Competitive strategies to use "warm start" algorithms with predictions*

Avrim Blum† Vaidehi Srinivas‡

Abstract

We consider the problem of learning and using predictions for *warm start* algorithms with predictions. In this setting, an algorithm is given an instance of a problem, and a *prediction* of the solution. The runtime of the algorithm is bounded by the distance from the predicted solution to the true solution of the instance. Previous work has shown that when instances are drawn iid from some distribution, it is possible to learn an approximately optimal fixed prediction [DIL+21], and in the adversarial online case, it is possible to compete with the best fixed prediction in hindsight [KBTV22].

In this work we give competitive guarantees against stronger benchmarks that consider a set of k predictions $\mathbf{P}$. That is, the "optimal offline cost" to solve an instance with respect to $\mathbf{P}$ is the distance from the true solution to the *closest* member of $\mathbf{P}$. This is analogous to the *k-medians* objective function. Our first two results are largely conceptual and demonstrate the promise of using k predictions. In the distributional setting, we show a simple strategy that incurs cost that is at most an $O(k)$ factor worse than the optimal offline cost. We then show a way to leverage learnable coarse information, in the form of partitions of the instance space into groups of "similar" instances, that allows us to potentially avoid this $O(k)$ factor.

Finally, as our main technical contribution, we consider an online version of the problem, where we compete against offline strategies that are allowed to maintain a *moving* set of k predictions or *trajectories*, and are charged for how much the predictions move. We give an algorithm that does at most $O(k^4 \ln^2 k)$ times as much work as any offline strategy of k trajectories. This algorithm is deterministic (robust to an adaptive adversary), and oblivious to the setting of k. Thus the guarantee holds for all k simultaneously.

1 Introduction

Warm start algorithms are a practically motivated paradigm of algorithms that make use of *predictions* to improve runtime. It is the most common framework in which *algorithms with predictions* (also *learning-augmented algorithms*, see [MV21]) are studied for static problems.[1] A warm start algorithm solves an instance I from an instance space $\mathbf{I}$ to find a solution S in a solution space $\mathbf{S}$. In addition to I, the algorithm is given a prediction $P \in \mathbf{S}$, which can be thought of as a "guess" of the solution. The runtime of the algorithm depends on how far S is from P. This has applications in settings in which one must solve a series of related instances of a problem, and can learn information about "typical" solutions. An example is a network routing problem that must be solved daily, for which the user knows that "today's network traffic is not too different than yesterday's."

In this setting, the solution space $\mathbf{S}$ is a metric space with distance $\mathbf{d}(\cdot, \cdot)$, and the runtime of an algorithm on instance I with prediction P can be bounded by $\mathbf{d}(P, S)$, where S is the true solution of instance I. (We wrap factors that depend on the size of the instance and other details into the distance function $\mathbf{d}$.) Examples that fit in this framework include algorithms for bipartite matching [DIL+21, CSVZ22], network flows [PZ22, DMVW23], and some related problems that can be solved via convex minimization [SO22, OS23].

Given warm starts as an algorithmic primitive, the question remains how to obtain and use high quality predictions to achieve good performance. Previous work on this problem has focused on competing with the best fixed prediction in hindsight. In the distributional setting, works including [DIL+21, DMVW23, CSVZ22] show that it is possible to PAC-learn a good fixed prediction to use with future instances. In the online setting, [KBTV22] show that it is possible to compete with the best fixed prediction in hindsight, and their work extends to algorithms with predictions in settings beyond warm starts.

---

*The full version of the paper can be accessed at https://arxiv.org/abs/2405.03661

†Toyota Technological Institute at Chicago: avrim@ttic.edu, supported in part by the National Science Foundation under grants CCF-2212968 and ECCS-2216899.

‡Northwestern University: vaidehi@u.northwestern.edu, supported by the National Science Foundation (NSF) under grants EECS-2216970 and CCF-2154100, and the Northwestern Presidential Fellowship. Part of this work was done while V.S. was visiting TTIC (Toyota Technological Institute at Chicago), as part of the IDEAL (Institute for Data, Econometrics, Algorithms, and Learning) summer exchange program.

[1]By "static," we mean problems for which we are given all inputs up front, and we wish to minimize runtime. This is as opposed to work in online, streaming, dynamic, and approximation algorithms.

These results demonstrate settings in which warm starts can give us an advantage. However, to see a large advantage, we must be in a setting where the best fixed prediction leads to much better performance than a generic default prediction. This means that the solutions of most instances coming from our distribution must be very close to the prediction, and therefore very similar to each other, which is a restrictive assumption. Our work explores less restrictive structure that we can take advantage of to achieve good performance.

We observe that warm start algorithms have many properties that we can take advantage of, that make them powerful and flexible. They are packaged local search routines that allow you to search the solution space by conceptually growing a ball around a prediction, where the radius of a ball corresponds to the runtime of the warm start algorithm. It is also possible to run multiple instantiations of the algorithm in parallel, and simultaneously search the space from multiple points. Another useful property of warm starts is that every time we solve an instance, we learn the optimal prediction for that instance, i.e. the solution. These observations motivate our main question:

Are there settings in which we can use warm-start algorithms to achieve significantly better performance than any single fixed prediction?

In this work, we leverage these properties of warm starts to compete against a variety of stronger baselines, answering this question in the affirmative. In particular, we consider competing with a set of k predictions $\mathbf{P} \in \mathbf{S}^k$. That is, the cost of $\mathbf{P}$ with respect to an instance I with solution S is the distance from S to the closest prediction: $\min_{P \in \mathbf{P}} \mathbf{d}(S, P)$. This is analogous to the *k-medians* objective. We also consider competing with the best *function* h from some given class, mapping each instance to one of k predictions. Our main technical contribution is designing an algorithm for the online setting.

It is somewhat surprising that it is possible to achieve any non-trivial guarantees against multiple points, especially in the online setting. Previous work approaches this problem by reducing to online convex optimization [KBTV22]. We note that in the standard online convex optimization setting, we cannot expect to compete with a collection of multiple points and achieve vanishing regret. In particular, even going from 1 trajectory to 2 trajectories makes the offline problem of choosing the best trajectories go from being convex to being non-convex. We observe that vanishing regret is actually a stronger property than what we need. By accruing constant factors in runtime, and taking advantage of structural properties such as running algorithms in parallel, we show that we can compete with far stronger baselines.

We note that there is a line of work that studies competing against multiple predictions for algorithms with predictions for *online* problems, see [AGKP22, DIL+22, ACE+23], which has a different set of challenges and techniques from the warm start setting, which are discussed more in Section 1.1. Our work is, to the best of our knowledge, the first to consider this problem for warm starts.

Competing against multiple points offline. Our first setting is simple, and has been observed in specific settings in prior work (see e.g. section 3 of [DIL+22]), but we find it beneficial to provide a formal treatment as it motivates our other two settings. In this setting, we consider instance-solution pairs drawn from a distribution $\mathcal{D}$ over $\mathbf{I} \times \mathbf{S}$. Here, we can use a simple strategy that learns k fixed predictions for $\mathcal{D}$ from samples. Then, for a subsequent instance $(I, S) \sim \mathcal{D}$, for each prediction we can run an instantiation of our warm start algorithm "in parallel,"[2] and output the solution of the thread that completes first.

Informal Theorem 1.1 (Competing against k fixed points offline, Formally Theorem 3.6)**.** *For a distribution $\mathcal{D}$ over $\mathbf{I} \times \mathbf{S}$ it is possible to learn a set of k predictions $\widehat{\mathbf{P}}$ that is an $O(1)$ approximation to*

$$\operatorname*{argmin}_{\mathbf{P} \in \mathbf{S}^k} \mathop{\mathbb{E}}_{(I,S) \sim \mathcal{D}} \left[\min_{P \in \mathbf{P}} \mathbf{d}(S, P) \right].$$

Furthermore, given a set of predictions $\widehat{\mathbf{P}}$, we can design a procedure such that the time to solve a new instance $I \in \mathbf{I}$ with (unknown) true solution $S \in \mathbf{S}$ is

$$O(k) \cdot \min_{P \in \widehat{\mathbf{P}}} \mathbf{d}(S, P).$$

We note that a $\Omega(k)$ approximation factor in the runtime is necessary without assuming more structure in the problem (Remark 3.7).

[2] By "in parallel," we refer to interleaving the k threads of computation, resulting in a sequential algorithm.

Learning "coarse information." This motivates our second setting, in which we avoid this extra $\Omega(k)$ factor in the runtime. Suppose we had access to extra "coarse information" about the mapping from instances and solutions. A warm start algorithm, or indeed any algorithm, already encodes the perfect mapping between instances and solutions. However, this comes at a high cost to uncover. We model coarse information as a k-wise partition of the instance space $h : \mathbf{I} \to [k]$, where we expect that two instances I_1 and I_2 that map to the same partition have similar solutions. In addition, we expect $h(I)$ to be significantly faster to compute than solving the instance I. Given an h that satisfies these properties, we can learn a fixed prediction $P^{(i)}$ for each partition i of h with respect to a distribution $\mathcal{D}$ over $\mathbf{I} \times \mathbf{S}$. Then, for a new instance $(I, S) \sim \mathcal{D}$, we can use the single prediction $P^{(h(I))}$. This allows us to potentially avoid the $O(k)$ factor in the runtime of the previous theorem.

Thus the question is whether we can learn a partition h of $\mathbf{I}$ that is informative and well-aligned with $\mathcal{D}$. We show that given a class of efficiently-computable candidate partitions $\mathcal{H}$, it is possible to learn an $h \in \mathcal{H}$ that is approximately best-aligned with the distribution $\mathcal{D}$.

Informal Theorem 1.2 (Competing with a hypothesis class of k-wise partitions offline, Formally Theorem 4.10)**.** *For a distribution $\mathcal{D}$ over $\mathbf{I} \times \mathbf{S}$, and a learnable hypothesis class $\mathcal{H}$ of k-wise partitions of $\mathbf{I}$, given access to an appropriate ERM oracle over $\mathcal{H}$, it is possible to learn an $h : \mathbf{I} \to [k]$ and set of predictions $\mathbf{P} = \left(P^{(1)}, \ldots, P^{(k)}\right)$ that are an $O(1)$ approximation to*

$$\min_{h \in \mathcal{H}} \min_{\mathbf{P} \in \mathbf{S}^k} \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} \left[\mathbf{d}(S, P^{(h(I))})\right].$$

Furthermore, given an $h : \mathbf{I} \to [k]$ and $\mathbf{P} = \left(P^{(1)}, \ldots, P^{(k)}\right)$, we can design a procedure such that the expected time to solve a new instance is

$$\mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} \left[\text{runtime}(h(I)) + \mathbf{d}(S, P^{(h(I))})\right].$$

We achieve this via a two step procedure, that first learns an approximately good set of k centers for $\mathcal{D}$, then uses an ERM oracle to find an $h \in \mathcal{H}$ that aligns well with this set of centers. While there may be no reason to expect that the k centers chosen in the first step are similar to the best centers for the best $h \in \mathcal{H}$, we show that this strategy is still approximately optimal.

Competitive algorithms to solve sequences of instances. Finally, as our main technical contribution, we consider an online setting where instances arrive sequentially. One of the main motivations of warm start algorithms is settings in which instances that arrive close in time are likely to be similar. One example is the setting explored in experiments by [DMVW23], in which they use warm starts to speed up solving a series of flow instances that correspond to image segmentation tasks on frames of a video.

We formulate the *online ball search* problem to model this setting. In this problem, on each day there is an hidden point $S \in \mathbf{S}$ (corresponding to the true solution of that day's instance) that the algorithm must find. The algorithm is allowed to search the space $\mathbf{S}$ by growing balls around points of its choosing in $\mathbf{S}$. The algorithm succeeds when one of the balls first contains S. Each of these balls corresponds to running an instance of an algorithm $\mathcal{A}(I, P)$ using a particular prediction P. The radius of the ball corresponds to the number of steps for which the algorithm was run.

In this setting, we design competitive algorithms that compete against the best strategy in hindsight. The strategies that we compete against are collections of k predictions that are allowed to move over time. We call these moving predictions *trajectories*. The cost of a collection of k trajectories is the total *hit cost* plus the total *movement cost*. The hit cost on a given day is the distance from that day's solution to the closest of the trajectories on that day. The total movement cost is the total distance that each of the k trajectories moves over time. This is a generalization of competing with k fixed points, that includes adaptive strategies.

Note that while the baseline strategies that we compete against are constrained to using k predictions, and must pay to move the predictions, the online algorithm has no constraint on how many predictions it can use (i.e., points from which it can search), nor does it pay to move predictions. We note that the restriction on the offline strategies is necessary, because if the offline strategy was allowed to move arbitrarily, it would move perfectly to the solution on each day and achieve cost 0, making it impossible to compete with.

Our objective is to minimize the total runtime to solve a sequence of instances. The runtime to solve the instances corresponds to the sum of radii that the algorithm searches, which is the number of steps it runs of the subroutine $\mathcal{A}$, plus any additional overhead in running the algorithm.

Informal Theorem 1.3 (Competing with trajectories online efficiently, Formally Theorem 5.18)**.** *We have an algorithm for online ball search that is $O(k^4 \ln^2 k)$-competitive with any set of k trajectories, where the total runtime of the algorithm is bounded by $O(1)$ times the number of steps it runs of the algorithms-with-predictions subroutine. That is, this algorithm does at most $O(k^4 \ln^2 k)$ times as much work as any set of k trajectories.*

Furthermore, the algorithm is deterministic and therefore robust to an adaptive adversary, and oblivious to the setting of k, so the guarantee holds for all k simultaneously.

This guarantee states that our online algorithm has runtime comparable, within an $O(k^4 \ln^2 k)$ factor, to any offline strategy that is allowed to maintain k moving predictions, and pay the cost on each day from that days solution to the *closest* of the predictions. In some sense, this algorithm does well when the solutions to the instances we see fall into k (moving) clusters, and our objective is related to the k-medians objective. Thus, it is particularly nice that the algorithm does not need to know k, as in practice we often do not know the number of clusters in our data in advance.

To approach the online problem, it is illustrative to first consider competing with only one trajectory. In this setting, an algorithm that always uses the previous day's solution as the next day's prediction is 2-competitive (via the triangle inequality). Note that this simple guarantee already generalizes what was previously known, as it competes not only with the best fixed prediction in hindsight, but also against adaptive strategies that move over time. For multiple trajectories, this simple strategy is no longer enough, as the previous day's solution could come from some other trajectory, and be arbitrarily far away from today's solution.

