$, restrict the domain of $\sigma$ to be the new $C$
until no clients come
```

end of each stage, we permanently open one copy of each facility in S°, and permanently connect clients in C° to copies of S° according to σ°, where C° and (S°, σ°) are the client set and solution at the beginning of the stage. Notice that we assume the original facilities in S° will still participate in the algorithm in the future; that is, they are subject to opening and closing. Thus each facility may be opened multiple times during the algorithm and we take the facility costs of all copies into consideration. This assumption is only for the sake of analysis; the actual algorithm only needs to open one copy and the costs can only be smaller compared to the described algorithm.

From now on, we focus on one stage of the algorithm and assume that the solution given at the beginning of each stage is $O(1)$-approximate. In the end we shall account for the loss due to the freezing of clients and facilities. Within a stage, the approximation ratio follows directly from Theorem 6: Focus on the moment after the while loop at time step t in Algorithm 2. Since there are no $\frac{\epsilon' \cdot \mathsf{cost}(S,\sigma)}{\alpha_{\mathsf{FL}}|C|}$-efficient local operations on (S, σ), we have by the theorem that $\mathsf{cost}(S,\sigma) \leq \alpha_{\mathsf{FL}} \left(\mathsf{opt} + |C| \cdot \frac{\epsilon' \cdot \mathsf{cost}(S,\sigma)}{\alpha_{\mathsf{FL}}|C|}\right) = \alpha_{\mathsf{FL}}\mathsf{opt} + \epsilon' \cdot \mathsf{cost}(S,\sigma)$, where opt is the cost of the optimum solution for C. Thus, at the end of each time, we have $\mathsf{cost}(S,\sigma) \leq \frac{\alpha_{\mathsf{FL}}}{1-\epsilon'} \cdot \mathsf{opt}$.

3.2 Bounding Amortized Recourse in One Stage

We then bound the amortized recourse in a stage; we assume that $\mathsf{cost}(S,\sigma) > 0$ at the beginning of the stage since otherwise there will be no recourse involved in the stage (since we terminate the stage when the cost becomes non-zero). We use T to denote the last time of the stage. For every time t, let C_t be the set C at the end of time t, and opt_t to be the cost of the optimum solution for the set C_t. For every $t \in [T]$, we define Δ_t to be the value of $\mathsf{cost}(S,\sigma)$ after Step 3 at time step t in Algorithm 2, minus that before Step 3. We can think of this as the cost increase due to the arrival of j_t.

The key lemma we can prove is the following:

▶ **Lemma 8.** *For every $T' \in [T]$, we have*

$$\sum_{t=1}^{T'} \Delta_t \leq O(\log T')\mathsf{opt}_{T'}.$$

With Lemma 8, we can now bound the amortized recourse of one stage. In time t, $\mathsf{cost}(S,\delta)$ first increases by Δ_t in Step 3. Then after that, it decreases by at least $\frac{\epsilon'\mathsf{cost}(S,\sigma)}{\alpha_{\mathsf{FL}}|C|} \geq \frac{\epsilon'\mathsf{opt}_t}{\alpha_{\mathsf{FL}}|C|} \geq \frac{\epsilon'\mathsf{opt}_t}{\alpha_{\mathsf{FL}}|C_{T'}|}$ for every reconnection we made. Let $\Phi_{T'} = \sum_{t=1}^{T'} \Delta_t$; Lemma 8 says $\Phi_t \leq \alpha\mathsf{opt}_t$ for some $\alpha = O(\log T)$ and every $t \in [T]$. Noticing that $(\mathsf{opt}_t)_{t\in T}$ is a non-decreasing sequence, the total number of reconnections is at most

$$\frac{\mathsf{init}}{\epsilon' \cdot \mathsf{opt}_1/(\alpha_{\mathsf{FL}}|C_T|)} + \sum_{t=1}^{T} \frac{\Delta_t}{\epsilon' \cdot \mathsf{opt}_t/(\alpha_{\mathsf{FL}}|C_T|)} = \frac{\alpha_{\mathsf{FL}}|C_T|}{\epsilon'} \left(\frac{\mathsf{init}}{\mathsf{opt}_1} + \sum_{t=1}^{T-1} \frac{\Delta_t}{\mathsf{opt}_t} + \frac{\Delta_T}{\mathsf{opt}_T} \right).$$

Notice that $\mathsf{init} \leq O(1)\mathsf{opt}_0 \leq O(1)\mathsf{opt}_1$. Applying Lemma 7 with T replaced by $T-1$, $b_t = \Delta_t, B_t = \Phi_t$ and $a_t = \mathsf{opt}_t$ for every t, we have that $\sum_{t=1}^{T-1} \frac{\Delta_t}{\mathsf{opt}_t} \leq \alpha \left(\ln \frac{\mathsf{opt}_{T-1}}{\mathsf{opt}_1} + 1 \right) = O\left(\log T \log \frac{1}{\epsilon'} \right)$, since we have $\mathsf{opt}_{T-1} \leq O(1/\epsilon') \cdot \mathsf{opt}_1$. Notice that $\Delta_T \leq \mathsf{opt}_T$ since $\mathsf{opt}_T \geq \min_{i \in F}(f_i + d(i, j_T)) \geq \Delta_T$. So, the total number of reconnections is at most $O\left(\frac{\log T}{\epsilon'} \log \frac{1}{\epsilon'} \right) \cdot |C_T|$. The amortized recourse per client is $O\left(\frac{\log T}{\epsilon'} \log \frac{1}{\epsilon'} \right) \leq O\left(\frac{\log n}{\epsilon'} \log \frac{1}{\epsilon'} \right)$, where in the amortization, we only considered clients involved in the stage. Recall that n is the total number of clients arrived.

As each client appears in at most 2 stages, the overall amortized recourse is $O\left(\frac{\log n}{\epsilon'} \log \frac{1}{\epsilon'} \right)$. Finally we consider the loss in the approximation ratio due to freezing of clients. Suppose we are in the p-th stage. Then the clients arrived at and before $(p-2)$-th stage has been frozen and removed. Let $\overline{\mathsf{opt}}$ be the cost of the optimum solution for all clients arrived at or before $(p-1)$-th stage. Then the frozen facilities and clients have cost at most $\overline{\mathsf{opt}} \cdot O\left(\epsilon' + \epsilon'^2 + \epsilon'^2 + \cdots \right) = O(\epsilon')\overline{\mathsf{opt}}$. In any time in the p-th stage, the optimum solution taking all arrived clients into consideration has cost $\overline{\mathsf{opt}}' \geq \overline{\mathsf{opt}}$, and our solution has cost at most $(\alpha_{\mathsf{FL}} + O(\epsilon'))\overline{\mathsf{opt}}'$ without considering the frozen clients and facilities. Thus, our solution still has approximation ratio $\frac{(\alpha_{\mathsf{FL}} + O(\epsilon'))\overline{\mathsf{opt}}' + O(\epsilon')\overline{\mathsf{opt}}}{\overline{\mathsf{opt}}'} = \alpha_{\mathsf{FL}} + O(\epsilon')$ when taking the frozen clients into consideration.

