Online Graph Algorithms with Predictions

Yossi Azar* Tel Aviv University
Debmalya Panigrahi† Duke University
Noam Touitou Tel Aviv University

Abstract

Online algorithms with predictions is a popular and elegant framework for bypassing pessimistic lower bounds in competitive analysis. In this model, online algorithms are supplied with future *predictions* and the goal is for the competitive ratio to smoothly interpolate between the best offline and online bounds as a function of the prediction error.

In this paper, we study online graph problems with predictions. Our contributions are the following:

- The first question is defining prediction error. For graph/metric problems, there can be two types of error, locations that are not predicted, and locations that are predicted but the predicted and actual locations do not coincide exactly. We design a novel definition of prediction error called *metric error with outliers* to simultaneously capture both types of errors, which thereby generalizes previous definitions of error that only capture one of the two error types.
- We give a general framework for obtaining online algorithms with predictions that combines, in a "black box" fashion, existing online and offline algorithms, under certain technical conditions. To the best of our knowledge, this is the first general-purpose tool for obtaining online algorithms with predictions.
- Using our framework, we obtain tight bounds on the competitive ratio of several classical graph problems as a function of metric error with outliers: Steiner tree, Steiner forest, priority Steiner tree/forest, and uncapacitated/capacitated facility location.

Both the definition of metric error with outliers and the general framework for combining offline and online algorithms are not specific to the problems that we consider in this paper. We hope that these will be useful for future work on other problems in this domain.

1 Introduction

Online algorithms has been the paradigm of choice in algorithms research for handling future uncertainty in input data. But, they have often been criticized for being overly pessimistic, and their failure to distinguish between algorithms with widely different empirical performance in many contexts is well documented. To overcome this deficiency, recent research has proposed augmenting online algorithms with future *predictions* (e.g., from machine learning models), the goal being to design algorithms that gracefully degrade from the best offline bounds for accurate predictions to the best online bounds for arbitrary predictions. Indeed, a robust literature is beginning to emerge in this area with applications to caching [40, 51, 35, 56], rent or buy [49, 23, 2], scheduling [49, 38, 47, 29], online learning [17, 9], covering problems [7], frequency estimation [28], low rank approximation [31], metrical task systems [3], auction pricing [43], budgeted allocation [45], Bloom filters [46], etc. In this paper, we ask: *how do we design online graph algorithms with predictions?*

*Supported in part by the Israel Science Foundation (grant No. 2304/20).
†Supported in part by an NSF grants CCF-1750140 (CAREER Award) and CCF-1955703, and ARO grant W911NF2110230.

For example, consider a cable company that needs to set up a network that can serve clients in a certain area. They can obtain a rough prediction of their future client locations via, e.g., market surveys. This prediction is available upfront, but the actual clients join the service over time, and the network has to be built up as the clients join. A natural goal for the cable company is to use the predictions to mitigate uncertainty, and design an algorithm that degrades gracefully with prediction error.

Another example is a video conferencing platform that has to allocate bandwidth for client meetings. The platform can make predictions on client locations for meetings based on past data (e.g., just their last known location, or by using a more sophisticated machine learning model), but the actual set of users connect to the meeting online. Of course, the prediction could be wrong in terms of the users who actually join the meeting, or even for a correctly predicted user, might get her precise location wrong. Again, the goal would be to smoothly degrade in performance with these types of prediction error.

1.1 Our Contributions Metric Error with Outliers. The first question is how to model prediction error. For concreteness, think of the example above, i.e., making predictions on a geographical map; on such a metric space/graph, there are, conceptually, two types of errors.

- Some predictions might be entirely wrong; for such cases, the appropriate notion of error is the number of false predictions, or the number of unpredicted requests. This is the error typically implied when solving problems with outliers. However, this error notion is unrealistic for metric spaces: even when the prediction for a location is "correct", one cannot hope that it would *precisely* match the actual location on the underlying graph/metric space.
- Another notion of error, which is able to handle such slightly-perturbed predictions, is the minimum transportation cost of moving from the predicted input to the actual input. (This is the error previously used for metric problems in the prediction model.) The shortcoming of this error notion is that even a single acute misprediction could lead to unbounded transportation cost.

It is clear from this discussion that both these types of error are required to faithfully capture prediction (in)accuracy. We introduce a new robust notion of prediction error called *metric error with outliers* that subsumes both these types of errors. This notion of error applies to any problem involving predictions on a metric space; with the prediction model becoming very popular in online algorithms, we hope that this definition of error will be useful for other problems in the future.

For convenience, we define this error on a metric space; this is easily extended to a graph using the shortest path metric. Let $M = (V, d)$ be a metric space. Let $R \subseteq V$ be a subset of vertices and $\hat{R} \subseteq V$ be a prediction for R. (In general, $\hat{R}$ and R may have different sizes.) The *metric error with outliers* for $\hat{R}$ is given by $\lambda = (\Delta, D)$ w.r.t. R if there exist sets $T \subseteq R$ and $\hat{T} \subseteq \hat{R}$ satisfying $|T| = \left|\hat{T}\right|$ such that (a) the minimum cost matching between T and $\hat{T}$ on M has cost D, and (b) $|R \setminus T| + \left|\hat{R} \setminus \hat{T}\right| = \Delta$. Intuitively, the idea is to define the error as the minimum cost matching D between the predicted and actual locations, but allow *outliers* in both the predicted and actual sets that count toward a numerical error Δ. (See Figure 1 for a visualization.)

We remark that for a given prediction, there are multiple different values of Δ and D that fit this (Δ, D)-error definition. In particular, for a given D, to find the best Δ, simply find the maximum number of pairs of predicted and actual locations whose transportation cost is at most D; the total number of remaining predicted and actual locations is Δ. This produces a Pareto frontier of (Δ, D) pairs (see Fig. 1). Our theorems hold for **all** possible values of Δ and D. In particular, if we set $D = 0$

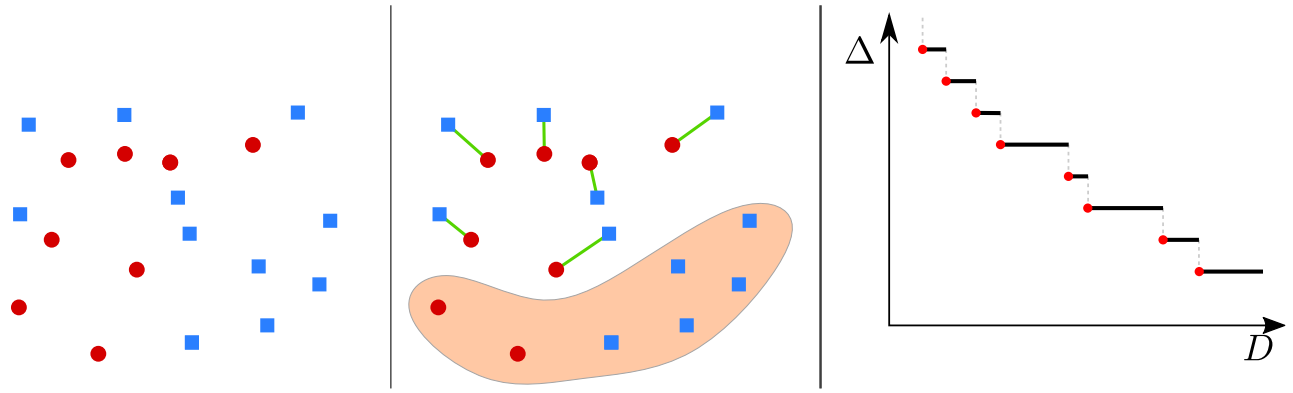

Figure 1: An illustration of metric error with outliers. The figure on the left shows a request set R (red circles) and a prediction $\hat{R}$ (blue squares). The figure in the middle shows a valid definition of metric error with outliers for $\hat{R}$ w.r.t. R. Here, $T \subset R$ are matched to $\hat{T} \subset \hat{R}$ using the green lines; D is the total length of the green lines. The outliers are shown in the orange shaded region; Δ is the number of points in this region.
The figure on the right shows the tradeoff between Δ and D in the error. The red points are the Pareto frontier formed by various choices of Δ and D.

and define error using only the numerical count of mispredicted locations, or set $\Delta = 0$ and define error using only metric distance between predicted and actual points, our theorems continue to hold for both cases. (These are the two ends of the Pareto frontier.)

A Framework for Combining Offline and Online Algorithms. The gold standard in online algorithms with predictions is to interpolate between online and offline bounds for a problem as a function of the prediction error. But, perhaps surprisingly, in spite of the recent surge of interest in this domain, there is still no general framework/methodology that can combine online and offline results in a black box fashion to achieve such interpolation. I.e., the existing results are individually tailored to meet the needs of the problem being considered, and therefore, do not use offline/online results in a black box manner. We make progress toward this goal of designing a general methodology in this paper by presenting a framework that requires the following ingredients:

- An *online* algorithm that satisfies a technical condition called *subset competitiveness*. We will describe this more precisely later, but intuitively, this means that the cost incurred by the algorithm on any subset of the input can be bounded against the optimal solution on that subset.
- An *offline* algorithm for the *prize-collecting* version of the problem, i.e., if one is allowed to not satisfy a constraint in lieu of paying a penalty for it.

We show that this framework achieves tight bounds for several classical problems in graph algorithms that we describe below. But, before describing these results, we note that the framework itself is quite general (i.e., does not use any specific property of problems defined on graphs), and uses the online and offline algorithms as black box results. Therefore, we hope that this framework will be useful for online problems with predictions in other domains as well.

Our Problems. We present online algorithms with predictions for some classic problems in graph algorithms using the above framework:

- *Steiner Tree* (e.g., [36, 58, 48, 50, 12]): In this problem, the goal is to obtain the minimum cost subgraph that connects a given set of vertices called *terminals*. In the online version, the terminals are revealed one at a time in each online step. The offline prediction provides the algorithm with a predicted set of terminals.

- *Steiner Forest* (e.g., [1, 22]): This is a generalization of the Steiner tree problem where the input comprises a set of vertex pairs called *terminal pairs* that have to be connected to each other (but not necessarily with other pairs). In the online version, the terminal pairs are revealed one at a time in each online step. Again, the offline prediction gives a predicted set of terminal pairs.
- *Facility Location* (e.g., [27, 53, 44, 24, 15, 37, 34, 13, 33, 32, 54, 42, 11, 39]): In this problem, a subset of vertices are designated as *facilities* and each of them is given an opening cost. Now, a subset of vertices are identified as *clients*, and each client must connect to an *open* facility and pay the shortest path distance to that vertex as connection cost. The open facilities also pay their corresponding opening costs. The goal is to minimize the total cost. In the online version, the clients are revealed one at a time in each online step. The offline prediction comprises a set of predicted clients.
- *Capacitated Facility Location* (e.g., [34, 41]): This is the same as facility location, except that each facility also has a maximum capacity for the number of clients that can connect to it. We consider the *soft* capacitated version of the problem, i.e., where multiple copies of a facility can be opened but each opened copy incurs the opening cost of the facility.

Our Results. Our main result is the following:

THEOREM 1.1. *Consider the Steiner tree, Steiner forest, and (capacitated) facility location problems. Suppose R and $\hat{R}$ are respectively the actual and the predicted inputs, and let* OPT *be the optimal solution for R. Then, for* **any** *metric error with outliers (Δ, D), we present (for each problem) an algorithm* ALG *with the following guarantee:*

$$\text{ALG} \leq O(\log \Delta) \cdot \text{OPT} + O(D). \tag{1.1}$$

Universality: As shown in Figure 1, the multiple choices for the error (Δ, D) create a Pareto frontier of choices which are non-dominated. The algorithms presented in this paper achieve the bounds in Theorem 1.1 for *every error* (Δ, D) *simultaneously*, and thus remain oblivious to the chosen error parameters.

Scale-freeness: The key property of these theorems is that the competitive ratio does not scale with the size of the input as long as the input is predicted correctly (or with small D). Conceptually, this is exactly what predictions should achieve in an online algorithm, that the competitive ratio should only be a function of the part of the input that was not predicted correctly.

Tightness of Bounds: The competitive ratios in Theorem 1.1 match the tight competitive ratios for the corresponding online problems, i.e., for $\hat{R} = \emptyset$. Hence, these bounds are also tight. Indeed, we show in the appendix that the online lower bound constructions can be implemented even if there is a prediction of *any* size by creating a copy of the lower bound instance inside the erroneous predictions.

Impossibility of Black Box Theorem: Given our theorems, one might wonder if it is *always* possible to replace the number of requests in an online graph algorithm with the number of prediction errors in the competitive ratio, i.e., if there is a *universal* theorem to this effect. Unfortunately, this is not true. For instance, in the bipartite matching problem even on a uniform metric space on n points, it is easy to see that even a single prediction error can lead to a competitive ratio of $\Omega(\log n)$. We give this example in the appendix.

Extension to Priority Steiner tree/forest: In Appendix B, we extend our results to *priority* Steiner tree/forest (e.g. [14, 16]). This is a generalization of Steiner tree/forest where every edge and every

terminal/terminal pair has a *priority*, a terminal/terminal pair can only use edges of its priority or higher to connect. In the online version, the terminals/terminal pairs and their priorities are revealed online. The offline prediction comprises a set of predicted terminals/terminal pairs with their respective priorities. In the video conferencing and network design applications, this reflects multiple *service level agreements* (SLAs) with different users.

We give an algorithm ALG with the following guarantee:

$$\text{ALG} \leq O(b \cdot \log(\Delta/b)) \cdot \text{OPT} + O(D), \tag{1.2}$$

where b is the number of priority classes in the input.

2 A General Framework for Online Algorithms with Predictions

In this section, we give our general framework for online algorithms with prediction. Abstractly, we are given a universe $\mathcal{S}$ offline, where each element has an associated cost $c : \mathcal{S} \to \mathbb{R}^+$. A set of requests R arrives online, one request per step, and the online algorithm must augment its solution $S \subseteq \mathcal{S}$ by adding new elements from $\mathcal{S}$ such that the new request is satisfied. We assume that for each request $r \in R$, the collection of sets satisfying r is *upward-closed*, i.e., if S satisfies r, so too does any $S' \supseteq S$. The algorithm is also given an offline prediction $\hat{R}$ for the set of requests.

