Randomization in On-line Algorithms

Octavian Procopiuc

April 30, 1998

MOTTO: "[T]ime is unidirectional, past events cannot be reversed, the future is uncertain, and Murphy's Law rules" [39]

1 Introduction

The classical theory of algorithms studies problems in which the input data is given a priori and the algorithms are designed to take advantage of this knowledge for improved performance. In some applications, hewever, the algorithm has only partial information about the input data. This data is presented to the algorithm in pieces, as the computation proceeds, and the algorithm has to make decisions based only on the information it has at any given moment. The quality of the decision may of course be influenced by future inputs, that the algorithm has not seen yet.

Algorithms operating in the framework described above are called *on-line algorithms* (also spelled online algorithms). Although on-line algorithms have been developed for a long time, they were related to specific problems with an inherent on-line behavior, like paging in a two-level memory storage system. Due to its practical importance, this problem is still a prominent example of on-line computation, so we will use it as a running example. Consider a two-level memory hierarchy: a fast memory (cache) that can store $k$fixed-size memory pages, and a slower main memory of infinite size (virtual memory). A paging algorithm decides which k pages to keep in cache at each point in time. The input to the algorithm is a sequence of requests for memory pages. If a page is in the cache at the moment when it is requested, a *hit* is said to occur and the algorithm incurs no cost on that request. Otherwise, a *miss* occurs and the requsted page must be brought in from the main memory at a unit cost. Also, one of the pages in the cache must be evicted to make place for the new one. Thus, the cost of a paging algorithm on a given sequence of requests is the number

of misses incurred. This number depends on the algorithm employed for evicting a page on a miss. An on-line paging algorithm must dicide what page to evict without knowledge of future requests. In contrast, an *offline* algorithm makes this decision with complete knowledge of the future. Here are the most ubiquitous deterministic on-line algorithms for paging:

1. Least Recently Used (LRU): evict the page that has been requested least recently.

2. First-in, First-out (FIFO): evict the page that has been in the cache the longest.

3. Least Frequently Used (LFU): evict the page that has been requested least often.

The early attempts to study the performance of these and other algorithms for on-line problems were based on probabilistic analysis [34].Although they yielded useful comparisons between various on-line algorithms for a certain problem, they were prone to input skew, due to the conditions they imposed on the input data. To overcome this shortcoming, Sleator and Tarjan [60] proposed an alternative performance measure that would later be called *competitive analysis* [53, 54] (see Section 2.1). This has since become the de facto standard for studying on-line algorithms and is the common denominator for most of the research done in this area.

But even without competitive analysis, one could easily observe a common drawback of the three deterministic algorithms described above: auxiliary space needed for storing state information. Such state memory is expensive and slow to update in hardware [61]. As an alternative, Raghavan and Snir [57, 56] proposed using randomization. The idea is to make probabilistic choices instead of choices based on state information. It turns out that such randomized on-line algorithms not only use less additional space, but they often outperform the best deterministic algorithms in terms of *competitiveness* (see Section 2.1). Randomization has thus become an important tool in the development of better on-line algorithms.

The rest of this survey is organized as follows. Section 2 explains competitive analysis and lists other performance measures for on-line algorithms, Section 3 presents the framework for developing on-line randomized algorithms. The subsequent sections deal with various classes of problems for which on-line algorithms have been developed.

2 Performance Measures

2.1 Competitive analysis

In their seminal paper [60], Sleator and Tarjan studied the list update (see Section 5) and paging problems from the following viewpoint: the input is generated by an adversary, and the performance of the on-line algorithm on a sequence is compared with the performance of a benchmark algorithm on the same sequence. In this manner, there is no need to make assumptions on the input sequence, as was the case with probabilistic anaysis, and, moreover, we avoid the overly pessimistic and usually impractical statements yielded by worst-case analysis.

To make the presentation easier, let's consider again the paging problem and let $\rho = (\rho_1, \ldots, \rho_N)$ be a request sequence presented to an on-line paging algorithm A. Let $f_A(\rho)$ be the cost (i.e., the number of cache misses) incurred by running algorithm A on the input ρ and let $f_0(\rho)$ be the minimum cost (for an optimal off-line algorithm) on the same sequence.