4 Fast Local Search via Randomized Sampling

From now on, we will be concerned with dynamic algorithms. Towards proving Theorem 2 for the incremental setting, we first develop a randomized procedure that allows us to perform local search operations fast. In the next section, we use this procedure and ideas from the previous section to develop the dynamic algorithm with the fast update time.

The high level idea is as follows: We partition the set of local operations into many "categories" depending on which facility it tries to open or swap in. In each iteration of the procedure, we sample the category according to some distribution and find the best local operation in this category. By only focusing on one category, one iteration of the procedure can run in time $O(|C| \log |F|)$. On the other hand, the categories and the distribution over them are designed in such a way that in each iteration, the cost of our solution will be decreased by a multiplicative factor of $1 - \Omega\left(\frac{1}{|F|}\right)$. This idea has been used in [8] to obtain their $\tilde{O}(n^2)$ algorithm for approximating facility location. However, their algorithm was based on a different local search algorithm and analysis; for consistency and convenience of description, we stick to original local search algorithm of [2] that leads to $(1+\sqrt{2})$-approximation for the problem. Our algorithm needs to use the heap data structure.

4.1 Maintaining Heaps for Clients

Unlike the online algorithm for facility location in Section 3, in the dynamic algorithm, we guarantee that the clients are connected to their nearest open facilities. That is, we always have $\sigma_j = \arg\min_{i \in S} d(j, i)$; we still keep σ for convenience of description. We maintain $|C|$ min-heaps, one for each client $j \in C$: The min-heap for j will contain the facilities in

Algorithm 4 Δ-open(i).

```
1: return $\lambda f_i - \sum_{j \in C} \max\{0, d(j,\sigma_j) - d(j,i)\}$
```

Algorithm 5 try-open(i).

```
1: if $\Delta$-open$(i) < 0$ then open $i$ by updating $S, \sigma$ and heaps accordingly
```

Algorithm 6 Δ-swap-in(i).

```
1: $C' \leftarrow \{j \in C : d(j,i) < d(j,\sigma_j)\}$ and $\Psi \leftarrow \lambda f_i - \sum_{j \in C'} \big(d(j,\sigma_j) - d(j,i)\big)$
2: $\Delta \leftarrow \min_{i' \in S} \left\{ \sum_{j \in \sigma^{-1}(i') \setminus C'} \big[\min\{d(j,i), d(j, \text{heap-top}(j))\} - d(j,i')\big] - \lambda f_{i'} \right\} + \Psi$
3: return $(\Delta$, the $i'$ above achieving the value of $\Delta)$
```

Algorithm 7 Δ-close.

```
1: $\Delta \leftarrow \min_{i' \in S} \left\{ \sum_{j \in \sigma^{-1}(i')} \big[d(j, \text{heap-top}(j)) - d(j,i')\big] - \lambda f_{i'} \right\}$
2: return $(\Delta$, the $i'$ above achieving the value of $\Delta)$
```

$S \setminus \{\sigma_j\}$, with priority value of i being $d(j,i)$. This allows us to efficiently retrieve the second nearest open facility to each j: This is the facility at the top of the heap for j and we use the procedure heap-top(j) to return it.

We define four simple procedures Δ-open, try-open, Δ-swap-in and Δ-close that are described in Algorithms 4, 5, 6 and 7 respectively. Recall that we use the *scaled cost* for the local search algorithm; so we are working on the scaled cost function in all these procedures. Δ-open(i) for any $i \notin S$ returns Δ, the increment of the scaled cost that will be incurred by opening i. (For it to be useful, Δ should be negative, in which case $|\Delta|$ indicates the cost decrement of opening i). This is just one line procedure as in Algorithm 4; try-open will open i if it can reduce the scaled cost. Δ-swap-in(i) for some $i \notin S$ returns a pair (Δ, i'), where Δ is the smallest scaled cost increment we can achieve by opening i and closing some facility $i' \in S$, and i' gives the facility achieving the smallest value. (Again, for Δ to be useful, it should be negative, in which case i' is the facility that gives the maximum scaled cost decrement $|\Delta|$.) Similarly, Δ-close returns a pair (Δ, i'), which tells us the maximum scaled cost decrement we can achieve by closing one facility and which facility can achieve the decrement. Notice that in all the procedures, the facility we shall open or swap in is given as a parameter, while the facility we shall close is chosen and returned by the procedures.

With the heaps, the procedures Δ-open, Δ-swap-in and Δ-close can run in $O(|C|)$ time. We only analyze Δ-swap-in(i) as the other two are easier. First, we define C' to be the set of clients j with $d(j,i) < d(j,\sigma_j)$; these are the clients that will surely be reconnected to i once i is swapped in. Let $\Psi = \lambda f_i - \sum_{j \in C'} (d(j,\sigma_j) - d(j,i))$ be the net scaled cost increase by opening i and connecting C' to i. The computation of C' and Ψ in Step 1 takes $O(|C|)$ time. If additionally we close some $i' \in S$, we need to reconnect each client in $\sigma^{-1}(i') \setminus C'$ to either i, or the top element in the heap for j, whichever is closer to j. Steps 2 and 3 compute and return the best scaled cost increment and the best i'. Since $\sum_{i' \in S} |\sigma^{-1}(i')| = |C|$, the running time of the step can be bounded by $O(|C|)$.

The running time for try-open, swapping two facilities and closing a facility (which are not defined explicitly as procedures, but used in Algorithms 8) can be bounded by $O(|C| \log |F|)$. The running times come from updating the heap structures: For each of the $|C|$ heaps, we need to delete and/or add at most 2 elements; each operation takes time $O(\log |F|)$.

Algorithm 8 sampled-local-search.