For every set $S \subseteq \mathcal{S}$ of elements, we define $c(S) := \sum_{e \in S} c(e)$. Our framework requires the following ingredients:

PROPERTY 2.1. *An online algorithm* ON *whose competitive ratio is* $f(|R|)$. *This algorithm must be* subset-competitive*; that is, for every request subset* $R' \subseteq R$ *we have*

$$\text{ON}(R') \leq O(f(|R'|)) \cdot \text{OPT}$$

where $\text{ON}(R')$ *denotes the total cost incurred by the algorithm in serving the requests of* R'.

PROPERTY 2.2. *A (constant)* γ*-approximation algorithm* PC *for the prize-collecting offline problem. In this problem, each request in* R *has a penalty given by* $\pi : R \to \mathbb{R}^+$. *A solution is a subset of elements* $S \subseteq \mathcal{S}$ *that minimizes* $c(S) + \sum_{r \in R'} \pi(r)$, *where* $R' \subseteq R$ *is the set of requests* not *satisfied by* S.

Given these two ingredients, we define our framework in Algorithm 2.1. Before describing the framework formally, let us give some intuition for it. There are two extreme cases. If the prediction is entirely accurate, we should simply run an offline approximation for the predicted requests. On the other hand, if the prediction is entirely erroneous, we should run a standard online algorithm and ignore the prediction completely. Between these extremes, where the prediction is only partially accurate, some hybrid of those algorithms seems a natural choice. Thus, our framework runs a standard online algorithm on the actual requests, while also satisfying increasingly larger parts of the prediction by periodically running an offline algorithm to satisfy predicted requests. Ideally, the offline solutions should satisfy the correctly predicted requests, while the online solution should satisfy the requests that were not predicted. But, in general, this is impossible to achieve: the offline algorithm does not know in advance which of the predicted requests will arrive in the future. Our main technical work is to show the surprising property that it suffices for the offline algorithm to choose the most "cost-effective" part of the prediction to satisfy *even if those requests eventually do not realize in the actual input.*

ALGORITHM 2.1. General Framework for Online Algorithms with Predictions.

Input:
ON – An online algorithm for the problem
PC – a γ-approximation for the prize-collecting problem. Denote by $\mathrm{PC}(Q, \pi)$ the application of PC to the set of requests Q and the penalty function π.
$\hat{R}$ – a prediction of requests

procedure INITIALIZATION
  Initialize $B \leftarrow 0$, $\hat{B} \leftarrow 0$ and $S \leftarrow \emptyset$.
  Initialize ON to be a new instance of the online algorithm.
end procedure

procedure UPONREQUEST(r) ▷ *Upon the next request r*
  Send r to the online algorithm ON, and augment S accordingly. Increase B by the resulting cost incurred by ON.

  ▷ *Whenever the cost of the online algorithm doubles, buy another offline solution*
  if $B \geq 2\hat{B}$ **then**
    Set $\hat{B} \leftarrow B$
    Calculate $0 \leq u \leq |\hat{R}|$, the minimum number such that $c(\textsc{Partial}(\hat{R}, u)) \leq 3\gamma\hat{B}$.
    Augment S with the elements of $\textsc{Partial}(\hat{R}, u)$.
    Start a new instance of the online algorithm ON where $c(e) = 0$ for all $e \in S$.
  end if
end procedure

procedure PARTIAL($\hat{R}$,u)
  if $\gamma u \geq |\hat{R}|$ **then return** $\emptyset$
  For every x, define π_x to be the prize-collecting penalty function such that $\forall r : \pi_x(r) = x$.
  Let i be the minimum integer such that $\mathrm{PC}(\hat{R}, \pi_{2^i})$ does not satisfy $\leq \gamma u$ requests. [1]
  Let $S_1 = \mathrm{PC}(\hat{R}, \pi_{2^{i-1}})$ and $S_2 = \mathrm{PC}(\hat{R}, \pi_{2^i})$.
  Let u_1, u_2 be the number of requests from $\hat{R}$ which are not satisfied by S_1, S_2, respectively.
  if $\gamma u \geq \frac{u_1+u_2}{2}$ **then return** S_1 **else return** S_2
end procedure

Framework and Partial Procedure. A high-level, informal description of the framework is to perform the following whenever a request r is released:

1. Send r to be served by the online algorithm ON.
2. If the total cost B incurred in Step 1 (over all requests) has doubled, buy an offline solution which satisfies as many requests from the prediction $\hat{R}$ as possible, while not exceeding a budget of $O(B)$.

The framework thus runs the online algorithm ON on each incoming request, but periodically adds an (offline) solution for the prediction $\hat{R}$. This offline solution is computed by a subroutine that we call PARTIAL. The input to PARTIAL is the prediction $\hat{R}$ and an integer u such that $0 \leq u \leq |\hat{R}|$, and it attempts to output a minimum-cost solution that satisfies all requests from $\hat{R}$ **except** for at most u requests. If PARTIAL were able to solve this minimization problem exactly, or even up to a constant approximation, then the algorithm would add the solution returned by $\textsc{Partial}(\hat{R}, u)$ for the minimum

[1] This can be done by enumerating all values of i.

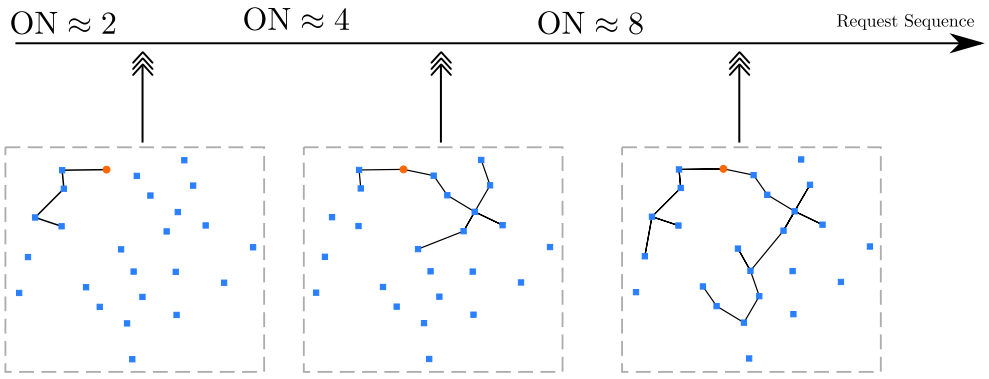

Figure 2: Visualization of Framework: As the request sequence advances, the requests are forwarded to an online algorithm. Each time the cost of this online algorithm doubles, the current solution is augmented with an offline solution to the prediction, which serves an increasingly-larger number of predicted requests.

value of u within the given budget. An example for a problem in which PARTIAL can output such a solution is Steiner tree, in which PARTIAL simply needs to solve the well-known k-MST problem, for which there are several constant approximation algorithms (e.g., [6, 10, 20, 5, 4, 21]).

However, for other problems, this may be infeasible – for Steiner forest, the approximability of the corresponding "k-minimum spanning forest" problem is known to be related to that of the k-densest subgraph problem, and hence a sub-polynomial approximation is unlikely [26, 52, 25].

Our key observation is that we can relax the guarantee that we seek from this minimization problem. Namely, we show that it suffices to obtain a *residual bi-criteria* constant approximation, which means that the algorithm satisfies the following two properties: (a) the cost of the solution is at most a constant times that of the optimal solution, and (b) the number of requests in $\hat{R}$ that are *not satisfied* by the algorithm is also at most a constant times that in the optimal solution. Surprisingly, this weaker guarantee allows us to entirely conceal the differences in approximability of the minimization problem being solved by PARTIAL. Indeed, we show that using a series of invocations of an offline prize-collecting algorithm PC in a completely *black box* manner, we can obtain an implementation of PARTIAL that provides the bicriteria guarantee that we need.

The formal algorithm for the framework is presented in Algorithm 2.1.

2.1 Framework Properties

The PARTIAL Subroutine Guarantee. Recall the desired property from the subroutine PARTIAL: given a prediction $\hat{R}$ and a number u (such that $0 \leq u \leq |\hat{R}|$) as input, compare the output S of PARTIAL$(\hat{R}, u)$ against the minimum-cost solution S^* which satisfies all requests from $\hat{R}$ except for u requests. This output S cannot fail to satisfy more than a constant factor more requests $\hat{R}$ than S^*; that is, it fails to satisfy only $O(u)$ requests (here, recall that γ is a constant). In addition, the cost of S can be at most a constant times the cost of S^*. This residual bi-criteria guarantee crucially uses Property 2.2, where the γ approximation ratio of the prize-collecting algorithm goes into both the cost of S and the amount of requests satisfied by S. This guarantee is formally stated in the following lemma (proof in Appendix A).

LEMMA 2.1. *The solution S returned by a call to* PARTIAL$(\hat{R}, u)$ *has the following properties:*

1. *S satisfies all requests from $\hat{R}$ except for at most $2\gamma u$ requests.*

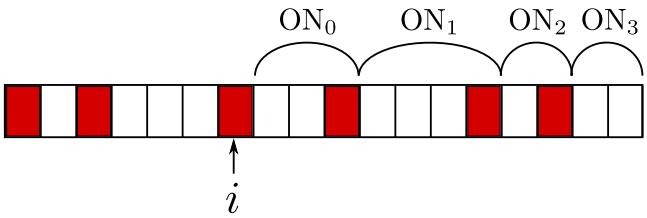

Figure 3: Phase Structure: This figure illustrates the iterations of the algorithm, where the major iterations are shown in red. Fixing any specific major iteration i induces a sequence of online phases $\mathrm{ON}_0, \cdots, \mathrm{ON}_m$.

2. *$c(S) \leq 3\gamma \cdot c(S^*)$, where S^* is the minimum cost solution satisfying all requests from $\hat{R}$ except for at most u requests.*

Bounding Cost of Framework. Next, we provide some bounds for various costs incurred by the framework. The following sections make use of these general bounds to prove competitiveness.

The requests of R are handled in $|R|$ iterations of the algorithm (calls to UponRequest). For the variable B in the algorithm, and any iteration j, we write B_j to refer to the value of the variable immediately after the j'th iteration of the algorithm (where B_0 is the initial value of B, namely 0). Similarly, we define $\hat{B}_j$ to be the value of the variable $\hat{B}$ after the j iteration, where $\hat{B}_0 = 0$.

If the **if** condition of Line 7 holds during iteration i, then i is called a *major iteration*. That is, major iterations are those iterations in which our solution is augmented by a solution for some predicted requests.

Fix a major iteration i, and observe the subsequent major iterations, denoted $i = i_0^\star, i_1^\star, ..., i_m^\star$ (where m is the number of such major iterations). For convenience, we define $i_{m+1}^\star = |R|$. For each index $j \in \{0, \cdots, m\}$, we define *phase* j to consist of iterations $\left\{i_j^\star + 1, \cdots, i_{j+1}^\star\right\}$. Observe that each phase uses a different instance of ON; we denote by ON_j the total cost of the ON instance of phase j.

The following lemma provides a bound on the cost of the algorithm (formal proof in Appendix A). It does so by dividing the costs of the algorithm into two parts, subject to the choice of any major iteration i: the first is the prefix cost, up to (and including) iteration i; the second is the suffix cost, from iteration $i+1$ onwards. Intuitively, a "correct" choice of i would be such that $\hat{B}_i \approx \mathrm{OPT}$; subject to such a choice of i, the first i iterations perform an exploratory doubling procedure, and thus the prefix cost would be $O(\mathrm{OPT})$. As for the suffix cost, we claim that once an offline solution of cost $\approx \mathrm{OPT}$ is added, the suffix cost cannot be large unless the prediction is rather inaccurate. These claims are made formal in the following sections; for now, we simply state and prove this prefix-suffix division of costs. Note that this lemma makes use of Property 2.1, which implies that serving no single request in the online algorithm is prohibitively expensive.

Lemma 2.2. *Fix any major iteration i. Denote the subsequent major iterations by $i = i_0^\star, i_1^\star, ..., i_m^\star$, and define phases and ON_j for any j as above (with respect to iteration i). The following holds:*

1. *The total cost of the first i iterations is at most $O(1) \cdot \mathrm{OPT} + (12\gamma + 2) \cdot \hat{B}_{i-1}$.*
2. *The total cost of the iterations $i+1, \cdots, |R|$ is at most $O(1) \cdot \max\{\mathrm{ON}_{m-1}, \mathrm{ON}_m\}$.*

The lemmas of this section reduce the cost analysis of the framework to bounding the terms $\mathrm{ON}_{m-1}, \mathrm{ON}_m$ and $\hat{B}_{i-1}$ for some major iteration i. This is done in the following sections, for each problem separately.

3 Online Steiner Tree with Predictions

In the Steiner tree problem, we are given an undirected graph $G = (V, E)$ with edge costs $c : E \to \mathbb{R}^+$. The set of requests $R = \{r_1, r_2, \ldots, r_{|R|}\}$ is a subset of vertices called *terminals*. The goal is to obtain a connected subset of edges S that spans all the terminals at minimum cost $\sum_{e \in S} c(e)$. In the *online* version, the terminals arrive online, one per step, and must be connected to a root vertex ρ given offline. (The rooted and unrooted versions are equivalent.) The offline prediction comprises a predicted set of terminals $\hat{R}$.

In this section, we apply the framework of Algorithm 2.1 using the two required components – an offline algorithm for prize-collecting Steiner tree and a subset-competitive online algorithm for Steiner tree.

Prize-collecting offline algorithm. Goemans and Williamson [22] gave a 2-approximation algorithm for the prize-collecting (unrooted) Steiner tree problem. Their algorithm can be trivially adapted for the rooted version of the problem by adding the root ρ to the terminals, and assigning it a penalty of ∞.

LEMMA 3.1. (GOEMANS AND WILLIAMSON [22]) *There exists a 2-approximation offline algorithm* PC^{ST} *for the rooted prize-collecting Steiner tree problem.*

Subset-competitive online algorithm. The greedy algorithm ON^{ST} for online Steiner tree [30] performs the following:

1. Upon the release of a new terminal r_i, let $v \in \{\rho, r_1, \cdots r_{i-1}\}$ be the closest terminal to r_i.
2. Buy the path $P_{r_i,v}$, the shortest path from r_i to v.