The following terms were introduced by Karlin et al. [47].

Definition 2.1 *A deterministic on-line paging algorithm A is said to be* $\mathcal{C}$-competitive *if there exists a constant b such that on every sequence of requests ρ,*

$$f_A(\rho) - \mathcal{C} \times f_0(\rho) \leq b,$$

where the constant b must be independent of N but may depend on k, the size of the cache. The competitiveness ratio *of A, denoted $\mathcal{C}_A$, is the infimum of $\mathcal{C}$ such that A is $\mathcal{C}$-competitive.*

In the same paper in which they introduced this model, Sleator and Tarjan [60] proved that the **FIFO** and **LRU** algorithms for paging are k-competitive and that no deterministic on-line algorithm can do better. The best (offline) algorithm for paging is **MIN** [12], which evicts the page whose next request occurs furthest in the future. From the above considerations, we can conclude that, on any given input sequence, an on-line deterministic algorithm incurs in the worst case k times more misses than **MIN**, the optimal algorithm.

2.2 Beyond Competitive Analysis

Competitive analysis as a measure for the performance of on-line algorithms has been criticized even by its creators, Sleator and Tarjan [60], who complained that the competitive ratios for paging strategies are too high to be

of practical interest. Simulations show that, in practice, good paging strategies usually incur a cost within a small constant factor of the minimum [65]. Another criticism was that competitive analysis can be counterintuitive. Ben-David and Borodin [13] gave the following example, when we choose to abandon all competitive strategies in favor of a noncompetitive one. Consider buying an insurance policy. Paying \$5 a month to insure the car is a noncompetitive strategy, because there is a scenario where one will never present a claim to the insurance agent. The off-line strategy would have never bought the policy to start with.

A first refinement of the original model was proposed by Young [65]. He introduces loose competitiveness for the k-server problem (see Section 4.1. A k-server strategy is *loosely $c(k)$-competitive* if, for any sequence, for almost all k, the cost incurred by the strategy *either* is no more than $c(k)$ times the minimum cost or is insignificant. He proves that the Mark algorithm for paging is loosely $c(k)$-competitive provided that $(k - 2\ln\ln k) \to \infty$ and both $2\ln k - c(k)$ and $c(k)$ are nondecreasing.

A second model, introduced by Borodin et al. [18], is another refinement of the competitiveness model. It is devised for the paging problem and introduces a practical element that underlies the philosophy behind paging: locality of refernce. This notion is modeled using an *access graph*, which is known by the on-line algorithm. The vertices of the access graph correspond to the pages, and the edges reflect the locality of the pages. The competitiveness criterion is defined on sequences that are walks in the access graph. The authors conjectured that, in this model, the competitive ratio of LRU is less than or equal to the competitiveness ratio of FIFO, a behavior often observed in practice. Chrobak and Noga [25] later proved this conjecture.

Another model, which departs from the competitiveness model, was proposed by Ben-David and Borodin [13]. Their *Max/Max ratio* compares the worst case amortized behavior of an algorithm with that of an optimal off-line algorithm. This is a revival of the worst-case amortized analysis, except that now it is normalized by the best that can be done using an optimal off-line algorithm. They show that this measure gives rise to a more intuitive behavior of algorithms for the k-server problem regarding the dependence of the algorithm on memory and the effects of finite lookahead.

Finally, a fourth model was introduced by Kalyanasundaram and Pruhs [44], motivated by the unrealistically high ratios proven for scheduling algorithms (see also Section 6). They compare the performance of an on-line algorithm to the performance of an optimal off-line algorithm when the on-line algorithm is given extra resources.

All of these models have yet to prove their generality and to provide

significant positive results.

3 Adversary models

The power of randomization in on-line problems was first demonstrated by Borodin et al. [19] in the context of *metrical task systems* (see Section 4). The framework presented below was developed by Raghavan and Snir [56] and Ben-David et al. [14].