```
if rand$(0,1) < 1/3$ then        ▷ rand$(0,1)$ returns a uniformly random number in $[0,1]$
    $(\Delta, i') \leftarrow \Delta$-close
    if $\Delta < 0$ then close $i'$ by updating $S, \delta$ and heaps accordingly
else
    $i \leftarrow$ random facility in $F \setminus S$
    $\Delta \leftarrow \Delta$-open$(i)$, $(\Delta', i') \leftarrow \Delta$-swap-in$(i)$
    if $\Delta \le \Delta'$ and $\Delta < 0$ then open $i$ by updating $S, \delta$ and heaps accordingly
    else if $\Delta' < 0$ then open $i$ and close $i'$ by updating $S, \delta$ and heaps accordingly
```

Algorithm 9 FL-iterate(M).

```
$(S^{\text{best}}, \sigma^{\text{best}}) \leftarrow (S, \sigma)$
for $\ell \leftarrow 1$ to $M$ do
    call sampled-local-search
    if cost$(S, \sigma) <$ cost$(S^{\text{best}}, \sigma^{\text{best}})$ then $(S^{\text{best}}, \sigma^{\text{best}}) \leftarrow (S, \sigma)$
return $(S^{\text{best}}, \sigma^{\text{best}})$
```

4.2 Random Sampling of Local Operations

With the support of the heaps, we can design a fast algorithm to implement randomized local search. sampled-local-search in Algorithm 8 gives one iteration of the local search. We first decide which operation we shall perform randomly. With probability $1/3$, we perform the close operation that will reduce the scaled cost the most (if it exists). With the remaining probability $2/3$, we perform either an open or a swap operation. To reduce the running time, we randomly choose a facility $i \in F \setminus S$ and find the best operation that opens or swaps in i, and perform the operation if it reduces the cost. One iteration of sampled-local-search calls the procedures in Algorithms 4 to 7 at most once and performs at most one operation, and thus has running time $O(|C| \log |F|)$.

In the procedure FL-iterate(M) described in Algorithm 9, we run the sampled-local-search M times. It returns the best solution obtained in these iterations, according to the *original (non-scaled) cost*, which is not necessarily the solution given in the last iteration. So we have

▶ **Observation 9.** *The running time of* FL-iterate(M) *is* $O(M|C| \log |F|)$*, where* C *is the set of clients when we run the procedure.*

Throughout this section, we fix a facility location instance. Let (S^*, σ^*) be the optimum solution (w.r.t the original cost) and opt $=$ cost(S^*, σ^*) be the optimum cost. Fixing one execution of sampled-local-search, we use (S^0, σ^0) to denote the solution before the execution. Then, we have

▶ **Lemma 10.** *Let* (S°, σ°) *be the* (S, σ) *at the beginning of an execution of* FL-iterate(M)*, and assume it is an* $O(1)$*-approximation to the instance. Let* $\Gamma \ge 2$ *and* $M = O\left(\frac{|F|}{\epsilon'} \log \Gamma\right)$ *is big enough. Then with probability at least* $1 - \frac{1}{\Gamma}$*, the solution returned by the procedure is* $(\alpha_{\mathsf{FL}} + \epsilon')$*-approximate.*

5 $(1+\sqrt{2}+\epsilon)$-Approximate Dynamic Algorithm for Facility Location in Incremental Setting

In this section, we prove Theorem 2 by combining the ideas from Sections 3 and 4 to derive a dynamic algorithm for facility location in the incremental setting. As for the online algorithm in Section 3, we divide our algorithm into stages. Whenever a client comes, we use a simple rule to accommodate it. Now we can not afford to consider all possible local operations as in Section 3. Instead we use the randomized local search idea from the algorithm in Section 4 by calling the procedure FL-iterate. We call the procedure only if the cost of our solution has increased by a factor of $1+\epsilon'$ (where $\epsilon' = \Theta(\epsilon)$ is small enough). In our analysis, we show a lemma similar to Lemma 8: The total increase of costs due to arrival of clients is small, compared to the optimum cost for these clients. Then, we can bound the number of times we call FL-iterate. Recall that we are given an integer $\Gamma = \mathrm{poly}\big(n, \log D, \frac{1}{\epsilon}\big)$ that is big enough: We are aiming at a success probability of $1 - 1/\Gamma$ for each call of FL-iterate. Our final running time will only depend on $O(\log \Gamma)$.

The main algorithm will be the same as Algorithm 3, except that we use Algorithm 10 as the algorithm for one stage. As before, we only need to design one stage of the algorithm. Recall that in a stage we are given an initial set C of clients, an $O(1)$-approximate solution (S, σ) for C. Clients come one by one and our goal is to maintain an $(\alpha_{\mathsf{FL}} + O(\epsilon'))$-approximate solution at any time. The stage terminates if no client comes or our solution has cost more than $1/\epsilon'$ times the cost of the initial solution.

Algorithm 10 One Stage of Dynamic Algorithm for Facility Location.

```
Input: ▪ $C$: the initial set of clients
       ▪ $(S, \sigma)$: initial solution for $C$, which is $O(1)$-approximate
1:  let $M = O\left(\frac{|F|}{\epsilon'} \log \Gamma\right)$ be large enough
2:  $(S, \sigma) \leftarrow \mathsf{FL\text{-}iterate}(M)$, $\mathsf{init} \leftarrow \mathsf{cost}(S, \sigma)$, $\mathsf{last} \leftarrow \mathsf{init}$
3:  for $t \leftarrow 1, 2, 3, \cdots$, terminating if no more clients arrive do
4:      for $q = \left\lceil \log \frac{\mathsf{last}}{|F|} \right\rceil$ to $\left\lceil \log \frac{\mathsf{last}}{\epsilon'} \right\rceil$ do
5:          if $i \leftarrow \arg\min_{i \in F \setminus S, f_i \le 2^q} d(j_t, i)$ exists, then call $\mathsf{try\text{-}open}'(i)$   ▷ $\mathsf{try\text{-}open}'$ is the
            same as try-open except we consider the cost instead of scaled cost.
