University of California
Los Angeles

Making Decisions Under Uncertainty for Large Data Domains

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Alan James Roytman

2014

Abstract of the Dissertation

Making Decisions Under Uncertainty for Large Data Domains

by

Alan James Roytman

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2014

Professor Rafail Ostrovsky, Chair

In this thesis, we study key questions that touch upon many important problems in practice which are data-intensive. How can we process this influx of data while using a small amount of memory without sacrificing solution quality? We study this question in the context of the classical k-means clustering problem for the streaming model under a data separability assumption. We design a near-optimal streaming approximation algorithm that uses small space and makes one pass over the stream.

The streaming model may be too restrictive for certain problems that demand more computational resources. Can we still provide provable guarantees for such applications, where the input arrives online? We consider this question in the context of load balancing problems for data centers and study various scheduling problems which are energy-aware. Moreover, we show that our algorithmic techniques have applications to the machine learning community for a fundamental online convex optimization problem by giving insight into fine-tuning the tradeoff between two performance benchmarks: the competitive ratio and regret.

The dissertation of Alan James Roytman is approved.

Vwani Roychowdhury

Alexander Sherstov

Eli Gafni

Rafail Ostrovsky, Committee Chair

University of California, Los Angeles

2014

To my parents and sister, for their everlasting love and support ...

Table of Contents

List of Tables

Acknowledgments

First, I would like to express my deepest gratitude to my advisor, Rafail Ostrovsky, for his continuing guidance and support throughout my graduate student career. Rafi has encouraged me and taught me how to tackle tough problems, and he has sharpened my abilities to identify important research directions. I would also like to sincerely thank Adam Meyerson, who served as my advisor for three years when I began my graduate career. Adam introduced me to the wonderful world of research and helped shape my own perspectives as a student, and I am grateful to have had the opportunity to learn from him.

I am extremely grateful to Aman Kansal, Sriram Govindan, Jie Liu, Suman Nath, Milan Vojnovic, Bozidar Radunovic, Nina Mishra, and Abhimanyu Das for hosting me during summer internships at Microsoft Research. They broadened my research horizons and made sure that my time during the summer was very well spent.

I would like to thank my collaborators Lachlan Andrew, Minghong Lin, and Adam Wierman. I am also thankful to my dissertation committee members Eli Gafni, Alexander Sherstov, and Vwani Roychowdhury. I would also like to thank Amit Sahai, for his helpful advice over the course of my graduate career.

Last but not least, I would like to especially thank my fellow graduate students Elias Bareinboim, Chongwon Cho, Aaron Coté, Sanjam Garg, Ran Gelles, Abishek Kumarasubramanian, Michael Shindler, Brian Tagiku, and Akshay Wadia. They have left an indelible mark on my life, and my years at UCLA would not have been the same without them. I would also like to thank my friends and colleagues – Shweta Agrawal, Prabhanjan Ananth, Vladimir Braverman, Nishanth Chandran, David Felber, Divya Gupta, Abhishek Jain, Bhavana Kanukurthi, Dakshita Khurana, Chen-Kuei (James) Lee, Hemanta Maji, Omkant Pandey, Anat Paskin-Cherniavsky, Vanishree Rao, Alessandra Scafuro, Tomer Weiss, Arman Yousefi, and Vassilis Zikas, who have all greatly contributed to my wonderful experience as a graduate student.

Vita

2008	B.A. in Computer Science, B.A. in Mathematics, University of California, Berkeley.
2008	Dorothea Klumpke Roberts Prize, University of California, Berkeley.
2008	Camp Fellowship, University of California, Los Angeles.
2011	M.S. in Computer Science, University of California, Los Angeles.

Publications

Milan Bradonjić, Gunes Ercal, Adam Meyerson, and Alan Roytman. The Price of Mediation. *Discrete Mathematics and Theoretical Computer Science*, **16**(1):31-60, 2014.

Ran Gelles, Rafail Ostrovsky, and Alan Roytman. Efficient Error-Correcting Codes for Sliding Windows. In *Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science*, 2014.

Adam Meyerson, Alan Roytman, and Brian Tagiku. Online Multidimensional Load Balancing. In *Proceedings of the 16th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*. Springer, Berlin Heidelberg, 2013.

Alan Roytman, Aman Kansal, Sriram Govindan, Jie Liu, and Suman Nath. PACMan: Performance Aware Virtual Machine Consolidation. In *Proceedings of the 10th International Conference on Autonomic Computing*, 2013.

Lachlan L. H. Andrew, Siddharth Barman, Katrina Ligett, Minghong Lin, Adam Meyerson, Alan Roytman, and Adam Wierman. A Tale of Two Metrics: Simultaneous Bounds on Competitiveness and Regret. In *Proceedings of the 26th annual ACM Conference on Learning Theory*, 2013.

Vladimir Braverman, Adam Meyerson, Rafail Ostrovsky, Alan Roytman, Michael Shindler, and Brian Tagiku. Streaming k-means on Well-Clusterable Data. In *Proceedings of the 22nd annual ACM-SIAM Symposium on Discrete Algorithms*, 2011.

Milan Bradonjić, Gunes Ercal-Ozkaya, Adam Meyerson, and Alan Roytman. On the Price of Mediation. In *Proceedings of the 10th ACM Conference on Electronic Commerce*, 2009.

CHAPTER 1

Introduction

In this thesis, we propose two key questions that touch upon many important applications in practice and address problems which are accurately modeled within the context of theoretical computer science. Applications which are extremely data-intensive are indeed growing in importance today, as more data becomes available and is being generated at an exponential rate. As data continues to grow, algorithms which are able to handle a constant stream of incoming data while simultaneously using a very small amount of memory can have a significant impact in practice (since storing such a large amount of data is infeasible). Applications which have vast amounts of information to analyze include stock market transactions, DNA sequences, and packets flowing through a network, and they naturally give rise to the so-called "streaming" model. Here, the input is a stream of elements revealed to the algorithm over time, so it is inherently an "on-the-fly" process. Hence, a very pressing question arises: how can we make a single pass over the incoming data while using a small amount of memory, and yet still guarantee near-optimal solutions?

We study this question in the context of clustering problems. For the classical k-means problem, we provide a streaming algorithm that uses $O(k \log n)$ memory and makes a single pass over the stream, where n is the number of points to be clustered. Under a data separability assumption, our algorithm produces a near-optimal clustering in high-dimensional Euclidean space. The data assumption is practical, as it captures the notion that using k means to serve as cluster centers is very meaningful.

The streaming model may be too restrictive for certain applications that may have larger memory requirements, but many applications can still benefit from algorithms which operate under uncertainty. Hence, this leads us to the second part of this thesis regarding

online algorithms. In practice, especially for applications with a large amount of data, it is unreasonable to assume that the input is given up front to the algorithm. As a motivating example, every year companies spend billions of dollars to maintain and run data centers, so even a modest improvement via energy-aware scheduling can lead to significant savings. However, in practice, computer systems are unaware of what tasks will need to be run in the future. A natural question is, how do we design algorithms that can handle input arriving on the fly while providing provable guarantees on solution quality?

To address this question, we model applications in a theoretical framework and design efficient algorithms that provide provable guarantees on solution quality relative to an optimal solution that knows the future. We study various scheduling and load balancing problems which are energy-aware and design algorithms that optimize an objective function which captures the notion of minimizing energy. We introduce the notion of multidimensionality (where each dimension represents a system component such as, say, the CPU) in the context of load balancing. For an arbitrary number of dimensions d, we give an $O(\log d)$-competitive algorithm on the makespan for the identical machines setting. We also study the setting in which machines have an activation cost and give polylogarithmic competitive results for both the makespan and the total activation cost.

In addition, we study a data center optimization problem (for one dimension) and give a 2-competitive algorithm. Moreover, we show how our algorithm is not only important for the online algorithms community, but it can also be modified to have applications for the machine learning community. In particular, we formulate and design an algorithm for a very general problem that is closely related to the classical metrical task systems problem studied within the online algorithms community and the fundamental online convex optimization problem studied within the machine learning community. We show that our algorithm is able to fine-tune the tradeoff between two popular benchmarks: the competitive ratio and regret. In fact, we prove that our algorithm is tight by showing an inherent incompatibility between the two benchmarks.

We now discuss the organization of this thesis and provide a general overview of our results in each chapter.

Chapter 2: One of the most fundamental problems in data analysis is known as the k-means clustering problem. This problem has received considerable attention in the literature, and more recently, the *streaming* version of this problem has been studied, culminating in a series of results [HM04, FS05, FMS07, Che09] that produced a $(1+\epsilon)$-approximation for k-means clustering in the streaming setting. Unfortunately, optimizing the k-means objective is Max-SNP hard. This implies that all algorithms which achieve a $(1+\epsilon)$-approximation must take time exponential in k, unless P = NP.

If we wish to avoid this exponential dependence on k, some additional assumptions must be made about the input in order to simultaneously guarantee a high quality approximation and polynomial running time. The paper of [ORS06] introduced the very natural assumption of *data separability*: the notion that using k means is somehow much better than using $k-1$ means. This assumption closely reflects how k-means is used in practice, as it implies that there is a particularly meaningful choice for k. In particular, the assumption allowed the authors to create a high quality approximation for k-means clustering in the non-streaming setting with polynomial running time, even for large values of k. Their work left open a natural and important question: are similar results possible in the *streaming* setting? This is the question we answer in this chapter, by providing a positive result using substantially different techniques.

In particular, we show a near-optimal streaming approximation algorithm for the k-means problem in high-dimensional Euclidean space with sublinear memory while making a single pass over the stream, under the same data separability assumption. Our algorithm offers significant improvements in both space and running time over previous work.

The novel techniques we develop along the way imply a number of additional results: we improve the performance guarantee for online facility location by showing that the algorithm achieves the same guarantees within constants of its expected behavior with high probability (in contrast, the algorithm in [Mey01] gave bounds only in expectation); we develop a constant approximation algorithm for the general class of semi-metric clustering problems; we improve (even without the data separability assumption) the space requirements of the best previous result for streaming k-median by a logarithmic factor; finally, we design a

"re-sampling method" in the Euclidean setting to convert our constant approximation algorithm for the general semi-metric clustering class of problems to a near-optimal solution when the data is well-clusterable. In particular, our approximation factor approaches one as the separability of the data increases.

The work presented in this chapter is based on [BMO11].

Chapter 3: Energy efficient algorithms are becoming critically important, as huge data centers and server farms have an increasing impact on monetary and environmental costs. Motivated by such issues, we study online load balancing from an energy-aware perspective in this chapter. We consider the setting in which we are given m machines, each with some energy activation cost c_i and d dimensions (where each dimension represents a component on the machines). There are n jobs which arrive on the fly and must be assigned to machines. Each job induces a load on its assigned machine along each dimension. We must select a set of machines to activate and assign jobs to active machines as they arrive so that the total activation cost of the machines falls within an energy budget B and the largest load over all machines and dimensions (i.e., the makespan) is at most Λ.

We first study the model in which machines are unrelated and can have arbitrary activation costs (that is, the load of a job on a dimension can change arbitrarily depending on its assigned machine). For this problem, which we call Machine Activation, our framework extends the recent work of [KLS10] to the online model, in which we assume that jobs arrive on the fly, and the work of [ABF13] to the multidimensional case. We consider a variant where the target makespan Λ and budget B are given. The first main result is an online algorithm which is $O(\log(md)\log(nm))$-competitive on the load Λ and $O(d\log^2(nm))$-competitive on the energy budget B. We also address cases where one parameter is given and we are asked to minimize the other, or where we want to minimize a convex combination of the two. Running our previous algorithm in phases gives results for these variants. We prove lower bounds indicating that the effect on the competitive ratio due to multiple phases is necessary.

Our second main result for this chapter is in the same setting, except that all machines are identical (i.e., the load of a job on a dimension is the same across all machines) and

have no activation cost. We call this problem Vector Load Balancing, for which we have two objectives: minimize the largest load induced over all machines and dimensions (the makespan objective), and minimize the sum of the largest loads induced on each machine (the energy objective). We give an online algorithm that is $O(\log d)$-competitive on the makespan objective, which improves even on the best prior offline result, and $O(\log d)$-competitive on the energy objective if the target makespan is given; without this knowledge, we show that it is impossible to get a competitive ratio independent of m.

The work presented in this chapter is based on [MRT13].

Chapter 4: In this chapter, we consider algorithms for very general "smoothed online convex optimization" problems. In particular, our formulation is a variant of the class of online convex optimization problems typically studied within the machine learning community and is strongly related to the classical metrical task systems problem studied within the online algorithms community. Prior literature on these problems has focused on two performance metrics: regret (typically studied in the machine learning literature) and the competitive ratio (typically studied in the online algorithms literature). There exist known algorithms with sublinear regret and known algorithms with constant competitive ratios; however, no known algorithm achieves both simultaneously. Moreover, because the two performance metrics are so different, the algorithms developed within the two communities tend to have very different styles and techniques.

We show that this is due to a fundamental incompatibility between these two performance metrics – no algorithm (deterministic or randomized) can simultaneously achieve a sublinear regret and a constant competitive ratio, even in the simple setting when the objective functions are linear and the decision space is one-dimensional. However, we also exhibit an algorithm that, for the important special case of one-dimensional decision spaces, provides sublinear regret while maintaining a competitive ratio that grows arbitrarily slowly. In this sense, our results are essentially tight in fine-tuning the tradeoff between the competitive ratio and regret.

The work presented in this chapter is based on [ABL13].

CHAPTER 2

Streaming k-means on Well-Clusterable Data

2.1 Introduction

In this chapter, we consider the problem of Euclidean k-means in the streaming model. There are n points in Euclidean space that are read sequentially; when the data stream finishes, we must select k of these to designate as *facilities*. Our cost is the sum of squared distances from each point in the stream to its nearest facility.

A series of recent results [HM04, FS05, FMS07, Che09] produced $(1+\epsilon)$-approximations for streaming k-means. The general approach first appeared in the paper of Har-Peled and Mazumdar [HM04]. They used the concept of a (k, ϵ)-coreset: a weighted set of points such that the cost of any set of k facilities on the coreset is within $1+\epsilon$ of the cost on the original points. Subsequent results improved the time and space bounds for computing coresets. Frahling and Sohler [FS05] designed a new way to construct coresets based on grids and Chen [Che09] designed a new way to generate coresets by randomly sampling from rings around an approximate set of facilities. In addition, Feldman, Monemizadeh, and Sohler [FMS07] used the concept of a weak coreset (due to [BHI02]), where the size of the coreset is independent of n.

While these recent results claimed a $(1+\epsilon)$-approximation for streaming k-means, this requires producing an exact solution on the coreset itself, which takes time $2^{\tilde{O}(k/\epsilon)}$. When k is part of the input, this is exponential time, and it cannot be substantially improved since the objective is Max-SNP hard to optimize [BBG09].

In this work, we are interested in algorithms with truly polynomial running times. We seek to produce good approximations while optimizing space and runtime requirements.

Since we cannot obtain a $(1+\epsilon)$-approximation in polynomial time, we will make the natural assumption of data separability, introduced by Ostrovsky, Rabani, Schulman, and Swamy [ORS06]; this closely reflects how k-means is used in practice and allowed the authors to create a good approximation in the non-streaming setting. Our main result is a streaming algorithm, which produces a set of k means after processing a stream of n data points that arrive one at a time while making only a single pass through the data. We guarantee that the space requirement and processing time per point are logarithmic in n, and we produce an approximation factor of $1 + O(\epsilon) + O(\sigma^2)$ when the original data is σ-separable. While it is possible to modify the prior coreset-based approaches to obtain similar approximation bounds, our algorithm improves substantially on both space and time requirements. In fact, our algorithm requires less space (by a factor of $\log n$) than the best previous constant approximation for the problem. We give both results in expectation and with high probability; our results are compared to previous coreset-based results for k-means in Table 2.1.

The techniques that we develop along the way allow us to establish additional results: we provide a high probability performance guarantee for online facility location (Meyerson's results [Mey01] gave bounds only in expectation); we develop a constant approximation method for the general class of semi-metric clustering problems; we improve (even without σ-separability) by a logarithmic factor the space requirements of the previous best streaming algorithm for k-median; finally we show a novel "re-sampling method" in the streaming setting to reduce any constant approximation for clustering to $1 + O(\sigma^2)$.

Related Work

The k-means problem was considered as early as 1956 by Steinhaus [Ste56]. A simple local search heuristic for the problem was proposed in 1957 by Lloyd [Llo82]. The heuristic begins with k arbitrarily chosen points as facilities. At each stage, it allocates the points into clusters (where each point is assigned to its closest facility) and then computes the center of mass for each cluster. These centers of mass become the new facilities for the next phase, and the process repeats until the solution stabilizes. Lloyd's algorithm has a rich history,

Result	Space (points)	Runtime	PROB
[HM04] + [ORS06]	$O\left(\frac{k}{\epsilon^d}\log^{2d+2} n\right)$	$O_d\left(n\left(k^5+\log^2\left(\frac{k}{\epsilon}\right)\right)\right)$	EXP
[FS05] + [ORS06]	$O\left(\frac{(\log\Delta+\log n)^3 k^2 \log^4\Delta}{\epsilon^{2d+6}}\right)$	$O_d\left(n\log^2\Delta\left(\log\Delta+\log n\right)\right)$	EXP
Ours	$O\left(\frac{k}{\epsilon}\log n\right)$	$O_d\left(nk\log n\right)$	EXP
[FMS07] + [ORS06]	$O\left(\frac{k^2}{\epsilon^5}\log^{10} n\right)$	$O_d\left(\frac{nk^2}{\epsilon}\log^2 n\right)$	WHP
[Che09] + [ORS06]	$O\left(\left(\frac{dk}{\epsilon}\right)^2\log^8 n\right)$	$O_d\left(nk\log^2 n\ \mathrm{polylog}\left(\frac{k}{\epsilon}\right)\right)$	WHP
Ours	$O\left(\frac{k}{\epsilon}\log^2 n\right)$	$O_d\left(nk\log n\right)$	WHP

Table 2.1: Streaming $(1+O(\epsilon)+O(\sigma^2))$-approximations to k-means

receiving attention from psychologists in 1959-67 [TB70] and the computer science community from 1960 to the modern day [Max60, Mac67, DLR77, LBG80, GG92, GN98, JMF99, Tot59, Zad64, For65, Fis58, BH65, Bal65, Jan66, ARS98, PM99, Phi02, KMN02, ORS06]. Unfortunately, Lloyd's algorithm has no provable approximation bound, and arbitrarily bad examples exist. Furthermore, the worst-case running time is superpolynomial [AV06]. Despite these drawbacks, Lloyd's algorithm (frequently known simply as `k-means`) remains common in practice.

The best polynomial-time approximation factor for k-means is by Kanungo, Mount, Netanyahu, Piatko, Silverman, and Wu [KMN02]. They based their result on the k-median algorithm of Arya, Garg, Khandekar, Meyerson, Munagala, and Pandit [AGK01]. Both papers use local search; the k-means case produces a $(9+\epsilon)$-approximation. However, Lloyd's experimentally observed runtime is superior, and this is a high priority for real applications.

Ostrovsky, Rabani, Schulman and Swamy [ORS06] observed that the value of k is typically selected such that the data is "well-clusterable" rather than being an arbitrary part of the input. They defined the notion of σ-separability, where the input to k-means is said to be σ-separable if reducing the number of facilities from k to $k-1$ would increase the cost of the optimum solution by a factor $\frac{1}{\sigma^2}$. They designed an algorithm with approximation ratio $1+O(\sigma^2)$. They also showed that their notion of σ-separability is robust and generalizes a number of other intuitive notions of "well-clusterable" data. The main idea of their

algorithm is a randomized seeding technique which guarantees (with high probability) one initial facility belonging to each optimum cluster. They then perform a "ball k-means" step (Lloyd-like re-clustering) using only points which are near facilities. Subsequently, Arthur and Vassilvitskii [AV07] showed that the same procedure produces an $O(\log k)$ approximation for arbitrary instances of k-means.

When a k-means type algorithm is run in practice, the goal is to group the data based on a natural clustering. Balcan, Blum, and Gupta [BBG09] used this observation to extend the notion of σ-separability to η-closeness: two clusterings are η-close if they disagree on only an η-fraction of the points, and an instance of the problem has the (c, η) property if any c-approximation is η-close to the target clustering for that instance. Their main contribution is to show how to use an existing constant approximation to modify a solution on an agreeable dataset to be a better solution. When the (c, η) property assumption holds, they are able to find very accurate approximations to the subjective correct clustering. In particular, any instance of k-means that has a $(1+\alpha, \eta)$-property can be clustered to be $O(\eta/\alpha)$ close to the target. However, their approach is memory intensive and not amenable to direct adaptation for the streaming model.

Each of these algorithms assumed that the entire input was available for processing in any form the algorithm designer needed. Our work focuses instead on the streaming model, where the set of points to cluster is extremely large and the algorithm is required to make only a single in-order pass through the data. This is typically used to model the case where there is a vast amount of data and it must be read without random access.

The early work on the more general problem of streaming k-service clustering focused on streaming k-median. In 2000, Guha, Mishra, Motwani, and O'Callaghan [GMM00] produced an $O(2^{1/\epsilon})$-approximation for streaming k-median using $O(n^{\epsilon})$ memory. Their algorithm reads the data in blocks, clustering each using some non-streaming approximation, and then gradually merges these blocks when enough of them arrive. An improved result for k-median was given by Charikar, O'Callaghan, and Panigrahy in 2003 [COP03], producing an $O(1)$-approximation using $O(k \log^2 n)$ space. Their work was based on guessing a lower bound on the optimum k-median cost and running $O(\log n)$ parallel versions of the online facility

location algorithm of Meyerson [Mey01] with facility cost based on the guessed lower bound. When these parallel calls exceeded the approximation bounds, they would be terminated and the guessed lower bound on the optimum k-median cost would increase.

A recent result for streaming k-means due to Ailon, Jaiswal, and Monteleoni [AJM09] is based on a divide and conquer approach, similar to the k-median algorithm of Guha, Meyerson, Mishra, Motwani, and O'Callaghan [GMM03]. It uses the result of Arthur and Vassilvitskii [AV07] as a subroutine, finding $3k \log k$ centers and producing an approximation ratio of 64 with probability at least $1/4$ in the non-streaming setting. By dividing the input stream and running this repeatedly on pieces of the stream, they achieve an $O\left(2^{O(1/\epsilon)} \log k\right)$-approximation using $O(n^{\epsilon})$ memory.

High Level Ideas

Our goal is to produce a fully polynomial-time streaming approximation for k-service clustering. A natural starting point is the algorithm of Charikar, O'Callaghan, and Panigrahy [COP03]; however, their result as stated applies only to the k-median problem. Since their algorithm depends heavily on calls to the online facility location algorithm of Meyerson [Mey01], we first consider (and improve) results for this problem.

We produce new high probability bounds on the performance of online facility location, showing that the algorithm achieves within constants of its expected behavior with probability $1 - \frac{1}{n}$ (Theorem 2.1). To achieve this result, we inductively bound the probability of any given service cost being obtained prior to opening a facility in each of a collection of facility-less regions. We then combine this with deterministic bounds on the service cost subsequent to opening a facility in the local region, and with Chernoff bounds on the number of facilities opened. Coupling our result with the algorithm of Charikar, O'Callaghan, and Panigrahy [COP03] improves our memory bound and processing time per point by a $\Theta(\log n)$-factor. Our analysis extends to cases where the triangle inequality holds only approximately, allowing us to apply the streaming algorithm to k-means as well. This yields the first streaming $O(1)$-approximation for k-means and k-median to store only $O(k \log n)$

points in memory (Theorem 2.2).

The execution of the algorithm of [COP03] is divided into phases, each of which corresponds to a "guess" at the optimum cost value. Each phase induces overhead to merge the existing clusters from the previous phase. The number of these phases is bounded by $O(n)$; we show that a modification of the algorithm along with an appropriate choice of constants can guarantee that each phase processes at least $k(1 + \log n)$ new points from the data stream (Theorem 2.3), thus reducing the number of phases to $O(n/(k \log n))$. This reduction improves the overall running time to $O(nk \log n)$.

Next, we would like to improve our approximation result to an FPTAS for the important case of Euclidean k-means. This is hard in general, as the problem is Max-SNP hard [BBG09]. We instead make the σ-separability assumption of Ostrovsky, Rabani, Schulman, and Swamy [ORS06] and show that we can obtain a $(1 + O(\epsilon) + O(\sigma^2))$-approximation that stores $O\left(\frac{k}{\epsilon} \log n\right)$ points in memory and has polynomial running time.

The first step is to consider applying a ball k-means step to our $O(1)$-approximation; this involves selecting the points which are much closer to one of our facilities than to any other (the "ball" of that facility) and computing the center of mass on those points. We show that, given any $O(1)$-approximation to k-means, applying the ball k-means step will reduce the approximation factor to $1 + O(\sigma^2)$. The main idea is that the optimum facilities for such an instance must be far apart; any $O(1)$-approximation must include a facility close to each of the optimal facilities. Combining these facts gives a one-to-one mapping between our facilities and the optimal facilities, and we show that the points which are very close to each of our facilities must therefore belong to distinct optimal clusters. This would enable us to produce a $(1 + O(\sigma^2))$-approximation to k-means by making two passes through the stream – the first pass would run the algorithm of Charikar, O'Callaghan, and Panigrahy [COP03] with our modifications, and the second pass would run the ball k-means step.

Of course, we wish to compute our entire solution with only one pass through the data. To do this, we prove that sampling works well for approximating the center of mass. In particular, a random sample of constant size (independent of the size of the cluster) provides

a constant approximation (Theorem 2.4). Our goal is thus to produce a suitable random sample of the points belonging to each of the balls for our final ball k-means step.

Unfortunately, we do not know what our final cluster centers will be until the termination of the stream, making it difficult to sample uniformly from the balls. Instead, we show that the clusters from our solution are formed by adding points one at a time to clusters and by merging existing clusters together. This process permits us to maintain at all times a uniformly random sample of the points belonging to each of our clusters (Section 2.3). Of course, randomly sampling from the points in these clusters is not the same as randomly sampling from the balls in the ball k-means step. However, we can show that the set we are actually sampling from (our cluster about a particular facility) and the set we should be sampling from (the points which are much closer to a particular facility than to any other one of our facilities) are roughly (within constants) the same set of points, and that as the separability value σ approaches zero, these sets of points converge and become effectively identical (Theorem 2.6).

Putting it all together, our overall result maintains a sample of size $\frac{1}{\epsilon}$ from each of our clusters at all times. The number of clusters will never exceed $O(k \log n)$, so the total memory requirement is $O\left(\frac{k}{\epsilon} \log n\right)$ points for a chosen constant ϵ. The approximation factor for our final solution is $1+O(\epsilon)+O(\sigma^2)$ for σ-separable data, and our overall running time is $O(nk \log n)$. While this result holds in expectation, we also give a similar result which holds with high probability, namely at least $1 - \frac{1}{n}$. Our space requirement for the high probability result is $O\left(\frac{k}{\epsilon} \log n \log(nd)\right)$, and by applying the result of Johnson and Lindenstrauss [JL84] we can reduce this to $O\left(\frac{k}{\epsilon} \log^2 n\right)$. We also note that the value of σ need not be known to our algorithm at runtime.

We stress that our result improves over *all* previous streaming algorithms for k-means (or k-median) in the memory requirement and running time, while obtaining very good approximation results provided the data set is "well-clusterable" (as per [ORS06]).

Our Techniques Versus Prior Work

Our improvement of the analysis from Meyerson's online facility location result [Mey01] uses similar techniques to the original paper. As before, the optimum clusters are divided up into "regions" based on proximity to the optimum center, and arguments are made about the cost prior to and subsequent to opening a facility in each region. Extending this approach to handle the approximate triangle inequality is straightforward. The main new idea involves producing a high probability bound, specifically on the service cost paid prior to opening facilities in each region. Here we use induction to produce an upper bound on the actual probability of paying at least a given cost prior to opening the facilities; by setting the target probability appropriately, we can show that the chance of exceeding the expected cost by more than a constant is exponentially small in the number of regions. Combining this with a straightforward application of Chernoff bounds (for the number of facilities that are opened) completes the result.