As shown in [30], this algorithm is $O(\log |R| + 2)$-competitive. The following lemma shows that it also upholds Property 2.1, i.e. it is subset competitive.

LEMMA 3.2. *The greedy algorithm* ON^{ST} *is subset-competitive, as in Property 2.1. That is, for an input* R *and every* $R' \subseteq R$*, it holds that*

$$\text{ON}^{\text{ST}}(R') \leq O\left(\log \left|R'\right| + 2\right) \cdot \text{OPT}$$

Proof. Denote by OPT' the optimal solution for the subset R'. We show that $\text{ON}^{\text{ST}}(R') \leq O\left(\log |R'| + 2\right) \cdot \text{OPT}'$, which implies the lemma since $\text{OPT}' \leq \text{OPT}$.

Note that for each terminal $r \in R'$, the cheapest path to a previous terminal in an instance with terminals R is at most as expensive as the cheapest path to a previous terminal in an instance with terminals $R' \subseteq R$. The lemma now follows from the competitive ratio of the instance with terminals R'. ∎

Steiner tree with predictions. The algorithm for Steiner tree with predictions is now constructed by simply plugging in ON^{ST} and PC^{ST} into the framework. Note that the solution S remains connected throughout the algorithm (specifically in Line 10 of Algorithm 2.1, both subgraphs whose union is taken contain the root vertex ρ).

We would now like to show that applying the framework to ON^{ST} and PC^{ST} yields a guarantee for metric errors with outliers. However, we must first define the matching cost of two requests, to complete the definition of errors in the Steiner tree problem. For Steiner tree, we use the natural choice that the

matching cost of two requests r_1, r_2 is simply the distance between them in the graph, henceforth denoted by $d(r_1, r_2)$.

Fix any prediction $\hat{R}$ and any input request sequence $R = (r_1, \ldots, r_{|R|})$. Let $\lambda = (\Delta, D)$ be some error for the input R and the prediction $\hat{R}$. We denote by $T^\lambda \subseteq R$ and $\hat{T}^\lambda \subseteq \hat{R}$ the subsets of matched input requests and matched predicted requests, respectively, in the definition of λ. Slightly abusing notation, we use λ as the matching between T^λ and $\hat{T}^\lambda$ such that for every $r \in T^\lambda$, we use $\lambda(r)$ to denote its matched predicted request in $\hat{T}^\lambda$. Similarly, for a subset of requests $R' \subseteq T^\lambda$, we define $\lambda(R') \subseteq \hat{T}^\lambda$ to be the set of predicted requests matched to requests in R'. In the other direction, we use $\lambda(\hat{r})$ to denote the request in T^λ matched to the predicted request $\hat{r} \in \hat{T}^\lambda$ (operating similarly for sets of predicted requests). Note that by the definition of (Δ, D)-error, it holds that $\Delta = \left|R \backslash T^\lambda\right| + \left|\hat{R} \backslash \hat{T}^\lambda\right|$, and that $D = \sum_{r \in T^\lambda} d(r, \lambda(r))$.

The error guarantee for Steiner tree is formally stated in Theorem 3.1.

THEOREM 3.1. *For every error $\lambda = (\Delta, D)$, Algorithm 2.1 for Steiner tree with predictions upholds*

$$\mathrm{ALG} \leq O\left(\log\left(\min\left\{|R|, \Delta\right\} + 2\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D.$$

Henceforth, we fix the error λ, and drop λ from T^λ and $\hat{T}^\lambda$. We define $k = \left|T^\lambda\right|$ the number of matched requests in λ, and fix i to be the major iteration in which u is first assigned a value which is at most $\left|\hat{R}\right| - k$ in Line 9 of Algorithm 2.1.

Proof. [Proof of Theorem 3.1] Having used the framework of Section 2, and observing that i is a major iteration, we can apply Lemma 2.2:

1. The cost of the first i iterations is at most $O(1) \cdot \mathrm{OPT} + (12\gamma + 2)\hat{B}_{i-1}$. Using Proposition 3.1, this expression is at most $O(1) \cdot \mathrm{OPT} + O(1) \cdot D$.
2. The cost of the last iterations, from $i+1$ onwards, is at most $O(1) \cdot \max\left\{\mathrm{ON}^{\mathrm{ST}}_{m-1}, \mathrm{ON}^{\mathrm{ST}}_{m}\right\}$. Using Lemma 3.3, this is at most $O\left(\log\left(\min\left\{|R|, \Delta\right\}\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D$.

Overall, we have that

$$\mathrm{ALG} \leq O\left(\log\left(\min\left\{|R|, \Delta\right\}\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D.$$

□

To complete the proof of Theorem 3.1, it remains to prove Proposition 3.1 and Lemma 3.3.

PROPOSITION 3.1. $\hat{B}_{i-1} \leq \mathrm{OPT} + D$.

Proof. Consider the iteration $i' < i$ in which $\hat{B}_{i-1}$ was set, i.e. the first iteration such that $\hat{B}_{i'} = \hat{B}_{i-1}$. If $i' = 0$, then $\hat{B}_{i-1}$ is the initial value of $\hat{B}$, which is 0, and the proposition trivially holds. Henceforth assume that $i' > 0$.

From the definition of i, we have that $u_{i'} > |\hat{R}| - k$, where $u_{i'}$ is the value of the variable u after iteration i'. Thus, $c(\textsc{Partial}(\hat{R}, |\hat{R}| - k)) > 3\gamma\hat{B}_{i'} = 3\gamma\hat{B}_{i-1}$. From Lemma 2.1, this implies that the least expensive solution which satisfies at least k requests from $\hat{R}$ costs at least $\hat{B}_{i-1}$. Since $k = |T| = |\hat{T}|$, we have that $\mathrm{OPT}_{\hat{T}}$ is such a solution, where $\mathrm{OPT}_{\hat{T}}$ is the optimal solution for $\hat{T}$. Therefore, $c(\mathrm{OPT}_{\hat{T}}) \geq \hat{B}_{i-1}$.

Now, observe that OPT can be augmented to a solution for $\hat{T}$ through connecting r to $\lambda(r)$ for every $r \in T$, at a cost of at most D. Thus, $c(\mathrm{OPT}) + D \geq c(\mathrm{OPT}_{\hat{T}}) \geq \hat{B}_{i-1}$, which completes the proof. □

Define the phases of the algorithm with respect to the major iteration i as in Section 2, where the cost of the online instance $\mathrm{ON}^{\mathrm{ST}}$ in the j'th phase is denoted by $\mathrm{ON}^{\mathrm{ST}}_j$. We again number the phases from 0 to m.

LEMMA 3.3. *It holds that*

$$\max\left\{\mathrm{ON}^{\mathrm{ST}}_{m-1}, \mathrm{ON}^{\mathrm{ST}}_m\right\} \leq O\left(\log\left(\min\left\{|R|, \Delta\right\}+2\right)\right)\cdot \mathrm{OPT} + O(1)\cdot D.$$

Proof. Let $j \in \{m-1, m\}$, and let $Q \subseteq R$ be the subsequence of requests considered in $\mathrm{ON}^{\mathrm{ST}}_j$. Denote by $\hat{R}' \subseteq \hat{R}$ the subset of predicted requests that are satisfied by the PARTIAL solution bought in iteration i.

The online algorithm $\mathrm{ON}^{\mathrm{ST}}_j$ operates in a modified metric space, in which the cost of a set of edges S_0 is set to 0; denote by d' the metric subject to these modified costs.

We partition Q into the following subsequences:

1. $Q_1 = Q \cap T$. We further partition Q_1 into the following sets:
   (a) $Q_{1,1} = \left\{r \in Q_1 | \lambda(r) \in \hat{R}'\right\}$
   (b) $Q_{1,2} = \left\{r \in Q_1 | \lambda(r) \notin \hat{R}\backslash\hat{R}'\right\}$
2. $Q_2 = Q\backslash T$.

Observe that $\mathrm{ON}^{\mathrm{ST}}_j(Q) = \mathrm{ON}^{\mathrm{ST}}_j(Q_{1,1}) + \mathrm{ON}^{\mathrm{ST}}_j(Q_{1,2}\cup Q_2)$. We now bound each component separately.

Bounding $\mathrm{ON}^{\mathrm{ST}}_j(Q_{1,1})$. Since $\lambda(Q_{1,1}) \subseteq \hat{R}'$, and $\hat{R}'$ is already served by the PARTIAL solution bought in iteration i, it holds for each request $r \in Q_{1,1}$ that $d'(\lambda(r), \rho) = 0$. Through triangle inequality, it thus holds that $d'(r,\rho) \leq d'(r, \lambda(r))$, and thus the cost of connecting r incurred by $\mathrm{ON}^{\mathrm{ST}}_j$ is at most $d'(r,\lambda(r))$. Therefore:

$$\mathrm{ON}^{\mathrm{ST}}_j(Q_{1,1}) \leq \sum_{r\in Q_{1,1}} d'(r,\lambda(r)) \leq \sum_{r\in Q_{1,1}} d(r,\lambda(r)) \leq D$$

Bounding $\mathrm{ON}^{\mathrm{ST}}_j(Q_{1,2}\cup Q_2)$. Using Property 2.1 (subset competitiveness), we have that

$$\mathrm{ON}^{\mathrm{ST}}_j(Q_{1,2}\cup Q_2) \leq O\left(\log\left(|Q_{1,2}\cup Q_2|+2\right)\right)\cdot \mathrm{OPT}'$$

where OPT' is the optimal solution to Q with the cost of S_0 set to 0. Clearly, $\mathrm{OPT}' \leq \mathrm{OPT}$.

Now, observe that $|Q_{1,2}| \leq \left|\hat{R}\backslash\hat{R}'\right|$. From the definition of the major iteration i, and from Lemma 2.1, it holds that $\left|\hat{R}\backslash\hat{R}'\right| \leq 2\gamma_{\mathrm{ST}}\cdot(|\hat{R}|-k) = 2\gamma_{\mathrm{ST}}\left|\hat{R}\backslash\hat{T}\right|$. As for Q_2, it holds that $Q_2 \leq |R\backslash T|$. These facts imply that

$$|Q_{1,2}\cup Q_2| \leq 2\gamma_{\mathrm{ST}}\Delta$$

In addition, it clearly holds that $|Q_{1,2}\cup Q_2| \leq |R|$. Therefore, it holds that

$$\mathrm{ON}^{\mathrm{ST}}_j(Q_{1,2}\cup Q_2) \leq O\left(\log\left(\min\left\{|R|, \Delta\right\}+2\right)\right)\cdot \mathrm{OPT}$$

Combining this with the previous bound for $\mathrm{ON}^{\mathrm{ST}}_j(Q_{1,1})$, we obtain

$$\mathrm{ON}^{\mathrm{ST}}_j(Q) \leq O\left(\log\left(\min\left\{|R|, \Delta\right\}+2\right)\right)\cdot \mathrm{OPT} + D$$

completing the proof. ∎

4 Online Steiner Forest with Predictions

In the online Steiner forest problem, we are given an undirected graph $G = (V, E)$ with non-negative edge costs $c : E \to \mathbb{R}^+$. The requests are pairs of vertices from V called *terminal pairs*. The algorithm maintains a subgraph S of G in which each terminal pair is connected. Upon the arrival of a new terminal pair, the algorithm must connect the two vertices by adding edges to S. The goal is to minimize the cost of the solution $c(S)$. We are also given a set of vertex pairs offline as the prediction $\hat{R}$ for the terminal pairs.

In this section, we apply the framework in Algorithm 2.1 to Steiner forest, using the two components: an offline approximation for prize-collecting Steiner forest, and an online algorithm for Steiner forest.

Prize-collecting offline algorithm. The following algorithm is due to Hajiaghayi and Jain [26].

LEMMA 4.1. (HAJIAGHAYI AND JAIN [26]) *There exists a γ_{SF}-approximation algorithm PC^{SF} for prize-collecting Steiner forest, where $\gamma_{\text{SF}} = 2.54$.*

Subset-competitive online algorithm. We now describe a version of the algorithm of Berman and Coulston [8] for online Steiner forest, denoted ON^{SF}. The precise statement of the Berman-Coulston algorithm is important, as we want a variant which is subset-competitive, which does not hold for all variants; for example, the variant of Umboh [55] is not subset-competitive. The crucial difference between the version we describe below and that of [55] is that when we connect a pair and observe that the balls around the two terminals intersect with existing balls, we connect each terminal to an *arbitrary* intersecting ball, while the algorithm in [55] connects to *all* intersecting balls. This may lead to a high cost at a single step, which invalidates subset competitiveness.

The ON^{SF} algorithm maintains a collection of duals[2] $\{D_j\}_{j=-\infty}^{\infty}$, where each D_j is a set of disjoint balls of radius exactly 2^{j-2} in the metric space induced by the graph. For each ball $B \in D_j$, there exists a terminal pair (s, t) such that either B is centered at s and t is outside B, or vice-versa. For the sake of analysis, we also define, for each D_j, a meta-graph M_j whose vertices correspond to the balls in D_j.

The ON^{SF} algorithm is given in Algorithm 4.1. Here, $P^F_{s,t}$ denotes the shortest path connecting vertices s and t in G, where edges in F have cost 0. Also, $B(v, r)$ denotes the (open) ball of radius r centered at vertex v.

ALGORITHM 4.1. Berman and Coulston's Algorithm for Online Steiner Forest (ON^{SF})

Input: $G = (V, E)$ – the input graph

procedure INITIALIZATION
  Initialize $D_j \leftarrow \emptyset$ for every $j \in \mathbb{Z}$.
  Initialize F to be the empty graph.
end procedure

procedure UPONREQUEST((s, t)) ▷ *Upon the next terminal pair (s, t) in the input*
  Set $F \leftarrow F \cup P^F_{s,t}$ and $\ell \leftarrow \lfloor \log c(P^F_{s,t}) \rfloor$.
  if $B(v, 2^{\ell-2})$ does not intersect[3] any ball in D_ℓ for some $v \in \{s, t\}$ **then**
    Add $B(v, 2^{\ell-2})$ to D_ℓ ▷ *This adds a vertex to M_j*
  else ▷ *both $B(s, 2^{\ell-2})$ and $B(t, 2^{\ell-2})$ intersected balls in D_ℓ*
    Denote by $B_1 = B(a_1, 2^{\ell-2})$ some ball that intersected $B(s, 2^{\ell-2})$, and by $B_2 = B(a_2, 2^{\ell-2})$ some ball that intersected $B(t, 2^{\ell-2})$.