The first ingredient in studying a randomized on-line algorithm is the notion of competitiveness for such an algorithm. Consider a randomized on-line algorithm A. On a miss, it makes a (possibly random) choice of which of the k pages to evict from the cache. So now the cost $f_A(\rho)$ incurred by the algorithm is a random variable. Following the ideas from the deterministic algorithms, we want to study the behavior of A when the request sequence is generated by an adversary. But how much information does this adversary have, now that the choices made by the algorithm are no longer predictable? Does it know the outcome of the previous steps when making a decision? This kind of information turns out to be crucial for the overall effectiveness of the algorithm. The weakest adversary we can envision doesn't have any information of the random choices made by the algorithm, ut knows the algorithm A. Such an adversary is called an *oblivious adversary.*

We say that a randomized on-line algorithm A is $\mathcal{C}$-competitive against an oblivious adversary if for every sequence of requests ρ,

$$E[f_A(\rho)] - \mathcal{C} \times f_0(\rho) \leq b,$$

for a constant b independent of N, the number of requests. The *oblivious competitiveness ratio* of A, denoted $f_A^o bl$, is the infimum $\mathcal{C}$ such that A is $\mathcal{C}$-competitive against an oblivious adversary. Here $f_0(\rho)$ is still the cost of **MIN**, the optimal offline algorithm for paging.

One of the first randomized algorithms to improve significantly over the optimal deterministic algorithm is algorithm Mark for paging. It proceeds in a series of rounds. Each of the k cache locations has a marker bit associated with it. At the beginning of every round all marker bits are reset. When a request comes for a page already in the cache, the marker bit of that location is set to one. If the request is a miss, the page to be evicted is chosen uniformly at random from all unmarked locations. The marker bit of the chosen location is set to one. After all the locations have been marked, the round is deemed over on the next request for a page not in the cache. Fiat et al. [32], who discovered the algorithm, proved that it is

$2H_k$-competitive against an oblivious adversary, where $H_k \approx \ln k$ is the k^{th} harmonic number.

A more powerful type of adversary is one who can choose each request after having observed the previous choices of the on-line algorithm. Formally, an *adaptive adversary* chooses ρ_{i+1} after observing the responses of the algorithm to the previous requests, $\rho_1, \ldots, \rho_i$. A new question arises now regarding how the adversary uses the request sequence to generate an optimal response. If it uses the whole generated sequence as input to the optimal (offline) algorithm, it is called an *adaptive off-line adversary*. If, however, the adversary has to also manage a cache on-line, we talk about an *adaptive on-line adverary*. Note that in both of these cases the cost $f_0(\rho)$ of the optimal algorithm is also a random variable, since the sequence of requests used by the adversary depends on the behavior of A and is thus a random sequence. In conclusion, we say that a randomized on-line algorithm A is $\mathcal{C}$-competitive against the adaptive (offline or online) adversary if

$$E[[f_A(\rho)] - \mathcal{C} \times E[f_0(\rho)] \leq b,$$

for a constant b independent of N. As above, we define the *adaptive off-line (respectively, on-line) competitiveness ratio* of A, denoted $\mathcal{C}_A^{aof}$ ($\mathcal{C}_A^{aon}$). A moment's thought reveals the relation between these ratios, for any algorithm A:

$$\mathcal{C}_A^{obl} \leq \mathcal{C}_A^{aon} \leq \mathcal{C}_A^{aof}.$$

4 Metrical Task Systems

The notion of task systems was introduced by Borodin et al. [19] as a general model for the processing of sequences of tasks. Their importance lies in the power of subsuming a number of on-line problems, computer-related (paging, k-server) or not (choosing an investment strategy).

A *task system* (S, d) consists of a set S of states and a cost matrix d, where $d(i, j)$ is the cost of changing from state i to state j. We assume that d satisfies the triangle inequality and all diagonal entries are 0. A schedule for a sequence $T^1, T^2, \ldots, T^N$ of tasks is a sequence $s_1, s_2, \ldots, s_N$ of states, where s_i is the state in which T^i is processed; the cost of a schedule is the sum of all task processing costs and state transition costs incurred. An on-line scheduling algorithm is one that chooses s_i only knowing $T^1, T^2, \ldots, T^i$. A *symmetric*, or *metrical*, task system is one in which the matrix d is symmetric.