While our overall algorithm bears some similarity to the result of Charikar, O'Callaghan, and Panigrahy [COP03], our techniques are quite different. They break their process into phases, then show that each phase "succeeds" with reasonably high probability. They then require substantial work to bound the number of phases to be linear in the number of points. In contrast, we show that we only require "success" of a randomized algorithm at a particular critical phase; prior phases are always guaranteed to have bounded cost. This allows a substantial improvement, and unlike their work, our performance and success probability do not depend on the number of phases. Nonetheless, bounding the number of phases is important for the running time. We obtain a sublinear bound by simply requiring each phase to read in at least a logarithmic number of new points; this analysis is much simpler and enables us to perform a simple matching at the end of each phase (reducing the number of facilities sufficiently) rather than approximating k-means on the facilities of the prior phase. Of course, our ideas about using sampling and a ball k-means step to improve the approximation were not part of [COP03], although the general idea (without the sampling or streaming aspect) appeared in Ostrovsky, Rabani, Schulman, and Swamy [ORS06].

Definitions and Notation

Throughout this chapter, we use n to denote the number of points in the stream to be clustered, k to denote the desired number of means, and d to denote the dimensionality of the points. We also use ϵ as a parameter of our algorithm that controls the quality of the approximation guarantee, and Δ to denote the diameter of a dataset. For a set of points A, we let $com(A)$ denote the center of mass of A, so that $com(A) = \frac{\sum_{x \in A} x}{|A|}$.

Definition 2.1 (k-service clustering)**.** *We have a finite set X of points, a possibly infinite set Y (with $X \subseteq Y$) of potential facilities, and a cost function $\delta : X \times Y \to \Re^+$. Our goal is to select $K \subseteq Y$ of size k to be designated as facilities, so as to minimize $\sum_{i \in X} \min_{j \in K} \{\delta(i, j)\}$. The cost function is known as the service cost to connect a point to a facility.*

This encapsulates a family of problems, including *k-median*, where δ is a metric on space Y, and *k-means*, where Y is Euclidean space and δ is the square of Euclidean distance. The related *facility location* problem is formed by removing the constraint that $|K| = k$, replacing it with a facility cost f, and adding $f|K|$ to the objective function. For a point a and set A, we let $C(a, A)$ be the one-means cost of using point a as a mean for set A, so that $C(a, A) = \sum_{x \in A} \delta(x, a)$. Note that the k-means service costs satisfy the 2-approximate triangle inequality:

Definition 2.2. (α-Approximate Triangle Inequality) *If, for any points a, b, c the following applies: $\alpha[\delta(a, b) + \delta(b, c)] \geq \delta(a, c)$, then we say that the α-approximate triangle inequality is satisfied.*

Definition 2.3 (σ-separable dataset)**.** *A set of input data for the k-service clustering problem is said to be σ-separable if the ratio of the optimal k-service clustering cost to the optimal $(k-1)$-service clustering cost is at most σ^2.*

This captures the notion that the k^{th} facility must be meaningful for the clustering to be as well. This has been applied to k-means by [ORS06].

2.2 A Constant Approximation

In this section, we will provide an $O(1)$-approximation for k-service clustering, for any instance in which $X \subseteq Y$ and where the α-approximate triangle inequality applies to δ.

Algorithm 2.1 summarizes our entire process for streaming k-service clustering. It takes as input a data stream known to contain n points and a value k for the number of desired means. The algorithm is defined in terms of the constants β, γ, which will be determined later (when analyzing the algorithm). The algorithm as described also requires a (non-streaming) $O(1)$-approximation for k-service clustering, which is used as a subroutine. One candidate algorithm for this when running k-means is the approximation of Kanungo et al. [KMN02].

At several points in our algorithm, we refer to placing points at the front of the data stream. An easy way to implement this is to maintain a stack structure. When placing an item at the front of the stream, push it to the stack. When reading from the stream, check first if the stack is empty: if it is not, read by popping from the stack. If the stack is empty, read from the stream as normal. This also allows us to place items with weight on the stream, and we consider each item from the stream to be of weight one.

Our algorithm is quite similar to that of Charikar, O'Callaghan, and Panigrahy [COP03]. Both approaches run online facility location [Mey01] (lines 5-12 in our algorithm) with facility costs based on gradually improving lower bounds on the optimum cost. We show an improved online facility location analysis, which enables us to run only a single copy of online facility location (instead of $O(\log n)$ copies as in [COP03]) while maintaining a high success probability. We also show that the randomized online facility location algorithm need not "succeed" at every phase, only at the critical final phase of the algorithm; this allows us to improve our approximation factor from that of [COP03]. Finally, we show that the number of phases is bounded by $O(n/(k \log n))$ rather than $O(n)$; this improves the running time of our algorithm substantially from that of [COP03], obtaining $O(nk \log n)$ time.

1: $L_1 \leftarrow 1$
2: $i \leftarrow 1$
3: **while** solution not found **do**
4: $K \leftarrow \emptyset$
5: cost $\leftarrow 0$
6: $f \leftarrow L_i/(k(1+\log n))$
7: **while** there are points still in the stream **do**
8: $x \leftarrow$ next point from stream
9: $y \leftarrow$ facility in K that minimizes $\delta(x, y)$
10: **if** probability $\min\{\frac{weight(x)\cdot\delta(x,y)}{f}, 1\}$ **then**
11: $K \leftarrow K \cup \{x\}$
12: **else**
13: cost $\leftarrow$ cost + weight$(x) \cdot \delta(x, y)$
14: weight$(y) \leftarrow$ weight(y) + weight(x)
15: **if** cost $> \gamma L_i$ or $|K| > (\gamma - 1)(1 + \log n)k$ **then**
16: break and raise flag
17: **if** flag raised **then**
18: push facilities in K onto stream
19: $L_{i+1} \leftarrow \beta L_i$
20: $i \leftarrow i + 1$
21: **else**
22: Cluster K to yield exactly k facilities
23: Declare solution found

Algorithm 2.1: One pass, constant approximation k-service clustering algorithm

Improved Analysis of Online Facility Location

The online facility location algorithm of Meyerson [Mey01] is used implicitly in lines 7-16 of Algorithm 2.1 and works as follows. We are given a facility cost f. As each point arrives, we measure the service cost δ for assigning that point to the nearest existing facility. With

probability $\min\{\frac{\delta}{f}, 1\}$ we create a new facility at the arriving point. Otherwise, we assign the point to the nearest existing facility and pay the service cost δ. Let OPT denote the optimal k-service clustering.

Theorem 2.1. *Suppose we run the online facility location algorithm of [Mey01] with* $f = \frac{L}{k(1+\log n)}$ *where* $L \leq OPT$ *and that the service costs satisfy the* α*-approximate triangle inequality. Then the expected total service cost is at most* $(3\alpha + 1)OPT$ *and the expected number of facilities generated by the algorithm is at most* $(3\alpha+1)k(1+\log n)\frac{OPT}{L}$*. Further, with probability at least* $1 - \frac{1}{n}$*, the service cost is at most* $\left(3\alpha + \frac{2e}{e-1}\right) OPT$ *and the number of facilities generated is at most* $(6\alpha+1)k(1+\log n)\frac{OPT}{L}$*.*

Proof. Consider each optimum facility c_i^*. Let C_i^* be the points assigned by OPT to c_i^*, A_i^* be the total service cost of optimum cluster C_i^*, and $a_i^* = A_i^*/|C_i^*|$. Let δ_p^* be the optimum service cost for point p. We divide the optimum cluster C_i^* into regions S_i^j for $j \geq 1$ where $|S_i^j| = |C_i^*|/2^j$ and all the points in S_i^j have optimum service cost at most the optimum service cost of points in S_i^{j+1}. This will probably produce "fractional" points (i.e., points which are split between many regions); however, this does not affect the analysis. Let A_i^j be the total optimum service cost of points in S_i^j, so that $\Sigma_j A_i^j = A_i^*$.

For each region S_i^j, we may eventually open a facility at some point q in this region. Once we do so, subsequent points p arriving in the region must have bounded service cost of at most $\alpha(\delta_p^* + \delta_q^*)$. Since $q \in S_i^j$ and all points in S_i^j have smaller optimum service cost than points in S_i^{j+1}, we can conclude that $\delta_q^* \leq A_i^{j+1}/|S_i^{j+1}|$. Summing the resulting expression over all points in S_i^j gives us a service cost of at most $\alpha(A_i^j + 2A_i^{j+1})$. Summing this over all the regions gives a service cost of at most $3\alpha A_i^*$ subsequent to the arrival of the first facility in the regions. Note that this is a deterministic guarantee.

It remains to bound the service cost paid prior to the first facility opened in each region. In expectation, each region will pay at most f in service cost before opening a facility. Further, regions labeled $j > \log n$ contain only one point in total, and the overall service cost for this point cannot exceed f. Thus, the expected total service cost is at most $k(1 + \log n)f + 3\alpha\Sigma_i A_i^* \leq L + 3\alpha OPT$. Since $L \leq OPT$, this gives an expected service cost of at

most $1+3\alpha$ times optimum. For the high probability guarantee, let $P[x, y]$ be the probability that given x regions which do not yet have a facility, the remaining service cost due to points in these regions arriving prior to the region having a facility is more than yf. We will prove by induction that $P[x, y] \leq e^{x-y\left(\frac{e-1}{e}\right)}$, where e is the base of the natural log. Note that this is immediate for $x = 0$ and for very small values of y (i.e., $y \leq x \cdot \frac{e}{e-1}$). To prove this is always true, suppose that x is the smallest value where this can be violated, and y is the smallest value where it can be violated for this x. Thus, $P[x, y] > e^{x-y\left(\frac{e-1}{e}\right)}$. Suppose that the first request in one of the facility-less regions computes a service cost of $\delta > 0$. Then we have: $P[x, y] = \frac{\delta}{f} P[x-1, y] + \left(1 - \frac{\delta}{f}\right) P[x, y - \frac{\delta}{f}]$.

The first term corresponds to opening a facility at this point, thus reducing the number of facility-less regions by one; the second term corresponds to paying the service cost. Applying the definition of x and y:

$$e^{x-y\left(\frac{e-1}{e}\right)} < P[x, y] \leq \frac{\delta}{f} e^{x-1-y\left(\frac{e-1}{e}\right)} + \left(1 - \frac{\delta}{f}\right) e^{x-\left(y-\frac{\delta}{f}\right)\left(\frac{e-1}{e}\right)}.$$

Dividing both sides by the left hand expression leaves $1 < \frac{\delta}{ef} + \left(1 - \frac{\delta}{f}\right) e^{\frac{\delta}{f}\left(\frac{e-1}{e}\right)}$. This provides a contradiction.

Thus, the probability that the total cost prior to opening facilities in each region is more than $\frac{e}{e-1}(2k)(1 + \log n)f$ is at most $P[k(1 + \log n), \frac{e}{e-1}(2k)(1 + \log n)] \leq e^{-k(1+\log n)} \leq \frac{1}{2n}$. Substituting for f gives the bound claimed.

We now consider the facility count. The first in each region gives us a total of $k(1+\log n)$ facilities; this is a deterministic guarantee. Now we must bound the number of facilities opened in the various regions subsequent to the first. Each point p has probability δ_p/f of becoming a new facility, where δ_p is the service cost when p arrives. Note that we already had a deterministic guarantee that for points arriving after a facility is opened in their region, we have $\Sigma_p \delta_p \leq 3\alpha OPT$. Thus, we have a sum of effectively independent Bernoulli trials with expectation at most $\frac{3\alpha OPT}{f} = 3\alpha k(1 + \log n)\frac{OPT}{L}$. We can now apply Chernoff bounds for the result.

□

Algorithm Analysis

We first need to define the constants β, γ. Let c_{OFL} be the constant factor on the service cost obtained from the high probability result for online facility location given by Theorem 2.1, and let k_{OFL} be such that the online facility location algorithm guarantees to generate at most $k_{OFL}k(1+\log n)\frac{OPT}{L}$ facilities. Note that c_{OFL}, k_{OFL} are constants which depend on α and on the desired "high probability" bound for success. We now define the constants as $\beta = 2\alpha^2 c_{OFL} + 2\alpha; \gamma = \max\{4\alpha^3 c_{OFL}^2 + 2\alpha^2 c_{OFL}, \beta k_{OFL} + 1\}$.

We assume that $c_{OFL} \geq 2\alpha$ from this point on; this is implicit in the proof of Theorem 2.1 and we can always replace c_{OFL} with a larger value since it is a worst-case guarantee.

Define a phase in Algorithm 2.1 to be a single iteration of the outermost loop. Within each phase i, we maintain a lower bound L_i on OPT and run the online facility location algorithm using facility cost $f = \frac{L_i}{k(1+\log n)}$. We try reading as many points as we can until either our service cost grows too high (more than γL_i) or we have too many facilities (more than $(\gamma-1)k(1+\log n)$). At this point, we conclude that our lower bound L_i is too small, so we increase it by a factor β and start a new phase.

In a phase, we pay at most $f = L_i/k(1+\log n)$ for a weighted point and there are at most $(\gamma-1)k(1+\log n)$ weighted points from the previous phase. Our service cost for these points can be at most $(\gamma-1)L_i$, so we successfully cluster all weighted points in a phase. Thus, at the start of each phase, the stream looks like some weighted points from only the preceding phase followed by unread points. Additionally, we can show that all these points on the stream have a clustering with service cost comparable to OPT.

Lemma 2.1. *Let X' be any subset of points in the stream at the start of phase i. Then the total service cost of the optimum k-service clustering of X' is at most $\alpha \cdot OPT + \gamma\left(\frac{\alpha^2}{\beta-\alpha}\right) L_i$.*

Proof. Consider an original point $x \in X$. Say that $y \in K$ *represents* x in phase ℓ if y is the assigned facility for x or for x's phase $\ell-1$ representative. Note that the weight of $y \in K$ is the number of points it represents. Moreover, once a point x becomes represented in phase ℓ, it is represented for all future phases.

At the start of phase i, the stream looks like the weighted facilities from phase $i-1$ followed by unread points. Let us examine our cost if we use the optimum facilities (for X) to serve all of these points. Fix a point $x \in X$ and let us bound the service cost due to this point. Let j be the phase in which x was first clustered. Let $y_j, y_{j+1}, \ldots, y_{i-1}$ be x's respective representatives in phases j up through $i-1$. Then the service cost due to x will be $\delta(y_{i-1}, y^*)$, where y^* is the cheapest optimum facility for y_{i-1}. By the α-approximate triangle inequality:

$$\delta(y_{i-1}, y^*) \leq \alpha\delta(x, y^*) + \alpha\delta(x, y_{i-1}) \leq \alpha\delta(x, y^*) + \sum_{\ell=2}^{i-j} \alpha^\ell \delta(y_{i-\ell}, y_{i-\ell+1}) + \alpha^{i-j}\delta(x, y_j).$$

Thus, summing over all points x in or represented by points in X', and noting that our service cost in phase ℓ is bounded by $\gamma L_\ell \leq \gamma L_i \frac{1}{\beta^{i-\ell}}$, gives a total service cost of at most

$$\alpha \cdot OPT + \gamma\alpha L_i \sum_{\ell=1}^{i-1} \left(\frac{\alpha}{\beta}\right)^{i-\ell} = \alpha \cdot OPT + \gamma \left(\frac{\alpha^2}{\beta - \alpha}\right) L_i.$$

□

The above lemma shows that there exists a low cost clustering for the points at each phase, provided we can guarantee that $L_i \leq OPT$. Call the last phase where $L_i \leq OPT$ the *critical phase*. We will show that we in fact terminate at or before the critical phase with high probability.

Lemma 2.2. *With probability at least the success probability of online facility location from Theorem 2.1, Algorithm 2.1 terminates at or before the critical phase.*

Proof. Let i be the critical phase, and let OPT_i be the optimum cost of clustering all the points (weighted or not) seen on the stream at the start of phase i. By Lemma 2.1 and the fact that $OPT \leq \beta L_i$, we have

$$OPT_i \leq \alpha \cdot OPT + \gamma \left(\frac{\alpha^2}{\beta - \alpha}\right) L_i \leq \left(\alpha\beta + \gamma\frac{\alpha^2}{\beta - \alpha}\right) L_i.$$

Theorem 2.1 guarantees the online facility location algorithm yields a solution with at most $\beta k_{OFL}(1 + \log n)k$ facilities and of cost at most $c_{OFL}OPT_i$ with high probability. Our

definitions for β, γ guarantee that $c_{OFL}OPT_i \leq \gamma L_i$. In addition, our definition for γ guarantees that $(\gamma - 1)k(1 + \log n) \geq \beta k_{OFL} k(1 + \log n)$. Thus, if online facility location "succeeds," the critical phase will allow the online facility location algorithm to run to completion. □

Corollary 2.1. *With high probability (the same as that for online facility location), Algorithm 2.1 completes the final phase with a solution of cost at most* $\frac{\alpha\beta\gamma}{\beta-\alpha} \cdot OPT$. *Applying the values for the constants gives an approximation factor of* $4\alpha^4 c_{OFL}^2 + 4\alpha^3 c_{OFL}$ *provided* $4\alpha^3 c_{OFL}^2 + 2\alpha^2 c_{OFL} \geq \beta k_{OFL} + 1$.

Proof. Consider a point $x \in X$. As in the proof of Lemma 2.1, let $y_j, y_{j+1}, \ldots, y_{i-1}, y_i$ be x's respective representatives in phases j up through i. The service cost due to x is $\delta(x, y_i)$. By the α-approximate triangle inequality, we can bound this by

$$\delta(x, y_i) \leq \alpha^{i-j}\delta(x, y_j) + \sum_{\ell=1}^{i-j} \alpha^\ell \delta(y_{i-\ell}, y_{i-\ell+1}).$$

Summing over all points x, and noting that our service cost in phase ℓ is bounded by γL_ℓ, combined with the knowledge that with high probability, we terminate at a phase where $L_i \leq OPT$, gives a total service cost of at most:

$$\alpha\gamma L_i + \alpha^2\gamma L_{i-1} + \alpha^3\gamma L_{i-2} + \cdots + \alpha^i \gamma L_1 \leq \alpha\gamma L_i \sum_{\ell=0}^{i-1} \left(\frac{\alpha}{\beta}\right)^\ell \leq \frac{\alpha\beta\gamma}{\beta - \alpha} \cdot OPT.$$

□

However, this solution uses much more than k facilities. To prune down to exactly k facilities, we can use any non-streaming $O(1)$-approximation to cluster our final (weighted) facilities (line 22 of the algorithm). If this non-streaming clustering algorithm has an approximation ratio of c_{KS}, our overall approximation ratio increases to $(\alpha + 4\alpha^5 c_{OFL}^2 + 4\alpha^4 c_{OFL})c_{KS}$.

Theorem 2.2. *With high probability, our algorithm achieves a constant approximation to k-service clustering if the α-approximate triangle inequality holds for a fixed constant α. This uses exactly k facilities and stores $O(k \log n)$ points in memory.*

Pruning the Runtime

As presented, Algorithm 2.1 can see as many as $O(\log_\beta OPT)$ phases in expectation, which gives the runtime an undesirable dependence on OPT. We now show how to modify Algorithm 2.1 so that it has at most $O(n/(k \log n))$ phases and running time at most $O(nk \log n)$.

Theorem 2.3. *For any fixed α, Algorithm 2.1 can be modified to run in $O(nk \log n)$ time.*

Proof. Consider any phase. The phase starts by reading the weighted facilities from the previous phase and paying a cost of at most $f = \frac{L_i}{k(1+\log n)}$ for each, at the end of which the cost is at most $(\gamma - 1)L_i$. Each additional point gives us a service cost of at most $\frac{L_i}{k(1+\log n)}$, so the phase must read at least $k(1+\log n)$ additional unread points before it can terminate due to the cost exceeding γL_i.

Now suppose that the phase ends due to having too many facilities without reading at least $k(1 + \log n)$ additional points. Since each new point can create at most one facility, the previous phase must have had at least $(\gamma - 2)k(1 + \log n)$ facilities already. Consider an optimal k-service clustering over the set X' of all the weighted points during this phase. Let OPT' denote the total service cost of this solution and OPT'_r denote the optimum total service cost if we are instead restricted to only selecting points from X'. Note that by the α-approximate triangle inequality, we have $OPT'_r \leq 2\alpha OPT'$. Thus, by Lemma 2.1, we have $OPT'_r \leq 2\alpha \left(\alpha + \gamma \frac{\alpha^2}{\beta - \alpha}\right) OPT$.

Since OPT'_r is only allowed k facilities, it must pay non-zero service cost for at least $(\gamma - 3)(1 + \log n)k$ weighted points. Define the nearest neighbor function $\pi : X' \to X'$ where for each point $x \in X'$, $\pi(x)$ denotes the closest other point (in terms of service costs) in X'. Then note that $\Delta_x = \text{weight}(x) \cdot \delta(x, \pi(x))$ gives a lower bound on the service cost for x if it is not chosen as a facility. Thus, the sum η of all but the k highest Δ_x gives a lower bound on OPT'_r. It follows that $\frac{\eta}{2\alpha\left(\alpha + \gamma \frac{\alpha^2}{\beta - \alpha}\right)} \leq OPT$.

We will set L_i to be the maximum of this new lower bound and βL_{i-1}, and eliminate $k(1 + \log n)$ facilities, increasing the service cost by at most L_i. This guarantees that the next time the number of facilities grows too large we will have read $\Omega(k \log n)$ new points,

bounding the number of phases by $O\left(\frac{n}{k \log n}\right)$.

Let $\hat{X} \subseteq X'$ denote the set of points with $\Delta_x \leq \eta[2\alpha(\alpha + \gamma\frac{\alpha^2}{\beta-\alpha})(1+\log n)k]^{-1}$. Suppose that $|\hat{X}| < 2k(1+\log n)$. The number of points which contribute to η is at least $(\gamma-3)k(1+\log n)$, and at least $(\gamma-5)k(1+\log n)$ of these points would not belong to $\hat{X}$. Thus, the sum of Δ_x for such points is bounded by $\frac{\eta(\gamma-5)}{2\alpha\left(\alpha+\gamma\frac{\alpha^2}{\beta-\alpha}\right)} \leq \eta$. Canceling and solving this equation for γ yields $\gamma\left(1 - \frac{2\alpha^3}{\beta-\alpha}\right) \leq 2\alpha^2 + 5$.

Plugging in the values for β and γ along with $c_{OFL} \geq 2\alpha$ and $\alpha \geq 1$ yields a contradiction. Thus, it follows that $|\hat{X}| \geq 2k(1+\log n)$. We assume $|\hat{X}|$ is even for simplicity of analysis. Some points in $\hat{X}$ have their nearest neighbor in $X' - \hat{X}$. For the remaining points in $\hat{X}$, consider the nearest neighbor graph induced by these points. This graph has no cycles of length 3 or longer. Thus, the graph is bipartite and we can find a vertex cover C of size at most $|\hat{X}|/2$. We can add additional points to C from $\hat{X}$ to get precisely $|\hat{X}|/2$ points. Note that all points in $\hat{X} - C$ have a nearest neighbor not in $\hat{X} - C$. Thus, we can remove $\hat{X} - C$ as facilities and increase our service cost by at most

$$\frac{\eta(1+\log n)k}{2\alpha(\alpha+\gamma\frac{\alpha^2}{\beta-\gamma})(1+\log n)k} = \frac{\eta}{2\alpha(\alpha+\gamma\frac{\alpha^2}{\beta-\alpha})} \leq L_i.$$

We can compute η in time $O(k^2 \log^2 n)$, $\hat{X}$ in time $O(k \log n)$, and the vertex cover in time linear in $|\hat{X}|$ (using a greedy algorithm; note that it need not be a minimum vertex cover). Additionally, all these can be computed using space to store $O(k \log n)$ points. The running time for reading a new (unweighted) point is $O(k \log n)$, so the total running time is the time to read unweighted points plus the overhead induced by starting new phases (and reading weighted points). Each of these is at most $O(nk \log n)$. □

2.3 Maintaining Samples During Streaming k-means

We now turn our attention to the important case of Euclidean space, where the distance function d is the Euclidean distance.

Definition 2.4. *Let S be a set of cardinality n. Let $\Re^q = \{p \in S^q : \forall i, j \in [q] \quad p_i \neq p_j\}$.*

A random element E is a q-random sample without replacement from S if E has uniform distribution over $\Re^q$.

First, we establish that we can maintain a uniformly random sample for each facility's service points. This is identical to the problem of sampling uniformly at random from a stream, as all the points assigned to the facility can be treated as a single stream. Methods for streaming sampling are well-known (e.g., see [Vit85]). In particular, the following lemma is well-known.

Lemma 2.3. *There exists an algorithm that, given a stream X and q, maintains a q-random sample without replacement from X and uses $O(q)$ memory.*

For our purposes, it will be useful to use the following alternative definition of a q-random sample without replacement. We prove that this definition is equivalent to the original definition.

Lemma 2.4. *Let S be a set and let $X_1, \ldots, X_q$ be the following random variables. X_1 is distributed uniformly on S, and, for $1 < i \leq q$, X_i is distributed uniformly over the set $S \setminus \{X_1, \ldots, X_{i-1}\}$. Then an ordered q-tuple $X = \langle X_1, \ldots, X_q \rangle$ is a q-random sample without replacement from S.*

Proof. Consider a fixed q-tuple $p = \langle p_1, \ldots, p_q \rangle \in \Re^q$. We shall show that $P(X = p) = \frac{1}{|\Re^q|} = \frac{1}{q!\binom{n}{q}}$. We have

$$P(X = p) = P(X_1 = p_1) \times P(X_2 = p_2 | X_1 = p_1) \times \cdots \times P(X_q = p_q | \forall_{j<q} X_j = p_j).$$

By the definition of X_1, $P(X_1 = p_1) = \frac{1}{n}$. By the definition of X_i, and since $p_i \notin \{p_1, \ldots, p_{i-1}\}$, we have for all fixed $X_1, \ldots X_{i-1}$: $P(X_i = p_i | \forall_{j<i} X_j = p_j) = \frac{1}{n-i+1}$. Thus, $P(X = p) = \prod_{i=1}^{q} \frac{1}{n-i+1} = \frac{1}{q!\binom{n}{q}}$. □

Recall that $com(A)$ denotes the center of mass for a point set A. Note the following relation in any cluster of the difference in cost for replacing the center of mass with an arbitrary other point:

Fact 2.1. $\sum_{x \in X} d^2(x, y) = |X| d^2(com(X), y) + \sum_x d^2(x, com(X))$.

Now, we show how to maintain a uniformly at random sample from the union of two clusters, for the two times in our algorithm in which this occurs.

Lemma 2.5. *Let S_1, S_2 be two disjoint sets. Given independent q-random samples without replacement from S_1 and S_2 and $|S_1|, |S_2|$, it is possible to obtain a q-random sample without replacement from $S_1 \cup S_2$. The algorithm requires $O(q)$ time and $O(q)$ additional memory.*

Proof. The algorithm is similar in spirit to the merging process in Merge Sort. Let $X = \{X_1, \ldots, X_q\}$ and $Y = \{Y_1, \ldots, Y_q\}$ be q-random samples without replacement from S_1 and S_2. Take the first element of X with probability $\frac{|S_1|}{|S_1|+|S_2|}$; otherwise, take the element from Y. Repeat this until q are chosen (reducing the set sizes $|S_1|, |S_2|$ appropriately depending on whether previous samples came from X or Y), and call the set formed by those taken Z.

The performance bounds follow from the description of the algorithm. To show correctness, it is enough to show that Z_i is distributed uniformly over $(S_1 \cup S_2) \setminus \{Z_1, \ldots, Z_{i-1}\}$ and then use Lemma 2.4. For $i = 1$, by Lemma 2.4, X_1 and Y_1 are distributed uniformly over S_1 and S_2 respectively. Consider an arbitrary, fixed $w \in S_1 \cup S_2$. Without loss of generality, assume $w \in S_1$. Since the algorithm's randomness is independent of X, we get: $P(Z_1 = w) = \frac{|S_1|}{|S_1|+|S_2|} P(X_1 = w) = \frac{1}{|S_1|+|S_2|}$. Thus, Z_1 is distributed uniformly over $S_1 \cup S_2$.