6:      $C \leftarrow C \cup \{j_t\}$ and call $\mathsf{try\text{-}open}'\big(\arg\min_{i \in F \setminus S}(d(j_t, i) + f_i)\big)$
7:      if $\mathsf{cost}(S, \sigma) > (1 + \epsilon') \cdot \mathsf{last}$ then
8:          $(S, \sigma) \leftarrow \mathsf{FL\text{-}iterate}(M)$
9:          if $\mathsf{cost}(S, \sigma) > \mathsf{last}$ then $\mathsf{last} \leftarrow \mathsf{cost}(S, \sigma)$
10:         if $\mathsf{last} > \mathsf{init}/\epsilon'$ then terminate the stage
```

Notice that in a stage, we are considering the original costs of solutions (instead of scaled costs as inside FL-iterate). During a stage we maintain a value last which gives an estimation on the cost of the current solution (S, σ). Whenever a client j_t comes, we apply some rules to open some facilities and connect j_t (Steps 4 to 6). These operations are needed to make the cost increase due to the arrival of j_t (defined as Δ_t later) small. In the algorithm try-open′ is the same as try-open, except that we use the original cost instead of the scaled cost (this is not important but only for the sake of convenience). If $\mathsf{cost}(S, \sigma)$ becomes too large, i.e, $\mathsf{cost}(S, \sigma) > (1 + \epsilon')\mathsf{last}$, then we call $(S, \sigma) \leftarrow \mathsf{FL\text{-}iterate}(M)$ for the M defined in Step 1 (Step 8), and update last to $\mathsf{cost}(S, \sigma)$ if we have $\mathsf{cost}(S, \sigma) > \mathsf{last}$ (Step 9). We terminate the algorithm when $\mathsf{last} \ge \mathsf{init}/\epsilon$, where init is $\mathsf{cost}(S, \sigma)$ at the beginning of the stage (Step 10).

We say an execution of $\mathsf{FL\text{-}iterate}(M)$ is successful if the event in Lemma 10 happens. Then we have

▶ **Lemma 11.** *If all executions of* $\mathsf{FL\text{-}iterate}$ *are successful, the solution* (S, σ) *at the end of each time is* $(1+\epsilon')(\alpha_{\mathsf{FL}}+\epsilon')$*-approximate.*

Now we argue each execution of $\mathsf{FL\text{-}iterate}(M)$ is successful with probability at least $1 - 1/\Gamma$. This will happen if (S, σ) is $O(1)$-approximate before the call. By Lemma 10, we only need to make sure that the (S, σ) before the execution is $O(1)$-approximate. This is easy to see: Before Step 6 in time t, we have $\mathsf{cost}(S, \sigma) \leq O(1)\mathsf{opt}$; the increase of $\mathsf{cost}(S, \sigma)$ in the step is at most the value of opt after the step (i.e, we consider the client j_t when defining opt). Thus, we have $\mathsf{cost}(S, \sigma) \leq O(1)\mathsf{opt}$ after the step.

5.1 Bounding Number of Times of Calling FL-iterate

It remains to bound the number of times we call $\mathsf{FL\text{-}iterate}$. Again, we use T to denote the last time step of Algorithm 10 (i.e, one stage of the dynamic algorithm) and Δ_t to denote the cost increase due to the arrival of j_t: it is the value of $\mathsf{cost}(S, \sigma)$ before Step 6 minus that after Step 6 in time t. For every time $t \in [T]$, let C_t be the set C at the end of time t, and let opt_t be the cost of the optimum solution for C_t. Let last_t be the value of last at the *beginning* of time t.

Like in the online setting, we can bound the total cost increase in a similar manner as Lemma 8:

▶ **Lemma 12.** *For every* $T' \in [T-1]$*, we have*

$$\sum_{t=1}^{T'} \Delta_t \leq O(\log T') \cdot \mathsf{opt}_{T'}$$

Then we can wrap up the proof of Theorem 2. Between two consecutive calls of $\mathsf{FL\text{-}iterate}$ in Step 8 at time t_1 and $t_2 > t_1$, $\mathsf{cost}(S, \sigma)$ should have increased by at least $\epsilon'\mathsf{last}_{t_2}$: At the end of time t_1, we have $\mathsf{cost}(S, \sigma) \leq \mathsf{last}_{t_1+1} = \mathsf{last}_{t_2}$ since otherwise last should have been updated in time t_1. We need to have $\mathsf{cost}(S, \sigma) > (1+\epsilon')\mathsf{last}_{t_2}$ after Step 6 at time t_2 in order to call $\mathsf{FL\text{-}iterate}$. Thus, the increase of the cost during this period is at least $\epsilon'\mathsf{last}_{t_2}$. Thus, we have $\sum_{t=t_1+1}^{t_2} \frac{\Delta_t}{\epsilon' \cdot \mathsf{last}_t} \geq 1$ since $\mathsf{last}_t = \mathsf{last}_{t_2}$ for every $t \in (t_1, t_2]$. The argument also holds when $t_1 = 0$ and $t_2 > t_1$ is the first time in which we call $\mathsf{FL\text{-}iterate}$. Counting the call of $\mathsf{FL\text{-}iterate}$ in Step 2, we can bound the total number of times we call the procedure by $1 + \frac{1}{\epsilon'}\sum_{t=1}^{T} \frac{\Delta_t}{\mathsf{last}_t}$.

Again let $\Phi_{T'} = \sum_{t=1}^{T'} \Delta_t$ for every $T' \in [0, T]$. Lemma 12 says $\Phi_t \leq O(\log t)\mathsf{opt}_t$ for every $t \in [0, T-1]$. For every $t \in [T]$, since $\Delta_t \leq \mathsf{opt}_t$, thus we have $\Phi_t = \Phi_{t-1} + \Delta_t \leq O(\log t)\mathsf{opt}_{t-1} \leq O(\log T)\mathsf{last}_t$ since last_t will be at least the cost of some solution for C_{t-1}. Applying Lemma 7 with $a_t = \mathsf{last}_t, b_t = \Delta_t$ and $B_t = \Phi_t$ for every t, the number of times we call $\mathsf{FL\text{-}iterate}$ can be bounded by

$$1 + \frac{1}{\epsilon'}\sum_{t=1}^{T} \frac{\Delta_t}{\mathsf{last}_t} \leq \frac{1}{\epsilon'} O(\log T)\left(\ln \frac{\mathsf{last}_T}{\mathsf{last}_1} + 1\right) = O\left(\frac{\log T}{\epsilon} \log \frac{1}{\epsilon}\right).$$

We can then analyze the running time and the success probability of our algorithm. Focus on each stage of the algorithm. By Observation 9, each call to $\mathsf{FL\text{-}iterate}(M)$ takes time $O(M|C|\log|F|) = O\left(\frac{|F|}{\epsilon'}(\log \Gamma)|C|\log n\right) = O\left(\frac{n \cdot |C_T|}{\epsilon}\log^2 n\right)$, where C is the set of clients in the algorithm at the time we call the procedure, $C_T \supseteq C$ is the number set of clients at the end of time T, and $M = O\left(\frac{|F|}{\epsilon'}\log\Gamma\right)$ is as defined in Step 1. The total number of times we call the procedure is at most $O\left(\frac{\log T}{\epsilon}\log\frac{1}{\epsilon}\right) \leq O\left(\frac{\log n}{\epsilon}\log\frac{1}{\epsilon}\right)$. Thus, the running time we spent on $\mathsf{FL\text{-}iterate}$ is $O\left(\frac{n\cdot|C_T|}{\epsilon^2}\log^3 n\log\frac{1}{\epsilon}\right)$. The running time for Steps 4 to 6 is at most $T\cdot O\big(\log\frac{|F|}{\epsilon'}\big)\cdot O\big(|C_T|\log|F|\big) = O(|C_T|T\log^2\frac{|F|}{\epsilon}) \leq O(n|C_T|\log^2\frac{n}{\epsilon})$. Thus, the total running time of a stage is at most $O\left(\frac{n\cdot|C_T|}{\epsilon^2}\log^3 n\log\frac{1}{\epsilon}\right)$. Now consider all the stages together. The sum of $|C_T|$ values over all stages is at most $2n$ since every client appears in at most 2 stages. So, the total running time of our algorithm is $O\left(\frac{n^2}{\epsilon^2}\log^3 n\log\frac{1}{\epsilon}\right)$.