Our algorithm for multiple trajectories addresses this by searching from *all* previous solutions in parallel. These "threads" are run at quadratically decaying rates, i.e. the thread of the ith most recent solution is run at rate $\frac{1}{i^2 \ln^2 i}$. This leaves the issue that the previous solution from the same trajectory as today could be arbitrarily far in the past, and be run at an extremely slow rate. This is addressed by "pruning" threads that are no longer fruitful. That is, when the ball searched around a solution i fully contains the ball searched around a previous solution j, the algorithm can stop running thread j, as that work is redundant with thread i. Then, we can increase the rates of the threads that are lower priority than j, and maintain that there is at most one thread being run at each rate $\frac{1}{i^2 \ln^2 i}$ for each integer i. In the analysis we show that either the algorithm runs long enough for the previous solution from the same trajectory as today to be elevated to rate $\geq \frac{1}{k^2 \ln^2 k}$ and eventually solve the instance, or the algorithm terminates more quickly than that, which is even better. This allows us to bound the competitive ratio.

To bound the runtime, we note that the amount of work done by each thread is actually dominated by the work done to check if it was been pruned yet. This blows up the work done by the ith fastest thread by a factor $O(i)$. Since the rate at which the ith thread takes steps of $\mathcal{A}$ is $\frac{1}{i^2 \ln^2 i}$, the rate at which it does work is $O(i) \cdot \frac{1}{i^2 \ln^2 i} = O(\frac{1}{i \ln^2 i})$. Thus, summing the work done over all of the threads results in a convergent series, and we are able to bound the total work done by the algorithm by the work done by the fastest thread.

We remark that if we relax our objective to only be competitive in sum of radii searched, and not worry about the runtime of the algorithm, the problem is still interesting. This could correspond to settings where steps of the subroutine are much more costly than steps of the online scheduling algorithm. In this setting, we can design an algorithm with an improved competitive ratio of $O(k^2)$ based on a reduction to the k-server problem (Corollary 5.8).

In this work, we demonstrate that warm starts are a powerful algorithmic primitive that can, in many settings, be used in ways that are much stronger than competing with one fixed point in hindsight. This opens new research directions in more effective ways to learn and use warm starts. We also observe that the properties of warm starts that we use may hold in many settings that use local-search type algorithms, and we hope that our techniques can be extended to broad-ranging applications.

1.1 Related Work

Algorithms with predictions. *Algorithms with predictions* (also *learning-augmented algorithms*) is a *beyond worst-case* paradigm of algorithm design that has become well-studied in recent years. In this paradigm, an algorithm solicits an untrusted *prediction* to help solve a worst-case instance. Generally speaking, the goals are to provide (i) *consistency*: better performance than a worst-case algorithm when the prediction is of high quality, (ii) *robustness*: performance no worse than a worst-case algorithm when the prediction is of low quality, and (iii) *graceful degradation*: performance degrades smoothly as a function of the quality of the predictions. The type of prediction and how we measure prediction error can vary vastly among different problem settings, as well

as the type of performance that we are trying to optimize (e.g. runtime for warm starts, competitive ratio for online algorithms, memory usage for streaming algorithms). Thus the algorithms and techniques that have been developed for algorithms with predictions are also diverse. For an overview of algorithms with predictions, the reader is referred to the book chapter of Mitzenmacher and Vassilvitskii [MV21], with the note that the body of work in this area has grown significantly even in the few years since this was published.

Data-driven algorithms. *Data-driven algorithm design* is a beyond worst-case paradigm of algorithm design that is very related to algorithms with predictions. In this setting, we consider a *parametrized family* of algorithms for a certain problem, and we wish to learn the best setting of parameters for instances drawn from a distribution $\mathcal{D}$ (or to achieve low regret in the online case). Work in this area typically focuses on proving learning guarantees, e.g., how many samples needed to learn an approximately-optimal setting of parameters. For an overview of data-driven algorithms, the reader is referred to the book chapter of Balcan [Bal21].

We can think of this as a different viewpoint for algorithms with predictions. For an algorithm with predictions $\mathcal{A}(I, P)$ where we can think of predictions as parameters, and each possible prediction P defines an algorithm in the family, i.e. the family of algorithms $\{\mathcal{A}_P(I) = \mathcal{A}(I, P) \mid P \in \mathbf{S}\}$. Work in algorithms with predictions typically focuses on showing that the performance of the algorithms in the family can be significantly better than the performance of a worst-case algorithm. Ideally, we would like to have both types of guarantees: (i) that we can learn good predictions, and (ii) that good predictions enable us to achieve far better performance. Our work attempts to bridge this gap by providing guarantee (i) to problems for which we have guarantee (ii).

Data-driven algorithms are often studied in online settings where one must learn a good parameter setting over time. This is usually studied in the framework of regret, where an algorithm must compete with the best fixed parameter setting in hindsight. There is a line of work that considers "mixture" settings, in which the performance of the algorithm competes against multiple points in hindsight. [SBD20] gives guarantees for *shifting regret*, where the algorithm must compete against offline strategies that can switch parameter settings some fixed number of times. [BKST21] studies *meta-learning*, in which an algorithm sees samples from multiple distributions one at a time, and uses information from previous distributions to learn good initializations for future distributions. These settings are different from ours, as the set of points that they compete with are considered sequentially, whereas we compete with a k-medians style objective. [BSV21] study a setting in which the cost of a prediction for an instance is a piecewise-constant function. [KCBT24] uses a *contextual bandit* framework to choose different parameter settings for different types of instances. This is related to the setting that we study in Section 4, in that we are learning information about the instance space, to provide parameter settings that are more fit to each instance.

Warm starts. *Warm starts* are a popular heuristic that are used in practice to speed up the computation of various problems, e.g. for linear and mixed integer programs [Gur23]. Due to their success in practice, a line of work has sought to provide rigorous theoretical guarantees for problems such as bipartite matching [DIL+21, CSVZ22] and max flow [PZ22, DMVW23]. Previous work has studied the problem of solving a sequence of instances using a learning-augmented algorithm to compete with the performance of the best fixed prediction in hindsight [KBTV22]. Our work is, to the best of our knowledge, the first to consider the problem of competing with multiple predictions in the warm start setting.

Dynamic algorithms. We note that the online setting of our problem, in which we solve a sequence of related instances, is related to the setting of *dynamic algorithms*. In dynamic algorithms, the input can change in small, structured ways from one day to the next (e.g. for a graph, one edge is inserted or deleted per day). A dynamic algorithm must update its solution based on the change in the input, and seeks to minimize its *update time*, or runtime to perform this update on a given day. In dynamic algorithms, we typically hope to achieve update time that is sublinear, or even much smaller, in the size of the input. This justifies that a dynamic algorithm is much more efficient than solving each day's instance from scratch.

In the warm start setting on the other hand, the input instance can change arbitrarily from day to day. Thus, we cannot hope for sublinear update times, as the algorithm must at least read the input and verify the predicted solution on each day. However, as we show in this work, in this less structured setting, it is possible to compete against stronger baselines. For example, consider a setting where on each day we receive an input corresponding to one of several slowly changing graphs. This falls outside the standard dynamic model, but can be addressed by our guarantee that competes with multiple trajectories. There is also a line of work that considers *dynamic algorithms with predictions* [vdBFNP24, HSSY24, LS23, AB24], but this setting is quite different than the one

we consider in this work.

Learning a function from instances to predictions. In Section 4 we explore the problem of learning a function from a hypothesis class $\mathcal{H}$, that maps instances to predictions. In particular, we consider a setting in which $\mathcal{H}$ is actually a set of k-wise partitions of the instance space, and we then map each partition to a prediction. [KBTV22] consider a related setting, in which the goal is to learn the best linear function from instances to predictions. While related, the two settings are incomparable. The problem of learning functions from instance to predictions has also been studied by a line of work in data-driven algorithms, where it is typically phrased as learning a function from instances to algorithms. We overview this line of work in an earlier section ("Data-driven algorithms)."

Multiple predictions for warm-start algorithms. There is some work that considers the problem of using multiple predictions for warm-start algorithms. For example [DIL+22] shows that the algorithm of [DIL+21] for bipartite matching can be extended to use up to $\sqrt{n}$ extra predictions, at essentially no extra cost. In our setting, we make no assumptions on the structure of the warm-start algorithm. Thus our results are applicable to a wider range of algorithms, compared to the result of [DIL+22] which takes advantage of the structure of a specific algorithm to achieve much stronger guarantees.

Multiple predictions for online algorithms. There is a line of work that studies competing with multiple predictions for *online algorithms with predictions*. Online algorithms usually have some notion of commitment, where on each day the algorithm must make an irrevocable decision that affects future performance. Typically, algorithms with predictions in this setting solicit information about future events. Given multiple such predictions, the challenge is combining them into *one* decision that can be made on a given time step. Because the decisions that the algorithm makes vary from problem to problem, techniques to combine predictions often have to be problem specific. Work in this area includes strategies for scheduling problems [DIL+22], set cover and facility location [AGKP22], and metric algorithms [ACE+23]. This setting is significantly different from ours. In our setting, we can asymptotically compete with multiple predictions by running multiple instantiations of the learning-augmented algorithm in parallel. This leads to a different set of strategies and techniques than in the online setting.

Online search and server problems. The *online ball search* problem that we formulate in this work has connections to other well-studied online problems. Previous work has approached this problem through the lens of *online convex optimization* [KBTV22], to provide competitive guarantees against a single fixed point in hindsight. We note that in going from competing against one trajectory to competing against multiple trajectories, our problem becomes non-convex, and we require a different set of tools to approach it.

For competing against k-trajectories, the closest connection is to the k-server problem [MMS88, MMS90], in which we must service a sequence of *requests* in a metric space, using k *servers*, while moving the servers as little as possible over the course of the algorithm. In our setting, we can think predictions as servers, and the solutions of arriving instances as being requests, a connection that we explore in Section 5.2. While we can solve online ball search with a reduction to the k-server problem, it is not clear if they are equivalent problems, and another approach we propose for online ball search (Section 5.3) does not go through this connection.

A related online search problem is the *oil searching problem* [MOP09], in which an algorithm can put in work to search n locations to certain depths to try to find a resource. This is related to our problem, in which the algorithm can put in work at various predictions to try to find a solution near that prediction. However, our problem allows the algorithm more freedom in choosing the staring location, and has a different objective function. Other related search problems are the *cow path problem* [KRT93], and the problem of *searching in the plane* [BYCR93]. In these problems the algorithm is allowed to traverse the space in different ways than in online ball search.

The objective of the online ball search problem is analogous to the k-medians objective function in clustering, in some sense. Thus, the online problem also has connections to online and dynamic *k-medians clustering* [BCLP23], and *online steiner tree* [IW91, AA92, GGK16] though the objectives of these problems are somewhat different from ours. We note that the online steiner tree problem has also been studied in the learning-augmented setting [XM21], a setting which is quite different from this work.

2 Preliminaries

We model an algorithm as solving instances from an instance space $\mathbf{I}$ to produce solutions from a solution space $\mathbf{S}$. That is, a (standard) algorithm $\mathcal{A}$ takes $I \in \mathbf{I}$ as input to produce $\mathcal{A}(I) = S \in \mathbf{S}$. To model a warm start algorithm, we assume a metric distance $\mathbf{d}$ on the solution space $\mathbf{S}$.

Definition 2.1 (Warm start algorithm)**.** A *warm start* algorithm $\mathcal{A}$ is one that takes a problem instance $I \in \mathbf{I}$ and a *predicted solution* $P \in \mathbf{S}$ as input. $\mathcal{A}(I, P)$ outputs the true solution $S \in \mathbf{S}$ of instance I in time $\leq \mathbf{d}(P, S)$.

To justify modelling the runtime of an algorithm with predictions as a metric distance over the solution space $\mathbf{S}$, consider the following examples. For bipartite matching, [DIL+21] give an algorithm with runtime $\widetilde{O}\left(m\sqrt{n} \cdot (1 + ||y^* - \widehat{y}||_1)\right)$, where $\widehat{y}$ is the predicted (dual) solution, and y^* is the optimal (dual) solution. [CSVZ22] improve the guarantee to $O\left(m\sqrt{n} + (m + n \log n)||y^* - \widehat{y}||_0\right)$. Both of these runtimes can be interpreted as a metric distance that is essentially the distance between the true solution and the predicted solution in the relevant norm, scaled by a factor that depends on the input size. Another example is the algorithm of [DMVW23] which solves instances of max-flow in time $O(|E| \cdot (1 + ||\widehat{f} - f^*||_1))$, where $|E|$ is the number of edges in the flow network, $\widehat{f}$ is the predicted flow (as a vector), and f^* is the optimal flow closest to $\widehat{f}$.

Note that we consider copies of the same solution to be distinct entities at some fixed distance from each other that depends on the input size. This is because even when the predicted solution is exactly the true solution, a warm-start algorithm will still take some amount of time to verify the solution. This is reflected in the time-bounds that are given above, which are lower bounded by a constant that depends on the input size. This is consistent with our characterization of the runtime as a metric on the solution space. For example, in many settings, we can take the distance between a prediction P and a solution S to be

$$\mathbf{d}(P, S) = |I| \cdot (1 + ||P - S||)$$

where $|I|$ is the size of the relevant instances, and $||P - S||$ is the distance between P and S as vectors taken in the relevant norm.

In the remainder of the paper, we will assume that the runtime of a warm start algorithm corresponds to some metric distance $\mathbf{d}$ that is known and easily computable. In particular, we make the following assumption.

Assumption 2.2 (Metric distance is easily computable)**.** It is possible to compute the distance, $\mathbf{d}(S_1, S_2)$, between any two points, $S_1, S_2 \in \mathbf{S}$, in time at most $\mathbf{d}_{\min}$, where $\mathbf{d}_{\min}$ is the minimum distance in $\mathbf{S}$. Recall that the distance $\mathbf{S}$ is lower bounded, as we consider two copies of the same point to be distinct entities at some fixed distance away from each other.

For ease of notation, we will consider the distance $\mathbf{d}$ to be scaled by a factor such that $\mathbf{d}_{\min} = 1$ and the time to compute the distance between points is $O(1)$. This is without loss of generality, as we give multiplicative runtime guarantees in terms of $\mathbf{d}$.