Let $i > 1$, and consider any fixed $Z_1, \ldots, Z_{i-1}$. By Lemma 2.4, X_{i_1} is distributed uniformly over $S_1 \setminus \{X_1, \ldots, X_{i_1-1}\}$ and Y_{i_2} is distributed uniformly over $S_2 \setminus \{Y_1, \ldots, Y_{i_2-1}\}$. Consider any fixed $w \in (S_1 \cup S_2) \setminus \{Z_1, \ldots, Z_{i-1}\}$. Without loss of generality, assume that $w \in S_1$. Since $\{Z_1, \ldots, Z_{i-1}\} = \{Y_1, \ldots, Y_{i_2-1}\} \cup \{X_1, \ldots, X_{i_1-1}\}$, it follows that $w \in S_1 \setminus \{X_1, \ldots, X_{i_1-1}\}$. Thus, we have

$$P(Z_i = w) = \frac{n_1}{n_1 + n_2} P(X_{i_1} = w) = \frac{n_1}{n_1 + n_2} \left(\frac{1}{n_1}\right) = \frac{1}{n_1 + n_2},$$

where $n_1 = |S_1 \setminus \{X_1, \ldots, X_{i_1-1}\}|$ and $n_2 = |S_2 \setminus \{Y_1, \ldots, Y_{i_2-1}\}|$. The values of n_1 and n_2 imply that the probability is uniform over $(S_1 \cup S_2) \setminus \{Z_1, \ldots, Z_{i-1}\}$. Indeed,

$$n_1 + n_2 = |S_1| - i_1 + 1 + |S_2| - i_2 + 1 = |S_1 \cup S_2| - i + 1.$$

Thus, we have shown that Z_1 is a uniform sample from $S_1 \cup S_2$ and for any $i > 1$, Z_i is a uniform sample from $(S_1 \cup S_2) \setminus \{Z_1, \ldots, Z_{i-1}\}$. The correctness follows from Lemma 2.4. □

We now show that samples of optimal clusters are sufficient to find approximate centers of mass, by first proving the following useful lemma. Recall that $C(a, A)$ is the one-means cost of using point a as a mean for set A.

Lemma 2.6. *Let X be a set of points and $Y \subseteq X$. Then $C(com(Y), X) \leq \frac{|X|}{|Y|} C(com(X), X)$.*

Proof. Let $d_1 = d(com(X), com(Y))$ and $d_2 = d(com(X - Y), com(X))$. By the triangle inequality, $d(com(Y), com(X - Y)) \leq d_1 + d_2$. Applying Fact 2.1 repeatedly gives:

$$C(com(Y), X) = C(com(Y), Y) + C(com(X - Y), X - Y) + |X - Y|(\delta_1 + \delta_2)^2,$$

$$C(com(X), X) = C(com(Y), Y) + |Y|d_1^2 + C(com(X - Y), X - Y) + |X - Y|d_2^2,$$

$$\frac{C(com(Y), X)}{C(com(X), X)} \leq \frac{|X - Y|(d_1 + d_2)^2}{|Y|d_1^2 + |X - Y|d_2^2}.$$

To maximize the ratio, we take the derivative with respect to d_2 and set the resulting expression to zero, obtaining $|Y|d_1^2 + |X - Y|d_2^2 = |X - Y|(d_1 + d_2)d_2$; solving this yields $d_2 = \frac{|Y|}{|X-Y|} d_1$. Note that the other boundary conditions for the expression are at $d_2 = 0$ and d_2 tending to infinity, both of which easily satisfy the required inequality. Substitution gives:

$$\frac{C(com(Y), X)}{C(com(X), X)} \leq \frac{|X|^2/|X - Y|}{|Y| + (|Y|^2/|X - Y|)} \leq \frac{|X|}{|Y|}.$$

□

Theorem 2.4. *Suppose we have a set X of points and are given some arbitrarily selected $Y \subseteq X$. If Z is a set of q points selected uniformly at random from Y (without replacement), then the center of mass for Z is a $\left(1 + \frac{1}{q} - \frac{q-1}{q(|Y|-1)}\right)\left(\frac{|X|}{|Y|}\right)$-approximation to the optimum one-mean solution for X in expectation.*

Proof. By linearity of expectation, it is sufficient to show that the above holds in one-dimensional space. Applying Fact 2.1 gives us $C(com(Z), X) \leq |X|d^2(com(Z), com(X)) +$

$C(com(X), X)$. We will need to bound the expected value of $d^2(com(Z), com(X))$. Since we can assume one-dimensional space, we use the definition of center of mass to get:

$$E[d^2(com(Z), com(X))] = E\left[\left(\frac{\sum_{z\in Z} z}{|Z|} - \frac{\sum_{x\in X} x}{|X|}\right)^2\right].$$

We can compute the square and use linearity of expectation, noticing that since the points of Z are uniformly chosen from Y, we have $E[(1/|Z|)\Sigma_{z\in Z} z] = (1/|Y|)\sum_{y\in Y} y$. We need to bound $E[(\sum_{z\in Z} z)^2]$. For each $y_1 \in Y$, there is a probability $|Z|/|Y|$ that this point appeared also in the randomly selected set Z. If so, we will obtain an expected contribution to the sum of squares which looks like $y_1^2 + y_1 E[\Sigma_{z\in Z-\{y_1\}} z | y_1 \in Z]$, where the latter term is just $\frac{|Z|-1}{|Y|-1}\sum_{y_2\in Y, y_2\neq y_1} y_2$. Summing these gives us:

$$E\left[\left(\sum_{z\in Z} z\right)^2\right] = \frac{|Z|}{|Y|}\sum_{y\in Y} y^2 + \frac{|Z|}{|Y|}\sum_{y_1\in Y}\sum_{y_2\in Y, y_2\neq y_1} \frac{|Z|-1}{|Y|-1} y_1 y_2.$$

We can rewrite this, adding and subtracting terms representing the sum of squared elements of Y, as:

$$E\left[\left(\sum_{z\in Z} z\right)^2\right] = \frac{|Z|}{|Y|}\left[\left(1 - \frac{|Z|-1}{|Y|-1}\right)\sum_{y\in Y} y^2 + \frac{|Z|-1}{|Y|-1}\left(\sum_{y\in Y} y\right)^2\right].$$

We have $C(com(Y), Y) = \Sigma_{y\in Y} y^2 - \frac{1}{|Y|}\left(\Sigma_{y\in Y} y\right)^2$, and we can substitute this to get:

$$E\left[\left(\sum_{z\in Z} z\right)^2\right] = \frac{|Z|}{|Y|}\left[\left(1 - \frac{|Z|-1}{|Y|-1}\right) C(com(Y), Y) + \frac{|Z|}{|Y|}\left(\sum_{y\in Y} y\right)^2\right].$$

We observe that $d^2(com(Y), X)$ can be formulated similarly to $d^2(com(Z), X)$, and when we combine the various terms we obtain the following bound:

$$E[d^2(com(Z), com(X))] = \frac{1}{|Y||Z|}\left[\left(1 - \frac{|Z|-1}{|Y|-1}\right) C(com(Y), Y)\right] + d^2(com(Y), com(X)).$$

To compute the cost of $C(com(Z), X)$, we multiply by $|X|$ and add $C(com(X), X)$. We also observe that since $Y \subseteq X$, we will have $C(com(Y), Y) \leq C(com(X), X)$, and we apply Lemma 2.6 to reach:

$$E[C(com(Z), X)] \leq \frac{|X|}{|Y|}\left(1 + \frac{1}{|Z|} - \frac{|Z|-1}{|Z|(|Y|-1)}\right) C(com(X), X).$$

□

There is an analog to this theorem that applies with high probability, which we will now show in the following series of lemmas.

Lemma 2.7. *If $X = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}$ and $Y = Z - S$, where Z is a random sample from X and $S = com(X) = \frac{1}{n}\sum_i x_i$, then $E[Y] = 0$ and $Var[Y] = \frac{1}{n}OPT(1)$, where $OPT(1) = C(com(X), X)$ (i.e., $OPT(1)$ is the optimal 1-means solution for X).*

Proof. The expected value of Y is $E[Y] = E[Z - S] = E[Z] - S = S - S = 0$. The variance of Y is given by

$$Var[Y] = E[Y^2] - E[Y]^2 = E[Y^2] = \frac{1}{n}\sum_{i=1}^{n}(x_i - S)^2 = \frac{1}{n}OPT(1).$$

□

Now consider taking the mean of q random samples from X, with replacement:

Lemma 2.8. *If $X = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}$ and $Y = \frac{1}{q}(Z_1 + \cdots + Z_q) - S$, where each Z_i is a random sample from X (with replacement) and $S = com(X)$, then $E[Y] = 0$ and $Var[Y] = \frac{1}{qn}OPT(1)$, where $OPT(1)$ is the optimal 1-means solution for X.*

Proof. The expected value of Y is $E[Y] = E\left[\frac{1}{q}\sum_{i=1}^{q} Z_i - S\right] = \frac{1}{q}\sum_{i=1}^{q} E[Z_i] - S = S - S = 0$ (here we used Lemma 2.7). Notice that we can rewrite Y as $Y = \frac{1}{q}\sum_{i=1}^{q} Y_i$, where the $Y_i = Z_i - S$ are independent random variables. Hence, the variance of Y (by Lemma 2.7) is given by

$$Var[Y] = \frac{1}{q^2}\sum_{i=1}^{q} Var[Y_i] = \frac{1}{q^2}\frac{q}{n}OPT(1).$$

□

Using the same notation, we now have the following constant probability bound:

Lemma 2.9. *If $B = \frac{1}{q}(Z_1 + \cdots + Z_q)$, where $q = \frac{100}{\epsilon}$, then $P\left[|B - S| \geq \sqrt{\frac{\epsilon OPT(1)}{n}}\right] \leq \frac{1}{100}$.*

Proof. By Chebyshev's inequality, we have:

$$P\left[|B - S| \geq \sqrt{\frac{\epsilon OPT(1)}{n}}\right] = Pr\left[|Y| \geq \sqrt{\frac{\epsilon OPT(1)}{n}}\right] \leq \frac{nVar[Y]}{\epsilon OPT(1)} = \frac{1}{q\epsilon} = \frac{1}{100},$$

where Y is the same as in Lemma 2.8. □

We now take the median of means:

Lemma 2.10. *Let $B_1, \ldots, B_t$ be independent random variables, each from Lemma 2.9, where $t = O(\log nd)$. Let $B = median(B_1, \ldots, B_t)$. Then $P\left[|B - S| \geq \sqrt{\frac{\epsilon OPT(1)}{n}}\right] \leq \frac{1}{nd}$.*

Proof. This follows from a standard application of Chernoff bounds. □

We now concentrate on points from $\mathbb{R}^d$ and give some notation and definitions. Let $X = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^d$ and let x_{ij} denote the j^{th} coordinate of x_i. Let $S = \frac{1}{n}\sum_i x_i$, and define $S_j = \frac{1}{n}\sum_i x_{ij}$ (so that $S = (S_1, \ldots, S_d)$). Define $OPT_j(1) = \sum_{i=1}^n (x_{ij} - S_j)^2$. Observe that $OPT(1) = \sum_{i=1}^n \sum_{j=1}^d (x_{ij} - S_j)^2 = \sum_{j=1}^d OPT_j(1)$, where $OPT(1)$ denotes the optimal 1-means solution for X. The analog of Theorem 2.4 for the high probability setting essentially follows from the lemma below.

Lemma 2.11. *Let $U \in \mathbb{R}^d$ be a vector with coordinates $U = (U_1, \ldots, U_d)$, where the U_i are independent and each has the same distribution as B from Lemma 2.10 with respect to the set of j^{th} coordinates $\{x_{1j}, \ldots, x_{nj}\}$. Let $A = \sum_{i=1}^n d^2(U, x_i)$. Then $A \leq (1+\epsilon)OPT(1)$ with probability at least $1 - \frac{1}{n}$.*

Proof. By Lemma 2.10, we know that $P\left[|U_j - S_j| \geq \sqrt{\frac{\epsilon OPT_j(1)}{n}}\right] \leq \frac{1}{dn}$. By applying the union bound, we know that with probability at least $1 - \frac{1}{n}$, the inequality $|U_j - S_j|^2 \leq \frac{\epsilon OPT_j(1)}{n}$ holds over all dimensions (i.e., for all $1 \leq j \leq d$). By Fact 2.1, we know that $A = \sum_{i=1}^n d^2(U, x_i) = nd^2(U, S) + OPT(1)$. We have the following upper bound on $d^2(U, S)$:

$$d^2(U, S) = \sum_{j=1}^d (U_j - S_j)^2 \leq \sum_{j=1}^d \frac{\epsilon OPT_j(1)}{n} = \frac{\epsilon OPT(1)}{n}.$$

The lemma follows. □

We can now apply similar methods used in our result for the expectation guarantee and achieve the same result for sufficiently large Y which are subsets of the set X.

2.4 From Constant to Converging to One

Given a c-approximation to k-means (where c is constant) for a σ-separable point set, we now show that we can perform a single recentering step, called a *ball k-means step*, and obtain an approximation ratio of $1 + \sigma^2$ which converges to one as σ approaches 0. While a full ball k-means step requires another pass through the point stream, we will establish that it is sufficient to use a smaller random sample of points.

In what follows, for each optimum mean i, we let C_i^* be the points OPT assigns to i, $\nu(i)$ be the closest mean to i in our c-approximate solution, and $B_{\nu(i)}$ be the ball around $\nu(i)$ (the formal definition of which is given later).

Theorem 2.5. *Suppose we have a c-approximation to k-means, and a σ-separable data set where $\frac{1}{\sigma^2} > 2\gamma(c+1)+1$ for $\gamma \geq \frac{169}{4}$. Then we can apply a ball k-means step, by associating with each of our approximate means $\nu(i)$ the set of points $B_{\nu(i)}$, then computing a new mean $\nu(i)' = com\left(B_{\nu(i)}\right)$. This yields an approximation to k-means which approaches one as σ approaches zero.*

We prove the above theorem with the following sequence of lemmas. First, we show that i and $\nu(i)$ are in fact very close together.

Lemma 2.12. *For any i, we have $d(\nu(i), i) \leq \sqrt{\frac{2(c+1)OPT}{|C_i^*|}}$.*

Proof. Consider the points $S = \{x \in C_i^* \mid d(i,x) \leq d(i,\nu(i))\}$. For each $x \in S$, we can bound $d^2(i,\nu(i))$ using the 2-approximate triangle inequality. Summing over all $x \in S$ gives

$$d^2(i,\nu(i)) \leq \frac{2}{|S|}\left(\sum_{x\in S} d^2(i,x) + \sum_{x\in S} d^2(x,\nu(i))\right).$$

The first term in the right hand side is bounded by $OPT - \sum_{x\notin S} d^2(i,x)$ whereas the second term is at most $c \cdot OPT$. Substituting and using the fact that $d^2(i,\nu(i)) < d^2(i,x)$ for all $x \notin S$ proves the claim. □

We now show that a σ-separable point set implies that the optimum means are far apart.

Lemma 2.13. *In a σ-separable point set, any two distinct optimum means i, j must satisfy* $d(i,j) \geq \sqrt{\frac{OPT}{\sigma^2|C_i^*|} - \frac{OPT}{|C_i^*|}}$.

Proof. For any two means i, j, we can always eliminate mean i and reassign any points in C_i^*. This produces a solution using $k-1$ means and, since i is the center of mass of C_i^*, increases the cost by at most $|C_i^*|d^2(i,j)$. By σ-separability, the total cost of this solution must be at least $\frac{OPT}{\sigma^2}$, which gives the above bound on $d(i,j)$. □

Lemma 2.12 and Lemma 2.13 show that for sufficiently small σ, each optimum mean i has unique $\nu(i)$ which is much closer to i than to any other optimum mean. In particular, if $\frac{1}{\sigma^2} > 2\gamma(c+1)+1$ for some γ to be specified later, it follows that for any optimum mean i, the next closest optimum mean is at least distance $\sqrt{\gamma}\sqrt{\frac{2(c+1)OPT}{|C_i^*|}}$ away and the closest mean $\nu(i)$ is at most distance $\sqrt{\frac{2(c+1)OPT}{|C_i^*|}}$ away.

Lemma 2.14. *Define $B_{\nu(i)}$ to consist of all points x such that $2d(x,\nu(i)) \leq d(x,\nu(j))$ for any $j \neq i$. Then $B_{\nu(i)} \subseteq C_i^*$ and $|B_{\nu(i)}| \geq \left(1 - \frac{9}{2(c+1)(\sqrt{\gamma}-5)^2}\right)|C_i^*|$ when $\gamma \geq \frac{169}{4}$.*

Proof. We first show $B_{\nu(i)} \subseteq C_i^*$. Fix i, j and suppose $x \in B_{\nu(i)}$. By definition, $2d(x,\nu(i)) \leq d(x,\nu(j))$. Applying the triangle inequality gives $2d(x,i) \leq 2d(i,\nu(i)) + d(j,\nu(j)) + d(x,j)$. If $d(x,j) \leq d(x,i)$, then it will follow that $d(x,i) \leq 2d(i,\nu(i)) + d(j,\nu(j))$. Each of these is bounded according to Lemma 2.12, so we can conclude that

$$d(i,j) \leq 2d(x,i) \leq 4\sqrt{\frac{2(c+1)OPT}{|C_i^*|}} + 2\sqrt{\frac{2(c+1)OPT}{|C_j^*|}} \leq 6\sqrt{\frac{2(c+1)OPT}{\min\{|C_i^*|,|C_j^*|\}}}.$$

However, Lemma 2.13 implies that $d(i,j) \geq \sqrt{\frac{2\gamma(c+1)OPT}{\min\{|C_i^*|,|C_j^*|\}}}$. If $\gamma > 36$, this gives a contradiction. We conclude that $d(x,i) < d(x,j)$ and that therefore $x \in C_i^*$ and $B_{\nu(i)} \subseteq C_i^*$.

Notice that the service cost of points in C_i^* is at most OPT. By Markov's inequality, for any $\mu \geq \frac{1}{2(c+1)}$, there are at least $\left(1 - \frac{1}{2(c+1)\mu}\right)|C_i^*|$ points of C_i^* within distance $\sqrt{\frac{2\mu(c+1)OPT}{|C_i^*|}}$ of i. For any such point x and optimum mean $j \neq i$, the triangle inequality along with

Lemma 2.12 and Lemma 2.13 give

$$\begin{aligned} d(x,\nu(j)) &\geq d(i,j) - d(i,\nu(i)) - d(j,\nu(j)) - d(x,\nu(i)) \\ &\geq (\sqrt{\gamma}-2)\sqrt{\frac{2(c+1)OPT}{\min\{|C_i^*|,|C_j^*|\}}} - d(x,\nu(i)). \end{aligned}$$

Setting $\mu = \frac{1}{9}\left(\sqrt{\gamma}-5\right)^2$ ensures that $d(x,\nu(j)) \geq 2d(x,\nu(i))$ and that $x \in B_{\nu(i)}$. However, this imposes the additional constraint that $\gamma \geq \frac{169}{4}$ in order to ensure that $\mu \geq \frac{1}{4} \geq \frac{1}{2(c+1)}$. $\square$

By Lemma 2.14, we have $B_{\nu(i)} \subseteq C_i^*$ and $|B_{\nu(i)}| \geq |C_i^*|\left(1 - \frac{9}{2(c+1)(\sqrt{\gamma}-5)^2}\right)$. It follows from Lemma 2.6 that we obtain an approximation to k-means of ratio at worst $1 + \frac{9}{2(c+1)(\sqrt{\gamma}-5)^2-9}$. As σ becomes smaller, γ becomes larger, and thus the approximation ratio converges to one.

We finally seek to prove the main theorem in this chapter.

Theorem 2.6. *Suppose we have a c-approximation to k-means for a σ-separable data set where $\frac{1}{\sigma^2} > 2\gamma(c+1)+1$ for $\gamma \geq \frac{169}{4}$. Additionally, suppose that instead of being given the entire point set, we are only given small random samples $Z_{\nu(i)}$ of size $\frac{1}{\epsilon}$ of each cluster $C_{\nu(i)}$ in this approximate solution. Then we can apply a ball k-means step computing a new mean $\nu(i)'$ by computing the center of mass of $(B_{\nu(i)} \cap Z_{\nu(i)})$. This yields a $\Theta((1+\epsilon)(1+\sigma^2 c))$-approximation to k-means which approaches one as ϵ and σ approach zero.*

To prove this theorem, first note that we have shown how to perform a ball k-means step on our approximate solution to achieve an approximation ratio which approaches 1 as σ approaches 0. While performing a full ball k-means step requires another pass through the point set, we can avoid this second pass if we are given a random sample of $\frac{1}{\epsilon}$ points from each of the balls $B_{\nu(i)}$ and perform the ball k-means step on just these sample points. By Theorem 2.4, this gives us an approximation ratio of $\left(1+\epsilon - \frac{\frac{1}{\epsilon}-1}{\frac{1}{\epsilon}(|B_{\nu(i)}\cap C_{\nu(i)}|-1)}\right)\frac{|C_i^*|}{|B_{\nu(i)}\cap C_{\nu(i)}|}$ within each cluster.

However, our algorithm only returns a random sample of q points in each of our clusters $C_{\nu(i)}$. Thus, we need to show that, in expectation, a constant fraction of these points are in $B_{\nu(i)}$. Indeed, we now show that this fraction approaches 1 and that $|B_{\nu(i)} \cap C_{\nu(i)}|$ approaches

$|B_{\nu(i)}|$ as σ tends toward 0. Thus, our overall approximation ratio still converges to 1 as ϵ and σ approach 0.

We first give an upper bound on the number of points in $B_{\nu(i)}$ that are not in $C_{\nu(i)}$. These points are never candidates in the randomly selected points from $C_{\nu(i)}$ and so may hurt our approximation if there are too many. Fortunately, we can prove that there is only a small number of them.

Lemma 2.15. $|B_{\nu(i)} - C_{\nu(i)}| \leq \frac{c}{8(\sqrt{\gamma}-2)^2(c+1)}|C_i^*|$.

Proof. Consider $x \in B_{\nu(i)} - C_{\nu(i)}$ and let $\nu(j)$ be the mean such that $x \in C_{\nu(j)}$. The triangle inequality and the fact that $x \notin B_{\nu(j)}$ give

$$d(x,\nu(j)) \geq d(i,j) - d(i,\nu(i)) - d(j,\nu(j)) - \frac{1}{2}d(x,\nu(j)).$$

Solving for $d(x,\nu(j))$, applying Lemma 2.12 and Lemma 2.13, and squaring give

$$d(x,\nu(j))^2 \geq \left(2\left(\sqrt{\gamma}-2\right)\sqrt{\frac{2(c+1)OPT}{\min\{n_i,n_j\}}}\right)^2 \geq 4\left(\sqrt{\gamma}-2\right)^2\left(\frac{2(c+1)OPT}{|C_i^*|}\right).$$

If we sum over all such x, then we should get no more than $c \cdot OPT$ since we have a c-approximation. This bounds $|B_{\nu(i)} - C_{\nu(i)}|$ as desired. □

We can now use Lemma 2.14 and Lemma 2.15 to give a lower bound on the fraction of $B_{\nu(i)}$ contained in $C_{\nu(i)}$. Accordingly, this fraction approaches 1 as σ diminishes, showing that roughly the entirety of $B_{\nu(i)}$ is in our sample space. We can also bound the cardinality of $B_{\nu(i)} \cap C_{\nu(i)}$ in terms of C_i^* which will become useful later.

Corollary 2.2. $|B_{\nu(i)} \cap C_{\nu(i)}| \geq \left(1 - \frac{c(\sqrt{\gamma}-5)^2}{4(\sqrt{\gamma}-2)^2(2(c+1)(\sqrt{\gamma}-5)^2-9)}\right)|B_{\nu(i)}|$.

Corollary 2.3. $|B_{\nu(i)} \cap C_{\nu(i)}| \geq \left(1 - \frac{9}{2(c+1)(\sqrt{\gamma}-5)^2} - \frac{c}{8(\sqrt{\gamma}-2)^2(c+1)}\right)|C_i^*|$.

Though roughly all of $B_{\nu(i)}$ lies in $C_{\nu(i)}$, there are other points in $C_{\nu(i)}$. If there are too many of these points, then we would expect that a very small fraction of the q' sampled points are actually in $B_{\nu(i)}$, driving our approximation ratio upwards. Thus, we must show that the number of these points tends towards 0.

Lemma 2.16. $|C_{\nu(i)} \cap B_{\nu(i)}| \geq \frac{\left(1-\frac{9}{2(c+1)(\sqrt{\gamma}-5)^2}-\frac{c}{8(\sqrt{\gamma}-2)^2(c+1)}\right)}{\left(1+\frac{c}{2(c+1)(\sqrt{\gamma}-2)^2}\right)}|C_{\nu(i)}|$.

Proof. Consider an $x \in C_{\nu(i)} - B_{\nu(i)}$. Since $j \notin B_{\nu(i)}$, we must have $2d(x, \nu(i)) \geq d(x, \nu(j))$ for some j. Proceeding in a fashion similar to the proof of Lemma 2.15 shows

$$|C_{\nu(i)} - B_{\nu(i)}| \leq \frac{c}{2(c+1)(\sqrt{\gamma}-2)^2}|C_i^*|.$$

Thus, we can bound the number of elements in our cluster by

$$|C_{\nu(i)}| = |C_{\nu(i)} \cap B_{\nu(i)}| + |C_{\nu(i)} - B_{\nu(i)}| \leq \left(1 + \frac{c}{2(c+1)(\sqrt{\gamma}-2)^2}\right)|C_i^*|.$$

Combining with Corollary 2.3 gives the desired result. □

Together, these results prove Theorem 2.6.

2.5 Concluding Remarks

This chapter studies the k-means clustering problem in the streaming model. Given that it is difficult to design an efficient approximation algorithm with near-optimal guarantees for this problem (under standard complexity assumptions), it is natural to place some restriction on the input that is practical. We show how the assumption of separable data allows us to break the difficulty barrier and obtain a near-optimal approximation guarantee for k-means in Euclidean space. Our assumption on the data is reasonable and not too restrictive, and there are still many practical applications that can be impacted by our results.

In the end, for σ-separable data, we design a streaming algorithm and show that the separability assumption is amenable to analysis, yielding a $(1+O(\epsilon)+O(\sigma^2))$-approximation. Our algorithm is efficient and its guarantees hold with high probability, using $O\left(\frac{k}{\epsilon}\log^2 n\right)$ space and running in time $O(nk \log n)$. Along the way, we give an $O(1)$-approximation algorithm for a general semi-metric clustering class of problems (even without σ-separability), and improve the online facility location algorithm's guarantees by showing they hold with high probability. Finally, we show how to maintain samples and turn our $O(1)$-approximation into a near-optimal solution under the separability assumption for Euclidean space.

CHAPTER 3

Online Multidimensional Load Balancing

3.1 Introduction

Motivation and Applications

Billions of dollars are spent every year to power computer systems, and any improvement in power efficiency could lead to significant savings [RSM09]. With the rise of huge data centers and server farms, energy costs and cooling costs have become a significant expense as demand for computing power, servers, and storage grows. Indeed, energy costs and cooling costs are likely to exceed the cost of acquiring new hardware and servers. Managers of data centers wish to optimize power consumption without sacrificing any performance to minimize energy costs and cooling costs due to heat dissipation [MCR05]. For these reasons, algorithms for energy efficient scheduling are very valuable, and even a small improvement could lead to significant savings and a positive impact on the environment.

As data centers and server farms grow in size, it becomes increasingly important to decide which machines should stay active and which should be shut down. In particular, there are opportunities for energy conservation and monetary savings due to cooling costs [MCR05]. Moreover, these larger data centers have huge fluctuations in work loads, ranging from very high peaks to very low valleys. Hence, when demand is low, it is possible to shut down some machines which would allow for significant savings [KLS10]. Certain jobs may need access to particular machines (due to data availability or device capability), and hence it makes sense to consider a subset of machines to activate for a given set of jobs. Once the appropriate machines have been activated, the pending jobs may then be scheduled. Note that the process of choosing which machines to activate is naturally an online problem, since

jobs arrive dynamically over time.