For the success probability, we call $\mathsf{FL\text{-}iterate}(M)$ at most $O\left(\log_{1/\epsilon}(nD)\frac{\log n}{\epsilon}\log\frac{1}{\epsilon}\right)$ $= \mathrm{poly}(\log n, \log D, \frac{1}{\epsilon})$ times. If we have Γ be at least n^2 times this number, which is still $\mathrm{poly}(n, \log D, \frac{1}{\epsilon})$, then the success probability of our algorithm is at least $1 - 1/n^2$.

Finally, we remark that the success of the algorithm only depends on the success of all executions of $\mathsf{FL\text{-}iterate}$. Each execution has success probability $1 - 1/\Gamma$ even if the adversary is adaptive. This finishes the proof of Theorem 2.

▶ Remark. We can indeed obtain an algorithm that has both $O(\log T)$ amortized client recourse and $\tilde{O}(n^2)$ running time, by defining $\phi = \frac{\mathsf{cost}(S,\sigma)}{\alpha_{\mathsf{FL}}\epsilon'}$ and only performing ϕ-efficient local operations. However, this will require us to put ϕ everywhere in our analysis and deteriorate the cleanness of the analysis. Thus, we choose to separate the two features in two algorithms: small recourse and $\tilde{O}(n^2)$ total running time.

We also remark that the total running time for all calls of $\mathsf{FL\text{-}iterate}$ is only $\tilde{O}(n|F|)$, and the $\tilde{O}(n^2)$ time comes from Steps 4 to 6. By losing a multiplicative factor of 2 and additive factor of 1 in the approximation ratio, we can assume every client is collocated with its nearest facility. Then at any time we only have $O(|F|)$ different positions for clients, and the running time of the algorithm can be improved to $O(\frac{n|F|}{\epsilon^2}\log^3 n\log\frac{1}{\epsilon})$.

6 Open Problems and Discussions

We initiated the study of facility location problem in general metric spaces in recourse and dynamic models. Several interesting problems remain open: The most obvious one is can we get $O(1)$-competitive online/dynamic algorithms with polylog amortized recourse or fast update times in the fully dynamic setting. Another interesting direction is can we extend our results to the capacitated facility location and capacitated k-median, where there is an upper bound on the number of clients that can be assigned to a single open facility. From technical point of view, it would be interesting to find more applications of local search and probabilistic tree embedding techniques in the dynamic algorithms model. Finally, as alluded in the introduction, an exciting research direction is to understand the power of recourse in the online model.

References

1 Aris Anagnostopoulos, Russell Bent, Eli Upfal, and Pascal Van Hentenryck. A simple and deterministic competitive algorithm for online facility location. *Inf. Comput.*, 194(2):175–202, November 2004. doi:10.1016/j.ic.2004.06.002.

2 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristic for k-median and facility location problems. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 21–29, 2001. doi:10.1145/380752.380755.

3 Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, FOCS '96, pages 184–, Washington, DC, USA, 1996. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=874062.875536.

4 Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 711–719, 1997. doi:10.1145/258533.258667.

5 Guy E. Blelloch, Yan Gu, and Yihan Sun. Efficient construction of probabilistic tree embeddings. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 26:1–26:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.26.

6 Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 3–16, 2018. doi:10.1145/3188745.3188798.

7 T-H. Hubert Chan, Arnaud Guerqin, and Mauro Sozio. Fully dynamic k-center clustering. In *Proceedings of the 2018 World Wide Web Conference*, WWW ?18, page 579?587, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. doi:10.1145/3178876.3186124.

8 Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for facility location problems. *SIAM J. Comput.*, 34(4):803–824, April 2005. doi:10.1137/S0097539701398594.

9 Vincent Cohen-Addad, Niklas Hjuler, Nikos Parotsidis, David Saulpic, and Chris Schwiegelshohn. Fully dynamic consistent facility location. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 3250–3260, 2019. URL: http://papers.nips.cc/paper/8588-fully-dynamic-consistent-facility-location.

10 Marek Cygan, Artur Czumaj, Marcin Mucha, and Piotr Sankowski. Online facility location with deletions. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 21:1–21:15, 2018. doi:10.4230/LIPIcs.ESA.2018.21.

11 Yunus Esencayi, Marco Gaboardi, Shi Li, and Di Wang. Facility location problem in differential privacy model revisited. *CoRR*, abs/1910.12050, 2019. arXiv:1910.12050.

12 Jittat Fakcharoenphol, Satish Rao, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 448–455, New York, NY, USA, 2003. ACM. doi:10.1145/780542.780608.

13 Dimitris Fotakis. Incremental algorithms for facility location and k-median. *Theor. Comput. Sci.*, 361(2-3):275–313, 2006. doi:10.1016/j.tcs.2006.05.015.

14 Dimitris Fotakis. A primal-dual algorithm for online non-uniform facility location. *J. of Discrete Algorithms*, 5(1):141–148, March 2007. doi:10.1016/j.jda.2006.03.001.

15 Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008. doi:10.1007/s00453-007-9049-y.

16 Dimitris Fotakis. Memoryless facility location in one pass. *ACM Trans. Algorithms*, 7(4):49:1–49:24, 2011. doi:10.1145/2000807.2000817.