In the same paper where they introduce this general model [19], Borodin et al. prove that any deterministic on-line algorithm for a metrical task system (S, d) has competitive ratio at least $2|S| - 1$ and they give an on-line algorithm that is $(2|S|-1)$-competitive. They also show that using randomization one can achieve a competitive ratio of $O(\log n)$ against an oblivious adversary for the special case of the uniform metric space. Several papers have since presented randomized algorithms for special metric spaces, such as an $O(\log n)$-competitive algorithm for "highly unbalanced" spaces [16], and an $O(2^{\sqrt{\log n \log\log n}})$-competitive algorithm for equally-spaced points on a line [17]. Irani and Seiden [42] present a randomized algorithm for general spaces that achieves competitive ratio of roughly $1.58n$. In a recent breakthrough, Bartal et al. [9] presented an $O(\log^6 n / \log\log n)$-competitive randomized algorithm for arbitrary metric spaces. The algorithm uses a result of Bartal [8] that an arbitrary metric space can be probabilistically approximated by a set of metric spaces called k-hierarchical well-separated trees.

It is still an open problem how much better we can do in general. The lower bounds proved so far—$\Omega(\log\log n)$ [48], later improved to $\Omega(\sqrt{\log n / \log\log n})$ [16]—lead to the belief that the correct answer is $\Theta(\log n)$. For special metric spaces such as the uniform space [19], and the superincreasing space [48] there are $\Omega(\log n)$ bounds on the competitive ratio of any randomized on-line algorithm.

All the above considerations pertain to randomized algorithms working against an oblivious adversary, the weakest type of the three introduced in section 3. Against an adaptive on-line adversary, Coppersmith et al. [26] proved a lower bound of $2n - 1$ for the competitive ratio and gave a simple, memoryless, randomized algorithm for any metric space that is $(2n - 1)$-competitive. The algorithm uses methods for the synthesis of random walks on graphs with positive real costs on the edges. This result proves in essence that randomized algorithms for MTS against an adaptive on-line adversary cannot do better than deterministic on-line algorithms.

4.1 The k-server Problem

Like the metrical task system problem, the k-server problem was introduced as a general on-line problem. However, the latter is actually a particular case of the former. It is general enough though to encomppass a number of well-known problems, like paging, caching, and two-headed disks, and at the same time is particular enough to make it more tractable and susceptible to interesting results than the metrical task system problem.

The *k-server problem* was introduced by Manasse et al. [54]. We are given initial locations of k servers in a metric space. Requests for service at points ρ_i come over time. Immediadely after the i^{th} request is received, one of the servers must be moved from its current location to ρ_i. The cost of moving a server from u to v is $d(u, v)$, the distance between u and v. Our goal is to minimize the total cost of an on-line algorithm.

Manasse et al. [54] proved that the competitive ratio of any deterministic algorithm for the k-server problem is at least k and gave an $(n-1)$-competitive deterministic algorithm for the $(n-1)$-server problem and a 2-competitive algorithm for the 2-server problem, where n is the number of points in the metric space.

For some time, it was open whether a constant competitive ratio algorithm exists. The question was answered affirmatively by Fiat et al. [33], who proved by induction on k that there is an $f(k)$-competitive algorithm, where $f(k) = O(e^{ck \log k})$. Later, Chrobak and Larmore [23] devised the Work Function Algorithm which Koutsoupias and Papadimitriou [51] proved it achieves a competitive ratio of at most $2k - 1$.

Randomization against an oblivious adversary has not been very successful for this problem. A discussion of the cases when such algorithms are better than the deterministic ones can be found in [46]. Also, the algorithm of Bartal et al [9] for metrical task systems yields an $O(c^6 \log^6 n)$-competitive algorithm for the k-server problem on metric spaces of $k + c$ points. The lower bounds mentioned for metrical task systems are true for the k-server problem as well (they were in fact originally proved for this problem).

The situation is slightly better in regard to adaptive adversaries. The Harmonic algorithm, due to Raghavan and Snir [56], is a memoryless randomized algorithm and has a competitive ratio less than $(5k2^k)/4$ [37]. But there are metric spaces for which Harmonic cannot achieve a competitiveness ratio lower than $k(k+1)/2$. The most general class of metric spaces for which we know of a k-competitive (optimal) algorithm against adaptive on-line adversaries follows from the work of Coppersmith et al. [26]; their algorithm works in a class of metric spaces called resistive metric spaces.