In [RSR08, RSM09, SMK08], methods were developed to measure power consumption at a high sampling rate. This allows us to measure energy requirements of recurring jobs at various speeds, and also to use machine learning techniques to estimate these requirements for new jobs. In [RSM09] measurements were made for the energy effects of resource contention between jobs. The results indicated that jobs which make heavy use of different system components can be parallelized in a power-efficient manner, whereas jobs which make heavy use of the same components do not parallelize well. Their results motivate our belief that optimizing the scheduling of jobs to minimize power consumption is a non-trivial problem in real systems. It is important to effectively distribute jobs among machines so that resource contention (and thus energy use) is minimized. Multidimensional load balancing has applications in cutting stock, resource allocation, and implementation of databases for share-nothing environments [DG92, GI95, GI96]. This problem also has applications in multidimensional resource scheduling for parallel query optimization in databases. Query execution typically involves multidimensionality, particularly among time-sharing system resources such as the CPU or disk [GI95]. This motivates representing jobs as having d dimensions where each dimension represents the load induced by the job on the component. Real jobs often involve more than two components (and real network speeds depend on internet congestion). It is important to model the load placed on various system components (processor, network, memory, etc.) and the key to obtaining good performance (both in terms of completion times and power) is to balance the loads appropriately.

Problem Definitions

We consider two problems in this domain. Our first problem considers online allocation of jobs to unrelated machines with arbitrary activation costs and d dimensions, which we call the Machine Activation problem:

Definition 3.1 (Machine Activation Problem)**.** *We are given a set of m unrelated machines each with d dimensions (i.e., components such as CPU, memory, network) and an activation*

cost c_i. Moreover, a set of n jobs j arrive online, each inducing a load of p_{ij}^k if assigned to machine i on dimension k. We must select a set A of machines to activate such that $\sum_{i \in A} c_i \leq B$ for a constraint budget B, and assign jobs to active machines such that the total p_{ij}^k for jobs assigned to machine i along dimension k is at most a load constraint Λ.

The main setting we study is when Λ and B are given to the algorithm. Our competitive guarantees hold if there is an offline integral solution with makespan Λ and budget B. Note that we also consider variants in which the load Λ or budget B may not necessarily be specified, in which case we seek to minimize the corresponding objective.

Our second problem considers online allocation of jobs to identical machines with multiple components and no activation costs.

Definition 3.2 (Vector Load Balancing Problem)**.** *We have m identical machines each with d components. Jobs $\vec{p}_j$ arrive online and are to be assigned to machines upon arrival. Here, the k^{th} coordinate p_j^k gives the load placed on component k by job j. Let ℓ_i^k denote the sum of p_j^k over all jobs j on machine i. The* load *ℓ_i of machine i is $\max_k \ell_i^k$. Our goal is to simultaneously minimize the makespan, $\max_i \ell_i$, and the energy, $\sum_i \ell_i$.*

Our Contributions and Techniques

For the Machine Activation problem, we design an online algorithm in Section 3.2 with competitive ratios $O(\log(md)\log(nm))$ on the load and $O(d\log^2(nm))$ on the energy budget for the case when the load Λ and the budget B are given. It extends the result of [KLS10] to an online setting and the work of [ABF13] to the multidimensional setting. Our main technique considers the linear relaxation of an integer program. We approximately solve the linear relaxation online as constraints arrive, such that everything except the budget and load constraints is feasible. We develop a novel algorithm and technique for analyzing the primal linear program, with a unique combination of multiplicative and additive updates. This innovative approach ensures that key inequalities are feasible and keeps the total number of iterations of the algorithm small. Our fractional solution is $O(\log(md)\log(nm))$-competitive on the load and $O(d\log(nm))$-competitive on the energy budget. Our analysis includes a

non-trivial potential function to obtain our competitive result on the load. We then describe an online randomized rounding scheme to produce a competitive ratio of $O(\log(md)\log(nm))$ on the load and $O(d\log^2(nm))$ on the budget. Combining our approach with the rounding scheme of the Generalized Assignment Problem [ST93] also gives an offline result similar to [KLS10] with a substantially simpler rounding scheme. Since our problem generalizes both set cover even in the one-dimensional setting (by setting the processing times to 0 or ∞) and load balancing (by setting all c_i to 0), we can apply lower bounds from the online versions of these problems from [AAA03, AAF93] to get polylogarithmic online lower bounds. We also show that no deterministic algorithm can be competitive.

In Section 3.3, we give upper and lower bounds for variants of the Machine Activation problem. We consider variants where one parameter (either B or Λ) is given up front and the goal is to minimize the other. We obtain our positive results by running our online algorithm from Section 3.2 in phases. Though this induces a logarithmic dependence on the value of the optimal solution, we show that such dependence is necessary for a fully online algorithm, suggesting a semi-online algorithm that is given a good estimate of the optimum performs much better. Lastly, we consider the case where neither parameter is given and the goal is to minimize a linear combination of the maximum load and the energy cost (say $c_\Lambda\Lambda^* + c_B B^*$, where Λ^*, B^* are the makespan and energy cost of a schedule, respectively). Our algorithm is $O(d\log^2(nm))$-competitive on this objective.

In Section 3.4, for the Vector Load Balancing problem, we design an online algorithm which is $O(\log d)$-competitive on the makespan, improving even the best known offline result [CK99]. Our algorithm is simultaneously $O(\log d)$-competitive on the energy usage if we are given a small piece of information (the maximum load induced by any single job on any single component). Without this information, we show our competitive ratios on the two criteria must have product $\Omega(\min(d, \log m))$. Our main technique involves adapting a result of Aspnes et al. [AAF93] which works by assigning jobs greedily based on an exponential cost function. A direct application of their proof technique requires a near-optimum offline solution (which we do not have) and obtains competitive ratio $O(\log md)$, matching random assignment [CK99]. We modify their technique to make use of suitable rough bounds on

the optimum load and exploit the fact that our machines are identical to obtain an $O(\log d)$ bound. Our analysis also includes the use of a non-trivial potential function argument. The work of [ACK13] considers a similar problem from a bin packing perspective. They also develop a greedy algorithm based on exponential cost functions and some of the techniques they use are similar, though the works were done independently of one another. It is perhaps interesting that similar algorithms can be used for both online multidimensional bin packing and load balancing to obtain strong competitive ratios for the two problems.

Related Work

For the Machine Activation problem, the recent work of [KLS10] studies the same problem in the offline setting. They give an algorithm for the unrelated Machine Activation problem which produces a schedule with makespan at most $(2+\epsilon)\Lambda$ and activation cost at most $2(1+\frac{1}{\epsilon})(\ln\frac{n}{OPT}+1)B$ for any $\epsilon > 0$ (where OPT is the number of active machines in the optimal solution), assuming there is a schedule with makespan Λ and activation cost B. They also give a polynomial time approximation scheme for the uniformly related parallel machines case (where machine i has speed s_i and $p_{ij} = \frac{p_j}{s_i}$), which outputs a schedule with activation cost at most B and makespan at most $(1+\epsilon)\Lambda$, for any $\epsilon > 0$. In [LK11], a generalized version of the Machine Activation problem is considered in the offline setting where each machine's activation cost is a function of the load assigned to the machine. An algorithm is given which assigns at least $n - \epsilon$ jobs fractionally with cost at most $(1 + \ln(n/\epsilon))OPT$.

In addition, [LK11] studies an offline version of the Machine Activation problem in which each machine has d linear constraints. For this version, they give a solution which is $O\left(\frac{1}{\epsilon}\log n\right)$ times the optimal activation cost while breaking the d machine constraints by at most a factor of $2d + \epsilon$. This can be compared with our online, multidimensional guarantees of $O(\log(nm)\log(md))$ on the load and $O(d\log^2(nm))$ on the activation cost. There is also the recent work of [ABF13], which we extend to the multidimensional setting. The work of [ABF13] studies several problems. For their generalized framework, which is referred to as the Online Mixed Packing and Covering (OMPC) problem, a deterministic

$O(\log P \log(v\rho\kappa))$-competitive algorithm is given, where P is the number of packing constraints, v is the maximum number of variables in any constraint, and ρ (respectively, κ) is the ratio of the maximum to the minimum non-zero packing (respectively, covering) coefficient respectively. Hence, if all coefficients are either 0 or 1, this is $O(\log P \log v)$-competitive. Note that we cannot simply apply the general scheme in [ABF13] for OMPC, since our packing constraints are not given offline (indeed, this is precisely where the online nature of our problem comes into play). The OMPC framework only models programs in which packing constraints are given offline. The work of [ABF13] also studies a problem called Unrelated Machine Scheduling with Startup Costs (UMSC), which is similar to our problem in the single-dimensional case. In particular, when $d = 1$, they give an $O(\log m)$-competitive result on the makespan and an $O(\log(mn) \log m)$-competitive result on the energy budget. We extend this result to the multidimensional setting. Note that it is not clear how to adapt their algorithm to the multidimensional setting, and we develop our own framework which uses a novel combination of additive and multiplicative updates in our fractional algorithm.

In [AAF93], they consider the online load balancing problem without activation costs. They give an $O(\log m)$-competitive algorithm for unrelated machines and an 8-competitive algorithm for related machines. In [AAG95], they consider the online load balancing problem without activation costs where the load on a machine is measured according to the L_p norm. Their main result is that the greedy algorithm is $O(p)$-competitive under the L_p norm, and any deterministic algorithm must be $\Omega(p)$-competitive. In [ST93], they give the first constant approximation (offline) for the unrelated machines case. In [BN06], the identical machines case is studied in the online setting without activation costs. It is shown that greedy is globally $O(\log m)$-balanced in the restricted assignment model and globally $O(\log m)$-fair in the 1-∞ model (see [BN06] for details). Our problem and techniques are substantially different, as we do not design or analyze a greedy algorithm. For a survey on power management and energy minimization, see [IP05]. Also, see [PST04] for a comprehensive survey on the online scheduling literature.

For the Vector Load Balancing problem, the single-dimensional case has an offline PTAS algorithm [HS87] and a $(2 - \epsilon)$-competitive online algorithm for a small fixed ϵ [BFK92].

The multidimensional version was introduced by [CK99], which includes an offline PTAS with running time exponential in the number of dimensions d and an offline $O(\log^2 d)$-approximation with polynomial running time. They also prove that a simple randomized algorithm is $O\left(\frac{\log md}{\log\log md}\right)$-competitive (with m machines and d dimensions), and that no polynomial-time offline algorithm can attain a constant-factor approximation under standard complexity assumptions.

Most prior work (including a substantial portion of [CK99]) focuses on vector bin packing, where we have a *hard constraint* on the makespan and must minimize the number of machines (bins). Some of these results include [BCS06, CK99, GI97, KLM84, KM77, LS07, LCH07] with the best being $O(\log d)$-approximations. The recent work of [ACK13] studied vector bin packing in the online setting. The algorithm they design for their positive result is similar to ours for Vector Load Balancing, though the two works were done independently.

3.2 Machine Activation

LP and Fractional Algorithm

We formulate the problem as an integer program, where $y_i = 1$ means machine i is activated, and $x_{ij} = 1$ means job j is assigned to machine i. We assume the target load Λ and budget B are given, and that there is an offline integral solution with makespan at most Λ and budget at most B.

1. For all $1 \le i \le m$, we have $0 \le y_i \le 1$.
2. For all $1 \le i \le m$ and $1 \le j \le n$, we have $x_{ij} \ge 0$.
3. For all j, we have $\sum_{i=1}^{m} x_{ij} \ge 1$.
4. For all i, j, we have $x_{ij} \le y_i$.
5. For all i, k, we have $\sum_{j=1}^{n} x_{ij} p_{ij}^k \le \Lambda y_i$.
6. We have $\sum_{i=1}^{m} y_i c_i \le B$.

Our goal is to design an online algorithm to solve the integer version of this linear system, while violating constraints 1 and 6 by at most a bounded factor. We will do this by first providing an online solution to the linear relaxation above, which may also violate the first constraint by possibly having $y_i \geq 1$, then describe an online rounding technique to produce an integer solution. We will assume that either $\frac{B}{m} \leq c_i \leq B$ or $c_i = 0$ for all i (if more then discard machine i as offline cannot use it, if less then simply buy machine i and assume $c_i = 0$ for a constant-factor increase in the total cost). Note that we can normalize B to anything we wish – we will choose $B = \Theta(m)$ so that any non-zero c_i is at least a constant (for instance, at least 1). We define $q_{ij}^k = p_{ij}^k/\Lambda$ and $\ell_i^k = \sum_j q_{ij}^k \max\{x_{ij} - \frac{1}{jm}, 0\}$. Let $a \geq 1$ be a value to be set later.

We first give some intuition for our algorithm. When a job j arrives, most constraints are satisfied except for $\sum_{i=1}^m x_{ij} \geq 1$. To fix this, we need to raise the x_{ij} variables until this inequality is satisfied. However, this may cause other inequalities such as $x_{ij} \leq y_i$ and $\sum_{j=1}^n x_{ij} p_{ij}^k \leq \Lambda y_i$ to be violated. Hence, we will only increase x_{ij} if $x_{ij} \leq y_i$ will continue to hold (if we do raise x_{ij}, then we also increase y_i to satisfy the load inequalities). If increasing x_{ij} would cause $x_{ij} > y_i$, then we simply increase y_i. We increase x_{ij} multiplicatively, so that the larger x_{ij} is, the larger the increase. Moreover, it seems intuitively clear that we should increase x_{ij} for job j less aggressively if p_{ij}^k, c_i, or ℓ_i^k is large (in fact, we penalize machine i with an exponential cost function for the load ℓ_i^k to obtain our competitive ratio on the load constraints). We also increase y_i multiplicatively whenever it is too small (again, intuitively, y_i should be increased less aggressively if c_i is large). When a variable is small, it may take many iterations for multiplicative updates to increase its value substantially. We use additive updates to avoid this issue, and couple the additive updates with multiplicative ones to achieve our feasibility and competitive guarantees by keeping the number of iterations in our algorithm small. See Algorithm 3.1 for details.

Analysis

We prove certain feasibility properties which our algorithm maintains.

1 Initialize $x_{ij} \leftarrow 0$ for all i, j and $y_i \leftarrow 0$ for all i

2 When job j arrives, set $x_{ij} \leftarrow \frac{1}{jm}$ for each i such that $p_{ij}^k \leq \Lambda$ for all k and $y_i \leftarrow y_i + \frac{1}{jm}$ for all i

3 **while** job j has $\sum_{i=1}^{m} x_{ij} < 1$ **do**

4 **for** each $1 \leq i \leq m : p_{ij}^k \leq \Lambda$ for all k **do**

5 Set $z_{ij} \leftarrow \frac{x_{ij}}{\sum_k p_{ij}^k (a^{\ell_i^k} + c_i) + 1}$

6 **if** $x_{ij} + z_{ij} \leq y_i$ **then**

7 Set $x_{ij} \leftarrow \min\{x_{ij} + z_{ij}, 1\}$ and $y_i \leftarrow y_i + z_{ij} \max_k q_{ij}^k$

8 **else**

9 Set $y_i \leftarrow y_i \left(1 + \frac{1}{d\Lambda c_i}\right)$

Algorithm 3.1: Fractional Assignment

Theorem 3.1. *Inequalities* 3, 4, *and* 5 *of the linear program are satisfied.*

Proof. Inequality 3 follows from the termination condition of step three of the algorithm. The other two inequalities hold initially since all variables are zero. When j arrives, we set $x_{ij} \leftarrow \frac{1}{jm}$ for every i where $q_{ij}^k \leq 1$ along each dimension k, but we also increase all y_i by $\frac{1}{jm}$, so both inequalities will continue to hold.

Later, we might increase x_{ij} by some z_{ij}. However, we will only do this if $x_{ij} + z_{ij} \leq y_i$, so $x_{ij} \leq y_i$ still holds. When we increase x_{ij} in this way, we will also increase y_i by $z_{ij} \max_k q_{ij}^k$, which guarantees that $\sum_j x_{ij} p_{ij}^k \leq \Lambda y_i$ will continue to hold for all i and all k. □

Each time through the loop at line three of the algorithm will be called a **reinforcement** step. Let r_j represent the number of reinforcement steps which occur on the arrival of j.

Lemma 3.1. *For $a \geq 1$, when a job j arrives, the r_j reinforcement steps due to the job's arrival increase $\sum_{i=1}^{m} \sum_{k=1}^{d} a^{\ell_i^k}$ by at most $\frac{a-1}{\Lambda} r_j$.*

Proof. Each reinforcement step increases ℓ_i^k by at most $z_{ij} q_{ij}^k$ for each i and k. Thus, the total increase in the summation is bounded by $\sum_{i=1}^{m} \sum_{i=1}^{k} a^{\ell_i^k} (a^{z_{ij} q_{ij}^k} - 1)$. By the definition of z_{ij}, we will always have $z_{ij} q_{ij}^k \leq 1$. In general for any $a \geq 1$ we will have $a^x - 1 \leq$

$(a-1)x$ whenever $0 \leq x \leq 1$, and applying this allows us to bound the summation by $\sum_{i=1}^{m}\sum_{k=1}^{d} a^{\ell_i^k}(a-1)(z_{ij}q_{ij}^k)$. We can substitute $z_{ij} \leq \frac{x_{ij}}{\sum_k p_{ij}^k a^{\ell_i^k}+1}$ and use the fact that $\sum_i x_{ij} < 1$ prior to the reinforcement to get that $\sum_{i=1}^{m} \frac{a-1}{\Lambda} x_{ij} \leq \frac{a-1}{\Lambda}$. □

Lemma 3.2. *The total number of reinforcement steps after all jobs have arrived is bounded by* $\sum_j r_j \leq \Lambda(\log nm)(\sum_i \sum_k a^{\ell_i^k} + 2dB) + n(1+\log mn)$.

Proof. We split the reinforcements occurring on the arrival of j into two sets. Suppose i is the machine to which j is assigned by the optimum solution. We have $\bar{r}_j$ **load reinforcing** steps where $x_{ij}+z_{ij} \leq y_i$ and $\hat{r}_j$ **cost reinforcing** steps where the opposite was true. Clearly, $r_j = \bar{r}_j + \hat{r}_j$.

Each time a load reinforcing step occurs, x_{ij} increases by a factor of at least $1 + \frac{1}{\sum_k p_{ij}^k (a^{\ell_i^k}+c_i)+1}$ (except for possibly the last and only reinforcement step for job j in which x_{ij} is set to 1 instead of $x_{ij}+z_{ij}$ at line seven). Thus, every $\sum_k p_{ij}^k(a^{\ell_i^k}+c_i)+1$ such steps increase x_{ij} by a constant factor. Since x_{ij} is initially at least $\frac{1}{mn}$, the total number of such steps is bounded by $\log(nm)(\sum_k p_{ij}^k(a^{\ell_i^k}+c_i)+1)+1$. Summing over all machines i used by the optimum solution, we get $\sum_j \bar{r}_j \leq \Lambda \log(nm)(\sum_{i\in OPT}\sum_k a^{\ell_i^k} + dB) + n\log(mn) + n$.

Consider all cost reinforcing steps for jobs which the optimum assigns to machine i. The initial value of y_i is $\frac{1}{m}$. Each time we apply a cost reinforcing step, we increase this by a multiplicative $1+\frac{1}{d\Lambda c_i}$. Note that we will never have $x_{ij}+z_{ij} \geq 2$, as we would not perform a reinforcing step unless $x_{ij} < 1$, from which $z_{ij} < 1$ follows. Thus, to perform a cost reinforcing step we must have $y_i \leq 2$. It follows that the total number of cost reinforcing steps performed for j with optimum assignment i is at most $d\Lambda c_i \log 2m$. If we sum this over all active machines i in the optimum solution (and observe that the optimum solution must not exceed the budget B) we get $\sum_j \hat{r}_j \leq dB\Lambda \log 2m$. Combining the equations along with the assumption $n \geq 2$ gives the lemma. □

Lemma 3.3. *If we set* $a = 1 + \frac{1}{2\log nm}$, *then the final value of the potential function* $\Phi = \sum_i \sum_k a^{\ell_i^k}$ *after all jobs have arrived is at most* $3md + 2dB$.

Proof. Initially, the potential function equals md (since all loads $\ell_i^k = 0$). The function Φ

increases with each reinforcement step, so by Lemma 3.1 and Lemma 3.2, the final value is $\Phi \leq md + \frac{a-1}{\Lambda}\sum_j r_j \leq md + \frac{a-1}{\Lambda}(\Lambda \log(nm)(\sum_i \sum_k a^{\ell_i^k} + 2dB) + n(1 + \log mn))$. For $a = 1 + \frac{1}{2\log nm}$, this implies that the final value is $\Phi \leq 3md + 2dB$ (we can normalize Λ to anything, so we choose $\Lambda = \Theta(n)$). □

Theorem 3.2. *For $a = 1 + \frac{1}{2\log nm}$, we have $y_i \leq \left[\frac{\log n}{m} + 4 + 2\log(nm)\log(3md + 2dB)\right]$ for all machines i.*

Proof. Applying Lemma 3.3 along with the value of a specified in the theorem, we know for any k that $\sum_i a^{\ell_i^k} \leq 3md + 2dB$ (where ℓ_i^k is the final load on machine i). Since each term in the summation is non-negative, we can bound the ℓ_i^k values after taking the log of both sides by $\ell_i^k \leq 2\log(nm)\log(3md + 2dB)$.

We observe that y_i increases at several points in the algorithm. The total increase at step two of the algorithm can be at most $\frac{\log n}{m}$. The increases at line nine can only occur if $y_i \leq 2$, and will not cause y_i to exceed four (since we can ensure $1 + \frac{1}{d\Lambda c_i} \leq 2$ by scaling Λ and B appropriately). The increases at line seven always increase ℓ_i^k by the same amount as y_i, so the total increase in y_i due to these steps is at most ℓ_i^k, which is bounded as above. Combining these gives the result. □

Theorem 3.3. *The algorithm satisfies $\sum_i y_i c_i \leq B\log n + (6md + 8dB + 4)\log nm$.*

Proof. Initially, the left side of the equation is 0. When j arrives, every y_i increases by $\frac{1}{jm}$. Since each y_i has $c_i \leq B$ (otherwise we drop that machine), the total increase in cost due to this is at most $\frac{B}{j}$. Thus, in total, all arrivals increase the cost by at most $\sum_j \frac{B}{j} = B\log n$.

We also increase the y_i values when we perform a reinforcing step. Some y_i values increase by an additive $z_{ij}\max_k q_{ij}^k \leq \frac{x_{ij}}{\Lambda c_i}$, while others increase by a multiplicative $1 + \frac{1}{d\Lambda c_i}$. The increase in cost due to additive increases is at most $\frac{\sum_i x_{ij}}{\Lambda}$, while for multiplicative increases it is at most $\sum_i \frac{y_i}{d\Lambda}$. For the multiplicative increase to happen we must have $y_i < x_{ij} + z_{ij} < 2x_{ij}$, so the multiplicative increase is at most $\frac{2\sum_i x_{ij}}{d\Lambda}$. Since each i appears in only one of the two summations, the total increase in cost is at most $\frac{2}{\Lambda}$ for each reinforcement

step. The number of reinforcement steps is bounded in Lemma 3.2 along with Lemma 3.3 (note that we normalize Λ as in Lemma 3.3). Combining these gives the result. □

We can normalize B to whatever value we like. Given the expressions for the competitive ratio on the load and the cost, it is natural to set $B = \Theta(m)$. This will guarantee a competitive ratio of $O(d \log nm)$ on the cost, and a competitive ratio of $O(\log(md) \log(nm))$ on the load.

Rounding

We now show how to round our fractional solution to an integral solution. Our integral solution is $O(\log(nm) \log(md))$-competitive on the load with high probability and $O(d \log^2 nm)$-competitive on cost. Suppose we have some online fractional algorithm which guarantees that the values of x_{ij} and y_i never decrease, and maintains inequalities from the linear program except that it relaxes the first inequality to $0 \leq y_i \leq \rho_\Lambda$ and the last inequality to $\sum_{i=1}^{m} y_i c_i \leq B\rho_B$. In fact, our fractional algorithm also guarantees that $x_{ij} \leq 1$. We will show that this can be rounded in an online manner to produce integral $\hat{x}_{ij}$ and $\hat{y}_i$. Observe that our fractional algorithm (Algorithm 3.1) will satisfy the constraints with $\rho_\Lambda = O(\log(nm) \log(md))$ and $\rho_B = O(d \log nm)$.

Our rounding procedure is as follows. For each machine i, we compute a uniformly random $r_i \in [0, 1]$. We set $\hat{y}_i \leftarrow 1$ as soon as $y_i \log 2nm \geq r_i$. We define $M(j)$ as the set of machines with $\hat{y}_i = 1$ immediately after job j arrives. For each job j, let $y_i(j) = \min\{y_i, 1\}$ immediately after job j arrives. We observe that $x_{ij} \leq y_i(j)$, since the fourth linear program equation must hold at all times and $x_{ij} \leq 1$. We define $s_j = \sum_{i \in M(j)} \frac{x_{ij}}{y_i(j)}$. If $s_j < \frac{1}{2}$, then we immediately set $\hat{y}_i \leftarrow 1$ for all machines i and recompute s_j. We select exactly one machine i from $M(j)$ to assign j, setting $\hat{x}_{ij} \leftarrow 1$ for this machine only. Each machine is selected with probability $\frac{x_{ij}}{y_i(j) s_j}$.

Lemma 3.4. *The probability that we ever have $s_j < \frac{1}{2}$ after any j arrives is at most $\frac{1}{m}$; thus the increase in the expected total cost of the solution due to this case is at most B.*

Proof. Let $A(j)$ be the set of machines for which $y_i(j) \geq \frac{1}{\log 2nm}$. Clearly, $A(j) \subseteq M(j)$ since all of these machines are active with probability one. Observe that if $\sum_{i \in A(j)} x_{ij} \geq \frac{1}{2}$, then $s_j \geq \frac{1}{2}$ with probability 1. Hence, we consider the case when $\sum_{i \in A(j)} x_{ij} < \frac{1}{2}$. Since $\sum_i x_{ij} \geq 1$, it follows that $\sum_{i \notin A(j)} x_{ij} > \frac{1}{2}$. The actual value of s_j will depend on the random choices of r_i, since s_j is computed by summing over only the active machines. We can write the equation $s_j \geq \sum_{i \notin A(j)} \frac{x_{ij}}{y_i(j)} \hat{y}_i$. Since $i \notin A(j)$, we can guarantee $E[\hat{y}_i] = y_i(j) \log 2nm$, implying $E[s_j] \geq \frac{1}{2} \log 2nm$.

The value of s_j is a sum of independent Bernoulli variables, each of which has value at most 1 (since $x_{ij} \leq y_i(j)$). Even though the variables range in between 0 and 1, we can still apply Chernoff type bounds to conclude: $P\left[s_j < \frac{1}{2}\right] \leq P\left[s_j < (1 - \frac{1}{2})E[s_j]\right] \leq \left(\frac{e^{-.5}}{.5^{.5}}\right)^{E[s_j]} \leq \sqrt{\frac{2}{e}}^{\frac{1}{2} \log 2nm} \leq \frac{1}{nm}$ (assuming the base of the logarithm is a sufficiently small constant). Applying the union bound, we can sum this over all j and conclude that the probability of ever having $s_j < \frac{1}{2}$ for any j is less than $\frac{1}{m}$. Hence, the increase in expected total cost due to this case is at most $\frac{1}{m} \sum_i c_i \leq B$. □

Lemma 3.5. *Every job is assigned to exactly one active machine. The expected total cost of the solution is bounded by* $E\left[\sum_i \hat{y}_i c_i\right] \leq B \rho_B \log 2nm$.