This assumption is reasonable, because in our applications of interest, the points in $\mathbf{S}$ are represented by vectors of dimension that scales with the input size, and the distance function is distance taken in an appropriate norm. The runtime of the warm-start algorithms also has a multiplicative factor that scales with the input size, so the time to calculate the distance is on the same order of magnitude as the time to perform "one iteration" of the warm-start algorithm.

We often refer to the *k-medians* cost of a set of points in solution space.

Definition 2.3 (k-medians cost)**.** For a distribution $\mathcal{D}$ over a metric space $\mathbf{S}$, the *k-medians cost* of a set of k "centers" $\mathbf{C} \in \mathbf{S}^k$ is

$$\mathop{\mathbb{E}}_{x \sim \mathcal{D}} \left[\min_{\mathbf{C}_i \in \mathbf{C}} \mathbf{d}(\mathbf{C}_i, x) \right].$$

That is, the k-medians cost captures the average distance from a point in the distribution to the nearest center in $\mathbf{C}$. Finding a set of centers that minimizes or approximately minimizes the k-medians cost for a given distribution is a well-studied problem in clustering. In particular, polynomial time algorithms are known to approximate the k-medians cost within a constant factor in general metric spaces [LS16, BPR+17], and the problem is known to be NP-hard [MS84], and hard to approximate within a factor $1 + \varepsilon$ assuming the unique games conjecture [BGJ21].

3 Competing against k fixed points offline

As a warm up and motivating example, we consider a distribution $\mathcal{D}$ over instance-solution pairs $\mathbf{I} \times \mathbf{S}$. We assume that there is a underlying ground-truth mapping from instances $I \in \mathbf{I}$ to solutions $S \in \mathbf{S}$ that is uncovered by our algorithm-with-predictions $\mathcal{A}$. Thus it is sufficient to assume a distribution only over $\mathbf{I}$, and we include the solution in the distribution for ease of notation. In many places it will be useful to consider the marginal distribution of solutions.

We show that, with access to i.i.d. samples from $\mathcal{D}$, a simple strategy of running the predictions in parallel can compete with the best k fixed predictions for the distribution, with an $O(k)$ approximation factor. This is a generalization of the results in [DIL+21] and [DMVW23], which show that it is possible to PAC learn the single best fixed prediction with respect to $\mathcal{D}$.

Our main observation is that a warm start algorithm-with-predictions allows us to run multiple threads in parallel, allowing us to compete with the performance of the best thread.

Lemma 3.1 (Using k predictions)**.** *Given an algorithm-with-predictions $\mathcal{A}$ for instances in $\mathbf{I}$, and a set of k predictions $\mathbf{P} = (P_1, \ldots, P_k), P_i \in \mathbf{S}$, for an instance I with (unknown) true solution S, we can solve I in time*

$$O(k \cdot \mathbf{d}(S, \mathbf{P}(S))),$$

where $\mathbf{P}(S) = \operatorname{argmin}_{P_j \in \mathbf{P}} \mathbf{d}(S, P_j)$.

Proof. We run the algorithm-with-predictions in parallel with each of the k predictions, and output the solution of the thread that completes first (Algorithm 3.2). Let $\mathbf{P}(S) = P_{j^*}$ be the prediction that minimizes $\mathbf{d}(S, P_{j^*})$. We know that the time that Algorithm 3.2 spends on thread j^*, i.e. the runtime of $\mathcal{A}(I, P_{j^*})$, is $O(\mathbf{d}(S, P_{j^*}))$. Since the threads are run in parallel at the same rate, this means that the time that Algorithm 3.2 spends on any thread j is $O(\mathbf{d}(S, P_{j^*}))$, and therefore the total runtime of the algorithm is bounded by $O(k \cdot \mathbf{d}(S, P_{j^*}))$. □

Algorithm 3.2 Using k predictions in parallel

1: Run $\mathcal{A}(I, P_j)$ for all $j \in [k]$ "in parallel"[3] until one of them completes
2: Output the solution of the $\mathcal{A}(I, P_j)$ that completed

We observe that it is possible to learn an approximately good set of predictions with respect to a distribution $\mathcal{D}$ over $\mathbf{S}$ from samples, by noting that this is the k-medians clustering problem for the distribution $\mathcal{D}$. We provide a standard sample compression argument that assumes little structure on $\mathbf{S}$ for completeness, and note that better bounds are known for specific settings of interest.

Definition 3.3 (Clustering cost of a set)**.** Let $\mathbf{X} = \{X_1, \ldots, X_m\}$ be a set of m points from $\mathbf{S}$, and $\mathbf{C} \in \mathbf{S}^k$ be an arbitrary set of k centers. We define the *cost* of $\mathbf{C}$ over $\mathbf{X}$ by

$$\operatorname{cost}(\mathbf{C}, \mathbf{X}) = \frac{1}{m} \sum_{i=1}^{m} \left[\mathbf{d}(\mathbf{C}(X_i), X_i)\right],$$

where $\mathbf{C}(X_i)$ is the closest center in $\mathbf{C}$ to X_i.

Definition 3.4 (Clustering cost of a distribution)**.** Let $\mathcal{D}$ be a distribution over $\mathbf{I} \times \mathbf{S}$, and $\mathbf{C} \in \mathbf{S}^k$ be an arbitrary set of k centers. We define the *cost* of $\mathbf{C}$ over $\mathcal{D}$ by

$$\operatorname{cost}(\mathbf{C}, \mathcal{D}) = \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} \left[\mathbf{d}(\mathbf{C}(S), S)\right],$$

where $\mathbf{C}(S)$ is the closest center in $\mathbf{C}$ to S.

[3] Here, we use "in parallel" to mean alternately running a constant number of steps of each of the $\mathcal{A}(I, P_j)$, resulting in a sequential algorithm.

Lemma 3.5 (Learning k fixed points)**.** *For a distribution $\mathcal{D}$ over instance-solution pairs, it is possible to learn an $O(1)$-approximate k-medians clustering of $\mathcal{D}$ from $m \geq \frac{12 \cdot \mathbf{d}_{\max} \cdot k \cdot \log(1/\delta)}{\text{cost}(\mathbf{C}^*, \mathcal{D})}$ samples drawn i.i.d. from $\mathcal{D}$, with probability $\geq 1 - 2\delta$, where $\mathbf{d}_{\max}$ is the width of $\mathbf{S}$.*

Proof. Let $\mathbf{C}^* = (\mathbf{C}^*_{(1)}, \ldots, \mathbf{C}^*_{(k)})$ be a best set of k centers for distribution $\mathcal{D}$. Let $\mathbf{d}_{\max}$ be the width (largest distance) of the metric space $\mathbf{S}$. Consider $\mathbf{X} = \{X_1, \ldots, X_m\}$, a set of m samples drawn i.i.d. from $\mathcal{D}$. First, we bound the probability that the average loss of $\mathbf{C}^*$ over $\mathbf{X}$ is far from the average loss of $\mathbf{C}^*$ over $\mathcal{D}$. Let

$$\text{cost}(\mathbf{C}^*, \mathcal{D}) = \mathop{\mathbb{E}}_{X \sim \mathcal{D}} \left[\mathbf{d}(\mathbf{C}^*(X), X) \right],$$

where $\mathbf{C}^*(X)$ is the closest $C \in \mathbf{C}^*$ to X. Similarly, define the empirical cost of $\mathbf{C}^*$ over $\mathbf{X}$ as

$$\text{cost}(\mathbf{C}^*, \mathbf{X}) = \frac{1}{m} \sum_{i=1}^{m} \mathbf{d}(\mathbf{C}^*(X_i), X_i).$$

Now we can use a Chernoff-Hoeffding bound to see that

$$\mathbb{P}\left[\text{cost}(\mathbf{C}^*, X) \geq 2 \cdot \text{cost}(\mathbf{C}^*, \mathcal{D})\right] = e^{-\frac{m \cdot \text{cost}(\mathbf{C}^*, \mathcal{D})}{3\mathbf{d}_{\max}}}.$$

Thus, by setting $m \geq \log(1/\delta) \cdot \frac{3\mathbf{d}_{\max}}{\text{cost}(\mathbf{C}^*, \mathcal{D})}$, for some $0 < \delta < 1$ we can achieve that the empirical loss of $\mathbf{C}^*$ over the samples X is at most twice the true loss over the distribution, with probability $\geq 1 - \delta$.

Now, we show that it is possible to learn an approximately good clustering for $\mathcal{D}$ from $\mathbf{X}$. Our learning algorithm proceeds as follows. It considers all $\binom{m}{k}$ subsets of $\mathbf{X}$ as possible centers. Of the possible centers, it chooses the $\widehat{\mathbf{C}}$ with the lowest empirical loss.

By a sample compression argument, see e.g. Theorem 30.2 in [SSBD14], we have that with probability at least $1 - \delta$

$$\text{cost}(\widehat{\mathbf{C}}, \mathcal{D}) \leq \text{cost}(\widehat{\mathbf{C}}, \mathbf{X} \setminus \widehat{\mathbf{C}}) + \sqrt{\text{cost}(\widehat{\mathbf{C}}, \mathbf{X} \setminus \widehat{\mathbf{C}}) \frac{4k \log(m/\delta)}{m}} + \frac{8k \log(m/\delta)}{m}.$$

Setting m to be sufficiently larger than $4k \log(1/\delta) \mathbf{d}_{\max}$ allows us to conclude that

$$\text{cost}(\widehat{\mathbf{C}}, \mathcal{D}) \leq 2 \cdot \text{cost}(\widehat{\mathbf{C}}, \mathbf{X} \setminus \widehat{\mathbf{C}}).$$

Now, we note that there is a clustering of $\mathbf{X}$ that uses points from $\mathbf{X}$ as centers, that has cost at most 2 times that of $\mathbf{C}^*$ on $\mathbf{X}$. To see this, consider mapping each center in $\mathbf{C}^*$ to its nearest point in $\mathbf{X}$, to get $\mathbf{C}'$,

$$\mathbf{C}'_{(i)} = \mathop{\text{argmin}}_{X \in \mathbf{X}} \mathbf{d}(\mathbf{C}^*_{(i)}, X).$$

We have that for each $X \in \mathbf{X}$

$$\begin{aligned} \mathbf{d}(\mathbf{C}'(X), X) &\leq \mathbf{d}(\mathbf{C}^*(X), X) + \min_{X' \in \mathbf{X}} \mathbf{d}(\mathbf{C}^*(X), X') \\ &\leq 2 \cdot \mathbf{d}(\mathbf{C}^*(X), X). \end{aligned}$$

Since $\widehat{\mathbf{C}}$ is the cost minimizer over all sets of centers that are subsets of $\mathbf{X}$, this means that $\text{cost}(\widehat{\mathbf{C}}, \mathbf{X}) \leq 2 \cdot \text{cost}(\mathbf{C}^*, \mathbf{X}))$. Furthermore, since $2k \leq m$, we have that $\text{cost}(\widehat{\mathbf{C}}, \mathbf{X} \setminus \widehat{\mathbf{C}}) \leq 2 \cdot \text{cost}(\widehat{\mathbf{C}}, \mathbf{X})$. Finally, we get that if $m \geq \frac{12 \cdot \mathbf{d}_{\max} \cdot k \cdot \log(1/\delta)}{\text{cost}(\mathbf{C}^*, \mathcal{D})}$, then with probability $\geq 1 - 2\delta$

$$\begin{aligned} \text{cost}(\widehat{\mathbf{C}}, \mathcal{D}) &\leq 2 \cdot \text{cost}(\widehat{\mathbf{C}}, \mathbf{X} \setminus \widehat{\mathbf{C}}) \\ &\leq 4 \cdot \text{cost}(\widehat{\mathbf{C}}, \mathbf{X}) \\ &\leq 8 \cdot \text{cost}(\mathbf{C}^*, \mathbf{X}) \\ &\leq 16 \cdot \text{cost}(\mathbf{C}^*, \mathcal{D}), \end{aligned}$$

so $\widehat{\mathbf{C}}$ is an $O(1)$-approximation to the minimum cost k-clustering of $\mathcal{D}$. □

Using the above lemmas, we can conclude the following theorem.

Theorem 3.6 (Competing against k fixed points offline)**.** *With access to* $m \geq \frac{12 \cdot \mathbf{d}_{\max} \cdot k \cdot \log(1/\delta)}{\text{cost}(\mathbf{C}^*, \mathcal{D})}$ *i.i.d. samples from* $\mathcal{D}$*, it is possible to design an algorithm that has expected runtime*

$$O(k) \cdot \text{cost}(\mathbf{C}^*, \mathcal{D})$$

on future instances drawn from $\mathcal{D}$*, where* $\mathbf{C}^*$ *is the set of* k *centers in* $\mathbf{S}$ *that minimizes the cost with respect to* $\mathcal{D}$*.*

Proof. Lemma 3.5 tells us that from $m \geq \frac{12 \cdot \mathbf{d}_{\max} \cdot k \cdot \log(1/\delta)}{\text{cost}(\mathbf{C}^*, \mathcal{D})}$ samples, with probability $\geq 1 - 2\delta$ we can learn a set of centers $\widehat{\mathbf{C}}$ such that

$$\text{cost}(\widehat{\mathbf{C}}, \mathcal{D}) = O(1) \cdot \text{cost}(\mathbf{C}^*, \mathcal{D}).$$

Once we have $\widehat{\mathbf{C}}$, on a subsequent instance-solution pair $(I, S) \sim \mathcal{D}$, we can use the algorithm from Lemma 3.1 to solve I in expected time

$$\begin{aligned} \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} \left[O(k \cdot \mathbf{d}(S, \widehat{\mathbf{C}}(S)) \right] &= O(k) \cdot \text{cost}(\widehat{\mathbf{C}}, \mathcal{D}) \\ &= O(k) \cdot \text{cost}(\mathbf{C}^*, \mathcal{D}). \end{aligned}$$

□

Remark 3.7. In the generality that we have modeled the problem in this section, we cannot hope to achieve an approximation factor that is $o(k)$. Consider the following bad example. There are k planted solutions that are arbitrarily far apart. Our distribution serves instances such that there is a uniform probability that any of these planted solutions is the true solution. Thus, an algorithm must either explore a constant fraction of the k planted solutions in expectation, incurring a $O(k)$-approximation, or pay the large distance between the solutions, resulting in an unbounded approximation.