[2] Although these collections of disjoint balls actually correspond to feasible LP dual solutions for Steiner forest, and hence we are calling them duals, we will give a self-contained proof of our theorem that does not need the LP formulation or LP duality.

▷ *Add an edge between B_1 and B_2 in M_j*
Set $F \leftarrow F \cup P^F_{a_1,s} \cup P^F_{a_2,t}$.
end if
end procedure

The $O(\log |R|)$-competitiveness of Algorithm 4.1 is due to Berman and Coulston [8]. We now show that it also upholds Property 2.1, i.e. it is subset competitive. This fact is stated in the following lemma, the proof of which is presented in Subsection 4.1.

LEMMA 4.2. *Algorithm 4.1 is subset-competitive as described in Property 2.1. That is, for input R and every subset $R' \subseteq R$, it holds that*

$$\mathrm{ON}^{\mathrm{SF}}(R') \leq O\left(\log\left(\left|R'\right| + 2\right)\right) \cdot \mathrm{OPT}.$$

The algorithm for the online Steiner forest with predictions is now constructed by simply plugging in $\mathrm{ON}^{\mathrm{SF}}$ and $\mathrm{PC}^{\mathrm{SF}}$ into the framework.

Steiner forest with predictions. Using $\mathrm{ON}^{\mathrm{SF}}$ and $\mathrm{PC}^{\mathrm{SF}}$ as the online and prize-collecting components of the framework in Section 2, we obtain the desired algorithm for Steiner forest with predictions. It remains to analyze its guarantees given any error $\lambda = (\Delta, D)$.

First, we must define the matching cost of requests and predictions for the error to be fully defined. We naturally define the matching cost of two terminal pairs to be the minimum distance one must move the two terminals of the first pair in the metric space in order for each terminal in the first pair to coincide with a distinct terminal in the second pair. Formally, the matching cost of request (s_1, t_1) and request (s_2, t_2) is

$$\min\left\{d\left(s_1, s_2\right) + d\left(t_1, t_2\right), d\left(s_1, t_2\right) + d\left(s_2, t_1\right)\right\}.$$

Fix any prediction $\hat{R}$ and any input request sequence $R = (r_1, \ldots, r_{|R|})$. As in the Steiner tree case, let $\lambda = (\Delta, D)$ be some error for the input R and the prediction $\hat{R}$. We again denote by $T^\lambda \subseteq R$ and $\hat{T}^\lambda \subseteq \hat{R}$ the subsets of matched input requests and matched predicted requests, respectively, in the definition of λ. As before, we use λ to map from a request in T to its match in $\hat{T}$ and vice-versa.

The main result of this section is the following theorem for Steiner forest with predictions.

THEOREM 4.1. *For every error $\lambda = (\Delta, D)$, Algorithm 2.1 for Steiner forest with predictions has the guarantee*

$$\mathrm{ALG} \leq O\left(\log\left(\min\left\{R, \Delta\right\} + 2\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D.$$

As in the Steiner tree case, we henceforth fix a (Δ, D)-error λ, and drop λ from T^λ and $\hat{T}^\lambda$. We define $k = |T|$. Now, fix i to be the major iteration in which u is first assigned a value which is at most $|\hat{R}| - k$ in Line 9 of Algorithm 2.1.

Proof. [Proof of Theorem 4.1] The proof of Theorem 4.1 follows the same lines as Theorem 3.1 for Steiner tree.

Having used the framework of Section 2, and observing that i is a major iteration, we can apply Lemma 2.2:

[3] Two (open) balls $B(v_1, r_1)$ and $B(v_2, r_2)$ are said to intersect if the shortest path from v_1 to v_2 costs less than $r_1 + r_2$.

1. The cost of the first i iterations is at most $O(1) \cdot \mathrm{OPT} + (12\gamma + 2)\hat{B}_{i-1}$. Using Proposition 4.1, this expression is at most $O(1) \cdot \mathrm{OPT} + O(1) \cdot D$.
2. The cost of the last iterations, from $i+1$ onwards, is at most $O(1) \cdot \max\left\{\mathrm{ON}^{\mathrm{ST}}_{m-1}, \mathrm{ON}^{\mathrm{ST}}_{m}\right\}$. Using Lemma 4.3, this is at most $O\left(\log\left(\min\left\{|R|, \Delta\right\}\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D$.

Overall, we have that

$$\mathrm{ALG} \leq O\left(\log\left(\min\left\{|R|, \Delta\right\}\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D.$$

□

It remains to prove Proposition 4.1 and Lemma 4.3.

Proposition 4.1. (Analog of Proposition 3.1) $\hat{B}_{i-1} \leq \mathrm{OPT} + D$.

Proof. Identical to the proof of Proposition 3.1 for Steiner tree with predictions. □

Lemma 4.3. (Analog of Lemma 3.3) *It holds that*

$$\max\left\{\mathrm{ON}^{\mathrm{SF}}_{m-1}, \mathrm{ON}^{\mathrm{SF}}_{m}\right\} \leq O\left(\log\left(\min\left\{|R|, \Delta\right\} + 2\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D.$$

Proof. The proof is nearly identical to that of Lemma 3.3 for Steiner tree. The only thing to note explicitly is that for Steiner tree, the cost of serving a request is at most its distance to any previous request, while $\mathrm{ON}^{\mathrm{SF}}$ can pay at most twice that distance. □

4.1 Subset Competitiveness of $\mathrm{ON}^{\mathrm{SF}}$. This subsection is devoted to proving that the online algorithm $\mathrm{ON}^{\mathrm{SF}}$ is subset competitive, as stated in the following lemma.

Lemma 4.4. *At any point during the algorithm, M_j is acyclic.*

Proof. Suppose for contradiction that at some iteration in the algorithm, an edge is created between two balls $B_1, B_2 \in D_j$ that were already connected in M_j at the beginning of that iteration, thus closing a cycle in M_j. Denote by a_1, a_2 the centers of B_1, B_2 respectively. Due to Line 11, we have that at the beginning of the iteration, a_1 and a_2 were connected in F.

Let (s,t) be the terminal pair considered in that iteration, and without loss of generality assume that $B(s, 2^{j-2})$ intersected B_1 and $B(t, 2^{j-2})$ intersected B_2. Then $d(s, a_1) < 2^{j-1}$ and $d(t, a_2) < 2^{j-1}$. But then since a_1 and a_2 are connected in F, we have that $c\left(P^F_{s,t}\right) < 2^j$ at the beginning of the iteration, in contradiction to the algorithm creating an edge in M_j in that iteration. □

Denote by n_j the number of iterations in which the variable ℓ was set to j. We also denote by n'_j the number of such iterations on the requests of R'.

Corollary 4.1. *At the end of the algorithm, for every j, we have that $n_j \leq 2 \cdot |D_j|$.*

Proof. In each iteration counted in n_j, either a node or an edge is added to M_j. Since M_j is acyclic, the number of its edges is at most the number of its nodes. □

Define $m^+ := \lfloor \log\left(\mathrm{ON}^{\mathrm{SF}}(R')\right) \rfloor$ and $m^- := m^+ - \lceil \log |R'| \rceil - 2$.

LEMMA 4.5. $\text{ON}^{\text{SF}}(R') \leq 24 \cdot \sum_{j=m^-}^{m^+} n'_j \cdot 2^{j-2}$.

Proof. First, observe that $\text{ON}^{\text{SF}}(R') \leq 3 \cdot \sum_{j=-\infty}^{\infty} n'_j \cdot 2^j$. This is since an iteration counted in n'_j costs at most 2^{j+1} in adding a path in Line 6, and at most 2^j in connecting conflicting nodes in Line 11.

Now, observe that $n'_j = 0$ for every $j > m^+$. This is since an iteration counted in n'_j for such a j would cost strictly more than $\text{ON}^{\text{SF}}(R')$, in contradiction. Additionally, it holds that

$$\sum_{j=-\infty}^{m^- - 1} n'_j \cdot 2^j \leq |R| \cdot 2^{m^- - 1} \leq \left|R'\right| \cdot \frac{\text{ON}^{\text{SF}}(R')}{8 \cdot |R'|} = \frac{\text{ON}^{\text{SF}}(R')}{8}.$$

Combining those observations, we get that

$$\text{ON}^{\text{SF}}(R') \leq 3 \cdot \sum_{j=-\infty}^{m^- - 1} n'_j \cdot 2^j + 3 \cdot \sum_{j=m^-}^{m^+} n'_j \cdot 2^j \leq \frac{\text{ON}^{\text{SF}}(R')}{2} + 3 \cdot \sum_{j=m^-}^{m^+} n'_j \cdot 2^j,$$

and thus $\text{ON}^{\text{SF}}(R') \leq 6 \cdot \sum_{j=m^-}^{m^+} n'_j \cdot 2^j = 24 \cdot \sum_{j=m^-}^{m^+} n'_j \cdot 2^{j-2}$. ∎

PROPOSITION 4.2. *For every $j \in \mathbb{Z}$, we have* $\text{OPT} \geq |D_j| \cdot 2^{j-2}$.

Proof. Each D_j is a set of disjoint balls of radius 2^{j-2}. Each such ball B is centered at some terminal a that must be connected outside the ball. Therefore, any feasible solution must contain edges (or parts of edges) in B in order to connect a outwards, at a total cost that is at least the radius of the ball. Since the balls are disjoint, these costs sum up in any feasible solution. ∎

Proof. [Proof of Lemma 4.2] Using Proposition 4.2 for each $j \in [m^-, m^+]$ and averaging, we get that

$$\text{OPT} \geq \frac{\sum_{j=m^-}^{m^+} |D_j| \cdot 2^{j-2}}{m^+ - m^- + 1} = \frac{\sum_{j=m^-}^{m^+} |D_j| \cdot 2^{j-2}}{\log |R'| + 3}.$$

Using Lemma 4.5 and Corollary 4.1, we have

$$\text{ON}^{\text{SF}}(R') \leq 24 \cdot \sum_{j=m^-}^{m^+} n'_j \cdot 2^{j-2} \leq 48 \sum_{j=m^-}^{m^+} |D_j| \cdot 2^{j-2} \leq O\left(\log\left(\left|R'\right| + 2\right)\right) \cdot \text{OPT}.$$

∎

5 Online Facility Location with Predictions

In the online facility location problem, we are given an undirected graph $G = (V, E)$ offline with cost c_e for edge e and facility opening cost f_v for vertex v. Requests (called *clients*) arrive at the vertices of the graph, one per online step. The algorithm maintains a set of open facilities at a subset of the vertices of the graph. Upon the arrival of a client r, the algorithm must connect the client to its closest open facility, at a connection cost equal to the shortest path in the graph between r and that facility. The algorithm is also allowed to open new facilities before making this connection; opening a new facility at a vertex v adds a cost of f_v to the solution. The goal is to minimize overall cost defined as the sum of opening costs of facilities and connection costs of clients.

We would like to use the framework of Section 2. Thus, we must describe the relevant components, i.e. a prize-collecting algorithm and a subset-competitive online algorithm.

Prize-collecting offline algorithm. The following lemma is due to Xu and Xu [57].

LEMMA 5.1. *There exists a γ_{FL}-approximation for prize-collecting facility location, where $\gamma_{\text{FL}} = 1.8526$.*

Subset-competitive online algorithm. Algorithm 5.1 (we call it ON^{FL}) is an $O(\log(|R|+2))$-competitive deterministic algorithm for online facility location due to Fotakis [18]. Here, $d(u,v)$ denotes the shortest path distance between vertices u and v, and for a subset of vertices F, we denote $d(F,v) := \min_{u\in F} d(u,v)$. For any value d, we use $d_+ := \max(d,0)$.
$F \leftarrow \emptyset$

ALGORITHM 5.1. Fotakis' Algorithm for Online Facility Location (ON^{FL})
Input: $G = (V,E)$ – the input graph

```
procedure Initialization
    F ← ∅, L ← ∅
    for all v ∈ V do
        set p(v) ← 0
    end for
end procedure

procedure UponRequest(r)                    ▷ Upon the next request r in the sequence
    Set L ← L ∪ r
    UpdatePotentials F,r
    w ← arg max_{v∈V}(p(v) − f_v)
    if p(w) > f_w then
        set F ← F ∪ w
        ComputeNewPotentials F,L
    end if
end procedure

procedure UpdatePotentials(F,r)
    for all v ∈ V do
        set p(v) ← p(v) + (d(F,r) − d(v,r))_+
    end for
end procedure

procedure ComputeNewPotentials(F,L)
    for all v ∈ V do
        set p(v) ← Σ_{r∈L} (d(F,r) − d(v,r))_+
    end for
end procedure
```

This algorithm is *not* subset-competitive with respect to its actual cost; however, it is subset-competitive with respect to an amortized cost, which is sufficient to achieve the desired results.

We use subscript i to denote the value of a variable in Algorithm 5.1 immediately after handling the i'th request (for $i = 0$ this refers to the initial value of the variable). This is used in this section for the variables F, L and the potential function p.

DEFINITION 5.1. (AMORTIZED COST) *For each request $r_i \in R$, define the* amortized cost *of r_i as*

$$\alpha(r_i) = 2 \cdot \min\left\{ d(F_{i-1}, r_i), \min_{v\in V} \left(f_v - p_{i-1}(v) + d(v,r_i)\right) \right\}.$$

In words, the amortized cost of a request r is twice the minimum between its distance to a facility which is already open upon r's arrival, and its distance to a facility v which is not open plus the "remaining" cost to opening that facility (f_v minus the potential $p(v)$).

The following two lemmas, the proof of which appears in Subsection 5.1, state that α is indeed an amortization of the algorithm's cost, and that α is subset competitive.

LEMMA 5.2. $\text{ON}^{\text{FL}} \leq \sum_{r \in R} \alpha(r)$.