Proof. The rounding scheme assigns each job to exactly one active machine, and the set of active machines only grows over time as the y_i values are non-decreasing. The manner in which the $\hat{y}_i$ are determined along with the inequality $\sum_i y_i c_i \leq B \rho_B$ produce the cost bound (since $E\left[\hat{y}_i\right] \leq y_i \log 2nm$). Note that the additional expected cost as specified in Lemma 3.4 in the case that $s_j < \frac{1}{2}$ is small and does not change the competitive ratio asymptotically. □

Lemma 3.6. *For any active machine i and dimension k, with high probability the total load satisfies* $\sum_j \hat{x}_{ij} p_{ij}^k \leq \Lambda[2\rho_\Lambda + 4 \log m + \beta \log(md)]$, *where β is a suitably chosen constant.*

Proof. Consider any active machine i (note that $\hat{y}_i = 1$). Each job j is assigned to this machine with probability $\frac{x_{ij}}{y_i(j) s_j}$. The total load of the machine on dimension k is $\sum_j \hat{x}_{ij} p_{ij}^k$; we will assume that $p_{ij}^k \leq \Lambda$ since otherwise $x_{ij} = 0$ and thus $\hat{x}_{ij} = 0$. The problem here is that the $y_i(j)$ values change over time; if we could replace them all with the final y_i then we

could use the linear program inequalities to conclude that the expected load is at most $\rho_\Lambda \Lambda$ and then apply Chernoff bounds.

Instead, we define phase α to consist of those times when $\frac{2^\alpha}{m} \leq y_i(j) < \frac{2^{\alpha+1}}{m}$, for each $0 \leq \alpha \leq \log m$. Even though $y_i > 1$ is possible, we will never have $y_i(j) > 1$ so every j arrives in some phase. Let $J(\alpha)$ be the jobs which arrive during phase α. For $\alpha < \log m$, we have: $E\left[\sum_{j \in J(\alpha)} \hat{x}_{ij} p_{ij}^k\right] \leq \sum_{j \in J(\alpha)} \frac{x_{ij}}{y_i(j) s_j} p_{ij}^k \leq \sum_{j \in J(\alpha)} 2 x_{ij} p_{ij}^k \frac{m}{2^\alpha}$. However, we know $\sum_{j \in J(\alpha)} x_{ij} p_{ij}^k \leq \Lambda \frac{2^{\alpha+1}}{m}$ for any phase except the last, so we can apply this inequality to conclude $E\left[\sum_{j \in J(\alpha)} \hat{x}_{ij} p_{ij}^k\right] \leq 4\Lambda$. Thus, the expected total load is at most $4\Lambda \log m$ from all phases but the last. For the last phase, the expected load is at most $2\rho_\Lambda \Lambda$, since $y_i(j) = 1$ throughout. We observe that the loads are sums of Bernoulli variables with values at most Λ, so we can apply Chernoff bounds to show that with high probability the load will not exceed its mean plus $\beta \Lambda \log(md)$ on any dimension of any machine. □

Offline Scheme and Lower Bound

For the offline case, we can simply solve the linear program, so $\rho_\Lambda = \rho_B = 1$. Instead of rounding up when $y_i \log 2nm \geq r_i$, we can round up when $y_i \log 2n \geq r_i$. This increases the probability of the bad event where some $s_j < \frac{1}{2}$ to be a small constant instead of $\frac{1}{m}$, but we can simply discard our solution and retry whenever this occurs. This means we can get a solution of cost $O(\log n)$ times B, such that a fractional solution exists which exceeds Λ by at most a constant factor on any machine. We can then convert our fractional solution to an integer solution using the rounding approach of the Generalized Assignment Problem [ST93]. This attains roughly the same bounds as [KLS10] for the offline case with a substantially simpler rounding scheme.

There is a simple example which indicates that no deterministic approach to this problem can succeed. We simply give a series of requests each of which can run on every machine *except the ones where the preceding requests were scheduled*. This forces a deterministic algorithm to pay for all m machines, whereas the offline optimum could activate only a single machine. The Online Set Cover result of Alon et al. [AAA03] got around this by

presuming that we know in advance the set of elements (jobs) which *might* be requested in the future, and allowed the competitive ratio to depend on the size of this set. This approach seems less reasonable for our problem than theirs, but in any case a modification of their derandomization should work in the same model.

3.3 Machine Activation Variants

We study four versions of online load balancing with activation costs. For the version where both Λ and B are given, Section 3.2 gives an algorithm that is $O(\log(nm)\log(md))$-competitive on the load and $O(d\log^2(nm))$-competitive on cost. We also consider variants where either B is not given up front or Λ is not given up front (or both are not given). Our positive results are obtained by guessing the value of the objective function to be optimized in an online manner using doubling techniques combined with our randomized algorithm from Section 3.2. The logarithmic dependence on the optimum load (or budget) in some of the results is undesirable, and we observe that it would not occur in the offline setting (where we can discard the solution of previous phases and start over with each new phase). However, we can show that this dependence is necessary for a fully online algorithm.

Theorem 3.4. *For the version where we are given a budget B and asked to minimize load Λ, we can produce a solution which spends at most $\rho_B B$ and has competitive ratio ρ_Λ on the load against the optimum offline with budget B. Here, $\rho_\Lambda = O(\log(md)\log(nm))$ but $\rho_B = O(d\log^2(nm)\log\Lambda^*)$, where Λ^* is the optimum load (assuming we have a lower bound on the load of one; otherwise it is the ratio of maximum to minimum possible non-zero load).*

Proof. We are given the value B up front, and we are asked to minimize the load Λ. This theorem is obtained by using standard doubling techniques in combination with running our online randomized algorithm from Section 3.2.

In particular, a natural approach to this version is to guess the value of the parameter Λ^*, and then run the online algorithm until it becomes apparent that this value is too small. We then double the parameter and continue. The algorithm therefore runs in a number

of phases, and the total activation cost budget will be the sum of the budgets spent at each phase (similarly, the total load is the sum of loads generated at each phase). For the parameter Λ, this sum is geometric and therefore increases the end value by only a constant; however, our activation cost increases by a factor of the number of phases, which is $\log \Lambda^*$. $\square$

Theorem 3.5. *For the version where we are given load Λ and asked to minimize the budget B, we can produce a solution with load at most $\rho_\Lambda \Lambda$ which has competitive ratio ρ_B against the optimum offline which does not exceed load Λ. Here, $\rho_B = O(d \log^2 nm)$ but $\rho_\Lambda = O(\log(md) \log(nm) \log B^*)$, where B^* is the ratio of the optimum budget to the minimum non-zero cost of a machine.*

Proof. The proof is similar to the proof of Theorem 3.4, except that we guess the optimal solution's activation cost B^* (as opposed to guessing the optimal load Λ^*). $\square$

Theorem 3.6. *Consider the version where we are given a budget B and asked to minimize load Λ. Suppose that the actual optimum load is Λ^*, and we are to guarantee that we spend a budget at most $\rho_B B$ and obtain load at most ρ_Λ times optimum. Then $\rho_B = \Omega(\min\{\frac{\log \Lambda^*}{\log \log \Lambda^* + \log \rho_\Lambda}, m\})$.*

Proof. Our lower bound will apply even in the single-dimensional case. We are given m machines, each of which has activation cost $c_i = 1$. Our budget is $B = 1$, and we are asked to minimize the load online. The adversary submits up to n jobs, where $n = \min\{\frac{\log \Lambda^*}{\log(2\rho_\Lambda \log \Lambda^*)}, m\}$. Each job j can run only on machines j through m. To run job j on machine i, we will induce a load $p_{ij} = L^{i-1}$. We will define $L = 2\rho_\Lambda \log \Lambda^*$. Note that $L^n \leq \Lambda^*$ and $n < L$.

We observe that the offline solution can handle jobs 1 through j using only machine j while staying within its budget, and the total load in this case will be $jL^{j-1} \leq nL^{n-1} < L^n \leq \Lambda^*$. If our algorithm were to place any job on machine $j+1$ or higher with probability more than half, then our expected load would be at least $\frac{1}{2}L^j > \rho_\Lambda j L^{j-1}$, violating our competitive ratio on the load. Thus, our algorithm places job j on machine j with probability at least half, implying that we activate machine j with probability at least half. This applies to the

first n machines, so our algorithm pays a cost of at least $\frac{1}{2}n$ in expectation, implying that $\rho_B = \Omega(\min\{\frac{\log \Lambda^*}{\log\log \Lambda^* + \log \rho_\Lambda}, m\})$. □

Theorem 3.7. *If a deterministic algorithm guarantees to be ρ_Λ-competitive on the makespan and uses at most $\rho_B B$ budget, then $\rho_B = \Omega\left(\frac{\log \Lambda^*(\log n - \log\log m - \log\log \Lambda^*)(\log m - \log\log \Lambda^*)}{(\log \rho_\Lambda + \log\log n + \log\log m + \log\log \Lambda^*)(\log\log n + \log\log m)}\right)$.*

Proof. Our lower bound will apply even in the single-dimensional setting. We first consider a variant of the problem where a universe U of n jobs is pre-announced, but only some subset of $n' \leq n$ jobs arrive. It is clear that this problem is no harder than the original problem since this is only providing more information to the online algorithm (and n only gets larger). Hence, a lower bound for this variant provides a lower bound for the original version. Let OPT denote the optimal solution and let ALG denote the deterministic algorithm. Let p, q, r be positive integers. Our universe $U = \{0, 1, 2, \ldots, n-1\}$ where $n = 2^p pq^2 r$. We will partition U into r "phases" and each phase into pq^2 "blocks." The first phase will contain precisely the first $2^p pq^2$ jobs, the second phase will contain the next $2^p pq^2$ jobs, and so on. Within each phase k, the first block X_1^k contains the first 2^p jobs of the phase, the second block X_2^k contains the second 2^p jobs of the phase, and so on up to block $X_{pq^2}^k$, which contains the last 2^p jobs of the phase.

For each set $B = \{b_1, b_2, \ldots, b_q\} \subseteq \{1, 2, \ldots, pq^2\}$ with $b_1 < b_2 < \cdots < b_q$ and each set $I = \{i_1, i_2, \ldots, i_q\}$ where $1 \leq i_t \leq p$ for all t and each $1 \leq k \leq r$, we will have a machine $M_{B,I}^k$. This machine will be able to run any job from phases $1, 2, \ldots, k-1$, and any job in $X_{b_t}^k$ that has the i_t^{th} least significant bit in its binary encoding set to 1. Each of these jobs will induce load $\frac{1}{pqk} L^k$ on machine $M_{B,I}^k$ for some $L > \rho_\Lambda pq(r+1)$. Each machine will have activation cost 1 and our total allowed budget will be 1. We note that the number of machines is $m = \binom{pq^2}{q} p^q r$.

The instance will work in phases. In each phase k, exactly pq jobs will arrive. We will ensure that at any time OPT can place all jobs (including those from previous phases) on a single machine within the current phase. Since in phase k a total of pqk jobs will have arrived, this means the optimal makespan is at most L^k at the end of phase k.

Jobs arrive within phase k as follows: take the job from X_1^k that has all p least significant

bits set to 1. The algorithm ALG must select one machine on which to place this job. This job cannot go on any machine in a previous phase. Moreover, if ALG places the job on a machine in a future phase then this will induce load:

$$\frac{1}{pq(k+1)}L^{k+1} > \frac{1}{pq(k+1)}(\rho_\Lambda pq(r+1))L^k \geq \rho_\Lambda L^k.$$

Thus, since ALG guarantees to be ρ_Λ-competitive on the makespan, this cannot happen. So it follows that ALG must place this job on some machine in the current phase. This machine is able to do all jobs in X_1^k with some bit b_1 set to 1. Thus, the next job to arrive will be the job from X_1^k that has all p least significant bits except b_1 set to 1. Clearly, ALG must now choose a different machine to process this job. This continues for k steps, at which point ALG has activated k machines, but OPT could have activated only one.

At this point, ALG has k different machines which can accommodate jobs from at most pq blocks in phase k. Since there are pq^2 blocks, there must be another block we can select and repeat the above process. In fact, we can repeat this q times, forcing ALG to activate pq machines. Notice that OPT can select the single machine that handles elements from exactly the blocks we select (and exactly the correct bit in each block). In total, ALG will be forced to activate pqr machines whereas OPT is able to activate only one machine.

Notice that $p < \log n$, $q < \log m$, and $r + 1 < \log \Lambda^*$. Hence, we can set the value $L = \rho_\Lambda \log n \log m \log \Lambda^* > \rho_\Lambda pq(r+1)$, so that $r = \frac{\log \Lambda^*}{\log \rho_\lambda + \log\log n + \log\log m + \log\log \Lambda^*}$. Since $n = 2^p pq^2 r$, we have $p \geq \Omega(\log n - \log\log m - \log\log \Lambda^*)$. Finally, since $m \leq (pq^2)^q p^q r$, we have $q \geq \frac{\log m - \log\log \Lambda^*}{\log\log n + \log\log m}$. It follows that

$$\rho_B \geq \Omega\left(\frac{\log \Lambda^*(\log n - \log\log m - \log\log \Lambda^*)(\log m - \log\log \Lambda^*)}{(\log \rho_\Lambda + \log\log n + \log\log m + \log\log \Lambda^*)(\log\log n + \log\log m)}\right).$$

□

Theorem 3.8. *Consider the version where we are given a load Λ and asked to minimize the cost B. Suppose that the optimum cost is B^* to stay within load Λ. We are to guarantee load at most $\Lambda\rho_\Lambda$ and cost at most ρ_B times B^*. Then $\rho_\Lambda = \Omega(\min\{\frac{\log B^*}{\log 2\rho_B}, m\})$.*

Proof. Our lower bound will again apply even for the single-dimensional setting. We are given m machines labeled $i = 0$ through $i = m - 1$, where machine i has an activation cost

of $c_i = (2\rho_B)^i$. We are asked to maintain a load of at most $\Lambda = 1$. The adversary gives at most n requests, where $n = \min\{\frac{\log B^*}{\log 2\rho_B}, m\}$. Request $j \geq 1$ has $p_{ij} = 1$ for machines $i = 0$ and $i = j$ and otherwise has infinite load. We observe that the optimum can obey the load constraints on the first k jobs by activating machines 0 through $k-1$, placing job k on machine 0 and job $j < k$ on machine j. This has cost $\sum_{i=0}^{k-1}(2\rho_B)^i < 2(2\rho_B)^{k-1} < B^*$. Thus, the online algorithm cannot activate any machine k or higher without violating the ρ_B bound on the competitive ratio, from which it follows that job k must be placed on machine 0. Yet this applies to every job, so we must place all n jobs on machine 0, violating the load by $\rho_\Lambda = \Omega(\min\{\frac{\log B^*}{\log 2\rho_B}, m\})$. □

Theorem 3.9. *There is an $O(d \log^2 nm)$-competitive algorithm for minimizing a linear combination of the cost B and load Λ, namely $c_B B + c_\Lambda \Lambda$. Here, the coefficients c_B, c_Λ are constants and $c_B B + c_\Lambda \Lambda$ is the value of the objective function on a schedule with energy cost B and makespan Λ.*

Proof. We will run in phases, where in phase i we assume that $B = \frac{2^i}{c_B}$ and $\Lambda = \frac{2^i}{c_\Lambda}$, and run the online randomized algorithm of Section 3.2 for each phase.

Suppose that the actual optimum value is $2^{\alpha-1} < OPT \leq 2^\alpha$. Thus, we have $B^* \leq \frac{2^\alpha}{c_B}$ and $\Lambda^* \leq \frac{2^\alpha}{c_\Lambda}$, so the algorithm will terminate on phase α or earlier.

The total load generated by the algorithm is the sum of loads from each phase up to α. The load from phase i is at most $(\log nm)(\log md)2^i$, and summing these gives $\Lambda \leq O(2^{\alpha+1}(\log nm)(\log md))$. The total cost can be bounded by $B \leq O(2^{\alpha+1} d (\log nm)^2)$. Combining these gives the result. □

3.4 Vector Load Balancing

We give an $O(\log d)$-competitive algorithm to minimize the makespan, improving over the offline $O(\log^2 d)$-approximation in [CK99] (for arbitrary d). Note that [CK99] shows that even in the offline case, no constant approximation is possible unless NP = ZPP. We extend our algorithm to be simultaneously $O(\log d)$-competitive on energy, provided that we are

Initialize $\Lambda \leftarrow 1$, $x \leftarrow 0$, $\ell_i^{kt} \leftarrow 0$ for all i, k, t
while jobs j arrive **do**
    Update Λ_{max} and Λ_{tot}
    if $\Lambda < \max\{\Lambda_{max}, \frac{1}{m}\Lambda_{tot}\}$ **then**
        End phase x; let $x \leftarrow \lceil \log_2 \max\{\Lambda_{max}, \frac{1}{m}\Lambda_{tot}\} \rceil$, $\Lambda \leftarrow 2^x$
    Place job j on machine $s = \text{argmin}_i \sum_{k=1}^{d} [a^{\ell_i^{kx} + q_j^k} - a^{\ell_i^{kx}}]$; let $\ell_s^{kx} \leftarrow \ell_s^{kx} + q_j^k$ for all k

Algorithm 3.2: Assign-Jobs

given $\Lambda_{max} = \max_{k,j} p_j^k$ in advance (p_j^k is the load of job j on component k). We show that this additional information is necessary.

Minimizing the Makespan

The algorithm of [AAF93] depends heavily on maintaining an estimate Λ of the optimum value. We use the maximum coordinate of any single job $\Lambda_{max} = \max_{k,j} p_j^k$ and the load induced by placing all jobs on one machine $\Lambda_{tot} = \max_k \sum_j p_j^k$ as lower bounds in Algorithm 3.2. We run in phases where for each phase, we use an estimate Λ of the optimum makespan. If Λ is too small, we adjust and start a new phase. We also make use of $a = 1 + \frac{1}{\gamma}$ for some $\gamma > 1$. Let $\ell_i^{kx}(j)$ be the load normalized by Λ on component k of machine i during phase x after all jobs up through j are assigned (we sometimes omit j if the context is clear). Let p_j^k be the load induced on dimension k by job j, and $q_j^k = p_j^k / \Lambda$. The proof of the following lemma is a modification of the proof in [AAF93].

Lemma 3.7. *In Algorithm 3.2, during each phase x:* $\sum_{k=1}^{d} \sum_{i=1}^{m} a^{\ell_i^{kx}} (\gamma - 1) \leq \gamma m d$.

Proof. We first let $\lambda_j^k = \frac{1}{m} \sum_{t=1}^{j} q_t^k$. We also define the following potential function $\Phi(j) = \sum_{k=1}^{d} \sum_{i=1}^{m} a^{\ell_i^{kx}(j)} (\gamma - \lambda_j^k)$.

Note that at the beginning of phase x, all the values $\ell_i^{kx}(j)$ are zero, so the value of the potential is at most $\gamma m d$. We claim that as the phase continues, this potential function can only decrease. Observing that throughout the phase $\lambda_j^k \leq 1$ (since otherwise we would have

$\frac{1}{m}\Lambda_{tot} > \Lambda$ and the phase would end) will then establish the lemma. Now suppose that job j during phase x is placed upon machine s. The potential function will change as follows:

$$\begin{aligned}
\Phi(j) - \Phi(j-1) &= \sum_{k=1}^{d}(\gamma - \lambda_{j-1}^k)(a^{\ell_s^{kx}(j)} - a^{\ell_s^{kx}(j-1)}) - \sum_{k=1}^{d}\sum_{i=1}^{m}(\lambda_j^k - \lambda_{j-1}^k)(a^{\ell_i^{kx}(j)}) \\
&\leq \sum_{k=1}^{d}\gamma(a^{\ell_s^{kx}(j)} - a^{\ell_s^{kx}(j-1)}) - \frac{1}{m}\sum_{k=1}^{d}\sum_{i=1}^{m} q_j^k(a^{\ell_i^{kx}(j)}) \\
&\leq \frac{1}{m}\sum_{k=1}^{d}\sum_{i=1}^{m}\gamma(a^{\ell_i^{kx}(j-1)+q_j^k} - a^{\ell_i^{kx}(j-1)}) - q_j^k(a^{\ell_i^{kx}(j)}) \\
&\leq \frac{1}{m}\sum_{k=1}^{d}\sum_{i=1}^{m} a^{\ell_i^{kx}(j-1)}(\gamma a^{q_j^k} - \gamma - q_j^k) \\
&\leq 0.
\end{aligned}$$

The first inequality follows from the definition of λ^k and the observation that $\lambda_{j-1}^k \leq \Lambda$. The second from our choice of machine s. The third inequality comes from the observation that for any i, $a^{\ell_i^{kx}(j)} \geq a^{\ell_i^{kx}(j-1)}$. The last inequality comes from the definition of a, γ; in particular, the fact that for any $y \in [0,1]$, we have $\gamma(a^y - 1) \leq y$ along with the fact that $q_j^k \in [0,1]$ because $\Lambda \geq \Lambda_{max} \geq p_j^k$. Since the potential can never increase, we conclude that the *final* potential is at most the *initial* potential of γmd. □

At this point, we can follow [AAF93] and use Lemma 3.7 for an $O(\log dm)$ bound. We will improve our competitive ratio by exploiting the identical nature of our machines. Let $\mu^x = \max_{i,k} \ell_i^{kx}$.

Lemma 3.8. *For all machines i, i' and phases x, we have $\sum_{k=1}^{d} a^{\ell_i^{kx}(f)} \leq d + \frac{\mu^x}{\gamma}\sum_{k=1}^{d} a^{\ell_{i'}^{kx}(f)}$, where f is the final job of phase x.*

Proof. Let J_i be the set of jobs given to machine i during phase x. If $j \in J_i$, machine i must minimize $\Delta_i(j) = \sum_{k=1}^{d} a^{\ell_i^{kx}(j-1)+q_j^k} - a^{\ell_i^{kx}(j-1)}$. Hence, for any i', $\sum_{k=1}^{d}(a^{\ell_i^{kx}(f)} - 1) = \sum_{j\in J_i}\Delta_i(j) \leq \sum_{j\in J_i}\Delta_{i'}(j) \leq \sum_{k=1}^{d} a^{\ell_{i'}^{kx}(f)}\sum_{j\in J_i}(a^{q_j^k} - 1)$. Since $p_j^k \leq \Lambda$, we have $0 \leq q_j^k \leq 1$ and thus $\gamma(a^{q_j^k} - 1) \leq q_j^k$. This allows us to bound $\sum_{k=1}^{d} a^{\ell_{i'}^{kx}(f)}\sum_{j\in J_i}(a^{q_j^k} - 1)$ by

$$\sum_{k=1}^{d} a^{\ell_{i'}^{kx}(f)}\sum_{j\in J_i}\frac{q_j^k}{\gamma} = \sum_{k=1}^{d} a^{\ell_{i'}^{kx}(f)}\frac{\ell_i^{kx}(f)}{\gamma} \leq \frac{\mu^x}{\gamma}\sum_{k=1}^{d} a^{\ell_{i'}^{kx}(f)}.$$

Putting our inequalities together and rearranging terms finishes the proof. □

Theorem 3.10. *Algorithm 3.2 is $O(\log d)$-competitive on the makespan.*

Proof. Fix phase x and let s and s' be the machines with maximal and minimal $\sum_{k=1}^{d} a^{\ell_i^{kx}}$ (respectively) at the end of phase x. Combining Lemma 3.7 and Lemma 3.8 gives

$$a^{\mu^x} \leq \sum_{k=1}^{d} a^{\ell_s^{kx}(f)} \leq d + \frac{\mu^x}{\gamma} \sum_{k=1}^{d} a^{\ell_{s'}^{kx}(f)} \leq d + \frac{\mu^x}{\gamma}\left(\frac{\gamma d}{\gamma - 1}\right) = d + \frac{\mu^x d}{\gamma - 1}.$$

Taking the logarithm of both sides gives $\mu^x \leq \log_a\left(\frac{d}{\gamma-1}\right) + \log_a(\mu^x + \gamma - 1)$. Thus, $\mu^x - \log_a(\mu^x+\gamma-1) = O(\log d)$. Note that if for some constant c we have $c - \log_a(c+\gamma-1) = \frac{c}{2}$, then for all $\mu^x \geq c$, we have $\mu^x = O(\log d)$. The makespan during phase x is $2^x O(\log d)$. For our last phase g, our total load is at most $2^{g+1}O(\log d)$. Since $g = \lceil \log(\max\{\Lambda_{max}, \frac{1}{m}\Lambda_{tot}\}) \rceil$ and $\max\{\Lambda_{max}, \frac{1}{m}\Lambda_{tot}\}$ is a lower bound on the optimal solution, we conclude that our algorithm is $O(\log d)$-competitive. □

Simultaneously Minimizing Energy

Given the value of Λ_{max} in advance, we compress all jobs onto a small number of machines, then gradually open up more machines as our estimate of Λ_{tot} increases. Algorithm 3.3 does this with virtual machines. As there are at most $2m$ virtual machines in total, we identify two virtual machines with each real machine. Theorem 3.12 establishes that advance knowledge of Λ_{max} (or some comparable advance knowledge) is necessary to have a competitive ratio independent of m.

Theorem 3.11. *Algorithm 3.3 is $O(\log d)$-competitive on both makespan and power.*

Proof. Within a call to Assign-Jobs, we guarantee that $\Lambda_{tot} \leq M\Lambda_{max}$. Thus, by Theorem 3.10, we place a load of at most $\Lambda_{max}O(\log d)$ on any virtual machine. Then, after all jobs are placed, the load of any real machine is at most the sum of the loads of two virtual machines (each at most $\Lambda_{max}O(\log d)$), plus any load placed by the last instance of Assign-Jobs when $M = m$. Thus, the makespan is at most $2\Lambda_{max}O(\log d) + \frac{1}{m}\Lambda_{tot}O(\log d)$.

```
Initialize M ← 1
while M < m do
    Run algorithm Assign-Jobs on M new virtual machines until Λ_tot > MΛ_max
    M ← min{2M, m}
while jobs are still arriving do
    Run algorithm Assign-Jobs on all real machines ignoring previous loads
```

Algorithm 3.3: Power-and-Makespan

We observe that at most $2M$ machines have non-zero load, so the total power is at most $4M\Lambda_{max}O(\log d) + 2\Lambda_{tot}O(\log d)$. The optimal power is Λ_{tot} and the algorithm guarantees $\Lambda_{tot} > \frac{M}{2}\Lambda_{max}$, which completes the proof.

□

Theorem 3.12. *Suppose we have an online algorithm that is α-competitive on the makespan and β-competitive on energy. Then $\beta \geq \frac{1}{2\alpha}\min(d, \log_{2\alpha} m)$.*

Proof. Jobs will arrive in $p = \min(d, \log_{2\alpha} m)$ phases. During phase i, we receive $\frac{m}{(2\alpha)^{i-1}}$ vectors each with $(2\alpha)^{i-1}$ in the i^{th} coordinate and zero elsewhere. After i phases, since vectors from different phases place load on different coordinates, the optimum makespan of $(2\alpha)^{i-1}$ can be achieved by assigning all jobs from any given phase to different machines. Then the algorithm must guarantee makespan at most $\alpha(2\alpha)^{i-1}$ at the end of phase i. It follows that we cannot place more than α of the phase i job vectors on the same machine, and thus at least $m_i \geq \frac{m}{\alpha(2\alpha)^{i-1}}$ machines received at least one phase i job.

The optimum energy is m, achieved by scheduling all jobs on one machine. However, for the algorithm to be β-competitive on energy, we need

$$\beta m \geq \sum_i m_i[(2\alpha)^{i-1} - (2\alpha)^{i-2}] \geq m\sum_i \frac{1}{2\alpha} \geq \frac{mp}{2\alpha}.$$

□

3.5 Concluding Remarks

This chapter studies online load balancing from an energy-aware perspective. Even a small improvement in energy efficiency can lead to significant savings, both monetary and environmental. Hence, we formalize the notion of multidimensionality, and model the fact that jobs may place a different load on machines along different dimensions (i.e., machine components). This is very useful in practice, as it has been observed that resource contention can lead to inefficient load balancing. In particular, jobs which place a large load on different dimensions can be parallelized in a power-efficient manner, whereas jobs which make heavy use of the same components hurt quality of service if placed together. Moreover, our results hold in the more practical online setting, which models the reality that systems are unaware of what jobs will need to be processed in the future.

To this end, we design algorithms for online load balancing in the multidimensional setting and measure our algorithm's performance by analyzing how well it competes against an adversary that knows the entire input in advance. For the Machine Activation problem, we design an algorithm that is $O(\log(md)\log(nm))$-competitive on the makespan objective and $O(d\log^2(nm))$-competitive on the activation cost objective. For the Vector Load Balancing problem, we design a single algorithm that is simultaneously $O(\log d)$-competitive on the makespan objective and $O(\log d)$-competitive on the energy objective. This is perhaps surprising, as these two objectives are at odds with each other. In particular, to do well on the makespan objective, it is important to spread jobs out on as many machines as possible, while to perform well on the energy objective, it is important to place jobs on as few machines as possible.