It is potentially possible to get around this lower bound by taking advantage of additional structure in the problem. We explore one such approach in Section 4.

4 Competing with a hypothesis class of k-wise partitions offline

In the previous section, we showed that in the offline setting, it is possible to construct an algorithm that competes with the best k-wise clustering cost for the distribution, with a factor $O(k)$ blow-up. We also show that, without additional assumptions, this is the best approximation factor achievable for this model. In this section, we investigate ways to leverage extra information to remove the multiplicative $O(k)$ factor. In particular, to achieve better performance than the lower bound, we must have some additional way to extract information about the location of S from I.

A learning-augmented algorithm, or indeed any algorithm to solve instances I, already encodes the mapping from instances to solutions. However, it encodes the exact mapping, and the cost of uncovering the mapping is high. It is reasonable that for many distributions of interest over $\mathbf{I} \times \mathbf{S}$, we can learn a *coarse* mapping between instances and solutions, that can help us search for the solution faster.

A natural way to model a coarse mapping between instances and solutions is as a k-wise partition over the instance space $\mathbf{I}$, i.e. $h : \mathbf{I} \to [k]$, where we expect that instances in a particular partition have solutions that are similar. With this additional assumption of access to a partition $h : \mathbf{I} \to [k]$, we can learn the best fixed prediction for each partition of h. Once we have the k predictions fixed, for a new instance I, we can evaluate $h(I)$ and use the corresponding prediction to solve I. This avoids the factor k blow up in runtime that we had to incur in the previous section.

We note that this is somewhat different from the approach in the previous section (Section 3). In the previous section, we can think of the k predictions $\mathbf{C}$ that we choose as defining an implicit partition of the solution space, where each partition corresponds to a section in the Voronoi diagram of $\mathbf{S}$ defined by $\mathbf{C}$. Because the partition is over the solution space, given a fresh instance I, it is not easy to see which partition I's solution belongs to. Thus we must run all of the predictions, and accrue an $\Omega(k)$ approximation factor. In this section, we are considering partitions h over *instance* space, so that given a new instance I, we can easily compute the partition that I

belongs to. This also implicitly defines subsets of the solution space $\mathbf{S}$, where the ith (potentially overlapping) subset consists of solutions corresponding to instances in the ith partition of h. We note that these subsets may not have any particular structure.

The question remains whether it is possible to learn a good partition $h : \mathbf{I} \to [k]$ for a distribution $\mathcal{D}$. We consider the setting where we must select a partition from a hypothesis class $\mathcal{H}$ of potential efficiently-computable partitions of $\mathbf{I}$. We aim to minimize the expected runtime on future instance-solution pairs from $\mathcal{D}$. We show that when the hypothesis class $\mathcal{H}$ of k-wise partitions of $\mathbf{I}$ is learnable, it is indeed possible to learn an approximately optimal $h \in \mathcal{H}$ and set of predictions, or "centers," $\mathbf{C} \in \mathbf{S}^k$ for the partitions of h.

We do this by constructing a loss function, $\mathbf{C}$-loss that can be used with an ERM oracle. In Section 4.1, we give the main result of this section, which is showing that the $\mathbf{C}$-loss of a partition $h : \mathbf{I} \to [k]$ approximates the cost of h over a distribution $\mathcal{D}$. In Section 4.2 we show that if the hypothesis class $\mathcal{H}$ of partitions is learnable, then we can learn an approximately optimal hypothesis $h \in \mathcal{H}$. In Section 4.3, we conclude an algorithmic guarantee.

Definition 4.1 (Clustering cost of k-wise partition)**.** Let $\mathcal{D}$ be a distribution over $\mathbf{I} \times \mathbf{S}$, and h be a k-wise partition of $\mathbf{S}$. We define the *cost* of h over $\mathcal{D}$ by

$$\operatorname{cost}(h, \mathcal{D}) = \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} \left[\mathbf{d}(\mathbf{C}_h^{(h(I))}, S) \right],$$

where $\mathbf{C}_h$ is the collection of the best centers for each partition of h, with respect to $\mathcal{D}$. That is,

$$\mathbf{C}_h^{(i)} = \mathop{\operatorname{argmin}}_{C\in\mathbf{S}} \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} \left[\mathbf{d}(C, S) \mid h(I) = i \right].$$

Definition 4.2 ($\mathbf{C}$-loss)**.** Let $\mathcal{D}$ be a distribution over $\mathbf{I} \times \mathbf{S}$, and h be a k-wise partition of $\mathbf{S}$, and $\mathbf{C} \in \mathbf{S}^k$ be an arbitrary set of k centers. We define the $\mathbf{C}$-*loss* of h on a point $(I, S) \in \mathbf{I} \times \mathbf{S}$ as

$$\ell_{\mathbf{C}}(h, (I, S)) = \mathbf{d}(S, \mathbf{C}^{(h(I))}).$$

We denote the expected $\mathbf{C}$-loss of h over $\mathcal{D}$ by

$$\ell_{\mathbf{C}}(h, \mathcal{D}) = \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} \left[\mathbf{d}(S, \mathbf{C}^{(h(I))}) \right].$$

Definition 4.3 (Rotation)**.** For $h \in \mathcal{H}$, where $\mathcal{H}$ is a hypothesis class of k-wise partitions, we say that the *rotation* of h by some $\varphi : [k] \to [k]$ is $\varphi \circ h$. (Note that φ does not have to be bijective.)

Definition 4.4 (Rotational completion)**.** We say that a hypothesis class $\mathcal{H}$ of k-wise partitions is *rotationally complete* if for every $h \in \mathcal{H}$ and $\varphi : [k] \to [k]$,

$$\varphi \circ h \in \mathcal{H}.$$

For any hypothesis class $\mathcal{H}$ of k-wise partitions, we denote the *rotational completion* of $\mathcal{H}$ by $\operatorname{rc}(\mathcal{H})$, where we define

$$\operatorname{rc}(\mathcal{H}) = \{\varphi \circ h \mid h \in \mathcal{H}, \varphi : [k] \to [k]\}.$$

4.1 Approximation The main technical component of this section is designing the $\mathbf{C}$-loss (Definition 4.2), that approximates the clustering cost of a k-wise partition, while also being decomposable and therefore usable with an ERM oracle.

Lemma 4.5 ($\ell_{\mathbf{C}}(h, \mathcal{D})$ approximates clustering cost)**.** *Consider a distribution $\mathcal{D}$ over pairs $(I, S) \in (\mathbf{I}, \mathbf{S})$, an arbitrary set of k centers $\mathbf{C} = (\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(k)}) \in \mathbf{S}^k$, and a hypothesis class $\mathcal{H}$ of k-wise partitions of $\mathbf{S}$. For every $h \in \mathcal{H}$, there exists a rotation $\varphi : [k] \to [k]$, such that*

$$\ell_{\mathbf{C}}(\varphi \circ h, \mathcal{D}) \leq O(1) \cdot (\operatorname{cost}(h, \mathcal{D}) + \operatorname{cost}(\mathbf{C}, \mathcal{D})).$$

Proof. First, we consider the cost associated with a particular partition i of h. We show how to choose a center j from $\mathbf{C}$ to assign this partition. Let $\mathbf{C}_h^{(i)}$ be the best possible center in $\mathbf{S}$ for partition i of h for $\mathcal{D}$. That is,

$$\mathbf{C}_h^{(i)} = \operatorname*{argmin}_{C \in \mathbf{S}} \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, C) \mid h(I) = i] .$$

Let $\mathbf{C}(S)$ be the best center from $\mathbf{C}$ for a solution $S \in \mathbf{S}$. That is,

$$\mathbf{C}(S) = \operatorname*{argmin}_{\mathbf{C}^{(j)}} \mathbf{d}(S, \mathbf{C}^{(j)}).$$

Now, we can associate $\mathbf{C}_h^{(i)}$ with its closest center in $\mathbf{C}$. Formally, let

$$\varphi(i) = \operatorname*{argmin}_{j \in [k]} \mathbf{d}(\mathbf{C}^{(j)}, \mathbf{C}_h^{(i)}).$$

Then we have

$$\begin{aligned}
&\mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, \mathbf{C}^{(\varphi(i))}) \mid h(I) = i] \\
\leq &\mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, \mathbf{C}_h^{(i)}) \mid h(I) = i] + \mathbf{d}(\mathbf{C}^{(\varphi(i))}, \mathbf{C}_h^{(i)}) && \text{triangle inequality} \\
\leq &\mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, \mathbf{C}_h^{(i)}) \mid h(I) = i] + \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(\mathbf{C}(S), \mathbf{C}_h^{(i)}) \mid h(I) = i] \\
\leq &\mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, \mathbf{C}_h^{(i)}) \mid h(I) = i] + \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, \mathbf{C}_h^{(i)}) + \mathbf{d}(S, \mathbf{C}(S)) \mid h(I) = i] && \text{triangle inequality} \\
= &2 \cdot \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, \mathbf{C}_h^{(i)}) \mid h(I) = i] + \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, \mathbf{C}(S)) \mid h(I) = i]
\end{aligned}$$

Taking the appropriate combination over the partitions i, we get

$$\begin{aligned}
&\ell_{\mathbf{C}}(\varphi \circ h, \mathcal{D}) \\
= &\mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, \mathbf{C}^{(\varphi(h(I)))})] \\
= &\sum_{i \in [k]} \mathop{\mathbb{P}}_{(I,S)\sim\mathcal{D}} [h(I) = i] \cdot \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, \mathbf{C}^{(\varphi(i))})) \mid h(I) = i] \\
\leq &\sum_{i \in [k]} \mathop{\mathbb{P}}_{(I,S)\sim\mathcal{D}} [h(I) = i] \cdot \left(2 \cdot \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, \mathbf{C}_h^{(i)}) \mid h(I) = i] + \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, \mathbf{C}(S)) \mid h(I) = i] \right) \\
= &2 \cdot \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, \mathbf{C}_h^{(h(I))})] + \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}} [\mathbf{d}(S, \mathbf{C}(S))] \\
= &2 \cdot \mathrm{cost}(h, \mathcal{D}) + \mathrm{cost}(\mathbf{C}, \mathcal{D}).
\end{aligned}$$

□

4.2 Learning the hypothesis class In this section, we show a series of lemmas that imply that learnability of the hypothesis class $\mathcal{H}$ is enough to imply that the minimum loss $h \in \mathrm{rc}(\mathcal{H})$ is learnable. The proofs of these lemmas are largely straightforward sample complexity arguments, which we include for completeness.

To characterize the learnability of hypothesis classes, we find it simplest to go through Ψ_B-dimension, as defined by [BDCBL92].

Lemma 4.6 (Ψ_B-dimension of $\mathrm{rc}(\mathcal{H})$)**.** *For a hypothesis class $\mathcal{H}$ of k-wise partitions over $\mathbf{I}$, the Ψ_B-dimension of $\mathrm{rc}(\mathcal{H})$ can only be an order k factor (up to logarithmic factors) larger than the Ψ_B-dimension of $\mathcal{H}$. That is,*

$$\Psi_B(\mathrm{rc}(\mathcal{H})) \in O\Big(k\Psi_B(\mathcal{H}) \log(k\Psi_B(\mathcal{H})) \Big).$$

Proof. Suppose that $\mathrm{rc}(\mathcal{H})$ can Ψ_B shatter n points $x_1, \ldots, x_n \in \mathbf{I}$. This means that there exist functions $\psi_1, \ldots, \psi_n : [k] \to \{0,1\}$ such that for every labeling $y \in \{0,1\}^n$, there exists an $h' \in \mathrm{rc}(\mathcal{H})$ such that

$$\psi_i(h'(x_i)) = y_i, \qquad \forall i.$$

Let S be a set containing one h' achieving each labeling y. Thus, $|S| = 2^n$. Associate each $h' \in S$ with a choice of φ and $h \in \mathcal{H}$ such that $h' = \varphi \circ h$.

There are at most k^k possible values of φ. Thus, there must be some $\varphi^* : [k] \to [k]$ such that at least $2^n/k^k$ elements of S are associated with φ^*. Let S' be the subset of S containing h' that are associated with φ^*. We have that each $h' \in S'$ maps to a distinct value of

$$(\psi_1(h'(x_1)), \ldots, \psi_n(h'(x_n))) = ((\psi_1 \circ \varphi^*)(h(x_1)), \ldots, (\psi_n \circ \varphi^*)(h(x_n))).$$

Let $\mathcal{H}'$ be the set of h that are associated with $h' \in S'$. The above tells us that for functions $(\psi_1 \circ \varphi^*), \ldots, (\psi_n \circ \varphi^*)$, the hypothesis classes in $\mathcal{H}'$ span at least $2^n/k^k = n^{\frac{n - k\log k}{\log n}}$ labelings of $x_1, \ldots, x_n$.

By the Sauer-Shelah lemma, this means that there must be a subset $X' \subseteq (x_1, \ldots, x_n)$ of size $\Omega((n - k\log k)/\log n)$ that is Ψ_B-shattered by S' with respect to the functions $(\psi_1 \circ \varphi), \ldots, (\psi_n \circ \varphi)$.

Let q be the Ψ_B-dimension of $\mathcal{H}$. Since $q \in \Omega(\Omega((n - k\log k)/\log n))$, we have that $n \in O(kq \log kq)$. Since this applies when n is Ψ_B-dimension of $\mathrm{rc}(\mathcal{H})$, we have that

$$\Psi_B(\mathrm{rc}(\mathcal{H})) \in O\Big(k\Psi_B(\mathcal{H}) \log(k\Psi_B(\mathcal{H}))\Big).$$

□

A simple reduction can construct an ERM oracle for $\mathrm{rc}(\mathcal{H})$ using calls to an ERM oracle for $\mathcal{H}$, where the number of calls depends only on k. However, we note that it may be possible to get a much more efficient ERM oracle, for example when $\mathcal{H}$ is rotationally complete to begin with.