LEMMA 5.3. *Algorithm 5.1 is subset-competitive, as required in Property 2.1, with respect to its amortized cost. That is, for input R and for every subset of requests R' of R, it holds that*

$$\sum_{r \in R} \alpha(r) \leq O(\log\left(|R'| + 2\right)) \cdot \text{OPT}.$$

Facility Location with Predictions. We use the components ON^{FL} and PC^{FL} to construct the algorithm for facility location with predictions. The algorithm uses ON^{FL} as a black box, as in the framework described in Algorithm 2.1. For the sake of describing the algorithm, and its analysis, we assume that the cost incurred by ON^{FL} upon receiving a request r is not the actual cost of opening facilities and connecting the request, but rather the *amortized* cost $\alpha(r)$, as defined in Definition 5.1. In particular, this altered cost affects the counting of the online algorithm's cost, which affects whether a PARTIAL solution is bought. As guaranteed in Lemmas 5.2 and 5.3, α is indeed an amortized cost function, which is subset-competitive. Also observe that $\alpha(r)$ can be calculated by ON^{FL} upon the release of r (which is required for counting the online cost towards buying PARTIAL solutions).

In this algorithm, the addition of the offline solution returned by PARTIAL is done by adding the facilities of that solution immediately, and connecting the requests of that solution upon their future arrival. This postponing of costs does not affect the analysis.

We would now like to analyze this algorithm for (Δ, D)-errors. First, we must first define a (Δ, D)-error completely by defining the matching cost of two requests. This matching cost, rather naturally, is defined to be the distance between the requests' nodes in the graph.

The main result of this section is the following theorem for facility location with predictions.

THEOREM 5.1. *For every error $\lambda = (\Delta, D)$, Algorithm 2.1 for facility location with predictions has the guarantee*

$$\text{ALG} \leq O\left(\log\left(\min\left\{|R|, \Delta\right\} + 2\right)\right) \cdot \text{OPT} + O(1) \cdot D.$$

When analyzing this algorithm, we regard the costs incurred by the online component ON^{FL} as the amortized costs, i.e. on request r the cost of ON^{FL} would be $\alpha(r)$. Since the sum of those costs upper-bounds the actual cost of the algorithm (Lemma 5.2), such an analysis is legal. Lemma 5.3 now implies that this amortized cost is subset-competitive, and thus the properties and lemmas of Section 2 hold.

As for Steiner tree and Steiner forest, we denote by T^λ and $\hat{T}^\lambda$ the matched requests and the matched predictions of the error λ.

The proof follows the same lines as the guarantee for robust Steiner tree (Theorem 3.1). As in the robust Steiner tree case, define $k = |T|$. Now, fix i to be the iteration in which the variable u is first assigned a value which is at least $\left|\hat{R}\right| - k$ in Line 9 of Algorithm 2.1.

PROPOSITION 5.1. (ANALOG OF PROPOSITION 3.1) $\hat{B}_{i-1} \leq \text{OPT} + D$.

Proof. Let $i' < i$ be the iteration in which $\hat{B}_{i-1}$ was set, i.e. the first iteration such that $\hat{B}_{i'} = \hat{B}_{i-1}$. If $i' = 0$, then $\hat{B}_{i-1}$ is the initial value of $\hat{B}$, which is 0, and the proposition holds. Henceforth assume that $i' > 0$.

From the definition of i, we have that $u_{i'} > \left|\hat{R}\right| - k$, where $u_{i'}$ is the value of the variable u after iteration i'. Thus, $c(\textsc{Partial}(\hat{R}, \left|\hat{R}\right| - k)) > 3\gamma\hat{B}_{i'} = 3\gamma\hat{B}_{i-1}$. From Lemma 2.1, this implies that the least expensive solution which satisfies at least k requests from $\hat{R}$ costs at least $\hat{B}_{i-1}$. Since $k = \left|\hat{T}\right| = |T|$, we have that $\mathrm{OPT}_{\hat{T}}$ is such a solution, where $\mathrm{OPT}_{\hat{T}}$ is the optimal solution for $\hat{T}$. Therefore, $c(\mathrm{OPT}_{\hat{T}}) \geq \hat{B}_{i-1}$.

Now, observe that OPT can be augmented to a solution for $\hat{T}$ through disconnecting each request $r \in T$ from the facility v_r to which it is connected, and instead connecting the request $\lambda(r) \in \hat{T}$ to v_r. This connects all requests in $\hat{T}$, and increases the cost of OPT by exactly

$$\sum_{r \in T} d(\lambda(r), v_r) - d(r, v_r) \underbrace{\leq}_{\text{triangle inequality}} \sum_{r \in T} d(r, \lambda(r)) = D$$

We therefore have a solution for $\hat{T}$ which costs at most $\mathrm{OPT} + D$, completing the proof.

□

Lemma 5.4. (Analog of Lemma 3.3) *It holds that*

$$\max\left\{\mathrm{ON}^{\mathrm{FL}}_{m-1}, \mathrm{ON}^{\mathrm{FL}}_{m}\right\} \leq O\left(\log\left(\min\left\{|R|, \Delta\right\} + 2\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D.$$

Proof. Let $j \in \{m-1, m\}$, and let $Q \subseteq R$ be the subsequence of requests considered in $\mathrm{ON}^{\mathrm{ST}}_j$. Denote by $\hat{R}' \subseteq \hat{R}$ the subset of predicted requests that are satisfied by the Partial solution considered in iteration i (the solution whose facilities were opened at the end of iteration i).

The online algorithm $\mathrm{ON}^{\mathrm{FL}}_j$ operates on a modified input, in which the cost of a set of facilities F_0 is set to 0.

We partition Q into the following subsequences:

1. $Q_1 = Q \cap T$. We further partition Q_1 into the following sets:
   (a) $Q_{1,1} = \left\{r \in Q_1 | \lambda(r) \in \hat{R}'\right\}$
   (b) $Q_{1,2} = \left\{r \in Q_1 | \lambda(r) \notin \hat{R} \backslash \hat{R}'\right\}$
2. $Q_2 = Q \backslash T$.

Observe that $\mathrm{ON}^{\mathrm{FL}}_j(Q) = \mathrm{ON}^{\mathrm{FL}}_j(Q_{1,1}) + \mathrm{ON}^{\mathrm{FL}}_j(Q_{1,2} \cup Q_2)$. We now bound each component separately.

Bounding $\mathrm{ON}^{\mathrm{FL}}_j(Q_{1,1})$. Consider a request $r \in Q_{1,1}$, and note from the definition of amortized cost that $\mathrm{ON}^{\mathrm{FL}}_j(r) \leq 2d(r, F')$, where F' is the set of facilities which are either open or have cost 0 immediately before considering r. Thus, we have that

$$\mathrm{ON}^{\mathrm{FL}}_j(Q_{1,1}) \leq \sum_{r \in Q_{1,1}} 2d(r, F_0) \leq \sum_{r \in Q_{1,1}} 2d(r, \lambda(r)) + 2d(\lambda(r), F_0) \leq 2D + \sum_{r \in \hat{R}'} d(r, F_0)$$

Now, observe that F_0 contains all facilities bought by the Partial solution at iteration i, and thus $\sum_{r \in \hat{T}} d(r, F_0)$ is at most the connection cost of that solution, which is at most the solution's cost. From the proof of Item 1 of Lemma 2.2, We know that the cost of that solution is at most $3\gamma_{\mathrm{FL}}(2\hat{B}_{i-1} + O(1) \cdot \mathrm{OPT}) = O(1) \cdot \mathrm{OPT} + O(1) \cdot D$. Overall, we have that $\mathrm{ON}^{\mathrm{FL}}_j(Q_{1,1}) \leq O(1) \cdot \mathrm{OPT} + O(1) \cdot D$.

Bounding $\mathrm{ON}_j^{\mathrm{FL}}(Q_{1,2} \cup Q_2)$**.** Using Property 2.1 (subset competitiveness), we have that

$$\mathrm{ON}_j^{\mathrm{FL}}(Q_{1,2} \cup Q_2) \leq O\left(\log\left(|Q_{1,2} \cup Q_2| + 2\right)\right) \cdot \mathrm{OPT}'$$

where OPT' is the optimal solution to Q with the cost of F_0 set to 0. Clearly, $\mathrm{OPT}' \leq \mathrm{OPT}$.

Now, observe that $|Q_{1,2}| \leq \left|\hat{R} \backslash \hat{R}'\right|$. From the definition of the major iteration i, and from Lemma 2.1, it holds that $\left|\hat{R} \backslash \hat{R}'\right| \leq 2\gamma_{\mathrm{FL}} \cdot (|\hat{R}| - k) = 2\gamma_{\mathrm{FL}} \left|\hat{R} \backslash \hat{T}\right|$. As for Q_2, it holds that $Q_2 \leq |R \backslash T|$. These facts imply that

$$|Q_{1,2} \cup Q_2| \leq 2\gamma_{\mathrm{FL}} \Delta$$

In addition, it clearly holds that $|Q_{1,2} \cup Q_2| \leq |R|$. Therefore, it holds that

$$\mathrm{ON}_j^{\mathrm{FL}}(Q_{1,2} \cup Q_2) \leq O\left(\log\left(\min\left\{|R|, \Delta\right\} + 2\right)\right) \cdot \mathrm{OPT}$$

Combining this with the previous bound for $\mathrm{ON}_j^{\mathrm{FL}}(Q_{1,1})$, we obtain

$$\mathrm{ON}_j^{\mathrm{FL}}(Q) \leq O\left(\log\left(\min\left\{|R|, \Delta\right\} + 2\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D$$

completing the proof.

□

Theorem 5.1 now follows from Lemma 2.2, Proposition 3.1 and Lemma 5.4, using an identical argument as for Theorem 3.1.

5.1 Subset Competitiveness of Algorithm 5.1. The goal of this subsection is to prove Lemmas 5.2 and 5.3. Henceforth, fix the input sequence $R = (r_1, \ldots, r_{|R|})$, and recall that for every variable x (e.g., F, L, p) in the algorithm, we denote by x_i the value of x immediately after iteration i (if $i = 0$ this denotes the initial value of the variable).

LEMMA 5.5. *For every $i \in \{1, 2, \ldots, |R|\}$, and for every $v \in V$, it holds that $p_i(v) \leq f_v$.*

Proof. Identical to the proof of [18, Lemma 1]. □

Denote by F^* the set of facilities bought by the optimal solution. For each facility $v^* \in F^*$, let C_{v^*} be the set of requests connected to v^* by the optimal solution.

We denote the cost incurred by the algorithm in opening facilities by $\mathrm{ON}_F^{\mathrm{FL}}$, and the cost of connecting requests by $\mathrm{ON}_C^{\mathrm{FL}}$. We define OPT_F and OPT_C similarly. For every subsequence of requests R', we define $\mathrm{ON}_F^{\mathrm{FL}}(R')$ and $\mathrm{ON}_C^{\mathrm{FL}}(R')$ as the total opening cost and the total connection cost incurred for the requests of R', respectively. We also use the same terminology for OPT.

COROLLARY 5.1. $|L_i \cap C_{v^*}| \cdot d(F_i, v^*) \leq f_{v^*} + 2 \cdot \mathrm{OPT}_C(C_{v^*})$ *for every $v^* \in F^*$.*

Proof. Identical to the proof of [18, Corollary 1]. □

PROPOSITION 5.2. *For each request $r_i \in R$, if $\mathrm{ON}^{\mathrm{FL}}$ opens a facility at any $w \in V$ in iteration i, then $\alpha(r_i) = 2\left(f_w - p_{i-1}(w) + d(w, r_i)\right)$. Otherwise, $\alpha(r_i) = 2d(F_{i-1}, r_i)$.*

Proof. Identical to the proof of [18, Lemma 3]. ∎

Proof. [Proof of Lemma 5.2] Define the potential function $\phi(i) = \sum_{r \in L_i} d(F_i, r)$ for every iteration i. We show by induction on i that

$$\mathrm{ON}^{\mathrm{FL}}((r_1, \ldots, r_i)) + \phi(i) \leq \sum_{r \in L_i} \alpha(r), \tag{5.3}$$

thereby proving the lemma. Suppose Equation 5.3 holds for iteration $i-1$. Iteration i would increase the RHS by $\alpha(r_i)$, and increase the LHS by $\mathrm{ON}^{\mathrm{FL}}(r_i) + \Delta\phi$, where we define $\Delta\phi = \phi(i) - \phi(i-1)$ to be the change in the potential function ϕ.

If the algorithm did not open a new facility in iteration i, then we have $\mathrm{ON}^{\mathrm{FL}}(r_i) = d(r_i, F_{i-1})$. In addition, we have that $\Delta\phi = d(r_i, F_{i-1})$. In the RHS, using Proposition 5.2, we have that $\alpha(r_i) = 2d(r_i, F_{i-1})$, completing the proof for this case.

For the other case, in which the algorithm opens a new facility at $w \in V$ during iteration i, we have that the algorithm's cost for that iteration is f_w for opening the facility, plus the cost of connecting r_i.

Observe that if we open a facility for the current client, it becomes the closest open facility to the client. This is since at the beginning of the iteration, we had that $p(w) \leq f_w$, using Lemma 5.5. Thus, if w was opened in iteration i, its potential has increased upon the arrival of r_i. But this only happens if w is closer to r_i than any open facility in F_{i-1}. Therefore, the connection cost of r_i in iteration i is exactly $d(r_i, w)$.

As for the change in the potential function ϕ, we consider first the addition of w to F and then the addition of r_i to the request set. The addition of w to F has reduced ϕ by exactly $p_{i-1}(w)$. Then, the addition of r_i increased ϕ by exactly $d(r_i, w)$. Thus, The LHS of Equation 5.3 increased by a total of $f_w - p_{i-1}(w) + d(r_i, w)$. But according to Proposition 5.2, the RHS has increased by twice that amount, and thus the equation holds.

Since the potential function ϕ is initially 0, and is always non-negative, the proof is complete. ∎

Proof. [Proof of Lemma 5.3] Fix a facility $f^* \in F^*$. Denote the requests of C_{f^*} by $q_1, \ldots, q_{|C_{f^*}|}$ ordered by their arrival. With $k' = |R' \cap C^*_f|$, denote by $i_1, \ldots, i_{k'}$ the indices in the requests of $R' \cap C^*_f$.