17 Dimitris Fotakis. Online and incremental algorithms for facility location. *SIGACT News*, 42(1):97–131, March 2011. doi:10.1145/1959045.1959065.

18 Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil R. Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel C. Kilper. Projector: Agile reconfigurable data center interconnect. In *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*, pages 216–229, 2016. doi:10.1145/2934872.2934911.

19 Gramoz Goranci, Monika Henzinger, and Dariusz Leniowski. A tree structure for dynamic facility location. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 39:1–39:13, 2018. doi:10.4230/LIPIcs.ESA.2018.39.

20 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 649–657, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=314613.315037.

21 Monika Henzinger, Dariusz Leniowski, and Claire Mathieu. Dynamic clustering to minimize the sum of radii. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 48:1–48:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.48.

22 Silvio Lattanzi and Sergei Vassilvitskii. Consistent k-clustering. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1975–1984. PMLR, 2017. URL: http://proceedings.mlr.press/v70/lattanzi17a.html.

23 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Inf. Comput.*, 222:45–58, 2013.

24 A. Meyerson. Online facility location. In *Proceedings of the 42Nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, pages 426–, Washington, DC, USA, 2001. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=874063.875567.

25 Kamesh Munagala. Local search for k-medians and facility location. In *Encyclopedia of Algorithms*, pages 1139–1143. Springer, 2016. doi:10.1007/978-1-4939-2864-4_212.

26 Seeun Umboh. Online network design algorithms via hierarchical decompositions. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1373–1387, 2015. doi:10.1137/1.9781611973730.91.

27 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.

A Fully Dynamic Algorithm for Facility Location on Hierarchically Well Separated Tree Metrics

In this section, we give our fully dynamic algorithm for facility location on hierarchically-well-separated-tree (HST) metrics. Our algorithm achieves $O(1)$-approximation and $O(\log^2 D)$ amortized update time.

A.1 Hierarchically Well Separated Trees

▶ **Definition 13.** *A hierarchically-well-separated tree (or HST for short) is an edge-weighted rooted tree with the following properties:*

- *all the root-to-leaf paths have the same number of edges,*
- *if we define the level of vertex v, $\mathsf{level}(v)$, to be the number of edges in a path from v to any of its leaf descendant, then for an non-root vertex v, the weight of the edge between v and its parent is exactly $2^{\mathsf{level}(v)}$.*

Given a HST T with the set of leaves being X, we use d_T to denote the shortest path metric of the tree T (with respect to the edge weights) restricted to X.

The classic results by Bartal [3] and Fakcharoenphol, Rao and Talwar [12] state that we can embed any N-point metric (X, d) (with minimum non-zero distance being 1) to a distribution π of *expanding*[3] HST metrics (X, d_T) with distortion $O(\log N)$: For every $u, v \in X$, we have $d_T(u, v) \geq d(u, v)$ and $\mathbb{E}_{u,v}[d_T(u, v)] \leq O(\log N)d(u, v)$. Moreover, there is an efficient randomized algorithm [5] that outputs a sample of the tree T from π. Thus applying standard arguments, Theorem 3 implies Theorem 4.

Notations. As we mentioned early, we assume each client is collocated with a facility. From now on, we fix the HST T and assume the leaves of T is $X = F$; let V be the set of all nodes in T. Let d_T be the metric induced by T over the set V of vertices. Recall that $\mathsf{level}(v)$ is the level of v in T. For every vertex $v \in V$, define Λ_v to be the set of children of v, X_v to be the set of leaf descendants of v, and T_v be the maximal sub-tree of T rooted at v. We extend the facility cost from X to all vertices in V: for every $v \in V \setminus X$, we define $f_v = \min_{i \in X_v} f_i$. We can assume that each internal vertex v is a facility; by opening v we mean opening a copy of the $i \in X_v$ with $f_i = f_v$. This assumption only loses a factor of 2 in the competitive ratio: On one hand, having more facilities can only make our problem easier; on the other hand, the cost of connecting a client to any $i \in X_v$ is at most twice that of connecting it to v. By the definition, the facility costs along a root-to-leaf path are non-decreasing.

A.2 Offline Algorithm for Facility Location on HST Metrics

In this section, we first give an offline $O(1)$-approximation algorithm for facility location on the HST metric d_T as a baseline. Notice that facility location on trees can be solved exactly using dynamic programming. However the algorithm is hard to analyze in the dynamic algorithm model since the solution is sensitive to client arrivals and departures. Our algorithm generalizes the algorithm in [11] for facility location with uniform facility cost, that was used to achieve the differential privacy requirement.

For every vertex $v \in V$, we let N_v be the number of clients at locations in X_v. Although according to the definition N_v's are integers, in most part of the analysis we assume there are non-negative *real numbers*. This will be useful when we design the dynamic algorithm. Let $\alpha \in \{1, 2\}^V$ and $\beta \in \{1, 2\}^{V \setminus X}$ be vectors given to our algorithm. They are introduced solely for the purpose of extending the algorithm to the dynamic setting; for the offline algorithm we can set α and β to be all-1 vectors.

Marked and Open Facilities. For every vertex $v \in V$, we say v is *marked* w.r.t the vectors N and α if

$$N_v \cdot 2^{\mathsf{level}(v)} > f_v / \alpha_v$$

and *unmarked* otherwise. The following observation can be made:

▶ **Observation 14.** *Let u be the parent of v. If v is marked w.r.t N and α, so is u.*

Proof. v is marked w.r.t N and α implies $N_v 2^{\mathsf{level}(v)} > f_v/\alpha_v$. Notice that $N_u \geq N_v$, $\mathsf{level}(u) = \mathsf{level}(v) + 1$, $\alpha_v \leq 2\alpha_u$ and $f_u \leq f_v$. So, $N_u 2^{\mathsf{level}(u)} \geq 2N_v 2^{\mathsf{level}(v)} > 2f_v/\alpha_v \geq f_u/\alpha_u$. ◀

Thus there is a monotonicity property on the marking status of vertices in T. We say a vertex v is highest unmarked (w.r.t N and α) if it is unmarked and its parent is marked; we say a vertex v is lowest marked if it is marked but all its children are unmarked. However,

[3] A metric (X, d_T) is expanding w.r.t (X, d) if for every $u, v \in X$, we have $d_T(u, v) \geq d(u, v)$.

sometimes we say a vertex u is the lowest marked ancestor of a leaf $v \in X$ if either $u = v$ is marked, or $u \neq v$ is marked and the child of u in the u-v path is unmarked; notice that in this case, u might not be a lowest marked vertex since it may have some other marked children. If we need to distinguish between the two cases, we shall use that u is lowest marked *globally* to mean u is a lowest marked vertex.