Lemma 4.7 (Converting ERM for $\mathcal{H}$ to ERM for $\mathrm{rc}(\mathcal{H})$)**.** *For a hypothesis class $\mathcal{H}$ of k-wise partitions of $\mathbf{I}$ and an empirical distribution $\mathcal{D}$ over $\mathbf{I} \times \mathbf{S}$, given access to an oracle $\mathcal{O}$ that can answer queries of the form*

$$\operatorname*{argmin}_{h \in \mathcal{H}} \ell_{\mathbf{C}}(h, \mathcal{D})$$

for any set of centers $\mathbf{C} \in \mathbf{S}^k$, we can construct an oracle $\mathcal{O}_{\mathrm{rc}}$ that can answer queries of the form

$$\operatorname*{argmin}_{h' \in \mathrm{rc}(\mathcal{H})} \ell_{\mathbf{C}}(h', \mathcal{D})$$

for any set of centers $\mathbf{C} \in \mathbf{S}^k$ using k^k calls to $\mathcal{O}$.

Proof. There are exactly k^k possible choices for $\varphi : [k] \to [k]$. Define

$$\varphi(\mathbf{C}) = (\mathbf{C}^{(\varphi(1))}, \ldots, \mathbf{C}^{(\varphi(k))}).$$

Then we have that

$$\ell_{\mathbf{C}}(\varphi \circ h, \mathcal{D}) = \ell_{\varphi(\mathbf{C})}(h, \mathcal{D}).$$

Therefore,

$$\begin{aligned}\operatorname*{argmin}_{h' \in \mathrm{rc}(\mathcal{H})} \ell_{\mathbf{C}}(h', \mathcal{D}) &= \min_{\varphi:[k]\to[k]} \ell_{\mathbf{C}}\left(\varphi \circ \left(\operatorname*{argmin}_{h \in \mathcal{H}} \ell_{\mathbf{C}}(\varphi \circ h, \mathcal{D})\right), \mathcal{D}\right) \\ &= \min_{\varphi:[k]\to[k]} \ell_{\mathbf{C}}\left(\varphi \circ \left(\operatorname*{argmin}_{h \in \mathcal{H}} \ell_{\varphi(\mathbf{C})}(h, \mathcal{D})\right), \mathcal{D}\right).\end{aligned}$$

Thus, we can evaluate this by enumerating over all choices of φ, and making one call to $\mathcal{O}$ for each φ. Then, we can evaluate the empirical loss of φ with the minimizing $h \in \mathcal{H}$, and choose the φ that achieves the lowest loss. □

Lemma 4.8 (Pseudo-dimension of loss functions)**.** *For a fixed* $\mathbf{C}$*, and hypothesis class* $\mathcal{H}$ *of* k*-wise partitions over* $\mathbf{I}$*, the pseudo-dimension of the set of loss functions* $\ell_{\mathbf{C}}(h,\cdot)$ *for* $h \in \mathcal{H}$ *is bounded by the* Ψ_B*-dimension of* $\mathcal{H}$*.*

Proof. Let d be the pseudo-dimension of the set of $\ell_{\mathbf{C}}(h,\cdot)$ for $h \in \mathcal{H}$. This means that there exist points $x_1,\ldots,x_d \in \mathbf{I}$ and thresholds $t_1,\ldots,t_d \in \mathbb{R}$ such that for every labeling $y \in \{0,1\}^n$, there exists an $h \in \mathcal{H}$ such that for all $i \in [d]$

$$\begin{cases} \ell_{\mathbf{C}}(h,x_i) \le t_i & \text{if } y_i = 0 \\ \ell_{\mathbf{C}}(h,x_i) > t_i & \text{if } y_i = 1 \end{cases}.$$

For each i, we construct a function $\psi_i : [k] \to \{0,1\}$ via

$$\psi_i(z) = \begin{cases} 0 & \text{if } \mathbf{d}(\mathbf{C}^{(i)},x_i) \le t_i \\ 1 & \text{if } \mathbf{d}(\mathbf{C}^{(i)},x_i) > t_i \end{cases}.$$

This ensures that for a hypothesis $h \in \mathcal{H}$, $\psi_i(h(x_i)) = 0$ if and only if $\ell_{\mathbf{C}}(h,x_i) \le t_i$. Thus, $\psi_1,\ldots,\psi_d$ witness the Ψ_B-shattering of $x_1,\ldots,x_d$, and the pseudo-dimension of the set of $\ell_{\mathbf{C}}(h,\cdot)$ for $h \in \mathcal{H}$ is at most the Ψ_B-dimension of $\mathcal{H}$. □

Lemma 4.9 (Learning guarantee)**.** *Given an arbitrary set of centers* $\mathbf{C}$ *over a bounded space* $\mathbf{S}$*, a learnable (finite* Ψ_B*-dimension) rotationally complete hypothesis class* $\mathcal{H}$ *of* k*-wise partitions of* $\mathbf{I}$*, an ERM oracle* $\mathcal{O}$ *that can compute* $\operatorname{argmin}_{h\in\mathcal{H}} \ell_{\mathbf{C}}(h,\mathcal{D}')$ *for empirical distributions* $\mathcal{D}'$*, and sample access to a distribution* $\mathcal{D}$ *over* $\mathbf{I}\times\mathbf{S}$*, we can learn an* $h \in \mathcal{H}$ *and a set of centers* $\mathbf{C}_h \in \mathbf{S}^k$ *such that*

$$\mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}}\left[\mathbf{d}(\mathbf{C}_h^{h(I)},S)\right] \le O(1)\left(1+\operatorname{cost}(\mathbf{C},\mathcal{D})+\min_{h'\in\mathcal{H}}\operatorname{cost}(h',\mathcal{D})\right).$$

In particular, if d *is the* Ψ_B*-dimension of* $\mathcal{H}$*, and* $\mathbf{d}_{\max}$ *is the largest distance in* $\mathbf{S}$*, with probability* $1-\delta$ *we can learn the above in*

$$O(\mathbf{d}_{\max}^2(d+\ln\frac{1}{\delta}))$$

samples, where $\mathbf{d}_{\max}$ *is the largest distance in* $\mathbf{S}$*.*

Proof. Since $\mathcal{H}$ has Ψ_B dimension d, by Lemma 4.8 the set of loss functions $\mathcal{L} = \{\ell_{\mathbf{C}}(h,\cdot) \mid h \in \mathcal{H}\}$ has pseudo-dimension at most d.

By uniform convergence bounds for Ψ_B-dimension [BDCBL92], we have that for any $\delta \in (0,1)$, any distribution $\mathcal{D}$ over $\mathbf{I}\times\mathbf{S}$, since all of the loss functions in $\mathcal{L}$ have value bounded in $[0,D]$, $m = O(\mathbf{d}_{\max}^2(d+\ln\frac{1}{\delta}))$ samples are sufficient to ensure that with probability $1-\frac{\delta}{4}$ over the draw of $\{(I,S)_1,\ldots,(I,S)_m\} \sim \mathcal{D}^m$, for all $h \in \mathcal{H}$, the difference between the average empirical loss over the samples and the expected loss over $\mathcal{D}$ is at most a constant, i.e.:

$$\left|\frac{1}{m}\sum_{j=1}^{m}\ell_{\mathbf{C}}(h,(I,S)_j) - \ell_{\mathbf{C}}(h,\mathcal{D})\right| \le 1. \tag{4.1}$$

Let $\mathcal{D}_m$ be the empirical distribution of m samples drawn i.i.d. from $\mathcal{D}$. The ERM oracle $\mathcal{O}$ then gives us $\widehat{h}$ such that

$$\widehat{h} = \operatorname*{argmin}_{h\in\mathcal{H}} \ell_{\mathbf{C}}(h,\mathcal{D}_m).$$

Lemma 4.5 implies that, for a rotationally complete $\mathcal{H}$, and

$$h^* = \operatorname*{argmin}_{h\in\mathcal{H}} \operatorname{cost}(h,\mathcal{D}),$$

the ERM minimizer $\widehat{h}$ achieves

$$\ell_{\mathbf{C}}(\widehat{h},\mathcal{D}_m) \le O(1)\cdot(\operatorname{cost}(h^*,\mathcal{D}_m)+\operatorname{cost}(\mathbf{C},\mathcal{D}_m)),$$

Combining this with Equation (4.1), we get that

$$\ell_{\mathbf{C}}(\widehat{h}, \mathcal{D}) \leq O(1) \cdot (1 + \text{cost}(h^*, \mathcal{D}_m) + \text{cost}(\mathbf{C}, \mathcal{D}_m)).$$

We bound the deviation of $\text{cost}(h^*, \mathcal{D}_m)$ and $\text{cost}(\mathbf{C}, \mathcal{D}_m)$ from $\text{cost}(h^*, \mathcal{D})$ and $\text{cost}(\mathbf{C}, \mathcal{D})$. Specifically, let $\mathbf{C}_{h^*}$ be the optimal set of centers for each partition of h^* with respect to $\mathcal{D}$. We can write $\text{cost}(h^*, \mathcal{D}_m) \leq \frac{1}{m}\sum_{i=1}^m X_i$, where $X_i = \mathbf{d}\left(C_{h^*}^{(h^*(I_i))}, S_i\right)$, where (I_i, S_i) are the independent samples drawn from $\mathcal{D}$. Let $X = \sum_{i=1}^m X_i$. By a Chernoff-Hoeffding bound, we have that

$$\begin{aligned}\mathbb{P}\left[X \geq \mathbb{E}[X] + m\right] &\leq \exp\left(-\frac{(\mathbb{E}[X] + m)^2}{\mathbb{E}[X] + m}\right) \\ &\leq \exp(-\frac{1}{2}(\mathbb{E}[X] + m)).\end{aligned}$$

Thus, we have that as long as $m \geq O(\log(\frac{1}{\delta}))$, we have

$$\text{cost}(h^*, \mathcal{D}_m) \leq 2 \cdot \text{cost}(h^*, \mathcal{D}) + 1,$$

with probability $\geq 1 - \frac{\delta}{4}$.

Similarly, let $Y = Y_1 + \cdots + Y_m$, where $Y_i = \mathbf{d}(\mathbf{C}(S_i), S_i)$ where the S_i are the m independent samples we drew from $\mathcal{D}$. Then $\text{cost}(\mathbf{C}, \mathcal{D}_m) = \frac{1}{m}\sum_{i=1}^m Y_i$. By a Chernoff-Hoeffding bound, we have

$$\mathbb{P}\left[Y \geq \mathbb{E}[Y] + m\right] \leq \exp(-\frac{1}{2}(\mathbb{E}[Y] + m)).$$

Thus, as long as $m \geq O(\log(\frac{1}{\delta}))$, we have

$$\text{cost}(\mathbf{C}, \mathcal{D}_m) \leq \text{cost}(\mathbf{C}, \mathcal{D}) + 1,$$

with probability $\geq 1 - \frac{\delta}{4}$.

Taking a union bound over these events, we have that with $m = O(\mathbf{d}_{\max}^2(d + \ln\frac{1}{\delta}))$ samples,

$$\mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}}\left[\mathbf{d}(\mathbf{C}^{\widehat{h}(I)}, S)\right] \leq O(1)\left(1 + \text{cost}(\mathbf{C}, \mathcal{D}) + \min_{h' \in \mathcal{H}} \text{cost}(h', \mathcal{D})\right),$$

with probability $\geq 1 - \delta$. □

4.3 Algorithmic guarantee Finally, we can use the above procedure to design an algorithm that can take advantage of course information in the form of partitions of $\mathbf{S}$, to achieve a potentially improved runtime guarantee for future instances.

Theorem 4.10 (Competing with a hypothesis class of k-wise partitions offline)**.** *Given an algorithm-with-predictions $\mathcal{A}$, a learnable (finite Ψ_B-dimension) rotationally complete hypothesis class $\mathcal{H}$ of k-wise partitions of $\mathbf{I}$, an ERM oracle $\mathcal{O}$ that can compute $\text{argmin}_{h\in\mathcal{H}}\ell_{\mathbf{C}}(h, \mathcal{D}')$ for empirical distributions $\mathcal{D}'$, and $O(\mathbf{d}_{\max}^2(d + \ln\frac{1}{\delta}))$ samples from distribution $\mathcal{D}$ over $\mathbf{I}\times\mathbf{S}$, it is possible to learn an $h : \mathbf{I} \to [k]$ and set of centers $\mathbf{C} = \left(C^{(1)}, \ldots, C^{(k)}\right)$ that are an $O(1)$ approximation to*

$$\min_{h\in\mathcal{H}} \min_{\mathbf{C}\in\mathbf{S}^k} \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}}\left[\mathbf{d}(S, C^{(h(I))})\right].$$

Furthermore, given an $h : \mathbf{I} \to [k]$ and $\mathbf{C} = \left(C^{(1)}, \ldots, C^{(k)}\right)$, we can design a procedure such that the expected time to solve a new instance is

$$\mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}}\left[\text{runtime}(h(I)) + \mathbf{d}(S, C^{(h(I))})\right].$$

Proof. By Lemma 3.5 we know that we can use samples from $\mathcal{D}$ to find a set of centers $\widehat{\mathbf{C}}$ that are an $O(1)$-approximation to

$$\operatorname*{argmin}_{\mathbf{C}} \operatorname{cost}(\mathbf{C}, \mathcal{D}).$$

By Lemma 4.9 we have that, using this $\widehat{\mathbf{C}}$, and additional samples from $\mathcal{D}$, we can find a hypothesis $\widehat{h} \in \mathcal{H}$ such that

$$\begin{aligned}
\mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}}\left[\mathbf{d}(\widehat{\mathbf{C}}^{\widehat{h}(I)}, S)\right] &\leq O(1)\left(1 + \operatorname{cost}(\widehat{\mathbf{C}}, \mathcal{D}) + \min_{h'\in\mathcal{H}} \operatorname{cost}(h', \mathcal{D})\right). \\
&\leq O(1)\left(1 + \min_{\mathbf{C}\in\mathbf{S}^k} \operatorname{cost}(\mathbf{C}, \mathcal{D}) + \min_{h'\in\mathcal{H}} \operatorname{cost}(h', \mathcal{D})\right) \\
&\leq O(1) \cdot \min_{h'\in\mathcal{H}} \operatorname{cost}(h', \mathcal{D}) \qquad (4.2) \\
&\leq O(1) \cdot \min_{h\in\mathcal{H}} \min_{\mathbf{C}\in\mathbf{S}^k} \mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}}\left[\mathbf{d}(S, C^{(h(I))})\right],
\end{aligned}$$

where the Equation (4.2) follows because the cost of the best hypothesis is lower bounded by the cost of the best clustering, and because we assume all distances in $\mathbf{S}$ are lower bounded by 1 (see preliminaries, Section 2).