For q_{i_1}, we have that $\alpha(q_{i_1}) \leq 2f_{v^*} + 2d(v^*, q_{i_1})$. Now, consider any $j > 1$, and let i be the iteration in which q_{i_j} is considered. For $j > 1$, using Corollary 5.1, we have that

$$\begin{aligned}
\alpha(q_{i_j}) &\leq 2 \cdot d(q_{i_j}, F_{i-1}) \\
&\leq 2 \cdot d(v^*, F_{i-1}) + 2 \cdot d(v^*, q_{i_j}) \quad \text{(by triangle inequality)} \\
&= 2 \cdot d(v^*, F_{i-1}) + 2 \cdot d(v^*, q_{i_j}) \\
&\leq \frac{2}{|L_{i-1} \cap C_{v^*}|} \cdot (f_{v^*} + 2 \cdot \mathrm{OPT}_C(C_{v^*})) + 2 \cdot d(v^*, q_{i_j}) \quad \text{(by Corollary 5.1)} \\
&\leq \frac{2}{|L_{i-1} \cap C_{v^*} \cap R'|} \cdot (f_{v^*} + 2 \cdot \mathrm{OPT}_C(C_{v^*})) + 2 \cdot d(v^*, q_{i_j}) \\
&= \frac{2}{j-1} \cdot (f_{v^*} + 2 \cdot \mathrm{OPT}_C(C_{v^*})) + 2 \cdot d(v^*, q_{i_j}) \quad \text{(by the definition of } j\text{)}.
\end{aligned}$$

Summing over j, we get

$$\begin{aligned}\sum_{j=1}^{k'} \alpha(q_{i_j}) &\leq 2 \cdot f_{v^*} + \sum_{j=2}^{k'} \frac{2}{j-1} \cdot (f_{v^*} + 2 \cdot \mathrm{OPT}_C(C_{v^*})) + 2 \cdot \mathrm{OPT}_C(C_{v^*} \cap R') \\ &\leq 2(\log k' + 1) \cdot f_{v^*} + 4(\log k' + 1) \cdot \mathrm{OPT}_C(C_{v^*} \cap R') \\ &\leq 2(\log k + 1) \cdot f_{v^*} + 4(\log k + 1) \cdot \mathrm{OPT}_C(C_{v^*} \cap R').\end{aligned}$$

Summing over all $v^* \in F^*$, we get that

$$\sum_{r \in R'} \alpha(r) \leq O(\log (k+2)) \cdot \mathrm{OPT}.$$

□

6 Online Capacitated Facility Location with Predictions

The results of Section 5 also extend to the soft-capacitated facility location problem. In this problem, each node $v \in V$ has, in addition to the facility opening cost f_v, a *capacity* β_v which is a natural number. After opening a facility at v, a solution can connect at most β_v clients to that facility. We consider the soft-capacitated case, in which a solution may open multiple facilities at a single node v, each at a cost of f_v, such that each facility can connect β_v requests.

The matching cost between two requests in this case is identical to the uncapacitated case – specifically, it is the distance between the two requests in the graph.

This problem can be related to the uncapacitated facility location problem using the following folklore reduction, which we nevertheless describe for completeness: given an instance for soft-capacitated facility location, which contains a graph $G = (V, E)$, facility costs $\{f_v\}$ and capacities $\{\beta_v\}$, construct a uncapacitated facility location instance over the graph $G' = (V \cup V', E \cup E')$, where:

- V' contains a copy v' for every $v \in V$. The opening cost in v' would be f_v, and the opening cost in v would be ∞.
- E' contains an edge $e = (v, v')$ for every $v \in V$ (and its copy $v' \in V'$), such that the cost of the edge is $\frac{f_v}{\beta_v}$.
- The cost of the edges of E remains as in the original instance.

Whenever a request is released in the original capacitated instance on a node v, release a request on v in the uncapacitated instance.

In essence, this reduction restricts opening facilities to copies of the original nodes, such that these copies are somewhat distant from the rest of the graph, as to discourage connecting requests frivolously (wasting the capacity).

The algorithm for an instance $\mathcal{I}$ of capacitated facility location would therefore be:

1. Reduce $\mathcal{I}$ to an uncapacitated instance $\mathcal{I}'$, on which the algorithm of Section 5 is run.
2. Whenever a request r is released on a node v in $\mathcal{I}$, release a request on v in $\mathcal{I}'$.
3. Whenever the algorithm opens a facility at v' in $\mathcal{I}'$, open a facility at v in $\mathcal{I}$.

4. Whenever the algorithm connects a request at r to a facility at v' in $\mathcal{I}'$, connect r to a facility at v in $\mathcal{I}$ (opening an additional facility at v if required).

Denote by ALG the cost of the algorithm on $\mathcal{I}$, ALG$'$ the cost of the algorithm of Section 5 on $\mathcal{I}'$, OPT$'$ the optimal solution for $\mathcal{I}'$, and OPT the optimal solution for $\mathcal{I}$. Observe that:

1. ALG $\leq$ ALG$'$. This is since for each connection to facility v, ALG$'$ pays exactly $x_v = \frac{f_v}{\beta_v}$ more than ALG. As for the opening costs, the opening of a first facility at v in ALG is charged the opening of the facility at v' in ALG$'$, and the opening of any subsequent facility at v in ALG is charged to β_v connections which cost x_v more, for a total of $\beta_v \cdot x_v = f_v$.

2. OPT$'$ $\leq$ 2OPT. We show that a OPT induces a solution for $\mathcal{I}'$ of at most double cost. The solution OPT$''$ for $\mathcal{I}'$ consists of opening a facility at v' for every v in which OPT opened a nonzero number of facilities, and connecting to v' all requests which OPT connected to facilities at v. Let y_v be the number of requests connected to node v in OPT. Then the connection cost of OPT$''$ is exactly $C_v + y_v \cdot \frac{f_v}{\beta_v}$, where C_v is the cost of connecting requests to v incurred by OPT. Now, observe that $y_v \cdot \frac{f_v}{\beta_v}$ is a lower bound on the buying cost of OPT on facilities in v; thus, the connection cost of OPT$''$ is at most OPT. Since the buying cost of OPT$''$ is also clearly at most OPT, we arrive at the desired conclusion.

Importantly, observe that the distance between requests in the metric spaces of G and G' is identical - thus, the matching cost D is the same in both $\mathcal{I}$ and $\mathcal{I}'$. We can thus apply Theorem 5.1 to $\mathcal{I}'$, which combined with the above conclusions, yields the following theorem.

THEOREM 6.1. *For error $\lambda = (\Delta, D)$ in soft-capacitated facility location, Algorithm 2.1 (combined with the above reduction) has the guarantee*

$$\text{ALG} \leq O\left(\log\left(\min\left\{|R|, \Delta\right\} + 2\right)\right) \cdot \text{OPT} + O(1) \cdot D.$$

7 Conclusions

In this paper, we presented algorithms for classical graph problems in the online problems with predictions framework. Our main contributions were: (a) a novel definition of prediction error for graph and metric problems that incorporates both the numerical value of errors and their magnitude in the metric space, (b) a new "black box" framework for converting online algorithms that satisfy a subset competitiveness condition and offline prize-collecting algorithms to new online algorithms with prediction, and (c) an application of this framework to a range of graph problems to obtain tight interpolations between offline and online approximations. In particular, this improves the dependence of the competitive ratio from being on the size of the instance to the number of prediction errors. We hope that the concepts introduced in this paper–the notion of metric error and the general framework for online algorithms with prediction–will be useful for other problems in the future.

References

[1] A. Agrawal, P. N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995.

[2] K. Anand, R. Ge, and D. Panigrahi. Customizing ml predictions for online algorithms. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020*, Proceedings of Machine Learning Research, 2020.

[3] A. Antoniadis, C. Coester, M. Elias, A. Polak, and B. Simon. Online metric algorithms with untrusted predictions. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020*, Proceedings of Machine Learning Research, 2020.

[4] S. Arora and G. Karakostas. A 2 + *epsilon* approximation algorithm for the k-mst problem. *Math. Program.*, 107(3):491–504, 2006.

[5] S. Arya and H. Ramesh. A 2.5-factor approximation algorithm for the k-mst problem. *Inf. Process. Lett.*, 65(3):117–118, 1998.

[6] B. Awerbuch, Y. Azar, A. Blum, and S. S. Vempala. New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM J. Comput.*, 28(1):254–262, 1998.

[7] É. Bamas, A. Maggiori, and O. Svensson. The primal-dual method for learning augmented algorithms. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[8] P. Berman and C. Coulston. On-line algorithms for steiner tree problems (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 344–353, 1997.

[9] A. Bhaskara, A. Cutkosky, R. Kumar, and M. Purohit. Online learning with imperfect hints. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 822–831. PMLR, 2020.

[10] A. Blum, R. Ravi, and S. Vempala. A constant-factor approximation algorithm for the k-mst problem. *J. Comput. Syst. Sci.*, 58(1):101–108, 1999.

[11] J. Byrka and K. Aardal. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. *SIAM J. Comput.*, 39(6):2212–2231, 2010.

[12] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, 2013.

[13] M. Charikar and S. Guha. Improved combinatorial algorithms for facility location problems. *SIAM J. Comput.*, 34(4):803–824, 2005.

[14] M. Charikar, J. Naor, and B. Schieber. Resource optimization in qos multicast routing of real-time multimedia. *IEEE/ACM Trans. Netw.*, 12(2):340–348, 2004.

[15] F. A. Chudak and D. B. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM J. Comput.*, 33(1):1–25, 2003.

[16] J. Chuzhoy, A. Gupta, J. Naor, and A. Sinha. On the approximability of some network design problems. *ACM Trans. Algorithms*, 4(2):23:1–23:17, 2008.

[17] O. Dekel, A. Flajolet, N. Haghtalab, and P. Jaillet. Online learning with a hint. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5299–5308, 2017.

[18] D. Fotakis. A primal-dual algorithm for online non-uniform facility location. *J. Discrete Algorithms*, 5(1):141–148, 2007.

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

[20] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 302–309, 1996.

[21] N. Garg. Saving an epsilon: a 2-approximation for the k-mst problem in graphs. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 396–402, 2005.

[22] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

[23] S. Gollapudi and D. Panigrahi. Online algorithms for rent-or-buy with expert advice. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California,*

USA, pages 2319–2327, 2019.
[24] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *J. Algorithms*, 31(1):228–248, 1999.
[25] A. Gupta, M. T. Hajiaghayi, V. Nagarajan, and R. Ravi. Dial a ride from k-forest. *ACM Trans. Algorithms*, 6(2):41:1–41:21, 2010.
[26] M. T. Hajiaghayi and K. Jain. The prize-collecting generalized steiner tree problem via a new approach of primal-dual schema. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 631–640, 2006.
[27] D. S. Hochbaum. Heuristics for the fixed cost median problem. *Math. Program.*, 22(1):148–162, 1982.
[28] C.-Y. Hsu, P. Indyk, D. Katabi, and A. Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.
[29] S. Im, R. Kumar, M. M. Qaem, and M. Purohit. Non-clairvoyant scheduling with predictions. In K. Agrawal and Y. Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 285–294. ACM, 2021.
[30] M. Imase and B. M. Waxman. Dynamic steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991.
[31] P. Indyk, A. Vakilian, and Y. Yuan. Learning-based low-rank approximations. *CoRR*, abs/1910.13984, 2019.
[32] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *J. ACM*, 50(6):795–824, 2003.
[33] K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 731–740, 2002.
[34] K. Jain and V. V. Vazirani. 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.
[35] Z. Jiang, D. Panigrahi, and K. Sun. Online algorithms for weighted paging with predictions. In A. Czumaj, A. Dawar, and E. Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 69:1–69:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
[36] M. Karpinski and A. Zelikovsky. New approximation algorithms for the steiner tree problems. *J. Comb. Optim.*, 1(1):47–65, 1997.
[37] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. *J. Algorithms*, 37(1):146–188, 2000.
[38] S. Lattanzi, T. Lavastida, B. Moseley, and S. Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, New Orleans, LA, USA, January 5 - 8, 2020.*, 2020.
[39] S. Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Inf. Comput.*, 222:45–58, 2013.
[40] T. Lykouris and S. Vassilvtiskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3302–3311, 2018.
[41] M. Mahdian, Y. Ye, and J. Zhang. A 2-approximation algorithm for the soft-capacitated facility location problem. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques, 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2003 and 7th International Workshop on Randomization and Approximation Techniques in Computer Science, RANDOM 2003, Princeton, NJ, USA, August 24-26, 2003, Proceedings*, volume 2764 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 2003.
[42] M. Mahdian, Y. Ye, and J. Zhang. Approximation algorithms for metric facility location problems. *SIAM J. Comput.*, 36(2):411–432, 2006.
[43] A. M. Medina and S. Vassilvitskii. Revenue optimization with approximate bid predictions. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1858–1866, 2017.
[44] A. Meyerson. Online facility location. In *FOCS*, pages 426–431, 2001.
[45] V. S. Mirrokni, S. O. Gharan, and M. Zadimoghaddam. Simultaneous approximations for adversarial and

stochastic online budgeted allocation. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1690–1701, 2012.
[46] M. Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, pages 464–473, 2018.
[47] M. Mitzenmacher. Scheduling with predictions and the price of misprediction. In T. Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 14:1–14:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
[48] H. J. Prömel and A. Steger. A new approximation algorithm for the steiner tree problem with performance ratio 5/3. *J. Algorithms*, 36(1):89–101, 2000.
[49] M. Purohit, Z. Svitkina, and R. Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.
[50] G. Robins and A. Zelikovsky. Tighter bounds for graph steiner tree approximation. *SIAM J. Discrete Math.*, 19(1):122–134, 2005.
[51] D. Rohatgi. Near-optimal bounds for online caching with machine learned advice. In S. Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1834–1845. SIAM, 2020.
[52] D. Segev and G. Segev. Approximate k-steiner forests via the lagrangian relaxation technique with internal preprocessing. In *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, pages 600–611, 2006.
[53] D. B. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 265–274, 1997.
[54] M. Sviridenko. An improved approximation algorithm for the metric uncapacitated facility location problem. In *Integer Programming and Combinatorial Optimization, 9th International IPCO Conference, Cambridge, MA, USA, May 27-29, 2002, Proceedings*, pages 240–257, 2002.
[55] S. Umboh. Online network design algorithms via hierarchical decompositions. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 1373–1387, Philadelphia, PA, USA, 2015. Society for Industrial and Applied Mathematics.
[56] A. Wei. Better and simpler learning-augmented online caching. In J. Byrka and R. Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 60:1–60:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
[57] G. Xu and J. Xu. An improved approximation algorithm for uncapacitated facility location problem with penalties. *J. Comb. Optim.*, 17(4):424–436, 2009.
[58] A. Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9(5):463–470, 1993.