A lot of research efforts have been focused on finding competitive algorithms for specific values of k. The smallest non-trivial value is $k = 2$. As mentioned above, the algorithm of Manasse et al. [54] is 2-competitive and is optimal amongst deterministic algorithms for the 2-server problem. It is not clear how much randomization helps reduce this bound. A number of papers [46, 24] improved on the lower bound for randomized algorithms against an oblivious adversary. The current best is $1+e^{-1/2} \approx 1.6$. However, there is no algorithm with competitive ratio smaller than 2. For the 3-server problem,

Chrobak and Larmore [22] gave an 11-competitive deterministic algorithm.

4.2 Paging and Related Problems

Paging is one of the most studied on-line problems. As mentioned above, it is a special case of the on-line k-server problem. We interpret a server at point p to mean that the page p is in the cache. The size of the cache is k and all distances are equal.

The lower bound of H_k for the competitive ratio of any algorithm for paging against an oblivious adversary was proved by Fiat et al. [32] and an algorithm achieving this bound was presented by McGeoch and Sleator [55]. A simpler algorithm achieving the same bound was proposed by Achlioptas et al. [1]. In the same paper they prove that the competitive ratio of algorithm Mark [32] is exactly $2H_k - 1$.

A slight generalization of this problem is the weighted cache problem. Here, each page has a positive real weight, representing the cost of loading the page into the cache. The total cost incurred by an algorithm on a sequence of requests is the sum of these costs. Raghavan and Snir [56] give a simple randomized algorithm that is k-competitive against any adaptive on-line adversary. Chrobak et al. [21] give a deterministic k-competitive algorithm.

5 The List Update Problem

List update is another heavily studied on-line problem. Its success is due to its simplicity and direct applicability in practice. We want to maintain, on-line, a linear list containing n items under a sequence of requests, where each request specifies one of the n items. If the item requested is at the i^{th} position in the list, the algorithm incurs a cost of i for that request. When an item is requested, the algorithm has the option of moving that item to the front of the list, or leaving it in place. This model was indroduced by Sleator and Tarjan [60] and they proved that the Move-to-Front algorithm, which always moves the requested item to the front of the list, has competitiveness ratio 2. It is known [58] that this is the best competitive ratio that can be achieved by a deterministic algorithm and by a randomized algorithm against an adaptive on-line adversary.

There has been extensive research, however, to find the tight lower bound for algorithms against an oblivious adversary. Irani [41] gave a 1.9375-competitive algorithm, quickly surpassed by a $\sqrt{3}$-competitive algorithm

by Reingolg et al. [58]. Later, Albers [2] proposed a ϕ-competitive randomized algorithm and a new 2-competitive deterministic algorithm, where $\phi = (1 + \sqrt{5})/2 \approx 1.62$ is the Golden Ratio. Finally, Albers et al. [4] presented a 1.6-competitive algorithm that makes an initial random choice between two known algorithms that have different worst-case request sequences. The best lower bound known is due to Teia [62], who proved that no randomized algorithm can be better than 1.5-competitive.

A number of variations of the standard model have been studied in the literature: list update with paid exchange [60, 58, 35], weighted list update [28], list update with retrieval of sets [27].

6 Scheduling and Load Balancing

There are many variants of the on-line scheduling problem. In one of the simplest settings, a sequence of tasks must be scheduled on n identical parallel machines. Whenever a task arrives, its processing time is known in advance and the task must be scheduled on one of the machines, without knowledge of any future tasks. Preemption is not allowed. The goal is to minimize the makespan.

First on-line algorithm is credited to Graham [36]. His greedy List algorithm takes the tasks one by one and always schedules them on the least loaded machine. The competitive ratio of this algorithm is $(2 - 1/n)$. Recently, Bartal et al. [10] gave a 1.986-competitive deterministic algorithm for all $n \geq 70$, which was generalized by Karger et al. [45], who also proved an upper bound of 1.945, for all n. The best known upper bound to date is 1.923 and was proved by Albers [3], who proposed a new 1.923-competitive algorithm. She also proves a new lower bound for any deterministic algorithm, namely 1.852 for all $n \geq 80$.