Finally, once we have a hypothesis $h \in \mathcal{H}$ and the centers $\mathbf{C}_h$, on subsequent instances I drawn from $\mathcal{D}$, we can compute $\mathcal{A}(I, \mathbf{C}_h^{(h(I))})$, in expected time

$$\mathop{\mathbb{E}}_{(I,S)\sim\mathcal{D}}\left[\text{runtime}(h(I)) + \mathbf{d}(S, C^{(h(I))})\right].$$

□

5 Competing with trajectories online

In this section, we consider the problem of solving a series of instance that arrive online. Consider an instance $I \in \mathbf{I}$ with (unknown) true solution $S \in \mathbf{S}$. As before, we assume that the solution space $\mathbf{S}$ is equipped with a metric distance $\mathbf{d}$, which we can calculate efficiently (see Assumption 2.2). We can think of an algorithm with predictions as searching the solution space $\mathbf{S}$ by growing a ball around a prediction $P \in \mathbf{S}$ of our choice. The cost of the algorithm is the radius of the ball necessary to find S. Furthermore, we can search from multiple points, and the total cost will be the sum of the radii searched from each of the points.

Definition 5.1 (Online Ball Search)**.** In the *online* setting, we consider instances arriving over T days. On each day $t \in [T]$ an instance $I_t \in \mathbf{I}$ arrives, with (unknown) solution $S_t \in \mathbf{S}$. We are given access to an algorithm with predictions $\mathcal{A}$ such that for any prediction $P \in \mathbf{S}$, the runtime of $\mathcal{A}(I_t, P)$ is bounded by $\mathbf{d}(S_t, P)$. On each day $t \in [T]$, the algorithm must output S_t.

In the online setting, we want to minimize the total work that the algorithm does over a sequence of T instances. We make *no distributional assumptions* about the instance-solution pairs that arrive. Instead, we approach the problem through the lens of *competitive analysis*, and provide algorithms that compete with the best strategies in hindsight.

In Section 5.1, we define the offline strategies that we are competing against. This requires careful consideration of various modeling constraints. In Section 5.2, we give an algorithm that is $O(k^2)$-competitive against these baselines in the total radius searched, via a reduction to k-server. We note that this does not immediately solve our problem of interest, as we wish to bound the *total work* done by our algorithm, which is only lower bounded by the total radius searched. In Section 5.3, we give an algorithm that is $O(k^4 \ln^2 k)$-competitive in the total radius searched, and which has total runtime bounded by an $O(1)$ factor times the total radius searched. To approach this, we use significantly different techniques than in the reduction to k-server.

5.1 Offline Baselines

We will analyze algorithms for this problem under the paradigm of competitive analysis, where we compare the performance of our algorithm to the performance of the best offline strategy in hindsight. To do this, we must define the set of offline strategies that we are competing with.

In our problem setting, on each day, the algorithm is allowed to search outward from any set of points, and pays for the total radius that is searched before that day's point is found. A natural idea is to allow the offline

strategies to have the same form. However, the offline strategy has full knowledge of all of the requests that arrive in the sequence. So, on each day, it could search exactly from that day's request, achieving zero cost. Thus, this set of offline strategies is not particularly meaningful.

This motivates us to consider a restricted set of offline strategies, that captures some structure in the input that we could hope to take advantage of. This is reasonable, as we must leverage some structure to get a fast algorithm for sequences of inputs. Otherwise, if we had a fast algorithm for arbitrary sequences of inputs, this would imply a faster algorithm for one arbitrary input.

Previous work imposes structure by providing a guarantee that competes with the best fixed prediction in hindsight [KBTV22]. This can be interpreted as imposing the structure that the data is well-clustered, and therefore the best fixed point performs well for the distribution of inputs. This objective function is analogous to a 1-medians objective, as our cost is the sum of the distances from the requests to the center (best fixed prediction in hindsight).

The first extension is to allow the offline strategy to move over time. We must be careful about how we do this, as allowing the offline point to move arbitrarily on each day results in the zero cost issue that we discussed earlier. Thus, we allow to the point to move, but we charge the offline strategy for the total distance that the point moves. That is, the cost to the offline strategy on a day t is the sum of the *movement cost*, the distance that the strategy moved its prediction on day t, and the *hit cost*, the distance from the request to the prediction after the prediction moves. We call this strategy a *trajectory*.

Definition 5.2 (Trajectory)**.** A *trajectory*, with respect to a sequence of instances $\{I_t : t \in [T]\}$, is a sequence of predictions $P_t \in \mathbf{S}$, for $t \in [T]$. The *cost* of a trajectory is the sum of the prediction costs for each day's prediction ("hit cost"), plus the total distance that the trajectory moves ("movement cost"),

$$\text{cost}(\{P_t : t \in [T]\}) = \left[\sum_{i=1}^{T} \mathbf{d}(P_t, S_t)\right] + \left[\sum_{i=1}^{T} \mathbf{d}(P_{t-1}, P_t)\right],$$

where S_t is the solution associated with instance I_t, and we define the initial $P_0 = \mathbf{0}$, is a fixed arbitrary starting prediction.

The second extension is that we compete against offline strategies that consist of k points. That is, on each day, the hit cost of the offline strategy is the distance from the request to the nearest of the k points. This setting is analogous to a k-medians objective, as the cost of each request is the distance from the request to the closest of k centers. We note that the extension from one center to k centers makes the offline problem of choosing the best strategy go from being convex, to being non-convex.

Definition 5.3 (Multiple trajectories)**.** A collection of k *trajectories*, with respect to a sequence of instances $\{I_t : t \in [T]\}$, is a sequence of predictions $\{P_t \in \mathbf{S} : t \in [T]\}$, and associates each day t with one of the trajectories $\in [k]$. We use $\mathcal{T}^{(i)} \subseteq [T]$ to refer the days that are associated with trajectory i, and we denote the collection of trajectories by $\left(\{P_t : t \in \mathcal{T}^{(1)}\}, \ldots, \{P_t : t \in \mathcal{T}^{(k)}\}\right)$.

For a day $t \in [T]$, we use $\text{prev}^{(i)}(t)$ to refer to the closest day previous to t that belongs to trajectory i, and $\text{next}^{(i)}(t)$ to refer to the closest day subsequent to t that belongs to trajectory i. If t is the first day in its trajectory, then we define $\text{prev}^{(i)}(t) = P_0 = \mathbf{0}$.

The *cost* of the collection of trajectories is the sum of the costs of the k trajectories. The cost of trajectory i is the single trajectory cost of $\{P_t : t \in \mathcal{T}^{(i)}\}$ with respect to $\{I_t : t \in \mathcal{T}^{(i)}\}$. That is,

$$\text{cost}(\{P_t : t \in \mathcal{T}^{(i)}\}) = \left[\sum_{t \in \mathcal{T}^{(i)}} \mathbf{d}(P_t, S_t)\right] + \left[\sum_{t \in \mathcal{T}^{(i)}} \mathbf{d}(P_{\text{prev}^{(i)}(t)}, P_t)\right],$$

$$\text{cost}\left(\{P_t : t \in \mathcal{T}^{(1)}\}, \ldots, \{P_t : t \in \mathcal{T}^{(k)}\}\right) = \sum_{i=1}^{k} \text{cost}(\{P_t : t \in \mathcal{T}^{(i)}\}).$$

We note that while the offline strategy is limited to maintaining a palette of k predictions, and must pay to move the predictions, we do not make this requirement of the online algorithm.

Another way to interpret a k-trajectory baseline for a sequence of requests, is that it is essentially the offline k-server cost for those requests. In the k-server problem, the algorithm must maintain a set of k-servers in a metric space $\mathbf{S}$. On each day, a request arrives at some point $S \in \mathbf{S}$, and the algorithm must move one of its servers to S. The cost to the algorithm is the total distance it moves its servers. The offline baseline is the best way to move the servers, when the sequence of requests is known ahead of time.

Lemma 5.4 (Multiple trajectories are approximately k-server baselines)**.** *Let $\mathcal{S} = (S_1, \ldots, S_T)$ be a sequence of T requests (solutions). Let $\{P_t : t \in \mathcal{T}^{(1)}\}, \ldots, \{P_t : t \in \mathcal{T}^{(k)}\}$ be the offline optimal collection of k trajectories for R. Let $\text{serveropt}_k(\mathcal{S})$ be the cost of the offline optimal k-server solution serving $\mathcal{S}$. We have that*

$$\text{cost}(\{P_t : t \in \mathcal{T}^{(1)}\}, \ldots, \{P_t : t \in \mathcal{T}^{(k)}\}) \leq \text{serveropt}_k(\mathcal{S}) \leq 2 \cdot \text{cost}(\{P_t : t \in \mathcal{T}^{(1)}\}, \ldots, \{P_t : t \in \mathcal{T}^{(k)}\}).$$

Proof. The first inequality follows from observing that a k-server solution that serves the requests $\mathcal{S}$ is a valid collection of k trajectories, that has movement cost equal to the cost of the k-server solution, and zero hit cost.

For the second inequality, we convert the collection of k trajectories into an offline k-server solution as follows. We assign each of trajectory one server, that services the requests that correspond to that trajectory. We can bound the cost to server i by the cost to trajectory i. We can bound the cost of this k-server solution by

$$\begin{aligned}
\sum_{t \in \mathcal{T}^{(i)}} \mathbf{d}(S_{\text{prev}^{(i)}(t)}, S_t) &\leq \sum_{t \in \mathcal{T}^{(i)}} \left[\mathbf{d}(S_{\text{prev}^{(i)}(t)}, P_{\text{prev}^{(i)}(t)}) + \mathbf{d}(P_{\text{prev}^{(i)}(t)}, P_t) + \mathbf{d}(P_t, S_t)\right] \\
&\leq 2 \left[\sum_{t \in \mathcal{T}^{(i)}} \mathbf{d}(S_t, P_t)\right] + \left[\sum_{t \in \mathcal{T}^{(i)}} \mathbf{d}(P_{\text{prev}^{(i)}(t)}, P_t)\right] \\
&\leq 2 \cdot \text{cost}(\{P_t : t \in \mathcal{T}^{(1)}\}, \ldots, \{P_t : t \in \mathcal{T}^{(k)}\}).
\end{aligned}$$

□

This observation already gives the following algorithm that is competitive against one trajectory.

Algorithm 5.5 "Predict yesterday's solution"

```
P = 0
for day t, instance I_t arrives do
    S_t = A(I_t, P)
    P = S_t
    Output solution S_t
```

Corollary 5.6 ("Predict yesterday's solution" competes with any single trajectory)**.** *The "predict yesterday's solution" strategy (Algorithm 5.5) is $O(1)$-competitive with the best single trajectory in hindsight. Furthermore, the total runtime of Algorithm 5.5 is bounded by $O(1)$ times the total radius searched.*

Proof. Lemma 5.4 tells us that the best offline trajectory has cost within a factor 2 of the optimal offline 1-server strategy. For any sequence of requests, the optimal 1-server strategy is simply the one that moves the single server to each request on each day, and pays cost

$$\sum_{t=1}^{T} \mathbf{d}(S_{t-1}, S_t).$$

This is also the cost of Algorithm 5.5. Thus, Algorithm 5.5 is 2-competitive against any fixed strategy.

Furthermore, the runtime of Algorithm 5.5 is dominated by the time to run the algorithms-with-predictions subroutine, so the total runtime of Algorithm 5.5 is bounded by $O(1)$ times the total radius searched. □

Remark. The guarantee given by this strategy is already stronger than what is shown in previous work. [KBTV22] give a strategy that competes with the best *fixed* prediction in hindsight. This strategy allows us to compete with trajectories that are adaptive, in that they move over time. We note that the approach of [KBTV22] based on online convex optimization (OCO) could be modified to give a similar guarantee (for the single trajectory case) if instantiated with an OCO algorithm that achieves low dynamic regret, for example the path length bound in [Zin03].

5.2 Reduction to k-server We consider the setting of competing against a collection of k trajectories. In this setting, we can think of the offline optimum as maintaining k prediction points. On each day, the cost is the distance from today's solution to the minimum of these points. Like the single trajectory setting, these points can move over time, and the offline cost is charged for the total distance that the k points move, capturing an adaptive strategy.

In this section, we show how get a competitive algorithm for the online ball search problem using a competitive algorithm for the k-server problem, with an k factor blow-up in the competitive ratio. We then discuss why this does not immediately imply a fast algorithm for solving sequences of instances.

In the k-server problem, an algorithm must maintain the positions of k servers in a metric space $\mathbf{S}$. On each day $t \in [T]$, a request $r_t \in \mathbf{S}$ is made, and the algorithm must move at least one server to r_t. The cost to the algorithm is the total distance that all of the servers move. The competitive ratio of the algorithm is given with respect to the best offline solution, i.e. the best way to service the requests in order with k servers in hindsight. For an introduction to the k-server problem, the reader is referred to the survey [Kou09], with the note that there has been a good deal of work on this problem since the survey was published.

Theorem 5.7 (Online ball search to k-server reduction)**.** *Given an α-competitive algorithm for the k-server problem on metric space $\mathbf{S}$, we can construct a $O(k\alpha)$-competitive algorithm for the online ball search problem on $\mathbf{S}$.*

Proof. We maintain k predictions $P_1, \ldots, P_k \in \mathbf{S}$, which can also be thought of as the position of k servers.

On each day $t \in [T]$, instance I_t arrives. Our algorithm runs $\mathcal{A}(I_t, P_i)$ for all $i \in [k]$ in parallel[4] at equal rates, until one of them terminates and finds S_t. Then, we take S_t to be the request for the k-server problem for day t, and run our competitive k-server algorithm to update the positions of the predictions/servers.

The cost to the online ball search problem is k times the distance from the closest prediction (at the beginning of day t) to S_t. Since the k-server algorithm must service S_t, the cost to the k-server algorithm on day t is at least the distance from the closest server/prediction (at the beginning of day t) to S_t. Thus the cost to the online ball search algorithm is bounded by at most k times the cost to the k-server algorithm.

Finally, the k-server algorithm is α-competitive against the best offline k-server solution that visits all of the requests S_t. By Lemma 5.4, we know that this has cost at most $O(1)$ times more than the k best trajectories for the sequence of S_ts. Thus, this algorithm for online ball search is $O(k\alpha)$-competitive against k trajectories in the number of steps it takes of the subroutine $\mathcal{A}$. □

Corollary 5.8 ($O(k^2)$-competitive algorithm for online ball search)**.** *There exists an $O(k^2)$-competitive algorithm for the online ball search problem.*

This follows from the fact that the work function algorithm is $O(k)$-competitive for the k-server problem [KP95].