If a leaf vertex $v \in X$ is marked, then we open v. For every marked vertex $v \in V \setminus X$, we open v if and only if

$$\left(\sum_{u \in \Lambda_v : u \text{ unmarked}} N_u\right) 2^{\mathsf{level}(v)} > f_v/(\alpha_v \beta_v).$$

Notice that all unmarked vertices are closed.

▶ **Observation 15.** *If v is lowest marked, then v is open.*

Proof. We can assume $v \notin X$ since otherwise v is open. So, $N_v 2^{\mathsf{level}(v)} > f_v/\alpha_v$ and all children of v are unmarked. Thus, $\sum_{u \in \Lambda_v : u \text{ unmarked}} N_u = \sum_{u \in \Lambda_v} N_u = N_v$. Therefore, $\left(\sum_{u \in \Lambda_v : u \text{ unmarked}} N_u\right) 2^{\mathsf{level}(v)} = N_v 2^{\mathsf{level}(v)} > f_v/\alpha_v \geq f_v/(\alpha_v \beta_v)$. Thus v will be open. ◀

With the set of open facilities defined, every client is connected to its nearest open facility according to d_T, using a consistent tie-breaking rule (e.g, the nearest open facility with the smallest index). We assume the root r of T has $\frac{f_v}{2^{\mathsf{level}(v)}} \leq 1$ by increasing the number of levels. So r will be marked whenever $N_r \geq 1$. This finishes the description of the offline algorithm.

Analysis of $O(1)$-Approximation Ratio. We show the algorithm achieves an $O(1)$-approximation. First we give a lower bound on the optimum cost. For every $v \in V$, let

$$\mathsf{LB}(v) = \min\left\{N_v 2^{\mathsf{level}(v)}, f_v\right\}.$$

Then we have

▶ **Lemma 16.** *Let U be a set of vertices in T without an ancestor-descendant pair; i.e, for every two distinct vertex u and v in U, u is not an ancestor of v. Then the cost of the optimum solution is at least $\sum_{v \in U} \mathsf{LB}(v)$.*

Then let U be the set of highest unmarked vertices and marked leaves; clearly U does not have an ancestor-descendant pair. By Lemma 16, the optimum cost is at least $\sum_{v \in U} \mathsf{LB}(v)$. We prove the following lemma.

▶ **Lemma 17.** *The solution produced by our algorithm has cost at most $O(1) \sum_{u \in U} \mathsf{LB}(u)$.*

Combining Lemmas 16 and 17 gives that our algorithm is an $O(1)$-approximation. One lemma that will be useful in the analysis of dynamic algorithm is the following:

▶ **Lemma 18.** *For any open facility v in our solution, the number of clients connected to v that are outside T_v is at most $O(\log D)\frac{f_v}{2^{\mathsf{level}(v)}}$.*

▶ Remark. The algorithm so far gives a *data structure* that supports the following operations in $O(\log D)$ time: i) updating N_v for some $v \in X$ and ii) returning the nearest open facility of a leaf $v \in X$. Indeed the algorithm can be made simpler: We set α to be the all-1 vector, and we open the set of lowest marked facilities (so both α and β are not needed). For every vertex $u \in V$, we maintain the nearest open facility ψ_u to u in T_u. Whenever a client at v arrives or departs, we only need change N_u, ψ_u, marking and opening status of u for ancestors u of v. To return the closest open facility to a leaf $v \in X$, we travel up the tree from v until we find an ancestor u with ψ_u defined, and return ψ_u. Both operations take $O(\log D)$ time. However, our goal is to maintain the solution (S, σ) *explicitly* in memory. Thus we also have to bound the the number of reconnections during the algorithm, since that will be a lower bound on the total running time.

A.3 Dynamic Algorithm for Facility Location on HST Metrics

In this section, we extend the offline algorithm to a dynamic algorithm with $O(\log^3 D)$-amortized update time; recall that D is the aspect ratio of the metric. We maintain α, β and N-vectors, and at any moment of the algorithm, the marking and opening status of vertices are exactly the same as that obtained from the offline algorithm for α, β and N.

Initially, let α and β be all-1 vectors, and N be the all-0 vector. So all the vertices are unmarked. Whenever a client at some $v \in X$ arrives or departs, the α, β values, the marking and opening status of ancestors of v may change and we show how to handle the changes. The vertices that are not ancestors of v are not affected during the process.

When a client at v arrives or departs, we increase or decrease the N_u values for all ancestors u of v by 1 *continuously* at the same rate (we can think of that the number of clients at v increases or decreases by 1 continuously). During the process, the marking and opening status of these vertices may change. If such an event happens, we change α and/or β values of the vertex so that it becomes harder for the status to change back in the future. Specifically, we use the following rules:

- If a vertex u changes to marked (from being unmarked), then we change α_u to 2 (notice that u remains marked w.r.t the new α), and β_u to 1. In this case, we do not consider the opening status change of u as an event.
- If a vertex u changes to unmarked (from being marked), we change α_u to 1 (notice that u remains unmarked w.r.t the new α). The β_u value becomes useless. In this case, we also do not consider the opening status change of u as an event.
- If a marked vertex u becomes open (from being closed), then we change β_u to 2 (notice that u remains open w.r.t the new β).
- If a marked vertex u becomes closed (from being open), then we change β_u to 1 (notice that u remains closed w.r.t the new β).

We call the 4 types of events above as marking, unmarking, opening and closing events.

Now we talk about the order the events happen. When we increase N_u values of ancestors of v continuously, one of the following two events may happen:

- The highest unmarked ancestor u of v may become globally lowest marked, and this may *induce* a closing event for the parent w of u.
- The lowest marked ancestor u of v may become open.

Similarly, when we decrease N_u values of ancestors of v continuously, one of the following two events may happen:

- The lowest marked ancestor u of v may become unmarked (we must that u was lowest marked globally), and this may *induce* an opening event for the parent w of u.
- The lowest marked ancestor u of v may become closed.

Above, if two events happen at the same time, we handle an arbitrary event. Notice that after we handle the event, the conditions for the other event might not hold any more, in which case we do not handle it.

Once we have finished the process of increasing or decreasing N_u values by 1, the clients will be connected to their respective nearest open facilities, breaking ties using the consistent rule. A reconnection happens if a client is connected to a different facility.