CHAPTER 4

Simultaneous Bounds on Competitiveness and Regret

4.1 Introduction

In an *online convex optimization* (OCO) problem, a learner interacts with an environment in a sequence of rounds. During each round t: (i) the learner must choose an action x^t from a convex decision space F; (ii) the environment then reveals a convex cost function c^t, and (iii) the learner experiences cost $c^t(x^t)$. The goal is to design learning algorithms that minimize the cost experienced over a (long) horizon of T rounds.

In this chapter, we study a generalization of online convex optimization that we term *smoothed online convex optimization* (SOCO). The only change in SOCO compared to OCO is that the cost experienced by the learner each round is now $c^t(x^t) + \|x^t - x^{t-1}\|$, where $\|\cdot\|$ is a seminorm (recall that a seminorm satisfies the axioms of a norm except that $\|x\| = 0$ does not imply $x = 0$). That is, the learner experiences a "smoothing cost" or "switching cost" associated with changing the action, in addition to the "operating cost" $c(\cdot)$.

Many applications typically modeled using online convex optimization have, in reality, some cost associated with a change of action. For example, switching costs in the k-armed bandit setting have received considerable attention [AT96, GM09]. Additionally, a strong motivation for studying SOCO comes from the recent developments in dynamic capacity provisioning algorithms for data centers [KK07, LWA11, LLW12, UUN11, LCA13, ZZZ12, YZH12], where the goal is to dynamically control the number and placement of active servers (x^t) in order to minimize a combination of the delay and energy costs (captured by c^t) and the switching costs involved in cycling servers into power saving modes and migrating data ($\|x^t - x^{t-1}\|$). Further, SOCO has applications even in contexts where there are no costs

associated with switching actions. For example, if there is concept drift in a penalized estimation problem, it is natural to make use of a regularizer (switching cost) term in order to control the speed of the drift of the estimator.

Two Communities, Two Performance Metrics

Though the precise formulation of SOCO does not appear to have been studied previously, there are two large bodies of literature on closely related problems: (i) the online convex optimization (OCO) literature within the machine learning community (e.g., [Zin03, HAK07]) and (ii) the metrical task system (MTS) literature within the algorithms community (e.g., [BLS92, MMS88]). We discuss these literatures in detail in Section 4.3. While there are several differences between the formulations in the two communities, a notable difference is that they focus on different performance metrics.

In the OCO literature, the goal is typically to minimize the ***regret***, which is the difference between the cost of the algorithm and the cost of the offline optimal static solution. In this context, a number of algorithms have been shown to provide sublinear regret (also called "no regret"). For example, online gradient descent can achieve $O(\sqrt{T})$ regret [Zin03]. Though such guarantees are proven only in the absence of switching costs, we show in Section 4.3 that the same regret bound also holds for SOCO.

In the MTS literature, the goal is typically to minimize the ***competitive ratio***, which is the maximum ratio between the cost of the algorithm and the cost of the offline optimal (dynamic) solution. In this setting, most results tend to be "negative" (e.g., when c^t is arbitrary, for any metric space the competitive ratio of an MTS algorithm with states chosen from that space grows without bound as the number of states grows [BLS92, BKR92]). However, these results still yield competitive ratios that do not increase with the number of tasks (i.e., with time). Throughout, we will neglect dependence of the competitive ratio on the number of states, and describe the competitive ratio as "constant" if it does not grow with time. Note also that positive results have emerged when the cost function and decision space are convex [LWA11].

Interestingly, the focus on different performance metrics in the OCO and MTS communities has led the communities to develop quite different styles of algorithms. The differences between the algorithms is highlighted by the fact that *all algorithms developed in the OCO community have poor competitive ratio and all algorithms developed in the MTS community have poor regret.*

However, it is natural to seek algorithms with both low regret and low competitive ratio. In learning theory, doing well for both corresponds to being able to learn both static and dynamic concepts well. In the design of a dynamic controller, low regret shows that the control is not more risky than static control, whereas low competitive ratio shows that the control is nearly as good as the best dynamic controller.

The first work to connect the two metrics was [BB00], who treat the special case where the switching costs are a fixed constant, instead of a norm. In this context, they show how to translate bounds on regret to bounds on the competitive ratio, and vice versa. A recent breakthrough was made in [BCN12], where a primal-dual approach was used to develop an algorithm that performs well for the "α-unfair competitive ratio," which is a hybrid of the competitive ratio and regret that provides comparison to the dynamic optimal when $\alpha = 1$ and to the static optimal when $\alpha = \infty$ (see Section 4.2). Their algorithm was not shown to perform well *simultaneously* for regret and the competitive ratio, but the result highlights the feasibility of unified approaches for algorithm design across competitive ratio and regret. There is also work on achieving simultaneous guarantees with respect to the static and dynamic optimal solutions in other settings (e.g., decision making on lists and trees [BCK02]), and there have been applications of algorithmic approaches from machine learning to MTS [BBK99, ABB10].

Summary of Contributions

This chapter explores the relationship between minimizing regret and minimizing the competitive ratio. To this end, we seek to answer the following question: "Can an algorithm simultaneously achieve both a constant competitive ratio and a sublinear regret?"

To answer this question, we show that *there is a fundamental incompatibility between regret and competitive ratio* — no algorithm can maintain both sublinear regret and a constant competitive ratio (Theorems 4.1, 4.2, and 4.3). This "incompatibility" does not stem from a pathological example: it holds even for the simple case when c^t is linear and x^t is scalar. Further, it holds for both deterministic and randomized algorithms and also when the α-unfair competitive ratio is considered.

Though providing both sublinear regret and a constant competitive ratio is impossible, we show that it is possible to "nearly" achieve this goal. We present an algorithm, *"Randomly Biased Greedy" (RBG), which achieves a competitive ratio of* $1+\gamma$ *while maintaining* $O(\max\{T/\gamma, \gamma\})$ *regret* for $\gamma \geq 1$ on one-dimensional action spaces. If γ can be chosen as a function of T, then this algorithm can balance between regret and the competitive ratio. For example, it can achieve sublinear regret while having an arbitrarily slowly growing competitive ratio, or it can achieve $O(\sqrt{T})$ regret while maintaining an $O(\sqrt{T})$ competitive ratio. Note that, unlike the scheme of [BCN12], this algorithm has a finite competitive ratio on continuous action spaces and provides a *simultaneous* guarantee on both regret and the competitive ratio.

4.2 Problem Formulation

An instance of smoothed online convex optimization (SOCO) consists of a convex decision or action space $F \subseteq (\mathbb{R}^+)^n$ and a sequence of cost functions $\{c^1, c^2, \dots\}$, where each function $c^t : F \to \mathbb{R}^+$. At each time t, a learner (i.e., algorithm) chooses an action vector $x^t \in F$ and the environment chooses a cost function c^t. Define the *α-penalized cost with lookahead i* for the sequence $\dots, x^t, c^t, x^{t+1}, c^{t+1}, \dots$ to be

$$C_i^\alpha(A, T) = E\left[\sum_{t=1}^{T} c^t(x^{t+i}) + \alpha\|x^{t+i} - x^{t+i-1}\|\right],$$

where $x^1, \dots, x^T$ are the decisions of algorithm A, the initial action is $x^i = 0$ without loss of generality, the expectation is over the randomness of the algorithm, and $\|\cdot\|$ is a seminorm on $\mathbb{R}^n$. The parameter T will usually be suppressed.

In the OCO and MTS literatures, the learners pay different special cases of this cost. In OCO, the algorithm "plays first" giving a 0-step lookahead and switching costs are ignored, yielding C_0^0. In MTS, the environment plays first giving the algorithm 1-step lookahead ($i = 1$), and uses $\alpha = 1$, yielding C_1^1. Note that we sometimes omit the superscript when $\alpha = 1$, and the subscript when $i = 0$.

One can relate the MTS and OCO costs by relating C_i^α to C_{i-1}^α, as done by [BB00] and [BCN12]. The penalty due to not having lookahead is

$$c^t(x^t) - c^t(x^{t+1}) \leq \nabla c^t(x^t)(x^t - x^{t+1}) \leq \|\nabla c^t(x^t)\|_2 \cdot \|x^t - x^{t+1}\|_2, \tag{4.1}$$

where $\|\cdot\|_2$ is the Euclidean norm. We adopt the assumption, common in the OCO literature, that $\|\nabla c^t(\cdot)\|_2$ are bounded on a given instance; which thus bounds the difference between the costs of MTS and OCO (with switching cost), C_1 and C_0.

Performance Metrics

The performance of a SOCO algorithm is typically evaluated by comparing its cost to that of an offline "optimal" solution, but the communities differ in their choice of benchmark, and whether to compare additively or multiplicatively.

The OCO literature typically compares against the optimal offline *static* action, i.e.,

$$OPT_s = \min_{x \in F} \sum_{t=1}^{T} c^t(x),$$

and evaluates the ***regret***, defined as the (additive) difference between the algorithm's cost and that of the optimal static action vector. Specifically, the regret $R_i(A)$ of Algorithm A with lookahead i on instances $\mathfrak{C}$, is less than $\rho(T)$ if for any sequence of cost functions $(c^1, \ldots, c^T) \in \mathfrak{C}^T$,

$$C_i^0(A) - OPT_s \leq \rho(T). \tag{4.2}$$

Note that for any problem and any $i \geq 1$, there exists an algorithm A for which $R_i(A)$ is non-positive; however, an algorithm that is not designed specifically to minimize regret may have $R_i(A) > 0$.

This traditional definition of regret omits switching costs and lookahead (i.e., $R_0(A)$). One can generalize regret to define $R'_i(A)$, by replacing $C_i^0(A)$ with $C_i^1(A)$ in (4.2). Specifically, $R'_i(A)$ is less than $\rho(T)$ if for any sequence of cost functions $(c^1, \ldots, c^T) \in \mathfrak{C}^T$,

$$C_i^1(A) - OPT_s \leq \rho(T). \tag{4.3}$$

Except where noted, we use the set $\mathfrak{C}^1$ of sequences of convex functions mapping $(\mathbb{R}^+)^n$ to $\mathbb{R}^+$ with (sub)gradient uniformly bounded over the sequence. Note that we do not require differentiability; throughout this chapter, references to gradients can be read as references to subgradients.

The MTS literature typically compares against the optimal offline (dynamic) solution,

$$OPT_d = \min_{x \in F^T} \sum_{t=1}^{T} c^t(x^t) + \|x^t - x^{t-1}\|,$$

and evaluates the ***competitive ratio***. The cost most commonly considered is C_1. More generally, we say the competitive ratio with lookahead i, denoted by $CR_i(A)$, is $\rho(T)$ if for any sequence of cost functions $(c^1, \ldots, c^T) \in \mathfrak{C}^T$,

$$C_i(A) \leq \rho(T) OPT_d + O(1). \tag{4.4}$$

Bridging Competitiveness and Regret

Many hybrid benchmarks have been proposed to bridge static and dynamic comparisons. For example, Adaptive-Regret [HS09] is the maximum regret over any interval, where the "static" optimum can differ for different intervals, and internal regret [BM05] compares the online policy against a simple perturbation of that policy. We adopt the static-dynamic hybrid proposed in the most closely related literature [BKR92, BB00, BCN12], the α***-unfair competitive ratio***, which we denote by $CR_i^\alpha(A)$ for lookahead i. For $\alpha \geq 1$, $CR_i^\alpha(A)$ is $\rho(T)$ if (4.4) holds with OPT_d replaced by

$$OPT_d^\alpha = \min_{x \in F^T} \sum_{t=1}^{T} c^t(x^t) + \alpha \|x^t - x^{t-1}\|.$$

Specifically, the α-unfair competitive ratio with lookahead i, $CR_i^\alpha(A)$, is $\rho(T)$ if for any sequence of cost functions $(c^1, \ldots, c^T) \in \mathfrak{C}^T$,

$$C_i(A) \leq \rho(T) OPT_d^\alpha + O(1). \tag{4.5}$$

For $\alpha = 1$, OPT_d^α is the dynamic optimal solution; as $\alpha \to \infty$, OPT_d^α approaches the static optimal solution.

To bridge the additive versus multiplicative comparisons used in the two literatures, we define the ***competitive difference***. The α-unfair competitive difference with lookahead i on instances $\mathfrak{C}$, $CD_i^\alpha(A)$, is $\rho(T)$ if for any sequence of cost functions $(c^1, \ldots, c^T) \in \mathfrak{C}^T$,

$$C_i(A) - OPT_d^\alpha \leq \rho(T). \tag{4.6}$$

This measure allows for a smooth transition between regret (when α is large enough) and an additive version of the competitive ratio when $\alpha = 1$.

4.3 Background

In the following, we briefly discuss related work on both online convex optimization and metrical task systems, to provide context for the results in this chapter.

Online Convex Optimization

The OCO problem has a rich history and a wide range of important applications. In computer science, OCO is perhaps most associated with the k-experts problem [HW98, LW94], a discrete-action version of online optimization wherein at each round t the learning algorithm must choose a number between 1 and k, which can be viewed as following the advice of one of k "experts." However, OCO also has a long history in other areas, such as portfolio management [Cov91, Cal08].

The identifying features of the OCO formulation are that (i) the typical performance metric considered is regret, (ii) switching costs are not considered, and (iii) the learner must act before the environment reveals the cost function. In our notation, the cost function

in the OCO literature is $C^0(A)$ and the performance metric is $R_0(A)$. Following [Zin03] and [HAK07], the typical assumptions are that the decision space F is non-empty, bounded and closed, and that the Euclidean norms of gradients $\|\nabla c^t(\cdot)\|_2$ are also bounded.

In this setting, a number of algorithms have been shown to achieve "no regret" (i.e., sublinear regret, $R_0(A) = o(T)$). An important example is *online gradient descent* (OGD), which is parameterized by learning rates η_t. OGD works as follows.

Algorithm 4.1 (Online Gradient Descent, OGD)**.** *Select arbitrary $x^1 \in F$. In time step $t \geq 1$, select $x^{t+1} = P(x^t - \eta_t \nabla c^t(x^t))$, where $P(y) = \arg\min_{x \in F} \|x - y\|_2$ is the projection under the Euclidean norm.*

With appropriate learning rates η_t, OGD achieves sublinear regret for OCO. In particular, the variant of [Zin03] uses $\eta_t = \Theta(1/\sqrt{t})$ and obtains $O(\sqrt{T})$ regret. Tighter bounds are possible in restricted settings. The work of [HAK07] achieves $O(\log T)$ regret by choosing $\eta_t = 1/(\gamma t)$ for settings when the cost function additionally is twice differentiable and has minimal curvature. That is, $\nabla^2 c^t(x) - \gamma I_n$ is positive semidefinite for all x and t, where I_n is the identity matrix of size n. In addition to algorithms based on gradient descent, more recent algorithms such as Online Newton Step and Follow the Approximate Leader [HAK07] also attain $O(\log T)$ regret bounds for a class of cost functions.

None of the work discussed above considers switching costs. To extend the literature discussed above from OCO to SOCO, we need to track the switching costs incurred by the algorithms. This leads to the following straightforward result.

Proposition 4.1. *Consider an online gradient descent algorithm A on a finite dimensional space with learning rates such that $\sum_{t=1}^{T} \eta_t = O(\rho_1(T))$. If $R_0(A) = O(\rho_2(T))$, then $R_0'(A) = O(\rho_1(T) + \rho_2(T))$.*

Proof. Recall that, by assumption, $\|\nabla c^t(\cdot)\|_2$ is bounded. So, let us define D such that $\|\nabla c^t(\cdot)\|_2 \leq D$. Next, due to the fact that all norms are equivalent in a finite dimensional space, there exist $m, M > 0$ such that for every x, $m\|x\|_a \leq \|x\|_b \leq M\|x\|_a$. Combining

these facts, we can bound the switching cost incurred by an OGD algorithm as follows:

$$\sum_{t=1}^{T} \|x^t - x^{t-1}\| \leq M \sum_{t=1}^{T} \|x^t - x^{t-1}\|_2 \leq M \sum_{t=1}^{T} \eta_t \|\nabla c^t(\cdot)\|_2 \leq MD \sum_{t=1}^{T} \eta_t.$$

The second inequality comes from the fact that projection to a convex set under the Euclidean norm is non-expansive (i.e., $\|P(x) - P(y)\|_2 \leq \|x - y\|_2$). Thus, the switching cost causes an additional regret of $\sum_{t=1}^{T} \eta_t = O(\rho_1(T))$ for the algorithm, completing the proof. □

Interestingly, the choices of η_t used by the algorithms designed for OCO also turn out to be good choices to control the switching costs of the algorithms. The algorithms of [Zin03] and [HAK07], which use $\eta_t = 1/\sqrt{t}$ and $\eta_t = 1/(\gamma t)$, still have $O(\sqrt{T})$ regret and $O(\log T)$ regret respectively when switching costs are considered, since in these cases $\rho_1(T) = O(\rho_2(T))$. Note that a similar result can be obtained for Online Newton Step [HAK07].

Importantly, though the regret of OGD algorithms is sublinear, it can easily be shown that the competitive ratio of these algorithms is unbounded.

Metrical Task Systems

Like OCO, MTS also has a rich history and a wide range of important applications. Historically, MTS is perhaps most associated with the k-server problem [CMP08]. In this problem, there are k servers, each in some state, and a sequence of requests is incrementally revealed. To serve a request, the system must move one of the servers to the state necessary to serve the request, which incurs a cost that depends on the source and destination states.

The formulation of SOCO in Section 4.2 is actually, in many ways, a special case of the most general MTS formulation. In general, the MTS formulation differs in that (i) the cost functions c^t are not assumed to be convex, (ii) the decision space is typically assumed to be discrete and is not necessarily embedded in a vector space, and (iii) the switching cost is an arbitrary metric $d(x^t, x^{t-1})$ rather than a seminorm $\|x^t - x^{t-1}\|$. In this context, the cost function studied by MTS is typically C_1 and the performance metric of interest is the competitive ratio, specifically $CR_1(A)$, although the α-unfair competitive ratio CR_1^α also receives attention.

The weakening of the assumptions on the cost functions, and the fact that the competitive ratio uses the dynamic optimum as the benchmark, means that most of the results in the MTS setting are "negative" when compared with those for OCO. In particular, it has been proven that, given an arbitrary metric decision space of size n, any deterministic algorithm must be $\Omega(n)$-competitive [BLS92]. Further, any randomized algorithm must be $\Omega(\sqrt{\log n / \log\log n})$-competitive [BKR92].

These results motivate imposing additional structure on the cost functions in order to attain positive results. For example, it is commonly assumed that the metric is the uniform metric, in which $d(x, y)$ is equal for all $x \neq y$; this assumption was made by [BB00] in a study of the tradeoff between competitive ratio and regret. For comparison with OCO, an alternative natural restriction is to impose convexity assumptions on the cost function and the decision space, as done in this work.

Upon restricting c^t to be convex, F to be convex, and $\|\cdot\|$ to be a seminorm, the MTS formulation becomes quite similar to the SOCO formulation. This restricted class has been the focus of a number of recent papers, and some positive results have emerged. For example, [LWA11] show that when F is a one-dimensional normed space, a deterministic online algorithm called Lazy Capacity Provisioning (LCP) is 3-competitive. Note that we need only consider the absolute value norm for such spaces, since for every seminorm $\|\cdot\|$ on $\mathbb{R}$, $\|x\| = \|1\||x|$.

Importantly, though the algorithms described above provide constant competitive ratios, in all cases it is easy to see that the regret of these algorithms is linear.

4.4 The Incompatibility of Regret and the Competitive Ratio

As noted in the introduction, there is considerable motivation to perform well for the concepts of regret and competitive ratio simultaneously. See also [BKR92, BB00, BCN12, HS09, BM05]. None of the algorithms discussed so far achieves this goal. For example, Online Gradient Descent has sublinear regret but its competitive ratio is infinite. Similarly, Lazy Capacity Provisioning is 3-competitive but has linear regret.

This is no accident. We show below that the two goals are fundamentally incompatible: any algorithm that has sublinear regret for OCO necessarily has an infinite competitive ratio for MTS; and any algorithm that has a constant competitive ratio for MTS necessarily has at least linear regret for OCO. Further, our results give lower bounds on the simultaneous guarantees that are possible.

In discussing this "incompatibility," there are some subtleties that arise due to differences in formulation between the OCO literature, where regret is the focus, and the MTS literature, where competitive ratio is the focus. In particular, there are four key differences which are important to highlight: (i) OCO uses lookahead $i = 0$ while MTS uses $i = 1$; (ii) OCO does not consider switching costs ($\alpha = 0$) while MTS does ($\alpha = 1$); (iii) regret uses an additive comparison while the competitive ratio uses a multiplicative comparison; and (iv) regret compares to the static optimal solution while competitive ratio compares to the dynamic optimal solution. Note that the first two are intrinsic to the costs, while the latter are intrinsic to the performance metric. The following results tease apart which of these differences create incompatibility and which do not. In particular, we prove that (i) and (iv) each create incompatibilities.

Our first result in this section states that there is an incompatibility between regret in the OCO setting and the competitive ratio in the MTS setting (i.e., between the two most commonly studied measures $R_0(A)$ and $CR_1(A)$). Naturally, the incompatibility remains if switching costs are added to regret, where $R'_0(A)$ is considered. Further, the incompatibility remains when the competitive difference is considered, and so both the comparison with the static optimal solution and the dynamic optimal solution are additive. In fact, the incompatibility remains even when the α-unfair competitive ratio and difference is considered. Perhaps most surprisingly, the incompatibility remains when there is lookahead, where C_i and C_{i+1} are considered.

Theorem 4.1. *Consider an arbitrary seminorm $\|\cdot\|$ on $\mathbb{R}^n$, constants $\gamma > 0$, $\alpha \geq 1$ and $i \in \mathbb{N}$. There is a $\mathfrak{C}$ containing a single sequence of cost functions such that, for all deterministic and randomized algorithms A, either $R_i(A) = \Omega(T)$ or, for large enough T, both $CR^{\alpha}_{i+1}(A) \geq \gamma$ and $CD^{\alpha}_{i+1}(A) \geq \gamma T$.*

The incompatibility arises even in "simple" instances; the proof of Theorem 4.1 uses linear cost functions and a one-dimensional decision space, and the construction of the cost functions does not depend on T or A.

The cost functions used by regret and the competitive ratio in Theorem 4.1 are "off by one," motivated by the different settings in OCO and MTS. However, the following shows that parallel results also hold when the cost functions are not "off by one," namely for $R_0(A)$ versus $CR_0^\alpha(A)$ and $R_1'(A)$ versus $CR_1^\alpha(A)$.

Theorem 4.2. *Consider an arbitrary seminorm $\|\cdot\|$ on $\mathbb{R}^n$, constants $\gamma > 0$ and $\alpha \geq 1$, and a deterministic or randomized online algorithm A. There is a $\mathfrak{C}$ containing two cost functions such that either $R_0(A) = \Omega(T)$ or, for large enough T, both $CR_0^\alpha(A) \geq \gamma$ and $CD_0^\alpha(A) \geq \gamma T$.*

Theorem 4.3. *Consider an arbitrary norm $\|\cdot\|$ on $\mathbb{R}^n$. There is a $\mathfrak{C}$ containing two cost functions such that, for any constants $\gamma > 0$ and $\alpha \geq 1$, and any deterministic or randomized online algorithm A, either $R_1'(A) = \Omega(T)$ or, for large enough T, $CR_1^\alpha(A) \geq \gamma$.*

The impact of these results can be stark. It is impossible for an algorithm to learn static concepts with sublinear regret in the OCO setting, while having a constant competitive ratio for learning dynamic concepts in the MTS setting. More strikingly, in control theory, any dynamic controller that has a constant competitive ratio must have at least linear regret, and so there are cases where it does much worse than the best static controller. Thus, one cannot simultaneously guarantee the dynamic policy is always as good as the best static policy and is nearly as good as the optimal dynamic policy.

Theorem 4.3 is perhaps the most interesting of these results. Theorem 4.1 is due to seeking to minimize different cost functions (c^t and c^{t+1}), while Theorem 4.2 is due to the hardness of attaining a small CR_0^α (i.e., mimicking the dynamic optimal solution without 1-step lookahead). In contrast, for Theorem 4.3, algorithms exist with strong performance guarantees for each measure individually, and the measures are aligned in time. However, Theorem 4.3 must consider the (non-standard) notion of regret that includes switching costs (i.e., R'), since otherwise the problem is trivial.

Proofs

We now prove the results above. We use one-dimensional examples; however these examples can easily be embedded into higher dimensions if desired. We show proofs only for competitive ratio; the proofs for competitive difference are similar.

Let $\bar{\alpha} = \max(1, \|\alpha\|)$. Given $a > 0$ and $b \geq 0$, define two possible cost functions on $F = [0, 1/\bar{\alpha}]$: $f_1^\alpha(x) = b + ax\bar{\alpha}$ and $f_2^\alpha(x) = b + a(1 - x\bar{\alpha})$. These functions are similar to those used by [GBZ12] for studying online gradient descent to learn a concept of bounded total variation. To simplify notation, let $D(t) = 1/2 - E[x^t]\,\bar{\alpha}$, and note that $D(t) \in [-1/2, 1/2]$.

Proof of Theorem 4.1

To prove Theorem 4.1, we prove the stronger claim that $CR_{i+1}^\alpha(A) + R_i(A)/T \geq \gamma$.

Consider a system with costs $c^t = f_1^\alpha$ if t is odd and f_2^α if t is even. Then $C_i(A) \geq (a/2 + b)T + a\sum_{t=1}^T (-1)^t D(t+i)$. The static optimum is not worse than the scheme that sets $x^t = 1/(2\bar{\alpha})$ for all t, which has total cost no more than $(a/2+b)T + \|1/2\|$. The α-unfair dynamic optimum for C_{i+1} is not worse than the scheme that sets $x^t = 0$ if t is odd and $x^t = 1/\bar{\alpha}$ if t is even, which has total α-unfair cost at most $(b+1)T$. Hence

$$\begin{aligned} R_i(A) \geq & a\sum_{t=1}^T (-1)^t D(t+i) - \|1/2\|, \\ CR_{i+1}^\alpha(A) \geq & \frac{(a/2+b)T + a\sum_{t=1}^T (-1)^t D(t+i+1)}{(b+1)T}. \end{aligned}$$

Thus, since $D(t) \in [-1/2, 1/2]$,

$$\begin{aligned} & (b+1)T(CR_{i+1}^\alpha(A) + R_i(A)/T) + (b+1)\|1/2\| - (a/2+b)T \\ & \geq a\sum_{t=1}^T (-1)^t (D(t+i+1) + (b+1)D(t+i)) \\ & = ab\sum_{t=1}^T (-1)^t D(t+i) - a\left(D(i+1) + (-1)^T D(T+i+1)\right) \geq -abT/2 - a. \end{aligned}$$

To establish the claim, it is then sufficient that $(a/2+b)T - (b+1)\|1/2\| - abT/2 - a \geq \gamma T(b+1)$. For $b = 1/2$ and $a = 30\gamma + 2 + \|6\|$, this holds for $T \geq 5$.

Proof of Theorem 4.2

To prove Theorem 4.2, we again prove the stronger claim $CR_0^\alpha(A) + R_0(A)/T \geq \gamma$.

Consider the cost function sequence over the decision space $[0, 1]$ with $c^t(\cdot) = f_2^0$ for $E[x^t] \leq 1/2$ and $c^t(\cdot) = f_1^0$ otherwise, where x^t is the (random) choice of the algorithm at round t. Here, the expectation is taken over the marginal distribution of x^t conditioned on $c_1, \ldots, c_{t-1}$, averaging out the dependence on the realizations of $x_1, \ldots, x_{t-1}$. Notice that this sequence can be constructed by an oblivious adversary before the execution of the algorithm.

We now prove the following lemma.