The strategy of reducing online ball search to the k-server problem is both promising and presents some challenges. On the one hand, the k-server problem is a natural and very well-studied problem in online algorithms. It is promising that improvements to algorithms for the k-server problem can naturally translate to the online ball search problem.

On the other hand, the online ball search problem appears to have some advantages that are lost in the reduction to k-server. For example, an online ball search algorithm is not limited to maintaining a palette of exactly k predictions. This allows us in Section 5.3 to design an algorithm that achieves competitive guarantees for all values of k simultaneously. Another issue with a k-server strategy, is that existing algorithms for the k server problem can be computationally intensive. The work function algorithm, for example, requires solving a max-flow instance on each day t with $O(t)$ vertices. This causes the work to scale badly as the number of days grows large. Finally, some algorithms for the k-server problem work well in the setting where the number of points n in the metric space $\mathbf{S}$ is finite. A notable example is the algorithm of Bansal, Buchbinder, Madry, and Naor which gives a randomized algorithm that achieves competitive ratio of $\widetilde{O}(\log^2 k \log^3 n)$ against an oblivious adversary [BBMN15]. However, this is not ideal for our applications of interest, in which the solution space $\mathbf{S}$ is often a space of vectors $\subseteq \mathbb{R}^d$ with distance $\mathbf{d}(u, v) = ||u - v||_q$ corresponding to a relevant norm q.

[4]Here, by "in parallel" we refer to interleaving the steps of the various threads, resulting in a sequential algorithm.

5.3 Algorithm with improved runtime In this section we show an algorithm that is $O(k^4 \ln^2 k)$-competitive against any set of k trajectories. Furthermore, the algorithm is deterministic (and thus resistant to an adaptive adversary), and is oblivious to the choice of k. That is, the competitive ratio holds for all k simultaneously. Importantly, this algorithm also has total runtime bounded that is $O(1)$ times the sum of radii searched. We note that the techniques we use in this algorithm are significantly different than those in the reduction to k-server (Section 5.2).

Algorithm 5.9 generalizes the single trajectory algorithm. For the single trajectory case, on each day the algorithm searched from "yesterday's solution." For multiple trajectories, this is no longer enough, as the previous day's solution could come from some other trajectory, and be arbitrarily far away from today's solution.

Algorithm 5.9 addresses this by searching from *all* previous solutions in parallel. These "threads" are run at approximately harmonic rates, i.e. the thread of the ith most recent solution is run at rate $\frac{1}{i^2 \ln^2 i}$. This leaves the issue that the previous solution from the same trajectory as today could be arbitrarily far in the past, and be run at an extremely slow rate. This is addressed by "pruning" threads that are no longer fruitful. That is, when the ball searched around a solution i fully contains the ball searched around a previous solution j, the algorithm can stop running thread j, as that work is redundant with thread i. In this case, we say thread i "subsumes" thread j. Then, we can increase the rates of the threads that are lower priority than j, and maintain that there is at most one thread being run at each rate $\frac{1}{i^2 \ln^2 i}$ for each integer i. In the analysis we show that either the algorithm runs long enough for the previous solution from the same trajectory as today to be elevated to rate $\geq \frac{1}{k^2 \ln^2 k}$ and eventually solve the instance, or the algorithm terminates more quickly than that, which is even better. This allows us to bound the competitive ratio by $O(k^4 \ln^2 k)$.

The main runtime overhead in Algorithm 5.9 comes from checking when to prune the slower threads. For each step that the algorithm takes of a thread, it must check whether the thread has been subsumed. For a thread running at rate $\frac{1}{i^2 \ln^2 i}$ it must check the $O(i)$ threads that are faster than it on each step. Thus, even though the thread is running steps of the subroutine $\mathcal{A}$ at rate $\frac{1}{i^2 \ln^2 i}$, it is doing total work at rate $O(i) \cdot \frac{1}{i^2 \ln^2 i} = O(\frac{1}{i \ln^2 i})$. The rates are chosen so that this series converges, and the total work of the algorithm can be bounded by $O(1)$ times the work it does for the fastest thread.

Algorithm 5.9 Quadratic decay

```
1:  for day t, instance I_t arrives do
2:
3:      Initialize linked-list of active threads
4:      for t' : t ≥ t' ≥ 1 do
5:          Create thread t' to run A(I_t, S_t')
6:          radius(t') := 0
7:          Append thread t' to list of active threads
8:
9:      Run active threads in parallel,
10:         with ith active thread in list running at rate 1/(i^2 ln^2 i) (rate 1 for i = 1)
11:         track radius of each thread
12:     while no thread completed do
13:         for step taken of thread running at ith rate do
14:             t_1 := thread associated with this rate, found by walking down linked list of active threads
15:             for active thread t_2 with faster rate than t_1 do
16:                 if d(S_t1, S_t2) ≤ radius(t_2) − radius(t_1) then
17:                     Kill thread t_1
18:                     Remove thread t_1 from linked list of active threads
19:
20:     S_t = solution of thread that completed
21:     Output S_t
```

Definition 5.10 (Subsumes)**.** If thread i kills thread j (Algorithm 5.9, line 16), then we say that thread i *subsumes* thread j. If thread j previously subsumed thread h, then thread i now also subsumes thread h.

Definition 5.11 (Radius of a thread)**.** For a thread i at some given time, radius(i) is the number of steps that thread i has been run so far. If thread i subsumes thread j, for analysis we will set rate(j) = rate(i). We also continue updating the radius of j i.e., radius(j) continues to increase at rate(j).

Definition 5.12 (Virtual radius)**.** The *virtual radius* of Algorithm 5.9 is the radius of the highest rate thread.

We will show that the virtual radius of the algorithm is within $O(1)$ both of the total radius searched (Lemma 5.15) and the total runtime $O(1)$ (Lemma 5.16), making it a useful abstraction.

We define the subsuming condition so that thread i subsumes thread j, exactly when the ball of radius radius(i) around S_i fully contains the ball of radius radius(j) around S_j. Thus the part of $\mathbf{S}$ that has been searched by thread j, has also been searched by thread i, and the work of thread j is redundant. The following lemma shows that this invariant is maintained over the run of the algorithm.

Lemma 5.13 (Subsuming identity)**.** *If thread i subsumes thread j, then*

$$\mathbf{d}(S_i, S_j) \leq \text{radius}(i) - \text{radius}(j).$$

Proof. We show this via induction on the number of times the "kill condition" (Line 16) is triggered in Algorithm 5.9. In the base case, at the outset of the algorithm, no thread subsumes any other thread, so the statement holds vacuously.

Now, consider the sth kill event. Consider a thread i that subsumes a thread j at this point (not necessarily the threads that just triggered the kill condition). There are two cases.

Case 1. Thread i subsumed thread j previously, at the $(s-1)$th kill event. This means that rate(j) = rate(i), so radius(i) and radius(j) increased by the same amount since the previous kill event. Thus,

$$\mathbf{d}(S_i, S_j) \leq \text{radius}(i) - \text{radius}(j)$$

continues to hold from the induction hypothesis.

Case 2. Thread i killed thread j, or thread i killed a thread h that subsumes thread j, at the sth kill event.

If thread i killed thread j, then by the kill condition (Algorithm 5.9, line 16), we have that

$$\mathbf{d}(S_i, S_j) \leq \text{radius}(i) - \text{radius}(j).$$

If thread i killed a thread h that subsumes thread j, then the kill condition gives us that

$$\mathbf{d}(S_i, S_h) \leq \text{radius}(i) - \text{radius}(h).$$

Since thread h already subsumes thread j, Case 1 gives us that

$$\mathbf{d}(S_h, S_j) \leq \text{radius}(h) - \text{radius}(j).$$

Therefore, we have

$$\begin{aligned} \mathbf{d}(S_i, S_j) &\leq \mathbf{d}(S_i, S_h) + \mathbf{d}(S_h, S_j) && \text{triangle inequality} \\ &\leq \text{radius}(i) - \text{radius}(h) + \text{radius}(h) - \text{radius}(j) \\ &\leq \text{radius}(i) - \text{radius}(j). \end{aligned}$$

□

The previous lemma implies that if a thread j is subsumed by a thread i, then thread i is at least as effective at finding that day's solution as thread j.

Lemma 5.14. *If a subsumed thread i would have completed, i.e.,*

$$\mathbf{d}(S_i, S_t) \leq \text{radius}(i),$$

then there must be some active thread of Algorithm 5.9 that has completed.

Proof. Let i^* be the thread that subsumes thread i. We have that

$$\begin{aligned} \mathbf{d}(S_{i^*}, S_t) &\leq \mathbf{d}(S_{i^*}, S_i) + \mathbf{d}(S_i, S_t) && \text{triangle inequality} \\ &\leq \mathbf{d}(S_{i^*}, S_i) + \text{radius}(i) \\ &\leq \text{radius}(i^*) && i^* \text{ subsumes } i, \text{Lemma 5.13}, \end{aligned}$$

and therefore, thread i^* completed. □

Because we choose the rates of the threads to form a converging series, it is sufficient to bound the virtual radius of the algorithm, as it is within $O(1)$ of the total radius searched by the algorithm.

Lemma 5.15 (Total radius in terms of virtual radius)**.** *The sum of the radii searched by Algorithm 5.9 is at most $O(1)$ times the virtual radius at the end of Algorithm 5.9.*

Proof. Denote the virtual radius at the end of Algorithm 5.9 by v. The virtual radius is the radius of the highest rate thread (rate 1). Thus, the highest rate thread contributes v to the total work.

Over the run of Algorithm 5.9, at any given point, there is at most one thread per $i \geq 2, i \in \mathbb{N}$ with rate equal to $\frac{1}{i^2 \ln^2 i}$. Consider the amount the thread(s) with rate $\frac{1}{i^2 \ln^2 i}$ increase the total radius searched over time. The ratio of the radius searched by this thread to the highest rate thread is $\frac{1}{i^2 \ln^2 i}$. So, over the time that the highest rate thread accrued radius v, this thread has increased the total radius searched by $\frac{v}{i^2 \ln^2 i}$.

Summing over all threads, we can bound the total radius searched by

$$v + \sum_{i=2}^{\infty} \frac{v}{i^2 \ln^2 i} < 2v = O(1) \cdot v.$$

□

Now, we account for the overhead of running the algorithm. We note that it is not only the total radius searched that contributes to the work done by the algorithm. We must be careful to account for the work that it takes to check for when threads are subsumed.

Lemma 5.16 (Runtime in terms of total radius)**.** *On a given day t, the runtime of Algorithm 5.9 is at most $O(1)$ times the total radius searched at the end of Algorithm 5.9.*

Proof. We assume that each operation of running a thread for "one step" (Algorithm 5.9, Line 10) can be done in $O(1)$ time. In particular we note that the interleaving schedule of the threads (based on rate) is fixed and does not depend on the input. Thus it can be computed in advance.

For each step that the ith fastest thread is run (Algorithm 5.9, Line 10), the algorithm performs the following overhead:

- The algorithm steps through the linked list to find the associated thread in $O(i)$ time.
- The algorithm checks the thread against all faster threads to see if it has been subsumed for each of the $i-1$ faster threads. This requires one distance query for each of the faster threads. By Assumption 2.2, we have that this takes $O(1)$ time.
- If the thread has been subsumed, the algorithm removes it from the linked-list of active threads in $O(1)$ time.

Thus, each step of this thread is accompanied by $O(i)$ work.

Denote the virtual radius at the end of Algorithm 5.9 by v. Over the run of Algorithm 5.9, the total work done by the thread(s) that run at the ith fastest rate is

$$v \cdot \frac{1}{i^2 \ln^2 i} \cdot O(i) = v \cdot O\left(\frac{1}{i \ln^2 i}\right).$$

Summing over all threads, we have that the total work can be bounded by

$$v + \sum_{i=2}^{\infty} v \cdot O\left(\frac{1}{i \ln^2 i}\right) = O(1) \cdot v.$$

By Lemma 5.15, we have that v is within $O(1)$ of the sum of the radii searched by Algorithm 5.9. Thus, the total work done by Algorithm 5.9 is at most $O(1)$ times the sum of the radii that it searches. □

A key part of our analysis is showing that even if all solutions close to today's solution S_t were seen very far in the past, our algorithm will eventually raise the rates of these threads to a reasonable amount. We do this by showing that enough other threads will be subsumed, clearing the way for threads lower down the chain.