Bounding Number of Reconnections. Now we analyze the reconnections made in the algorithm. When a client at $v \in X$ arrives or departs, at most $O(\log D)$ vertices u will have their N_u values changed by 1. We distribute 4 tokens to each ancestor u of v, that are of type-A, type-B, type-C and type-D respectively.[4] We are going to use these tokens to charge the events happened.

First focus on the sequence of marking/unmarking events happened at a vertex u. Right before u becomes unmarked we have $N_u \leq f_u/(2 \times 2^{\mathsf{level}(u)})$ since at the moment we have $\alpha_u = 2$. Immediate after that α_u is changed to 1. For u to become marked again, we need $N_u \leq f_u/2^{\mathsf{level}(u)}$. So during the period N_u must have been increased by at least $f_u/(2 \times 2^{\mathsf{level}(u)})$. Similarly, right before u becomes marked we have $N_u \geq f_u/2^{\mathsf{level}(u)}$ since at the moment we have $\alpha_u = 1$. Then we change α_u to 2 immediately. For u to become unmarked again, N_u should be decreased by at least $f_u/(2 \times 2^{\mathsf{level}(u)})$. So, when a marking/unmarking event happens at u, we can spend $\Omega(f_u/2^{\mathsf{level}(u)})$ type-A tokens owned by u.

Then we focus on the sequence $\mathcal{S}$ of opening/closing events at u between two adjacent marking/unmarking events at u. At these moments, u is marked and $\alpha_u = 2$. For the first event in $\mathcal{S}$, we can spend $\Omega(f_u/2^{\mathsf{level}(u)})$ type-B tokens owned by u. If some opening/closing event e in $\mathcal{S}$ is induced by an unmarking/marking event of some child u' of u, then we can spend $\Omega(f_{u'}/2^{\mathsf{level}(u')}) \geq \Omega(f_u/2^{\mathsf{level}(u)})$ type-C tokens owned by u' for e, and the event e' after e in $\mathcal{S}$ if it exists. Notice that we already argued that u' has collected enough number of type-C tokens.

Then we focus on an event e' in $\mathcal{S}$ such that both e and the event e before e' in $\mathcal{S}$ are not induced. First, assume e is an opening event and e' is a closing event. Then, after e we have $\sum_{u' \in \Lambda_u : u' \text{ unmarked}} N_{u'} = f_u/(2 \times 2^{\mathsf{level}(u)})$ and before e' we have $\sum_{u' \in \Lambda_u : u' \text{ unmarked}} N_{u'} = f_u/(4 \times 2^{\mathsf{level}(u)})$. Notice that the set of unmarked children of u may change, and let U' and U'' be the sets of unmarked children of u at the moments after e and before e' respectively. Again if there is some $u' \in (U' \setminus U'') \cup (U'' \setminus U')$, we spend $\Omega(\frac{f_{u'}}{2^{\mathsf{level}(u')}}) \geq \Omega(\frac{f_u}{2^{\mathsf{level}(u)}})$ type-C tokens owned by u'. Otherwise, $U = U'$ and $f_u/(4 \times 2^{(\mathsf{level}(u))})$ clients in T_u must have departed between e and e' and we can then spend $\Omega(f_u/2^{\mathsf{level}(u)})$ type-D tokens for e'. The case when e is an closing event and e' is an opening event can be argued in the same way.

Thus, whenever an event happens at u, we can spend $\Omega(f_u/2^{\mathsf{level}(u)})$ tokens; moreover if an opening/closing event at u was induced by an unmarking/marking event at some child u' of u, then we can spend $\Omega(f_{u'}/2^{\mathsf{level}(u')})$ tokens for the event at u. A facility u changes its opening status when an event happens at u. Notice that, we reconnect a client only if it was connected to a ready-to-close facility, or it needs to be connected to newly open facility. By Lemma 18, at any moment the number of clients connected to u from outside T_u is at most $O(\log D) \cdot \frac{f_u}{2^{\mathsf{level}(u)}}$. At the moment u changes its opening status because of an non-induced event, then before and after the event the number of clients connected to u from T_u is of order $O\left(\frac{f_u}{2^{\mathsf{level}(u)}}\right)$. u changes its opening status due to a marking/unmarking event happened at some child u' of u, then before and after the event the number of clients connected to u from T_u is of order $\Theta\left(\frac{f_{u'}}{2^{\mathsf{level}(u')}}\right)$. Thus, on average, for each token we spent we connect at most $O(\log D)$ clients. Since each client arrival or departure distributes at most $O(\log D)$ tokens, we have that the amortized number of reconnections (per client arrival/departure) is at most $O(\log^2 D)$.

Analyzing Update Time. Then with the bound on the number of reconnections (recourse), we can bound the update time easily. Indeed, we can maintain a ψ_u for every $u \in V$, which indicates the nearest open facility to u in $T_u \setminus u$ (ψ_u could be undefined). We also maintain

[4] The types are only defined for convenience.

a value N'_u for marked vertices u where $N'_u = \sum_{v \in \Lambda_v, v \text{ unmarked}} N_v$. Whenever a client at v arrives or departs, we need to change $\alpha_u, \beta_u, N_u, N'_u, \psi_u$, marking and opening status of u only for ancestors u of v. The update can be made in $O(\log D)$ time for every client arrival or departure using the information on the vertices. The bottleneck of the algorithm comes from reconnecting clients. We already argued that the amortized number of reconnections per client arrival/departure is $O(\log^2 D)$ and thus it suffices to give an algorithm that can find the clients to be connected efficiently.

For every vertex u, we maintain a double-linked-list of unmarked children u' of u with $N_{u'} \geq 1$. With this structure it is easy to see that for every client that needs to be reconnected, we need $O(\log D)$ time to locate it. If u becomes open, we need to consider each unmarked children u' of u and reconnect clients in $T_{u'}$ to u. The time needed to locate these clients can be made $O(\log D)$ times the number of clients. For every strict ancestor w of u for which there are no open facilities in between we can use the ψ_w information to see if we need to reconnect clients in T_w. If yes, then for every unmarked child w' of w with $N_{w'} \geq 1$ that is not an ancestor of u, we need to connect the clients in $T_{w'}$ to u. Again enumerating these clients takes time $O(\log D)$ times the number of clients. Similarly, if u becomes closed, we then need to connect all clients connected to u to the nearest open facility to u, which can be computed using ψ values of u and its ancestors. Enumerating the clients takes time $O(\log D)$ times the number of clients. Overall, the amortized running time per client arrival/departure is $O(\log^3 D)$.