Lemma 4.1. *Given any algorithm, the sequence of cost functions chosen by the above oblivious adversary makes*

$$R_0(A), R_0'(A) \geq a \sum_{t=1}^{T} |1/2 - E\left[x^t\right]| - \|1/2\|, \tag{4.7}$$

$$CR_0^\alpha(A) \geq \frac{(a/2 + b)T + a\sum_{t=1}^{T} |1/2 - E[x^t]|}{(b + \|\alpha\|)T}. \tag{4.8}$$

Proof. Recall that the oblivious adversary chooses $c^t(\cdot) = f_2^0$ for $E[x^t] \leq 1/2$ and $c^t(\cdot) = f_1^0$ otherwise, where x^t is the (random) choice of the algorithm at round t. Therefore,

$$\begin{aligned}
C_0(A) &\geq E\left[\sum_{t=1}^{T} \begin{cases} a(1 - x^t) + b & \text{if } E[x^t] \leq 1/2 \\ ax^t + b & \text{otherwise} \end{cases}\right] \\
&= E\left[bT + a\sum_{t=1}^{T} \left(1/2 + (1/2 - x^t)\mathrm{sgn}(1/2 - E\left[x^t\right])\right)\right] \\
&= bT + a\sum_{t=1}^{T} \left(1/2 + (1/2 - E\left[x^t\right])\mathrm{sgn}(1/2 - E\left[x^t\right])\right) \\
&= (a/2 + b)T + a\sum_{t=1}^{T} |1/2 - E\left[x^t\right]|,
\end{aligned}$$

where $\mathrm{sgn}(x) = 1$ if $x > 0$ and -1 otherwise. The static optimal solution is not worse than the scheme that sets $x^t = 1/2$ for all t, which has total cost $(a/2 + b)T + \|1/2\|$. This establishes (4.7).

The dynamic scheme which chooses $x^{t+1} = 0$ if $c^t = f_1^0$ and $x^{t+1} = 1$ if $c^t = f_2^0$ has total α-unfair cost not more than $(b + \|\alpha\|)T$. This establishes (4.8). $\square$

From (4.7) and (4.8) in Lemma 4.1, we have $CR_0^\alpha(A) + R_0(A)/T \geq \frac{(a/2+b)T}{(b+\|\alpha\|)T} - \frac{\|1/2\|}{T}$. For $a > 2\gamma(b+\|\alpha\|)$, the right hand side is bigger than γ for sufficiently large T, which establishes the theorem.

Proof of Theorem 4.3

Let $a = \|1\|/2$ and $b = 0$. Let $M = 4\alpha\gamma\|1\|/a = 8\alpha\gamma$. For $T \gg M$, divide $[1, T]$ into segments of length $3M$. For the last $2M$ of each segment, set $c^t = f_1^\alpha$. This ensures that the static optimal solution is $x = 0$. Moreover, if for all t in the first M time steps, c^t is either f_1^α or f_2^α, then the optimal dynamic solution is also $x^t = 0$ for the last $2M$ time steps.

Consider a solution on which each segment has non-negative regret. Then to obtain sublinear regret, for any positive threshold ϵ at least $T/(3M) - o(T)$ of these segments must have regret below $\epsilon\|1/\bar{\alpha}\|$. We will then show that these segments must have high competitive ratio. To make this more formal, consider the single segment $[1, 3M]$ without loss of generality.

Let $\tilde{c}$ be such that $\tilde{c}^t = f_2^\alpha$ for all $t \in [1, M]$ and $\tilde{c}^t = f_1^\alpha$ for $t > M$. Then the optimal dynamic solution on $[1, 3M]$ is $x_d^t = \mathbf{1}_{t \leq M}/\bar{\alpha}$, which has total cost $2\alpha\|1/\bar{\alpha}\|$ consisting entirely of switching costs.

We prove the following lemma.

Lemma 4.2. *For any $\delta \in (0, 1/\bar{\alpha})$ and integer $\tau > 0$, there exists an $\epsilon(\delta, \tau) > 0$ such that, if $c^t = f_2^\alpha$ for all $1 \leq t \leq \tau$ and $x^t > \delta$ for any $1 \leq t \leq \tau$, then there exists an $m \leq \tau$ such that $C_1(x, m) - C_1(OPT_s, m) > \epsilon(\delta, \tau)\|1/\bar{\alpha}\|$.*

Proof. We will consider only the case that $\bar{\alpha} = 1$; other cases are analogous. We prove the contrapositive (that if $C_1(x; m) - C_1(OPT_s, m) \leq \epsilon\|1\|$ for all m, then $x^t \leq \delta$ for all $t \in [1, \tau]$). We consider the case that x^t are non-decreasing; if not, the switching and operating costs can both be reduced by setting $(x^t)' = \max_{t' \leq t} x^{t'}$.

Note that OPT_s sets $x^t = 0$ for all t, so that $C_1(OPT_s, m) = am$, and

$$C_1(x; m) = x^m \|1\| - a \sum_{i=1}^{m} x^i + am.$$

Thus, we want to show that if $x^m\|1\| - a\sum_{i=1}^m x^i \le \epsilon$ for all $m \le \tau$, then we have $x^t < \delta$ for all $t \in [1, \tau]$.

Define $f_i(\cdot)$ inductively by $f_1(y) = 1/(1-y)$, and

$$f_i(y) = \frac{1}{1-y}\left(1 + y\sum_{j=1}^{i-1} f_j(y)\right).$$

If $y < 1$, then $\{f_i(y)\}$ are increasing in i.

Notice that $\{f_i\}$ satisfy

$$f_m(y)(1-y) - y\sum_{i=1}^{m-1} f_i(y) = 1.$$

Expanding the first term, we get that, for any $\hat{\epsilon}$,

$$\hat{\epsilon} f_m(a/\|1\|) - \frac{a}{\|1\|}\sum_{i=1}^{m} \hat{\epsilon} f_i(a/\|1\|) = \hat{\epsilon}. \tag{4.9}$$

If for some $\hat{\epsilon} > 0$,

$$x^m - \frac{a}{\|1\|}\sum_{j=1}^{m} x^j \le \hat{\epsilon} \tag{4.10}$$

for all $m \le \tau$, then by induction $x^i \le \hat{\epsilon} f_i(a/\|1\|) \le \hat{\epsilon} f_\tau(a/\|1\|)$ for all $i \le \tau$, where the last inequality uses the fact that $a < \|1\|$, and hence $\{f_i(a/\|1\|)\}$ are increasing in i.

The left hand side of (4.10) is $(C_1(x; m) - C_1(OPT_s, m))/\|1\|$. We define $\epsilon = \hat{\epsilon} = \delta/(2f_\tau(a/\|1\|))$. If $(C_1(x; m) - C_1(OPT_s, m)) \le \epsilon\|1\|$ for all m, then (4.10) holds for all m, and hence $x^t \le \hat{\epsilon} f_\tau(a/\|1\|) = \delta/2 < \delta$ for all $t \in [1, \tau]$. □

Let $\delta = 1/[5\bar{\alpha}] \in (0, 1)$. For any decisions such that $x^t < \delta$ for all $t \in [1, M]$, the operating cost of x under $\tilde{c}$ is at least $3\alpha\gamma\|1/\bar{\alpha}\|$. Let the adversary choose a c on this segment such that $c^t = f_2^\alpha$ until (a) the first time $t_0 < M$ that the algorithm's solution x satisfies $C_1(x, t_0) - C_1(OPT_s, t_0) > \epsilon(\delta, M)\|1/\bar{\alpha}\|$, or (b) $t = M$. After this point in time, it chooses $c^t = f_1^\alpha$.

In case (a), $C_1(x, 3M) - C_1(OPT_s, 3M) > \epsilon(\delta, M)\|1/\bar{\alpha}\|$ by Lemma 4.2, since OPT_s incurs no cost after t_0. Moreover, $C_1(x, 3M) \geq C_1(OPT_d, 3M)$.

In case (b), $C_1(x, 3M)/C_1(OPT_d, 3M) \geq 3\alpha\gamma\|1/\bar{\alpha}\|/(2\alpha\|1/\bar{\alpha}\|) = 3\gamma/2$.

To complete the argument, consider all segments. Let $g(T)$ be the number of segments for which case (a) occurs. The regret then satisfies

$$R'_1(A) \geq \epsilon(\delta, M)\|1/\bar{\alpha}\|g(T).$$

Similarly, the ratio of the total cost to that of the optimum cost is at least

$$\frac{C_1(x,T)}{C_1(OPT_d,T)} \geq \frac{[T/(3M) - g(T)]3\alpha\gamma\|1/\bar{\alpha}\|}{[T/(3M)]2\alpha\|1/\bar{\alpha}\|} = \frac{3}{2}\gamma\left(1 - \frac{3Mg(T)}{T}\right).$$

If $g(T) = \Omega(T)$, then $R'_1(A) = \Omega(T)$. Conversely, if $g(T) = o(T)$, then for sufficiently large T, $3Mg(T)/T < 1/3$, and so $CR_1^{\alpha}(A) > \gamma$.

4.5 Balancing Regret and the Competitive Ratio

Given the above incompatibility, it is necessary to reevaluate the goals for algorithm design. In particular, it is natural now to seek tradeoffs such as being able to obtain ϵT regret for arbitrarily small ϵ while remaining $O(1)$-competitive, or being $\log\log T$-competitive while retaining sublinear regret.

To this end, in the following we present a novel algorithm, Randomly Biased Greedy (RBG), which can achieve simultaneous bounds on regret R'_0 and competitive ratio CR_1, when the decision space F is one-dimensional. The one-dimensional setting is the natural starting point for seeking such a tradeoff given that the proofs of the incompatibility results all focus on one-dimensional examples and that the one-dimensional case has recently been of practical significance (e.g., [LWA11]). The algorithm takes a norm N as its input:

Algorithm 4.2 (Randomly Biased Greedy, RBG(N))**.**
Given a norm N, define $w^0(x) = N(x)$ for all x and $w^t(x) = \min_y\{w^{t-1}(y)+c^t(y)+N(x-y)\}$. Generate a random number r uniformly in $(-1, 1)$. For each time step t, go to the state x^t which minimizes $Y^t(x^t) = w^{t-1}(x^t) + rN(x^t)$.

RBG is motivated by [CMP08], and makes very limited use of randomness – it parameterizes its "bias" using a single random $r \in (-1, 1)$. It then chooses actions to greedily minimize its "work function" $w^t(x)$.

As stated, RBG performs well for the α-unfair competitive ratio, but performs poorly for regret. Theorem 4.4 will show that RBG($\|\cdot\|$) is 2-competitive, and hence has at best linear regret. Note that this guarantee improves the best known competitive ratio within this setting from 3 (achieved by Lazy Capacity Provisioning) to 2. However, the key idea behind balancing regret and competitive ratio is to run RBG with a "larger" norm to encourage its actions to change less. This can make the coefficient of regret arbitrarily small, at the expense of a larger (but still constant) competitive ratio.

Theorem 4.4. *For a SOCO problem in a one-dimensional normed space $\|\cdot\|$, running RBG(N) with a one-dimensional norm having $N(1) = \theta\|1\|$ as input (where $\theta \geq 1$) attains an α-unfair competitive ratio CR_1^α of $(1+\theta)/\min\{\theta, \alpha\}$ and a regret R_0' of $O(\max\{T/\theta, \theta\})$.*

Note that Theorem 4.4 holds for the usual metrics of MTS and OCO, which are the "most incompatible" case since the cost functions are mismatched (see Theorem 4.1). Thus, the conclusion of Theorem 4.4 still holds when R_0 or R_1 is considered in place of R_0'.

The best CR_1^α, $1 + 1/\alpha$, achieved by RBG is obtained with $N(\cdot) = \alpha\|\cdot\|$. However, choosing $N(\cdot) = \|\cdot\|/\epsilon$ for arbitrarily small ϵ gives ϵT regret, albeit larger CR_1^α. Similarly, if T is known in advance, choosing $N(1) = \theta(T)$ for some increasing function achieves an $O(\theta(T))$ α-unfair competitive ratio and $O(\max\{T/\theta(T), \theta(T)\})$ regret; taking $\theta(T) = O(\sqrt{T})$ gives $O(\sqrt{T})$ regret, which is optimal for arbitrary convex costs [Zin03]. If T is not known in advance, $N(1)$ can increase in t, and bounds similar to those in Theorem 4.4 still hold.

Proof of Theorem 4.4

To prove Theorem 4.4, we derive a more general tool for designing algorithms that simultaneously balance regret and the α-unfair competitive ratio. In particular, for any algorithm A, let the operating cost be $OC(A) = \sum_{t=1}^T c^t(x^{t+1})$ and the switching cost be $SC(A) = \sum_{t=1}^T \|x^{t+1} - x^t\|$, so that $C_1(A) = OC(A) + SC(A)$. Define OPT_N to be the

dynamic optimal solution under the norm $N(1) = \theta\|1\|$ ($\theta \geq 1$) with $\alpha = 1$.

Lemma 4.3. *Consider a one-dimensional SOCO problem with norm $\|\cdot\|$ and an online algorithm A which, when run with norm N, satisfies $OC(A(N)) \leq OPT_N + O(1)$ along with $SC(A(N)) \leq \beta OPT_N + O(1)$ with $\beta = O(1)$. Fix a norm N such that $N(1) = \theta\|1\|$ with $\theta \geq 1$. Then $A(N)$ has α-unfair competitive ratio $CR_1^\alpha(A(N)) = (1+\beta)\max\{\frac{\theta}{\alpha}, 1\}$ and regret $R_0'(A(N)) = O(\max\{\beta T, (1+\beta)\theta\})$ for the original SOCO problem with norm $\|\cdot\|$.*

Proof. We first prove the α-unfair competitive ratio result. Let $\hat{x}^1, \hat{x}^2, \ldots, \hat{x}^T$ denote the actions chosen by algorithm ALG when running on a normed space with $N = \|\cdot\|_{ALG}$ as input. Let $\hat{y}^1, \hat{y}^2, \ldots, \hat{y}^T$ be the actions chosen by the optimal dynamic offline algorithm, which pays α times more for switching costs, on a normed space with $\|\cdot\|$ (i.e., OPT_d^α). Similarly, let $\hat{z}^1, \hat{z}^2, \ldots, \hat{z}^T$ be the actions chosen by the optimal solution on a normed space with $\|\cdot\|_{ALG}$, namely $OPT_{\|\cdot\|_{ALG}}$ (without an unfairness cost). Recall that we have $C_1(ALG) = \sum_{t=1}^T c^t(\hat{x}^{t+1}) + \|\hat{x}^{t+1} - \hat{x}^t\|$, $OPT_d^\alpha = \sum_{t=1}^T c^t(\hat{y}^t) + \alpha\|\hat{y}^t - \hat{y}^{t-1}\|$, and $OPT_{\|\cdot\|_{ALG}} = \sum_{t=1}^T c^t(\hat{z}^t) + \|\hat{z}^t - \hat{z}^{t-1}\|_{ALG}$. By the assumptions in our lemma, we know that $C_1(ALG) \leq (1+\beta)OPT_{\|\cdot\|_{ALG}} + O(1)$. Moreover,

$$\begin{aligned} OPT_d^\alpha &= \sum_{t=1}^T c^t(\hat{y}^t) + \alpha\|\hat{y}^t - \hat{y}^{t-1}\| \\ &\geq \sum_{t=1}^T c^t(\hat{y}^t) + \frac{\alpha}{\theta}\|\hat{y}^t - \hat{y}^{t-1}\|_{ALG} \geq \frac{OPT_{\|\cdot\|_{ALG}}}{\max\{1, \frac{\theta}{\alpha}\}}. \end{aligned}$$

The first inequality holds since $\|\cdot\|_{ALG} = \theta\|\cdot\|$ with $\theta \geq 1$. Therefore, $C_1(ALG) \leq (1+\beta)\max\{1, \frac{\theta}{\alpha}\}OPT_d^\alpha$.

We now prove the regret bound. Let $d_{\max}$ denote the diameter of the decision space (i.e., the length of the interval). Recall that $C_0(ALG) = \sum_{t=1}^T c^t(\hat{x}^t) + \|\hat{x}^t - \hat{x}^{t-1}\|$ and $OPT_s = \min_x \sum_{t=1}^T c^t(x)$. Then we know that $C_0(ALG) \leq C_1(ALG) + D\sum_{t=1}^T \|x^{t+1} - x^t\| + \|d_{\max}\|$ for some constant D by (4.1). Based on our assumptions, we have $\sum_t c^t(\hat{x}^{t+1}) \leq OPT_{\|\cdot\|_{ALG}} + O(1)$ and $\sum_t \|\hat{x}^{t+1} - \hat{x}^t\| \leq \beta OPT_{\|\cdot\|_{ALG}} + O(1)$. For convenience, we let $E = D + 1 = O(1)$.

Then $C_0(ALG)$ is at most:

$$\sum_{t=1}^{T} c^t(\hat{x}^{t+1}) + E\|\hat{x}^{t+1} - \hat{x}^t\| + \|d_{\max}\| + O(1)$$

$$\leq (1 + E\beta)OPT_{\|\cdot\|_{ALG}} + \|d_{\max}\| + O(1)$$

$$\leq (1 + E\beta)(OPT_s + \|d_{\max}\|_{ALG}) + \|d_{\max}\| + O(1).$$

Therefore, we get a regret $C_0(ALG) - OPT_s$ of at most

$$E\beta OPT_s + \|d_{\max}\|(1 + E(1 + \beta)\theta) + O(1)$$

$$= O(\beta OPT_s + (1 + \beta)\theta) = O(\max\{\beta OPT_s, (1 + \beta)\theta\}).$$

In the OCO setting, the cost functions $c^t(x)$ are bounded from below by 0 and are typically bounded from above by a value independent of T [HW98, LW94], so that $OPT_s = O(T)$. This immediately gives the result that the regret is at most $O(\max\{\beta T, (1 + \beta)\theta\})$. □

Theorem 4.4 then follows from the following two lemmas.

Lemma 4.4. *Given a SOCO problem with norm $\|\cdot\|$, the RBG algorithm with input norm N satisfies $E[OC(RBG(N))] \leq OPT_N$.*

Proof. We argue that the expected operating cost of RBG (when evaluated under $\|\cdot\|$) with input norm $N(\cdot) = \theta\|\cdot\|$, $\theta \geq 1$, is at most the cost of the optimal dynamic offline algorithm under norm N (i.e., OPT_N). Let M denote our decision space, and let $\hat{x}^1, \hat{x}^2, \ldots, \hat{x}^{T+1}$ denote the actions chosen by RBG (similarly, let $x^1_{OPT}, \ldots, x^{T+1}_{OPT}$ denote the actions chosen by OPT_N). Before proving this result, we make the following observation: $w^t(\hat{x}^{t+1}) = w^{t-1}(\hat{x}^{t+1}) + c^t(\hat{x}^{t+1})$. To see why, we note that for any state $x \in M$, we have $w^t(x) = \min_{y \in M}\{w^{t-1}(y) + c^t(y) + \theta\|x - y\|\}$. Suppose instead $w^t(\hat{x}^{t+1}) = w^{t-1}(y) + c^t(y) + \theta\|\hat{x}^{t+1} - y\|$ for some $y \neq \hat{x}^{t+1}$. Then

$$Y^{t+1}(\hat{x}^{t+1}) = w^t(\hat{x}^{t+1}) + \theta r\|\hat{x}^{t+1}\|$$

$$= w^{t-1}(y) + c^t(y) + \theta\|\hat{x}^{t+1} - y\| + \theta r\|\hat{x}^{t+1}\|$$

$$> w^{t-1}(y) + c^t(y) + \theta r\|y\|$$

$$= Y^{t+1}(y),$$

which contradicts $\hat{x}^{t+1} = \arg\min_{y \in M} Y^{t+1}(y)$. Therefore, $w^t(\hat{x}^{t+1}) = w^{t-1}(\hat{x}^{t+1}) + c^t(\hat{x}^{t+1})$.

Now let us prove the expected operating cost of RBG is at most the total cost of the optimal solution, OPT_N.

$$\begin{aligned} Y^{t+1}(\hat{x}^{t+1}) - Y^t(\hat{x}^t) &\geq Y^{t+1}(\hat{x}^{t+1}) - Y^t(\hat{x}^{t+1}) \\ &= (w^t(\hat{x}^{t+1}) + \theta r \|\hat{x}^{t+1}\|) - (w^{t-1}(\hat{x}^{t+1}) + \theta r \|\hat{x}^{t+1}\|) \\ &= c^t(\hat{x}^{t+1}). \end{aligned}$$

The lemma follows by summing up the above inequality for $t = 1, \ldots, T$, since $Y^{T+1}(\hat{x}^{T+1}) \leq Y^{T+1}(x^{T+1}_{OPT})$ and $E\left[Y^{T+1}(x^{T+1}_{OPT})\right] = OPT_N$ by $E[r] = 0$.

Note that this approach also holds when the decision space $F \subset \mathbb{R}^n$ for $n > 1$. □

Lemma 4.5. *Given a one-dimensional SOCO problem with norm $\|\cdot\|$, the RBG algorithm with input norm N satisfies $E[SC(RBG(N))] \leq OPT_N/\theta$ with probability 1.*

To prove Lemma 4.5, we make a detour and consider a version of the problem with a discrete state space. We first show that on such spaces the lemma holds for a discretization of RBG, which we name DRBG. Next, we show that as the discretization becomes finer, the solution (and hence the switching cost) of DRBG approaches that of RBG. The lemma is then proven by showing that the optimal cost of the discrete approximation approaches the optimal cost of the continuous problem.

To begin, define a discrete variant of SOCO where the number of states is finite as follows. Actions can be chosen from m states, denoted by the set $M = \{x_1, \ldots, x_m\}$, and the distances $\delta = x_{i+1} - x_i$ are the same for all i. Without loss of generality we define $x_1 = 0$. Consider the following algorithm.

Algorithm 4.3 (Discrete RBG, DRBG(N))**.**
Given a norm N and discrete states $M = \{x_1, \ldots, x_m\}$, define $w^0(x) = N(x)$ and $w^t(x) = \min_{y \in M}\{w^{t-1}(y) + c^t(y) + N(x - y)\}$ for all $x \in M$. Generate a random number $r \in (-1, 1)$. For each time step t, go to the state x^t which minimizes $Y^t(x^t) = w^{t-1}(x^t) + rN(x^t)$.

Note that DRBG looks nearly identical to RBG except that the states are discrete. DRBG is introduced only for the proof and need never be implemented; thus we do not need to worry about the computational issues when the number of states m becomes large.

Bounding the Switching Cost of DRBG

We now argue that the expected switching cost of DRBG (evaluated under the norm $\|\cdot\|$ and run with input norm $N(\cdot) = \theta\|\cdot\|$) is at most the total cost of the optimal solution in the discrete system (under norm N). We first prove a couple of useful lemmas. The first lemma states that if the optimal way to get to some state x at time t is to come to state y in the previous time step, incur the operating cost at state y, and travel from y to x, then in fact the optimal way to get to state y at time t is to come to y at the previous time step and incur the operating cost at state y.

Lemma 4.6. *If* $\exists x, y : w^t(x) = w^{t-1}(y) + c^t(y) + \theta\|x - y\|$, *then* $w^t(y) = w^{t-1}(y) + c^t(y)$.

Proof. Suppose towards a contradiction that $w^t(y) < w^{t-1}(y) + c^t(y)$. Then we have:

$$\begin{aligned} w^t(y) + \theta\|x - y\| &< w^{t-1}(y) + c^t(y) + \theta\|x - y\| \\ &= w^t(x) \leq w^t(y) + \theta\|x - y\| \end{aligned}$$

(since one way to get to state x at time t is to get to state y at time t and then travel from y to x). This is a contradiction, which proves the lemma. □

Let $SC^t = \sum_{i=1}^{t} \|x^i - x^{i-1}\|$ denote the total switching cost incurred by DRBG up until time t, and define the potential function $\phi^t = \frac{1}{2\theta}(w^t(x_1) + w^t(x_m)) - \frac{\|x_m - x_1\|}{2}$. Then we can show the following lemma.

Lemma 4.7. *For every time step* t, $E[SC^t] \leq \phi^t$.

Proof. We will prove this lemma by induction on t. At time $t = 0$, clearly it is true since the left hand side $E[SC^0] = 0$, while the right hand side $\phi^0 = \frac{1}{2\theta}(w^0(x_1) + w^0(x_m)) - \frac{\|x_m - x_1\|}{2} = \frac{1}{2\theta}(0 + \theta\|x_m - x_1\|) - \frac{\|x_m - x_1\|}{2} = 0$. We now argue that at each time step, the increase in the left hand side is at most the increase in the right hand side.

Since the operating cost is convex, it is non-increasing until some point x_{min} and then non-decreasing over the set M. We can imagine our cost vector arriving in ϵ-sized increments as follows. We imagine sorting the cost values so that $c^t(i_1) \leq c^t(i_2) \leq \cdots \leq c^t(i_m)$, and then view time step t as a series of smaller time steps where we apply a cost of ϵ to all states for the first $c^t(i_1)/\epsilon$ time steps, followed by applying a cost of ϵ to all states except state i_1 for the next $(c^t(i_2) - c^t(i_1))/\epsilon$ time steps, etc., where ϵ has a very small value. If adding this epsilon-sized cost vector would cause us to exceed the original cost $c^t(i_k)$ for some k, then we just use the residual $\epsilon' < \epsilon$ in the last round in which state i_k has non-zero cost. Eventually, these ϵ-sized cost vectors will add up precisely to the original cost vector c^t.

Under these new cost vectors, the behavior of our algorithm will not change (and the optimal solution cannot get worse). Moreover, we would never move to a state in which ϵ cost was added. If the left hand side does not increase at all from time step $t-1$ to t, then the lemma holds (since the right hand side can only increase). Our expected switching cost is the probability that the algorithm moves multiplied by the distance moved. Suppose the algorithm is currently in state x. Observe that there is only one state the algorithm could be moving from (state x) and only one state y the algorithm could be moving to (we can choose ϵ to be sufficiently small in order to guarantee this). Notice that the algorithm would only move to a state y to which no cost was added. First, we consider the case $x \geq x_{min}$.

The only reason we would move from state x is if $w^t(x)$ increases from the previous time step, so that we have $w^t(x) = w^{t-1}(x) + \epsilon$. Notice that for any state $z > x$, we must have $w^t(z) = w^{t-1}(z) + \epsilon$. If not (i.e., $w^t(z) < w^{t-1}(z) + \epsilon$), then we get a contradiction as follows. The optimal way to get to z at time step t, $w^t(z)$, must go through some point j in the previous time step and incur the operating cost at j. If $j \geq x$, then we know $w^{t-1}(j) + \epsilon + \theta\|z - j\| = w^t(z) < w^{t-1}(z) + \epsilon \leq w^{t-1}(j) + \theta\|z - j\| + \epsilon$, which cannot happen. On the other hand, by Lemma 4.6, if $j < x$ then we get $w^t(x) + \theta\|z - x\| \leq w^t(j) + \theta\|1\|\|x - j| + \theta\|1\|\|z - x| = w^t(j) + \theta\|z - j\| = w^{t-1}(j) + c^t(j) + \theta\|z - j\| = w^t(z) < w^{t-1}(z) + \epsilon \leq w^{t-1}(x) + \theta\|z - x\| + \epsilon$, which cannot happen either. Hence, we know $w^t(z) = w^{t-1}(z) + \epsilon$ for all $z \geq x$.

By the above argument, we can conclude a couple of facts. The state y we move to cannot

be such that $y \geq x$. Moreover, we also know that $w^t(x_m) = w^{t-1}(x_m) + \epsilon$ (since $x_m \geq x$). Notice that for us to move from state x to state y, the random value r must fall within a very specific range. In particular, we must have $Y^t(x) \leq Y^t(y)$ and $Y^{t+1}(y) \leq Y^{t+1}(x)$:

$$(w^{t-1}(x) + \theta r\|1\|x \leq w^{t-1}(y) + \theta r\|1\|y) \ \wedge \ (w^t(y) + \theta r\|1\|y \leq w^t(x) + \theta r\|1\|x)$$

$$\implies w^{t-1}(y) - w^{t-1}(x) - \epsilon \leq w^t(y) - w^t(x) \leq \theta r\|x - y\| \leq w^{t-1}(y) - w^{t-1}(x).$$

This means r must fall within an interval of length at most $\epsilon/\theta\|x - y\|$. Since r is chosen from an interval of length 2, this happens with probability at most $\epsilon/(2\theta\|x - y\|)$. Hence, the increase in our expected switching cost is at most $\|x - y\| \cdot \epsilon/(2\theta\|x - y\|) = \epsilon/2\theta$. On the other hand, the increase in the right hand side is $\phi^t - \phi^{t-1} = \frac{1}{2\theta}(w^t(x_1) - w^{t-1}(x_1) + w^t(x_m) - w^{t-1}(x_m)) \geq \epsilon/2\theta$ (since $w^t(x_m) = w^{t-1}(x_m) + \epsilon$). The case when $x < x_{min}$ is symmetric. This finishes the inductive claim. □

Now we prove the expected switching cost of DRBG is at most the total cost of the optimal solution for the discrete problem.