Lemma 5.17 (Slow threads have bounded lifetime)**.** *Let i be a thread that is running at rate $\geq \frac{1}{(k-1)^2 \ln^2(k-1)}$ in Algorithm 5.9 at some point. Let j be another thread running at a slower rate than i. Thread j is subsumed in at most $k^3 \ln^2 k \cdot \mathbf{d}(S_i, S_j)$ additional units of virtual radius.*

Proof. Assume for the sake of contradiction that j is not subsumed in the next $k^3 \ln^2 k \cdot \mathbf{d}(S_i, S_j)$ units of virtual radius. For i faster than j, and thread j not subsumed, Algorithm 5.9 maintains that $\text{rate}(i) > \text{rate}(j)$. Thus, $(\text{radius}(i) - \text{radius}(j))$ is always nonnegative, and monotonically nondecreasing with virtual radius. Over the next $k^3 \ln^2 k \cdot \mathbf{d}(S_i, S_j)$ units of virtual radius, we have that

$$\Delta(\text{radius}(i) - \text{radius}(j)) \geq k^3 \ln^2 k \cdot \mathbf{d}(S_i, S_j)\,(\text{rate}(i) - \text{rate}(j)),$$

where $\Delta(\text{radius}(i) - \text{radius}(j))$ is the change in $\text{radius}(i) - \text{radius}(j)$. Since $\text{rate}(i) \geq \frac{1}{(k-1)^2 \ln^2(k-1)}$ and i is faster than j, we have that

$$\begin{aligned}
\text{rate}(i) - \text{rate}(j) &\geq \frac{1}{(k-1)^2 \ln^2(k-1)} - \frac{1}{k^2 \ln^2 k} \\
&\geq \frac{1}{(k-1)^2 \ln^2 k} - \frac{1}{k^2 \ln^2 k} \\
&= \frac{2k+1}{k^2(k-1)^2 \ln^2 k} \\
&\geq \frac{1}{k^3 \ln^2 k}.
\end{aligned}$$

Thus, after $k^3 \ln^2 k \cdot \mathbf{d}(S_i, S_j)$ units of virtual radius, we have that

$$\begin{aligned}
\Delta(\text{radius}(i) - \text{radius}(j)) &\geq \mathbf{d}(S_i, S_j) \\
\text{radius}(i) - \text{radius}(j) &\geq \mathbf{d}(S_i, S_j).
\end{aligned}$$

If thread i is still active at the end of these iterations, this implies that thread i kills thread j (Algorithm 5.9, line 16). Otherwise let thread i^* be the active thread that subsumes thread i. By Lemma 5.13, we have that

$$\mathbf{d}(S_i, S_{i^*}) \leq \text{radius}(i^*) - \text{radius}(i).$$

Thus, we get that

$$\begin{aligned}
\mathbf{d}(S_{i^*}, S_j) &\leq \mathbf{d}(S_{i^*}, S_i) + \mathbf{d}(S_i, S_j) \\
&\leq \text{radius}(i^*) - \text{radius}(i) + \text{radius}(i) - \text{radius}(j) \\
&\leq \text{radius}(i^*) - \text{radius}(j),
\end{aligned}$$

which implies that thread i^* will kill thread j. Thus, thread j must be subsumed, and we reach a contradiction. □

Theorem 5.18 (Competing with k trajectories online in radius *and* runtime)**.** *Algorithm 5.9 is $O(k^4 \ln^2 k)$-competitive with any set of k trajectories in the total radius that it searches. Furthermore, the total runtime of the algorithm over T days can be bounded by $O(1)$ times the total radius that it searches.*

Proof. Fix a collection of trajectories $\left(\{P_t : t \in \mathcal{T}^{(1)}\}, \ldots, \{P_t : t \in \mathcal{T}^{(k)}\}\right)$. We begin by bounding the cost of Algorithm 5.9 on days that correspond to trajectory 1.

Consider a particular day t that corresponds to trajectory 1. We claim that eventually, if the algorithm runs longs enough, there will be at most $k-1$ active threads with higher rates than thread $\text{prev}^{(1)}(t)$.

We proceed iteratively. If there are $\leq k-1$ active threads with higher rates than thread $\text{prev}^{(1)}(t)$, we are done. Otherwise, of the k threads with rate $1, \ldots, \frac{1}{k^2 \ln^2 k}$, there must be two of them that belong to the same trajectory q. Let $\mathcal{T}_t^{(q)}$ be the set of days between $\text{prev}^{(1)}(t)$ and t that are associated with trajectory q. We can bound the total distance between any two solutions in $\mathcal{T}_t^{(q)}$ by the total distance the qth trajectory moves in this time as

$$\leq \sum_{t' \in \mathcal{T}_t^{(q)} \setminus \{\text{prev}^{(q)}(t)\}} \mathbf{d}\left(t', \text{next}^{(q)}(t')\right).$$

Let $t^{(q)*}$ be the thread of $\mathcal{T}_t^{(q)}$ with the highest rate. By the selection of q, we know that $t^{(q)*}$ has rate at least $\frac{1}{(k-1)^2 \ln^2(k-1)}$. By Lemma 5.17, this means that all other members of $\mathcal{T}_t^{(q)}$ will be subsumed in

$$\leq k^3 \ln^2 k \cdot \sum_{t' \in \mathcal{T}_t^{(q)} \setminus \{\text{prev}^{(q)}(t)\}} \mathbf{d}\left(t', \text{next}^{(q)}(t')\right)$$

additional units of virtual radius. Thus, after this many units of virtual radius, there can be at most one thread belonging to trajectory q that has a higher rate than $\text{prev}^{(1)}(t)$. We say that trajectory q has "collapsed" between $\text{prev}^{(1)}(t)$ and t.

Iterating this argument at most $k-1$ times gives us that after at most

$$k^3 \ln^2 k \cdot \sum_{q=2}^{k} \sum_{t' \in \mathcal{T}_t^{(q)} \setminus \{\text{prev}^{(q)}(t)\}} \mathbf{d}\left(t', \text{next}^{(q)}(t')\right)$$

iterations of the algorithm, there can be at most $k-1$ active threads with higher rate than thread $\text{prev}^{(1)}(t)$. Thus, thread $\text{prev}^{(1)}(t)$ has rate $\geq \frac{1}{k^2 \ln^2 k}$.

This means that in at most $k^2 \ln^2 k \cdot \mathbf{d}(S_{\text{prev}^{(1)}(t)}, S_t)$ additional units of virtual radius, we have that

$$\mathbf{d}(S_{\text{prev}^{(1)}(t)}, S_t) \leq \text{radius}(\text{prev}^{(1)}(t)).$$

Therefore, by Lemma 5.14, there must be some thread that solves day t in at most

$$k^3 \ln^2 k \cdot \sum_{q=2}^{k} \sum_{t' \in \mathcal{T}_t^{(q)} \setminus \{\text{prev}^{(q)}(t)\}} \mathbf{d}\left(t', \text{next}^{(q)}(t')\right) \;+\; k^2 \ln^2 k \cdot \mathbf{d}(S_{\text{prev}^{(1)}(t)}, S_t)$$

units of virtual radius, and by Lemma 5.15 the virtual radius of an algorithm is within a $O(1)$ factor of the total radius searched.

Now, we can sum the total radius searched over all days t belonging to trajectory 1. The total radius searched over these days can be bounded by

$$\leq \sum_{t \in \mathcal{T}^{(1)}} \left[O(k^3 \ln^2 k) \cdot \sum_{q=2}^{k} \sum_{t' \in \mathcal{T}_t^{(q)} \setminus \{\text{prev}^{(q)}(t)\}} \mathbf{d}\left(t', \text{next}^{(q)}(t')\right) + O(k^2 \ln^2 k) \cdot \mathbf{d}(S_{\text{prev}^{(1)}(t)}, S_t) \right]$$

$$\leq O(k^3 \ln^2 k) \sum_{q=1}^{k} \sum_{t \in \mathcal{T}^{(q)}} \mathbf{d}(S_{\text{prev}^{(q)}(t)}, S_t)$$

$$\leq O(k^3 \ln^2 k) \cdot \sum_{q=1}^{k} \text{cost}(\{P_t : t \in \mathcal{T}^{(q)}\}),$$

where the last line follows by Corollary 5.6 because $\sum_{t \in \mathcal{T}^{(q)}} \mathbf{d}(S_{\text{prev}^{(q)}(t)}, S_t)$ is the cost of "Predict yesterday's solution" run on trajectory q alone.

Finally, the same bound holds for all k trajectories, so we can conclude that the total radius searched by Algorithm 5.9 over all days is

$$\leq O(k^4 \ln^2 k) \cdot \sum_{q=1}^{k} \text{cost}(\{P_t : t \in \mathcal{T}^{(q)}\}),$$

and therefore Algorithm 5.9 is $O(k^4 \ln^2 k)$ competitive against the k trajectories. Finally, Lemma 5.16 tells us that the total runtime of Algorithm 5.9 is at most $O(1)$ times the total radius searched by Algorithm 5.9. □

Remark 5.19. Instead of running the threads at quadratically decaying rates, i.e., the ith fastest thread runs at rate $\frac{1}{i^2 \ln^2 i}$, we could run them at harmonically decaying rates, i.e., the ith fastest thread runs at rate $\frac{1}{i \ln^2 i}$. This would result in an improved competitive ratio of $O(k^3 \ln^2 k)$ against any k trajectories, in the sum of the radii searched (number of steps taken of the subroutine $\mathcal{A}$). However, the total runtime could scale as badly as an $O(T)$ factor times the sum of the radii, where T is the time horizon.

The competitive ratio of the harmonic rate decay strategy is still worse than that of the reduction to k-server, which achieves competitive ratio $O(k^2)$ (Corollary 5.8). However, the rate decay strategy also has the benefit that it is oblivious to the setting of k, and therefore achieves the guarantee for all k simultaneously.

6 Future Directions

Our work leaves open a number of interesting research directions.

- Investigate whether it is possible to get more efficient algorithms for settings with a specific structure. In this work, we considered the setting where we can group instances using a k-wise partition. Are there other forms of structure that we can take advantage of?
- Provide an algorithm with an improved competitive ratio for the online setting, or provide a lower bound for this setting. Currently, the best lower bound we have is $\Omega(k)$, which carries over from the offline setting (Remark 3.7). It would also be interesting to match the competitive ratio $O(k^2)$ of the k-server based algorithm, but achieve it for all k simultaneously.
- Find other warm-start and/or local-search type settings to which our techniques are applicable.
- In this work, we define the Online Ball Search problem (Definition 5.1), and investigate some connections to the k-server problem. Can algorithms for Online Ball Search be transformed into algorithms for k-server? Alternatively, can we show a separation between these two problems?

References

[AA92] Noga Alon and Yossi Azar. On-line steiner trees in the euclidean plane. In *Proceedings of the Eighth Annual Symposium on Computational Geometry*, SCG '92, page 337–343, New York, NY, USA, 1992. Association for Computing Machinery.

[AB24] Arpit Agarwal and Eric Balkanski. Learning-augmented dynamic submodular maximization, 2024.

[ACE+23] Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Mixing predictions for online metric algorithms. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 969–983. PMLR, 23–29 Jul 2023.

[AGKP22] Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. Online algorithms with multiple predictions. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 582–598. PMLR, 17–23 Jul 2022.

[Bal21] Maria-Florina Balcan. *Data-Driven Algorithm Design*, page 626–644. Cambridge University Press, 2021.

[BBMN15] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph (Seffi) Naor. A polylogarithmic-competitive algorithm for the k-server problem. *J. ACM*, 62(5), nov 2015.

[BCLP23] Sayan Bhattacharya, Martin Costa, Silvio Lattanzi, and Nikos Parotsidis. Fully dynamic k-clustering in tildeo(k) update time. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[BDCBL92] Shai Ben-David, Nicolò Cesa-Bianchi, and Philip M. Long. Characterizations of learnability for classes of O, . . . , n-valued functions. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 333–340, New York, NY, USA, 1992. Association for Computing Machinery.

[BGJ21] Anup Bhattacharya, Dishant Goyal, and Ragesh Jaiswal. Hardness of Approximation for Euclidean k-Median. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021)*, volume 207 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:23, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[BKST21] Nina Balcan, Mikhail Khodak, Dravyansh Sharma, and Ameet Talwalkar. Learning-to-learn non-convex piecewise-lipschitz functions. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[BPR+17] Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 13(2), March 2017.

[BSV21] Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Generalization in portfolio-based algorithm selection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35:12225–12232, 05 2021.

[BYCR93] Ricardo Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. Searching in the plane. *Inf. Comput.*, 106:234–252, 1993.

[CSVZ22] Justin Chen, Sandeep Silwal, Ali Vakilian, and Fred Zhang. Faster fundamental graph algorithms via learned predictions. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 3583–3602. PMLR, 17–23 Jul 2022.

[DIL+21] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[DIL+22] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Algorithms with prediction portfolios. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[DMVW23] Sami Davies, Benjamin Moseley, Sergei Vassilvitskii, and Yuyan Wang. Predictive flows for faster ford-fulkerson. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 7231–7248. PMLR, 23–29 Jul 2023.

[GGK16] Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: Maintaining a constant-competitive steiner tree online. *SIAM Journal on Computing*, 45(1):1–28, 2016.

[Gur23] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[HSSY24] Monika Henzinger, Barna Saha, Martin P. Seybold, and Christopher Ye. On the Complexity of Algorithms with Predictions for Dynamic Graph Problems. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, volume 287 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:25, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[IW91] Makoto Imase and Bernard M. Waxman. Dynamic steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.

[KBTV22] Mikhail Khodak, Nina Balcan, Ameet Talwalkar, and Sergei Vassilvitskii. Learning predictions for algorithms with predictions. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[KCBT24] Mikhail Khodak, Edmond Chow, Maria Florina Balcan, and Ameet Talwalkar. Learning to relax: Setting solver parameters across a sequence of linear system instances. In *The Twelfth International Conference on Learning Representations*, 2024.

[Kou09] Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.

[KP95] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, sep 1995.

[KRT93] Ming-Yang Kao, John H. Reif, and Stephen R. Tate. Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, page 441–447, USA, 1993. Society for Industrial and Applied Mathematics.

[LS16] Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. *SIAM Journal on Computing*, 45(2):530–547, 2016.

[LS23] Quanquan C. Liu and Vaidehi Srinivas. The predicted-updates dynamic model: Offline, incremental, and decremental to fully dynamic transformations, 2023.

[MMS88] Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive algorithms for on-line problems. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 322–333, New York, NY, USA, 1988. Association for Computing Machinery.

[MMS90] Mark S Manasse, Lyle A McGeoch, and Daniel D Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

[MOP09] Andrew McGregor, Krzysztof Onak, and Rina Panigrahy. The oil searching problem. pages 504–515, 09 2009.

[MS84] Nimrod Megiddo and Kenneth J. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1):182–196, 1984.

[MV21] Michael Mitzenmacher and Sergei Vassilvitskii. *Algorithms with Predictions*, page 646–662. Cambridge University Press, 2021.

[OS23] Taihei Oki and Shinsaku Sakaue. Faster discrete convex function minimization with predictions: The m-convex case. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 68576–68588. Curran Associates, Inc., 2023.

[PZ22] Adam Polak and Maksym Zub. Learning-augmented maximum flow, 2022.

[SBD20] Dravyansh Sharma, Maria-Florina Balcan, and Travis Dick. Learning piecewise lipschitz functions in changing environments. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 3567–3577. PMLR, 26–28 Aug 2020.

[SO22] Shinsaku Sakaue and Taihei Oki. Discrete-convex-analysis-based framework for warm-starting algorithms with predictions. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 20988–21000. Curran Associates, Inc., 2022.

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[vdBFNP24] Jan van den Brand, Sebastian Forster, Yasamin Nazari, and Adam Polak. *On Dynamic Graph Algorithms with Predictions*, pages 3534–3557. 2024.

[XM21] Chenyang Xu and Benjamin Moseley. Learning-augmented algorithms for online steiner tree. In *AAAI Conference on Artificial Intelligence*, 2021.

[Zin03] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 928–935. AAAI Press, 2003.