A Additional Proofs from Section 2

Proof. [Proof of Lemma 2.1] If $\gamma u \geq |\hat{R}|$, then PARTIAL returns $\emptyset$, which satisfies both claims of the lemma. Assume henceforth that $\gamma u < |\hat{R}|$.

To prove the first claim, consider the two cases in the subroutine. If the returned solution is S_2, then the number of unsatisfied requests is u_2, which is at most $\gamma u \leq 2\gamma u$, as required. Otherwise, S_1 is returned, and thus $\gamma u \geq \frac{u_1+u_2}{2}$. This implies that $2\gamma u \geq u_1 + u_2 \geq u_1$, which completes the proof of the first claim.

We now prove the second claim. Fix i to be the index chosen in partial, such that $S_2 = \mathrm{PC}(\hat{R}, \pi_{2^i})$ does not satisfy at most γu requests from $\hat{R}$. Fix u^* to be the number of requests in $\hat{R}$ which are not satisfied by S^* (such that $u^* \leq u$). Observe that S_1 defines a solution of cost $c(S_1) + 2^{i-1} \cdot u_1$ to the prize-collecting problem with penalty function $\pi_{2^{i-1}}$. Since S_1 was chosen by PC for that penalty,

Property 2.2 yields

$$c(S_1) + 2^{i-1} \cdot u_1 \leq \gamma \cdot (c(S^*) + 2^{i-1}u^*), \tag{A.1}$$

where the inequality used the fact that S^* also defines a solution for the prize-collecting problem.
If $S = S_1$, then since $u_1 \geq \gamma u$, we have

$$c(S_1) + \gamma \cdot 2^{i-1}u^* \leq c(S_1) + \gamma \cdot 2^{i-1}u \leq c(S_1) + 2^{i-1}u_1.$$

Combining the previous equation with Eq. (A.1) yields $c(S_1) \leq \gamma \cdot c(S^*)$, as required.
The second case is that $S = S_2$. In this case, we repeat the argument of Eq. A.1 for S_2 with penalty 2^i:

$$\begin{aligned}\gamma \cdot \left(c(S^*) + 2^i \cdot u^*\right) &\geq c(S_2) + 2^i \cdot u_2 = c(S_2) + 2^i\gamma u + 2^i \cdot (u_2 - \gamma u) \\ &\geq c(S_2) + 2^i\gamma u + 2^i \cdot \frac{u_2 - u_1}{2} \quad \left(\text{since } \gamma u < \frac{u_1 + u_2}{2}\right).\end{aligned}$$

This yields $c(S_2) \leq \gamma \cdot c(S^*) + \gamma 2^i(u^* - u) + 2^i \cdot \frac{u_1 - u_2}{2} \leq \gamma \cdot c(S^*) + 2^i \cdot \frac{u_1 - u_2}{2}$. Now, due to Eq. A.1, we have

$$\gamma \cdot c(S^*) \geq \underbrace{c(S_1)}_{\geq 0} + 2^{i-1}(u_1 - \gamma u^*) \geq 2^{i-1}(u_1 - \gamma u) \geq 2^{i-1} \cdot \frac{u_1 - u_2}{2}.$$

We thus get that $c(S_2) \leq 3\gamma \cdot c(S^*)$, completing the proof of the second claim.

□

Proof. [Proof of Lemma 2.2] **Proof of Item 1.** First, we claim that the total cost of the first $i - 1$ iterations is at most $(6\gamma + 2) \cdot \hat{B}_{i-1}$.

To prove this claim, consider that at any point during the algorithm, the value of B is exactly the total cost incurred by all instances of ON in Line 6 until that point. Thus, B_{i-1} is the cost incurred in the first $i - 1$ iterations due to Line 6. Since at the beginning (and end) of each iteration it holds that $B < 2\hat{B}$, we have $B_{i-1} \leq 2\hat{B}_{i-1}$.

As for the cost of adding items from calls to PARTIAL in Line 10, let $S_1, ..., S_\ell$ be the solutions added in Line 10 in the first $i - 1$ iterations, in order of their addition. Observe that $c(S_j) \leq 3\gamma\hat{B}_{i-1} \cdot 2^{-(\ell-j)}$. Thus, the sum of costs of those solutions is at most $6\gamma\hat{B}_{i-1}$.

Now, we claim that the cost of iteration i is at most $O(1) \cdot \mathrm{OPT} + 6\gamma \cdot \hat{B}_{i-1}$. To prove this, consider the cost of Line 6 in iteration i, in which the online algorithm ON serves some request r_i. The request r_i is a part of a phase of requests which are served by this specific instance of ON (between two subsequent resets of ON in Line 11); call the requests of this phase R'. The requests of R' are all served by ON in the modified input, in which the cost of some set of elements S_0 is set to 0 (specifically, S_0 is the value of the variable S immediately after its augmentation with S_ℓ). Using Property 2.1, we have that $\mathrm{ON}(r_i) \leq O(1) \cdot \mathrm{OPT}'$, where $\mathrm{ON}(r_i)$ is the cost incurred by ON when serving r_i, and OPT' is the optimal solution for R' (with the cost of elements in S_0 set to 0). Clearly, $\mathrm{OPT}' \leq \mathrm{OPT}$, which yields that $\mathrm{ON}(r_i) \leq O(1) \cdot \mathrm{OPT}$.

As for the solution added in Line 10, its cost is at most $3\gamma\hat{B}_i = 3\gamma B_i$. Consider that $B_i = B_{i-1} + \mathrm{ON}(r_i) \leq 2\hat{B}_{i-1} + O(1) \cdot \mathrm{OPT}$. This completes the proof of the first item.

Proof of Item 2. First, we claim that the total cost of PARTIAL solutions from iteration $i + 1$ onward is at most $6\gamma \cdot \sum_{j=0}^{m} \mathrm{ON}_j$.

Observe that for every j such that $1 \leq j \leq m$, we have that $\hat{B}_{i_j^\star} \geq 2\hat{B}_{i_{j-1}^\star}$. Thus,

$$\hat{B}_{i_j^\star} - \hat{B}_{i_{j-1}^\star} \geq \hat{B}_{i_{j-1}^\star} = \hat{B}_{i_j^\star} - \left(\hat{B}_{i_j^\star} - \hat{B}_{i_{j-1}^\star}\right),$$

and therefore $\hat{B}_{i_j^\star} \leq 2\left(\hat{B}_{i_j^\star} - \hat{B}_{i_{j-1}^\star}\right) = 2\left(B_{i_j^\star} - B_{i_{j-1}^\star}\right) = 2 \cdot \mathrm{ON}_{j-1}$.

Consider that for every j such that $0 \leq j \leq m$, the cost of the PARTIAL solution bought in iteration $i_j^\star$ is at most $3\gamma \cdot \hat{B}_{i_j^\star}$. Thus, the total cost of PARTIAL solutions bought from iteration $i+1$ onward is at most

$$6\gamma \cdot \sum_{j=0}^{m-1} \mathrm{ON}_j \leq 6\gamma \cdot \sum_{j=0}^{m} \mathrm{ON}_j.$$

Which completes the proof of the claim.

Combining this claim with the definition of ON_j for every j, we have that the total cost of the algorithm from iteration $i+1$ onwards is at most $(6\gamma+1)\sum_{j=0}^{m} \mathrm{ON}_j$. Now, we claim that $\sum_{j=0}^{m-2} \mathrm{ON}_j \leq \mathrm{ON}_{m-1}$; this would imply that $\sum_{j=0}^{m} \mathrm{ON}_j \leq 3\max\{\mathrm{ON}_{m-1}, \mathrm{ON}_m\}$, which completes the proof.

To show that $\sum_{j=0}^{m-2} \mathrm{ON}_j \leq \mathrm{ON}_{m-1}$, observe that for every $j \in [m-1]$, it holds that $\mathrm{ON}_j = \hat{B}_{i_{j+1}^\star} - \hat{B}_{i_j^\star}$. Summing over j from 0 to $m-2$ yields a telescopic sum:

$$\sum_{j=0}^{m-2} \mathrm{ON}_j = \hat{B}_{i_{m-1}^\star} - \hat{B}_{i_0^\star} \leq \hat{B}_{i_{m-1}^\star}.$$

Now, using the same property for $j = m-1$, we get $\mathrm{ON}_{m-1} = \hat{B}_{i_m^\star} - \hat{B}_{i_{m-1}^\star} \geq 2\hat{B}_{i_{m-1}^\star} - \hat{B}_{i_{m-1}^\star} = \hat{B}_{i_{m-1}^\star}$. Thus, we get

$$\mathrm{ON}_{m-1} \geq \sum_{j=0}^{m-2} \mathrm{ON}_j.$$

□

B Online Priority Steiner Forest with Predictions

A generalization of the Steiner forest problem is priority Steiner forest. In this problem, each edge $e \in E$ has an associated integer priority $\alpha_e \in [b]$. In addition, every request $r \in R$ contains, in addition to the pair of terminals to connect, a priority demand which we also denote by $\alpha_r \in [b]$. A feasible solution for this problem is such that for every request r of terminals s, t there exists a path connecting s, t comprising only edges of priority at least α_r. Throughout the discussion of the priority Steiner forest problem, we partition the input $R = \left(r_1, \cdots, r_{|R|}\right)$ into $\{R_j\}_{j=1}^{b}$, where R_j is the subset of requests of priority j.

To define errors in priority Steiner forest, we must define the matching cost between two requests r_1 and r_2. If $\alpha_{r_1} \neq \alpha_{r_2}$, we say that the matching cost is infinite. In other words, we do not allow matching requests of different priorities. If both r_1 and r_2 have the same priority j, then we define the matching cost between r_1 and r_2 to be the distance between the two pairs, as previously defined for Steiner forest. However, this distance is not computed over the original graph G, but rather its subgraph G_j, which only contains the edges in G which have priority at least j. Note the intuition for this definition – there could be some very low priority class in which all points are very close, but this does not ameliorate the error for higher-priority requests.

Divide R into $R_1, \cdots, R_b$ according to priority class, and divide $\hat{R}$ into $\hat{R}_1, \cdots, \hat{R}_b$ in a similar way. Consider the following algorithm for priority Steiner forest with predictions:

1. Run b parallel instances of the algorithm for Steiner forest with predictions, such that the j'th instance runs on the graph G_j (here again G_j is the subgraph of edges of priority at least j), and is given the prediction $\hat{R}_j$.
2. Upon the release of a request of priority j, send it to the j'th instance.
3. At any point during the sequence, maintain the union of the solutions held by all b instances.

THEOREM B.1. *For every error $\lambda = (\Delta, D)$, the algorithm above for priority Steiner forest with predictions yields the following:*

$$\mathrm{ALG} \leq O\left(b \log\left(\frac{\min\{|R|, \Delta\}}{b} + 2\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D$$

Proof. Observe that for every priority class j, restricting the error λ to R_j and $\hat{R}_j$ induces a Steiner forest error λ_j for the j'th instance, such that $\lambda_j = (\Delta_j, D_j)$ where $\sum_{j=1}^{b} \Delta_j = \Delta$ and $\sum_{j=1}^{b} D_j = D$. Now, one can apply Theorem 4.1 for every j, and obtain $\mathrm{ALG}_j \leq O\left(\log\left(\min\{|R_j|, \Delta_j\} + 2\right)\right) \cdot \mathrm{OPT}_j + O(1) \cdot D_j$, where ALG_j is the cost of the Steiner forest with predictions algorithm on the requests of R_j (given prediction $\hat{R}_j$), and OPT_j is the cost of the optimal solution for that same Steiner forest instance. Summing over j, we obtain

$$\begin{aligned}
\mathrm{ALG} &\leq \sum_{j=1}^{b} \mathrm{ALG}_j \\
&\leq \sum_{j=1}^{b} O\left(\log\left(\min\{|R_j|, \Delta_j\} + 2\right)\right) \cdot \mathrm{OPT}_j + \sum_{j=1}^{b} O(1) \cdot D_j \\
&\leq \sum_{j=1}^{b} O\left(\log\left(\min\{|R_j|, \Delta_j\} + 2\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D \\
&\leq O\left(b \log\left(\frac{\sum_{j=1}^{b}\left(\min\{|R_j|, \Delta_j\} + 2\right)}{b}\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D \\
&\leq O\left(b \log\left(\frac{\min\left\{\sum_{j=1}^{b}|R_j|, \sum_{j=1}^{b}\Delta_j\right\}}{b} + 2\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D \\
&= O\left(b \log\left(\frac{\min\{|R|, \Delta\}}{b} + 2\right)\right) \cdot \mathrm{OPT} + O(1) \cdot D
\end{aligned}$$

where the third inequality is due to the fact that $\mathrm{OPT}_j \leq \mathrm{OPT}$ for every j and that $\sum_{j=1}^{b} D_j = D$, and the fourth inequality is due to Jensen's inequality and the concavity of log. This completes the proof of the theorem. ∎

C Online Facility Location Algorithm [18] is NOT Subset Competitive

In this section, we show that the algorithm of Fotakis for online facility location [18] is not subset-competitive. This algorithm is denoted by ON^{FL}, and described in Section 5. This deterministic algorithm is $O(\log |R|)$-competitive. To show that it is not subset competitive, we show that for every constant c there exists an input R and a subsequence $R' \subseteq R$ such that $\text{ON}(R') > c \cdot \log |R'|$.