By Lemma 4.7, for all times t we have $E\left[SC^t\right] \leq \phi^t$. Denote by OPT^t the optimal solution at time t (so that $OPT^t = \min_x w^t(x)$ and $OPT^T = OPT_N$). Let $x^* = argmin_x w^t(x)$ be the final state which realizes OPT^t at time t. For all times t, we have:

$$\begin{aligned} E\left[SC^t\right] \leq \phi^t &= \frac{1}{2\theta}(w^t(x_1) + w^t(x_m)) - \frac{\|x_m - x_1\|}{2} \\ &\leq \frac{1}{2\theta}(w^t(x^*) + \theta\|x^* - x_1\| + w^t(x^*) + \theta\|x_m - x^*\|) - \frac{\|x_m - x_1\|}{2} \\ &= \frac{1}{\theta}OPT^t. \end{aligned}$$

In particular, the equation holds at time T, which gives the bound.

Convergence of DRBG to RBG

We are now going to show that if we keep splitting δ, the output of DRBG, which we denote by x_D^t, converges to the output of RBG, which we denote by x_C^t.

Lemma 4.8. *Consider a SOCO problem with $F = [x_L, x_H]$. Consider a sequence of discrete systems such that the state spacing $\delta \to 0$ and for each system, $[x_1, x_m] = F$. Let x_i denote*

the output of DRBG in the i^{th} discrete system, and $\hat{x}$ denote the output of RBG in the continuous system. Then the sequence $\{x_i\}$ converges to $\hat{x}$ with probability 1 as i increases. That is, for all t, $\lim_{i\to\infty} |x_i^t - \hat{x}^t| = 0$ with probability 1.

Proof. To prove the lemma, we just need to show that x_i converges pointwise to $\hat{x}$ with probability 1. For a given δ, let Y_D^t denote the function Y^t used by DRBG in the discrete system (with state space $M = \{x_1, \ldots, x_m\} \subset F$) and Y_C^t denote the function Y^t used by RBG in the continuous system (with state space F). The output of DRBG and RBG at time t are denoted by x_D^t and x_C^t respectively. The subsequence on which $|x_C^t - x_D^t| \leq 2\delta$ clearly has x_D^t converge to x_C^t. Now consider the subsequence on which this does not hold. For each such system, we can find a value $\bar{x}_C^t \in \{x_1, \ldots, x_m\}$ and $|\bar{x}_C^t - x_C^t| < \delta$ (and thus $|\bar{x}_C^t - x_D^t| \geq \delta$) such that $Y_C^t(\bar{x}_C^t) \leq Y_C^t(x_D^t)$, by the convexity of Y_C^t. Note that the minimum of a convex function over a convex set is convex, so w^t is a convex function by induction. Therefore, Y_C^t is convex as well. Moreover, $Y_D^t(x_D^t) \leq Y_D^t(\bar{x}_C^t)$ and $Y_C^t(x_D^t) \leq Y_D^t(x_D^t)$. So far, we have only rounded the t^{th} component. Now let us consider a scheme that will round to the set M all components $\tau < t$ of a solution to the continuous problem.

For an arbitrary trajectory $x = (x^t)_{t=1}^T$, define a sequence $x_R(x)$ with $x_R^\tau \in \{x_1, \ldots, x_m\}$ as follows. Let $l = \max\{k : x_k \leq x^\tau\}$. Set x_R^τ to x_l if $c^\tau(x_l) \leq c^\tau(x_{l+1})$ or $l = m$, and x_{l+1} otherwise. This rounding will increase the switching cost by at most $2\theta\|\delta\|$ for each timeslot. If $l = m$, then the operating cost is unchanged. Next, we bound the increase in operating cost when $l < m$.

For each timeslot τ, depending on the shape of c^τ on (x_l, x_{l+1}), we may have two cases: (i) c^τ is monotone; (ii) c^τ is non-monotone. In case (i), the rounding does not make the operating cost increase for this timeslot. Note that if $x_C^\tau \in \{x_L, x_H\}$, then for all sufficiently small values δ, case (ii) cannot occur, since the location of the minimum of c^τ is independent of δ. We now consider case (ii) with $x_C^\tau \in (x_L, x_H)$. Note that there must be a finite left and right derivative of c^τ at all points in (x_L, x_H) for c^τ to be finite on F. Hence, these derivatives must be bounded on any compact subset of (x_L, x_H). Since $x_C^\tau \in (x_L, x_H)$, there exists a set $[x'_L, x'_H] \subset (x_L, x_H)$ independent of δ such that, for sufficiently small δ, we have

$[x_l, x_{l+1}] \subset [x'_L, x'_H]$. Hence, there exists an H^τ such that, for sufficiently small δ, the gradient of c^τ is bounded by H^τ on $[x_l, x_{l+1}]$. Thus, for sufficiently small δ, the rounding will make the operating cost increase by at most $H^\tau\delta$ in timeslot τ.

Define $H = \max_\tau\{H^\tau\}$. If we apply this scheme to the trajectory which achieves $Y^t_C(\bar{x}^t_C)$, we get a decision sequence in the discrete system with $cost + r\theta\|\bar{x}^t_C\|$ not more than $Y^t_C(\bar{x}^t_C) + (H\delta + 2\theta\|\delta\|)t$ (by the foregoing bound on the increase in costs) and not less than $Y^t_D(\bar{x}^t_C)$ (because the solution of $Y^t_D(\bar{x}^t_C)$ minimizes $cost + r\theta\|\bar{x}^t_C\|$). Specifically, we have $Y^t_D(\bar{x}^t_C) \leq Y^t_C(\bar{x}^t_C) + (H\delta + 2\theta\|\delta\|)t$. Therefore,

$$Y^t_C(\bar{x}^t_C) \leq Y^t_C(x^t_i) = Y^t_C(x^t_D) \leq Y^t_D(x^t_D) \leq Y^t_D(\bar{x}^t_C) \leq Y^t_C(\bar{x}^t_C) + (H\delta + 2\theta\|\delta\|)t.$$

Notice that the gradient bound H is independent of δ and so $(H\delta + 2\theta\|\delta\|)t \to 0$ as $\delta \to 0$. Therefore, $|Y^t_C(x^t_i) - Y^t_C(\bar{x}^t_C)|$ converges to 0 as i increases.

Independent of the random choice r, the domain of $w^t_C(\cdot)$ can be divided into countably many non-singleton intervals on which $w^t_C(\cdot)$ is affine, joined by intervals on which it is strictly convex. Then $Y^t_C(\cdot)$ has a unique minimum unless $-r$ is equal to the slope of one of the former intervals, since $Y^t_C(\cdot)$ is convex. Hence, it has a unique minimum with probability one with respect to the choice of r.

Hence, with probability one, x^t_C is the unique minimum of Y^t_C. To see that $Y^t_C(\cdot)$ is continuous at any point a, apply the squeeze principle to the inequality $w^t_C(a) \leq w^t_C(x) + \theta\|x - a\| \leq w^t_C(a) + 2\theta\|x - a\|$, and note that $Y^t_C(\cdot)$ is $w^t(\cdot)$ plus a continuous function. The convergence of $|\bar{x}^t_C - x^t_C|$ then implies $|Y^t_C(\bar{x}^t_C) - Y^t_C(x^t_C)| \to 0$ and thus $|Y^t_C(x^t_i) - Y^t_C(x^t_C)| \to 0$, or equivalently $Y^t_C(x^t_i) \to Y^t_C(x^t_C)$. Note that the restriction of Y^t_C to $[x_L, x^t_C]$ has a well-defined inverse Y^{-1}, which is continuous at $Y^t_C(x^t_C)$. Hence, for the subsequence of x^t_i such that $x^t_i \leq x^t_C$, we have $x^t_i = Y^{-1}(Y^t_C(x^t_i)) \to Y^{-1}(Y^t_C(x^t_C)) = x^t_C$. Similarly, the subsequence such that $x^t_i \geq x^t_C$ also converges to x^t_C. □

Convergence of the Optimal Solution

To show that the competitive ratio holds for RBG, we also need to show that the optimal costs converge to those of the continuous system.

Lemma 4.9. *Consider a SOCO problem with $F = [x_L, x_H]$. Consider a sequence of discrete systems such that the state spacing $\delta \to 0$ and for each system, $[x_1, x_m] = F$. Let OPT_D^i denote the optimal cost in the i^{th} discrete system, and OPT_C denote the optimal cost in the continuous system (both under the norm N). Then the sequence $\{OPT_D^i\}$ converges to OPT_C as i increases. That is, $\lim_{i\to\infty} |OPT_D^i - OPT_C| = 0$.*

Proof. We can apply the same rounding scheme in the proof of Lemma 4.8 to the solution vector of OPT_C to get a discrete output with total cost bounded by $OPT_C + (H\delta + 2\theta\|\delta\|)T$, and thus we have $OPT_D^i \leq OPT_C + (H\delta + 2\theta\|\delta\|)T$. Notice that the gradient bound H is independent of δ and so $(H\delta + 2\theta\|\delta\|)T \to 0$ as $\delta \to 0$. Therefore, OPT_D^i converges to OPT_C as i increases. □

4.6 Concluding Remarks

This chapter studies the relationship between regret and competitive ratio when applied to the class of SOCO problems. It shows that these metrics, from the learning and algorithms communities respectively, are fundamentally incompatible, in the sense that algorithms with sublinear regret must have infinite competitive ratio, and those with constant competitive ratio have at least linear regret. Thus, the choice of performance measure significantly affects the style of algorithm designed. It also introduces a generic approach for balancing these competing metrics, exemplified by a specific algorithm, RBG.

There are many interesting directions that this work motivates. In particular, the SOCO formulation is still under-explored, and many variants of the formulation discussed here are still not understood. For example, is it possible to tradeoff regret and the competitive ratio in bandit versions of SOCO? More generally, the message from this chapter is that regret and the competitive ratio are incompatible within the formulation of SOCO. It is quite interesting to try to understand how generally this holds. For example, does the "incompatibility result" proven here extend to settings where the cost functions are random instead of adversarial (e.g., variations of SOCO such as k-armed bandit problems with switching costs)?

References

[AAA03] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. "The online set cover problem." In *Proceedings of the 35th annual ACM Symposium on Theory of Computing*, 2003.

[AAF93] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. "On-line load balancing with applications to machine scheduling and virtual circuit routing." In *Proceedings of the 25th annual ACM Symposium on Theory of Computing*, 1993.

[AAG95] Baruch Awerbuch, Yossi Azar, Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey S. Vitter. "Load balancing in the L_p norm." In *Proceedings of the 36th annual IEEE Symposium on Foundations of Computer Science*, 1995.

[ABB10] Jacob Abernethy, Peter L. Bartlett, Niv Buchbinder, and Isabelle Stanton. "A regularization approach to metrical task systems." In *Proceedings of the 21st International Conference on Algorithmic Learning Theory*, 2010.

[ABF13] Yossi Azar, Umang Bhaskar, Lisa Fleischer, and Debmalya Panigrahi. "Online mixed packing and covering." In *Proceedings of the 24th annual ACM-SIAM Symposium on Discrete Algorithms*, 2013.

[ABL13] Lachlan L. H. Andrew, Siddharth Barman, Katrina Ligett, Minghong Lin, Adam Meyerson, Alan Roytman, and Adam Wierman. "A tale of two metrics: simultaneous bounds on competitiveness and regret." In *Proceedings of the 26th annual ACM Conference on Learning Theory*, 2013.

[ACK13] Yossi Azar, Ilan R. Cohen, Seny Kamara, and Bruce Shepherd. "Tight bounds for online vector bin packing." In *Proceedings of the 45th annual ACM Symposium on Theory of Computing*, 2013.

[AGK01] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. "Local search heuristic for k-median and facility location problems." In *Proceedings of the 33rd annual ACM Symposium on Theory of Computing*, 2001.

[AJM09] Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. "Streaming k-means approximation." In *Proceedings of the 23rd annual Conference on Neural Information Processing Systems*, 2009.

[ARS98] Khaled Alsabti, Sanjay Ranka, and Vineet Singh. "An efficient k-means clustering algorithm." In *Proceedings of the 1st Workshop on High Performance Data Mining*, 1998.

[AT96] Manjari Asawa and Demosthenis Teneketzis. "Multi-armed bandits with switching penalties." *IEEE Transactions on Automatic Control*, **41**(3):328–348, 1996.

[AV06] David Arthur and Sergei Vassilvitskii. "How slow is the k-means method?" In *Proceedings of the 22nd annual ACM Symposium on Computational Geometry*, 2006.

[AV07] David Arthur and Sergei Vassilvitskii. "k-means++: the advantages of careful seeding." In *Proceedings of the 18th annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[Bal65] Geoffrey H. Ball. "Data analysis in the social sciences: what about the details?" In *Proceedings of the American Federation of Information Processing Societies Conference*, 1965.

[BB00] Avrim Blum and Carl Burch. "On-line learning and the metrical task system problem." *Machine Learning*, **39**(1):35–58, 2000.

[BBG09] Maria-Florina Balcan, Avrim Blum, and Anupam Gupta. "Approximate clustering without the approximation." In *Proceedings of the 20th annual ACM-SIAM Symposium on Discrete Algorithms*, 2009.

[BBK99] Avrim Blum, Carl Burch, and Adam Kalai. "Finely-competitive paging." In *Proceedings of the 40th annual IEEE Symposium on Foundations of Computer Science*, 1999.

[BCK02] Avrim Blum, Shuchi Chawla, and Adam Kalai. "Static optimality and dynamic search-optimality in lists and trees." In *Proceedings of the 13th annual ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[BCN12] Niv Buchbinder, Shahar Chen, Joseph Naor, and Ohad Shamir. "Unified algorithms for online learning and competitive analysis." In *Proceedings of the 25th annual ACM Conference on Learning Theory*, 2012.

[BCS06] Nikhil Bansal, Alberto Caprara, and Maxim Sviridenko. "Improved approximation algorithms for multidimensional bin packing problems." In *Proceedings of the 47th annual IEEE Symposium on Foundations of Computer Science*, 2006.

[BFK92] Yair Bartal, Amos Fiat, Howard Karloff, and Rakesh Vohra. "New algorithms for an ancient scheduling problem." In *Proceedings of the 24th annual ACM Symposium on Theory of Computing*, 1992.

[BH65] Geoffrey H. Ball and David J. Hall. "ISODATA: a novel method of data analysis and pattern classification." Technical report, Stanford Research Institute, 1965.

[BHI02] Mihai Bādoiu, Sariel Har-Peled, and Piotr Indyk. "Approximate clustering via core-sets." In *Proceedings of the 34th annual ACM Symposium on Theory of Computing*, 2002.

[BKR92] Avrim Blum, Howard Karloff, Yuval Rabani, and Michael Saks. "A decomposition theorem and bounds for randomized server problems." In *Proceedings of the 33rd annual IEEE Symposium on Foundations of Computer Science*, 1992.

[BLS92] Allan Borodin, Nathan Linial, and Michael Saks. "An optimal on-line algorithm for metrical task system." *Journal of the ACM*, **39**(4):745–763, 1992.

[BM05] Avrim Blum and Yishay Mansour. "From external to internal regret." In *Proceedings of the 18th annual ACM Conference on Learning Theory*, 2005.

[BMO11] Vladimir Braverman, Adam Meyerson, Rafail Ostrovsky, Alan Roytman, Michael Shindler, and Brian Tagiku. "Streaming k-means on well-clusterable data." In *Proceedings of the 22nd annual ACM-SIAM Symposium on Discrete Algorithms*, 2011.

[BN06] Niv Buchbinder and Joseph Naor. "Fair online load balancing." In *Proceedings of the 18th annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2006.

[Cal08] Giuseppe C. Calafiore. "Multi-period portfolio optimization with linear control policies." *Automatica*, **44**(10):2463–2473, 2008.

[Che09] Ke Chen. "On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications." *SIAM Journal on Computing*, **39**(3):923–947, 2009.

[CK99] Chandra Chekuri and Sanjeev Khanna. "On multi-dimensional packing problems." In *Proceedings of the 10th annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[CMP08] Aaron Coté, Adam Meyerson, and Laura Poplawski. "Randomized k-server on hierarchical binary trees." In *Proceedings of the 40th annual ACM Symposium on Theory of Computing*, 2008.

[COP03] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. "Better streaming algorithms for clustering problems." In *Proceedings of the 35th annual ACM Symposium on Theory of Computing*, 2003.

[Cov91] Thomas M. Cover. "Universal portfolios." *Mathematical Finance*, **1**(1):1–29, 1991.

[DG92] David DeWitt and Jim Gray. "Parallel database systems: the future of high performance database systems." *Communications of the ACM*, **35**(6):85–98, 1992.

[DLR77] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm." *Journal of the Royal Statistical Society, Series B (Methodological)*, **39**(1):1–38, 1977.

[Fis58] Walter D. Fisher. "On grouping for maximum homogeneity." *Journal of the American Statistical Association*, **53**(284):789–798, 1958.

[FMS07] Dan Feldman, Morteza Monemizadeh, and Christian Sohler. "A PTAS for k-means clustering based on weak coresets." In *Proceedings of the 23rd annual ACM Symposium on Computational Geometry*, 2007.

[For65] Edward Forgy. "Cluster analysis of multivariate data: efficiency vs. interpretability of classification." *Biometrics*, **21**:768–769, 1965.

[FS05] Gereon Frahling and Christian Sohler. "Coresets in dynamic geometric data streams." In *Proceedings of the 37th annual ACM Symposium on Theory of Computing*, 2005.

[GBZ12] Yonatan Gur, Omar Besbes, and Assaf Zeevi. "Non-stationary online stochastic optimization." In *INFORMS general meeting*, 2012.

[GG92] Allen Gersho and Robert M. Gray. *Vector quantization and signal compression.* Kluwer, 1992.

[GI95] Minos N. Garofalakis and Yannis E. Ioannidis. "Scheduling issues in multimedia query optimization." *ACM Computing Surveys*, **27**(4):590–592, 1995.

[GI96] Minos N. Garofalakis and Yannis E. Ioannidis. "Multi-dimensional resource scheduling for parallel queries." In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1996.

[GI97] Minos N. Garofalakis and Yannis E. Ioannidis. "Parallel query scheduling and optimization with time- and space-shared resources." In *Proceedings of the 23rd International Conference on Very Large Data Bases*, 1997.

[GM09] Sudipto Guha and Kamesh Munagala. "Multi-armed bandits with metric switching costs." In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming*. Springer, Berlin Heidelberg, 2009.

[GMM00] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. "Clustering data streams." In *Proceedings of the 41st annual IEEE Symposium on Foundations of Computer Science*, 2000.

[GMM03] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. "Clustering data streams: theory and practice." *IEEE Transactions on Knowledge and Data Engineering*, **15**(3):515–528, 2003.

[GN98] Robert M. Gray and David L. Neuhoff. "Quantization." *IEEE Transactions on Information Theory*, **44**(6):2325–2383, 1998.

[HAK07] Elad Hazan, Amit Agarwal, and Satyen Kale. "Logarithmic regret algorithms for online convex optimization." *Machine Learning*, **69**(2-3):169–192, 2007.

[HM04] Sariel Har-Peled and Soham Mazumdar. "On coresets for k-means and k-median clustering." In *Proceedings of the 36th annual ACM Symposium on Theory of Computing*, 2004.

[HS87] Dorit S. Hochbaum and David B. Shmoys. "Using dual approximation algorithms for scheduling problems: theoretical and practical results." *Journal of the ACM*, **34**(1):144–162, 1987.

[HS09] Elad Hazan and C. Seshadhri. "Efficient learning algorithms for changing environments." In *Proceedings of the 26th International Conference on Machine Learning*, 2009.

[HW98] Mark Herbster and Manfred K. Warmuth. "Tracking the best expert." *Machine Learning*, **32**(2):151–178, 1998.

[IP05] Sandy Irani and Kirk R. Pruhs. "Algorithmic problems in power management." *SIGACT News*, **36**(2):63–76, 2005.

[Jan66] Robert C. Jancey. "Multidimensional group analysis." *Australian Journal of Botany*, **14**(1):127–130, 1966.

[JL84] William B. Johnson and Joram Lindenstrauss. "Extensions of lipschitz maps into a hilbert space." *Contemporary Mathematics*, **26**:189–206, 1984.

[JMF99] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. "Data clustering: a review." *ACM Computing Surveys*, **31**(3):264–323, 1999.

[KK07] Dara Kusic and Nagarajan Kandasamy. "Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems." *Cluster Computing*, **10**(4):395–408, 2007.

[KLM84] Richard M. Karp, Michael Luby, and Alberto Marchetti-Spaccamela. "A probabilistic analysis of multidimensional bin packing problems." In *Proceedings of the 16th annual ACM Symposium on Theory of Computing*, 1984.

[KLS10] Samir Khuller, Jian Li, and Barna Saha. "Energy efficient scheduling via partial shutdown." In *Proceedings of the 21st annual ACM-SIAM Symposium on Discrete Algorithms*, 2010.

[KM77] Lawrence T. Kou and George Markowsky. "Multidimensional bin packing algorithms." *IBM Journal of Research and Development*, **21**(5):443–448, 1977.

[KMN02] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. "A local search approximation algorithm for k-means clustering." In *Proceedings of the 18th annual ACM Symposium on Computational Geometry*, 2002.

[LBG80] Yoseph Linde, Andrés Buzo, and Robert M. Gray. "An algorithm for vector quantizer design." *IEEE Transactions on Communications*, **28**(1):84–95, 1980.

[LCA13] Tan Lu, Minghua Chen, and Lachlan L. H. Andrew. "Simple and effective dynamic provisioning for power-proportional data centers." *IEEE Transactions on Parallel and Distributed Systems*, **24**(6):1161–1171, 2013.

[LCH07] Shih-Tang Lo, Ruey-Maw Chen, and Yueh-Min Huang. "Multi-constraint system scheduling using dynamic and delay ant colony system." In *Proceedings of the 20th International Conference on Industrial, Engineering, and Other Applications of Applied Intelligent Systems*. Springer, Berlin Heidelberg, 2007.

[LK11] Jian Li and Samir Khuller. "Generalized machine activation problems." In *Proceedings of the 22nd annual ACM-SIAM Symposium on Discrete Algorithms*, 2011.

[Llo82] Stuart Lloyd. "Least squares quantization in PCM." *IEEE Transactions on Information Theory*, **28**(2):129–137, 1982.

[LLW12] Minghong Lin, Zhenhua Liu, Adam Wierman, and Lachlan L. H. Andrew. "Online algorithms for geographical load balancing." In *Proceedings of the 3rd International Green Computing Conference*, 2012.

[LS07] Ron Lavi and Chaitanya Swamy. "Truthful mechanism design for multidimensional scheduling via cycle monotonicity." In *Proceedings of the 8th annual ACM Conference on Electronic Commerce*, 2007.

[LW94] Nick Littlestone and Manfred K. Warmuth. "The weighted majority algorithm." *Information and Computation*, **108**(2):212–261, 1994.

[LWA11] Minghong Lin, Adam Wierman, Lachlan L. H. Andrew, and Eno Thereska. "Dynamic right-sizing for power-proportional data centers." In *Proceedings of the 30th IEEE International Conference on Computer Communications*, 2011.

[Mac67] James MacQueen. "Some methods for classification and analysis of multivariate observations." In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[Max60] Joel Max. "Quantizing for minimum distortion." *IEEE Transactions on Information Theory*, **6**(1):7–12, 1960.

[MCR05] Justin Moore, Jeff Chase, Parthasarathy Ranganathan, and Ratnesh Sharma. "Making scheduling "cool": temperature-aware workload placement in data centers." In *Proceedings of the USENIX annual Technical Conference*, 2005.

[Mey01] Adam Meyerson. "Online facility location." In *Proceedings of the 42nd annual IEEE Symposium on Foundations of Computer Science*, 2001.

[MMS88] Mark Manasse, Lyle McGeoch, and Daniel Sleator. "Competitive algorithms for on-line problems." In *Proceedings of the 20th annual ACM Symposium on Theory of Computing*, 1988.

[MRT13] Adam Meyerson, Alan Roytman, and Brian Tagiku. "Online multidimensional load balancing." In *Proceedings of the 16th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*. Springer, Berlin Heidelberg, 2013.

[ORS06] Rafail Ostrovsky, Yuval Rabani, Leonard Schulman, and Chaitanya Swamy. "The effectiveness of Lloyd-type methods for the k-means problem." In *Proceedings of the 47th annual IEEE Symposium on Foundations of Computer Science*, 2006.

[Phi02] Steven J. Phillips. "Acceleration of k-means and related clustering problems." In *Proceedings of the 4th International Workshop on Algorithm Engineering and Experiments*, 2002.

[PM99] Dan Pelleg and Andrew Moore. "Accelerating exact k-means algorithms with geometric reasoning." In *Proceedings of the 5th ACM International Conference on Knowledge Discovery and Data Mining*, 1999.

[PST04] Kirk Pruhs, Jiří Sgall, and Eric Torng. "Online scheduling." *Handbook of scheduling: algorithms, models, and performance analysis*, 2004.

[RSM09] Sebi Ryffel, Thanos Stathopoulos, Dustin McIntire, William J. Kaiser, and Lothar Thiele. "Accurate energy attribution and accounting for multi-core systems." In *Technical Report 67, Center for Embedded Network Sensing*, 2009.

[RSR08] Mahsan Rofouei, Thanos Stathopoulos, Sebi Ryffel, William J. Kaiser, and Majid Sarrafzadeh. "Energy-aware high performance computing with graphic processing units." In *Proceedings of the USENIX Workshop on Power Aware Computing Systems*, 2008.

[SMK08] Thanos Stathopoulos, Dustin McIntire, and William J. Kaiser. "The energy endoscope: real-time detailed energy accounting for wireless sensor nodes." In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, 2008.

[ST93] David B. Shmoys and Éva Tardos. "An approximation algorithm for the generalized assignment problem." *Mathematical Programming*, **62**(3):461–474, 1993.

[Ste56] Hugo Steinhaus. "Sur la division des corp materiels en parties." *Bulletin of the Polish Academy of Sciences*, **4**:801–804, 1956.

[TB70] Robert C. Tryon and Daniel E. Bailey. *Cluster analysis.* McGraw-Hill, 1970.

[Tot59] László F. Tóth. "Sur la representation d'une population infinie par un nombre fini d'elements." *Acta Mathematica Academiae Scientiarum Hungaricae*, **10**(3-4):299–304, 1959.

[UUN11] Rahul Urgaonkar, Bhuvan Urgaonkar, Michael J. Neely, and Anand Sivasubramaniam. "Optimal power cost management using stored energy in data centers." In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems*, 2011.

[Vit85] Jeffrey S. Vitter. "Random sampling with a reservoir." *ACM Transactions on Mathematical Software*, **11**(1):37–57, 1985.

[YZH12] Jian Yang, Ke Zeng, Han Hu, and Hongsheng Xi. "Dynamic cluster reconfiguration for energy conservation in computation intensive service." *IEEE Transactions on Computers*, **61**(10):1401–1416, 2012.

[Zad64] Paul L. Zador. *Development and evaluation of procedures for quantizing multivariate distributions*. Ph.D. thesis, Stanford University, 1964.

[Zin03] Martin Zinkevich. "Online convex programming and generalized infinitesimal gradient ascent." In *Proceedings of the 20th International Conference on Machine Learning*, 2003.

[ZZZ12] Qi Zhang, Mohamed F. Zhani, Shuo Zhang, Quanyan Zhu, Raouf Boutaba, and Joseph L. Hellerstein. "Dynamic energy-aware capacity provisioning for cloud computing environments." In *Proceedings of the 9th International Conference on Autonomic Computing*, 2012.