We consider the lower bound of $\Omega(\frac{\log |R|}{\log \log |R|})$ of [19]. The lower bound consists of an input on which every algorithm is $\Omega(\frac{\log k}{\log \log k})$-competitive, where k is such that $|R| \leq k$.

The input is composed of a full binary tree of height $m = \frac{\log k}{\log \log k}$, where the cost to open a facility at each node is f. The weights on the edges of the tree are equal for each level, and decrease exponentially by a factor of m in each level, where the edges going out of the root have a weight of $\frac{f}{m}$. The request sequence is composed of $m+1$ phases, where the i'th phase releases m^{i-1} requests on a node v_i at depth $i-1$. The first phase releases a single request on the root v_0; For each subsequent phase i, the node v_i is the child node of v_{i-1} such that the algorithm did not open a facility in the subtree of that child node (if the algorithm opened a facility in the subtrees of both child nodes, v_i is an arbitrary child node of v_{i-1}).

Now, consider the operation of the online algorithm of [18] on this input. Upon the single request of the first phase, the algorithm opens a facility at the root v_1, and connects that request. The potential of all points at the end of this phase is zero.

At each subsequent phase i, the algorithm connects all requests except for the last request to the facility at v_{i-1}. At the last request of the phase, the potential of v_i at this point is $m^{i-1} \cdot \frac{f}{m^{i-1}} = f$, and thus a facility is opened at v_i. Note that the total connection cost of this phase is $(m^{i-1}-1) \cdot \frac{f}{m^{i-1}}$, so at most f.

Overall, the algorithm opened facilities at $v_1, \cdots, v_{m+1}$, at an opening cost which is more than its connection cost, and is thus at least half its total cost. From the guarantee of the lower bound, the opening cost of the algorithm is thus $\Omega(\frac{\log k}{\log \log k}) \cdot \text{OPT}$, where k is at least the number of requests, and thus $k \geq m^m$.

Now, consider the subsequence R' which consists of the last request in each phase. There are $m+1$ such requests, but the entire facility opening cost of the algorithm is incurred on those requests. Thus, $\text{ON}^{\text{FL}}(R') = \Omega(\frac{m \log m}{\log m}) = \Omega(m) = \Omega(|R'|)$. In particular, $\text{ON}^{\text{FL}}(R') = \omega(\log |R'|)$, proving that the algorithm is not subset-competitive.

D Lower Bounds for Online Algorithms with Predictions

In this section, we show additional lower bounds for the Steiner tree, Steiner forest and facility location in the online algorithms with predictions setting.

D.1 Steiner Tree

We start with a lower bound for Steiner tree, which also applies to Steiner forest. Suppose we are given the parameters n, k, Δ_1, Δ_2. We define a $(n, k, \Delta_1, \Delta_2)$-adversary to be an adversary for Steiner tree with predictions that gives a prediction $\hat{R}$ and a request sequence R such that $|\hat{R}| = n$, $|R| = k$, $|\hat{R} \backslash R| = \Delta_1$, $|R \backslash \hat{R}| = \Delta_2$. The following theorem yields the desired lower bound.

THEOREM D.1. *For every n, k, Δ_1, Δ_2, there exists a $(n, k, \Delta_1, \Delta_2)$-adversary for Steiner tree with predictions such that every randomized algorithm on this instance is $\Omega(\min(k, \Delta))$-competitive, where $\Delta = \Delta_1 + \Delta_2$.*

For this theorem, we use the lower bound for the online Steiner tree problem, due to Imase and Waxman [30], which we restate for the rooted version of Steiner tree.

THEOREM D.2. *For every m, there exists a graph $G_m = (V, E)$ of $m^2 + m$ nodes, a root $\rho \in V$ and an oblivious adversary* ADV *that releases a sequence of requests R such that $|R| = m$ on which every algorithm is $\Omega(\log m)$ competitive.*

Proof. The construction of [30] consists of an adversary I_i for every i such that every algorithm is $\Omega(i)$ competitive for I_i. The instances are built recursively – I_0 consists of a single edge between the root and another node. Each I_i is constructed from I_{i-1} by breaking each edge into a diamond – that is, adding two nodes and replacing the edge with 4 edges.

Therefore, the number of edges in I_i is 4^i. The number of nodes in I_i is the number of nodes in I_{i-1} plus twice the number of edges in I_{i-1} – thus, at most 4^i for every $i \geq 1$. The number of requests released in I_i is exactly 2^i. Thus, the number of nodes is at most the square of the number of requests.

Now, to obtain the desired instance, set $i = \lfloor \log m \rfloor$. The number of nodes will be $4^i \leq m^2$. Pad the number of nodes to exactly $m^2 + m$ by adding copies of the root ρ. Pad the number of requests to exactly m by requesting some of those copies of ρ. ∎

Fix any values of n, k, Δ_1, Δ_2. Define $\ell = \frac{n+k-\Delta_1-\Delta_2}{2}$, the size of the intersection of R and $\hat{R}$ in the adversary. Lemmas D.1 and D.2 imply Theorem D.1.

LEMMA D.1. *There exists an oblivious $(n, k, \Delta_1, \Delta_2)$-adversary on which every algorithm is $\Omega(\log \Delta_2)$-competitive.*

LEMMA D.2. *There exists an oblivious $(n, k, \Delta_1, \Delta_2)$-adversary on which every algorithm is $\Omega(\log(\min(\ell, \Delta_1)))$-competitive.*

Proof. [Proof of Theorem D.1] If $\Delta_2 \geq \min(\ell, \Delta_1)$, then $2\Delta_2 \geq \min(\ell + \Delta_2, \Delta_1 + \Delta_2) = \min(k, \Delta)$. Thus, the adversary of Lemma D.1 yields the desired result.

Otherwise, $\Delta_2 \leq \min(\ell, \Delta_1)$. Using $\Delta_2 \leq \ell$, we have that $\ell \geq \frac{\ell+\Delta_2}{2} = \frac{k}{2}$. Using $\Delta_2 \leq \Delta_1$, we have that $\Delta_1 \geq \frac{\Delta_1+\Delta_2}{2} = \frac{\Delta}{2}$. Thus, $\min(\ell, \Delta_1) \geq \frac{\min(k,\Delta)}{2}$, which completes the proof of the theorem. ∎

It remains to prove Lemmas D.1 and D.2.

Proof. [Proof of Lemma D.1] We construct the adversary in the following way. The graph on which the requests are released is the graph G_m of Theorem D.2 for $m = \Delta_2$, with multiple copies of the root ρ. The n predictions of $\hat{R}$ all arrive on copies of ρ. The input sequence R starts with ℓ requests on predicted copies of ρ (which have a connection cost of 0 for the optimal solution). We now use the Imase-Waxman adversary on G_m to release the remaining $k - \ell = \Delta_2$ requests, on which any algorithm is $\Omega(\log \Delta_2)$ competitive. ∎

Proof. [Proof of Lemma D.2] We construct the adversary in the following way. The graph on which the requests are released is the graph G_m for $m = \min(\ell, \sqrt{n - \ell})$, with multiple copies of the root node ρ. The prediction $\hat{R}$ consists of all $m^2 + m$ nodes of G_m (note that $m^2 + m \leq n$), as well as $n - (m^2 + m)$ copies of ρ.

The adversary starts the request sequence by requesting Δ_1 unpredicted copies of ρ. Serving these requests costs 0 to the optimal solution. It then requests $\ell - m$ *predicted* copies of ρ (note that there exist $\ell - m$ predicted copies of ρ, since $n - (m^2 + m) \leq \ell - m$.

It remains to release m predicted requests. The adversary now calls the Imase-Waxman adversary of Theorem D.1 to release these m requests on G_m.

Every algorithm is $\Omega(\log(m))$ competitive on this adversary. But $m = \min(\ell, \sqrt{n-\ell}) \geq \sqrt{\min(\ell, n-\ell)} = \sqrt{\min(\ell, \Delta_1)}$, and thus $\log m \geq \frac{\log(\min(\ell,\Delta_1))}{2}$. This completes the proof. ∎

D.2 Facility Location We now give a similar result to the previous lower bound for facility location.

THEOREM D.3. *For every n, k, Δ_1, Δ_2, there exists a $(n, k, \Delta_1, \Delta_2)$-adversary for facility location with predictions such that every randomized algorithm on this instance is $\Omega\left(\frac{\log \min(k,\Delta)}{\log\log \min(k,\Delta)}\right)$-competitive, where $\Delta = \Delta_1 + \Delta_2$.*

The following lower bound for online facility location is due to Fotakis [19].

THEOREM D.4. *For every m, there exists a graph $G_m = (V, E)$ of $m^2 + m$ nodes and an oblivious adversary* ADV *that releases a sequence of requests R such that $|R| = m$ on which every algorithm is $\Omega\left(\frac{\log m}{\log\log m}\right)$ competitive.*

Proof. The adversary given in [19] gives a graph which is a complete binary tree of some depth h. For $a = \frac{\log m}{\log\log m}$, each node of depth b in the tree has a^b copies. The adversary first requests the root, then requests the a copies of a node of depth 1, then the a^2 copies of a node of depth 2, and so on. The total number of requests released is at most m. The number of nodes at each depth b is $2^b \cdot a^b \leq (a^b)^2$, and thus the number of nodes in the graph is at most m^2. The analysis of [19] guarantees that every algorithm is $\Omega\left(\frac{\log m}{\log\log m}\right)$ competitive on this instance.

To reach the exact desired number of nodes and requests, we pad the number of nodes to exactly m^2+m by adding nodes of facility opening cost 0 at infinite distance from the rest of the construction. We pad the number of requests to m by requesting these nodes added in the padding. ∎

Fix any values of n, k, Δ_1, Δ_2. Define $\ell = \frac{n+k-\Delta_1-\Delta_2}{2}$, the size of the intersection of R and $\hat{R}$ in the adversary. The following two lemmas are analogous to the lemmas from the Steiner tree lower bound, and imply Theorem D.3 in the same way that their analogues implied Theorem D.1.

LEMMA D.3. (ANALOGUE OF LEMMA D.1) *There exists an oblivious $(n, k, \Delta_1, \Delta_2)$-adversary on which every algorithm is $\Omega\left(\frac{\log \Delta_2}{\log\log \Delta_2}\right)$-competitive.*

Proof. The proof is similar to the proof of Lemma D.1. The instance comprises the graph G_m as defined in Theorem D.4, for $m = \Delta_2$, and from multiple nodes of facility cost 0 and distance ∞ from every other node. The n predictions are all of nodes at distance ∞, and so are the first ℓ requests. The following Δ_2 requests are given to the adversary of G_m. ∎

LEMMA D.4. (ANALOGUE OF LEMMA D.2) *There exists an oblivious $(n, k, \Delta_1, \Delta_2)$-adversary on which every algorithm is $\Omega\left(\frac{\log \min(\ell,\Delta_1)}{\log\log \min(\ell,\Delta_1)}\right)$-competitive.*

Proof. [Proof of Lemma D.4] The proof is similar to the proof of Lemma D.2, with the only difference being as in the proof of Lemma D.1 – we use nodes at distance infinity with facility cost 0 to pad predictions and requests, in lieu of the copies of the root used for the Steiner tree problem. ∎

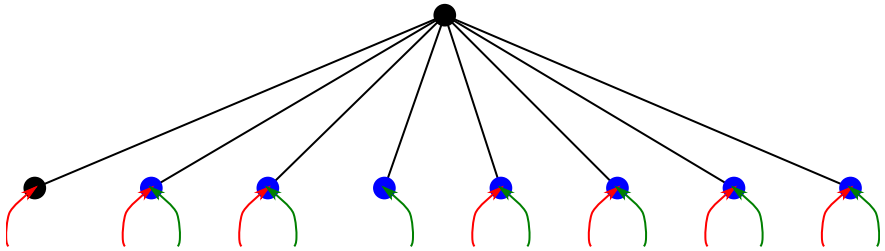

Figure 4: This figure visualizes the lower bound for online metric matching, which applies also for metric matching with predictions. The figure shows an unweighted star graph, where all but one of its spokes have blue points. The green arrows are the predicted red points, while the red arrows are the actual red points in the input.

E Lower Bound for Online Metric Matching with Predictions

In this section, we consider the online metric matching problem, and show that in contrast to the network design problems in this paper, it has no tolerance for errors. Specifically, we show that while the offline problem can be solved optimally, the online problem with predictions performs as badly as the online problem without predictions, even in the presence of *only 2 prediction errors*.

In the online metric matching problem, one is given a metric space $M = (X, d)$, as well as a set of k "blue" points in M. One after the other, "red" points are released, and the algorithm must match each arriving red point to an unmatched blue point, at a cost which is the distance in M between the two points. In this online version, after the algorithm matches two points, that matching cannot be undone.

The best known lower bounds for this problem, both in the randomized and deterministic settings, are both on a metric induced by an unweighted star graph with $k + 1$ spokes (each edge has a cost of 1), where the blue points are on k spokes of the graph. The first red point is released on the remaining spoke, which is matched to a blue point by the algorithm.

For a deterministic algorithm, the lower bound proceeds as follows: the adversary releases a red point on the blue point to which the algorithm has matched the previous red point, and repeats this for $k-1$ iterations. In the end of the instance, there remains only a single blue point v on which no red point has been released. The cost of the algorithm for this sequence is $2k$, as the cost of each matching is 2. The optimal solution, on the other hand, has a cost of 2 - it matches the first red request to v at a cost of 2, and matches each remaining red point to its colocated blue point. This example thus shows a lower bound of $\Omega(k)$ on the competitive ratio of a deterministic algorithm in this setting.

Now, consider the online algorithm augmented with a prediction for the red points, where the prediction predicts a single red point on every blue point. Clearly, this prediction does not provide any information to the algorithm, and does not improve its performance on the input. However, it has $(2, 0)$ error - one unpredicted red point arrives, and one predicted red point does not arrive (as the remaining points arrive exactly, the matching cost is 0). This example is shown in Figure 4.

In the randomized setting, the outcome is similar. In the randomized setting, the adversary is oblivious, releasing a red point on a random blue point, chosen uniformly from the blue points on which no red point has been released. A standard survivor-game analysis yields $\Omega(\log k)$ competitiveness on this instance; as before, predicting all the blue points as input does not help competitiveness, but has an error of $(2, 0)$.