For the case when the processing time of the incoming task is not known a priori, Shmoys et al. [59] prove that Garaham's algorithm is optimal among deterministic algorithms. They also show that randomization is of little help here: the best competitive ratio one can hope against an oblivious adversary is $(2 - O(1/\sqrt{m}))$.

A more general problem is load balancing, where each task has a weight and the goal is to minimize the maximum load on the machines. Most of the variants that that come up in the literature can be classified using two independent dichotomies that characterize the tasks [5]. The first is according to how long a task runs: if it runs for a limited period it is called *temporary*; if it runs indefinitely is called *permanent*. The second is according

to the set of servers on which a task can run: if it can run on any server, we say it is *unrestricted*, whereas if it can be assigned only to a proper subset of machines, we say it is *restricted*.

The problem described at the beginning of the section is for temporary, unrestricted tasks. Graham's greedy algorithm has the same competitive ratio, namely $(2 - 1/n)$, for permanent, unrestricted tasks. For permanent, restricted tasks, Azar et al. [6] showed that the competitive ratio of the greedy algorithm is $\log n$ and that no algorithm can do better. Finally, for temporary, restricted tasks, Azar et al [5] proved that the competitive ratio of the greedy algorithm is $((3n)^{2/3}/2)(1 + o(1))$ and that this is tight for the greedy algorithm. They also prove that any randomized (or deterministic) algorithm is at least $\Omega(n^{1/2})$-competitive.

Other versions of these problems have been studied by allowing tasks to be preempted [20, 29] or by allowing parallel tasks [31, 30].

7 Graph Problems

In an on-line graph optimization problem, the vertices and/or edges of the graph arrive on-line, and the algorithm has to make some relevant decision before another vertex/edge arrives. The adversary chooses both the graph and the order in which to send the vertices/edges. Problems studied in this context include vertex coloring (see Section 7.1), edge coloring [7], Steiner tree [63, 15], spanning tree [63] bipartite matching [49, 43]. Most of the algorithms given for these problems use randomization against an oblivious adversary.

By far, the most studied graph problem in the on-line setting is graph coloring, which gets a subsection of its own.

7.1 Graph Coloring

The problem of coloring a graph is that of assigning the vertices to the fewest bins possible so that no two vertices assigned to the same bin are are adjacent. In the on-line setting, a vertex is given along with its edges to the previous vertices. The algorithm must irrevocably assign the vertex to a bin before proceeding to the next vertex. The competitive ratio is measured in total number of bins (colors) needed by the algorithm. Lovász et al. [52] gave an $O(n/\log^* n)$-competitive algorithm, slightly improving the trivial bound of n. Vishwanathan [64] gave a randomized algorithm which attains a competitive ratio of $O(n/\sqrt{\log n})$ against an oblivious adversary. His algorithm was later modified by Halldórsson to improve the ratio to

$O(n/\log n)$. The best lower bound was proved by Halldórsson and Szegedy [39]: $2n/\log^2 n$ for deterministic algorithms and $O(n/log^2 n)$ for randomized algorithms.

In the case when all the graph is known to the algorithm a priori, some preprocessing can be done to improve the performance. Bartal et al. [11] gave an $O(\sqrt{n})$-competitive deterministic algorithm and prove that no randomized algorithm can do beter than $\Omega(n^{1-\log_4 3})$.

Also, on-line coloring algorithms have been developed for special families of graphs [38, 40, 50].

References

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. In Josep Díaz and Maria Serna, editors, *Algorithms—ESA '96, Fourth Annual European Symposium*, volume 1136 of *Lecture Notes in Computer Science*, pages 419–430, Barcelona, Spain, 25–27 September 1996. Springer.

[2] Susanne Albers. Improved randomized on-line algorithms for the list update problem. In *Proceedings of the 6th Annual Symposium on Discrete Algorithms*, pages 412–419, New York, NY, USA, January 1995. ACM Press.

[3] Susanne Albers. Better bounds for online scheduling. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 130–139, El Paso, Texas, 4–6 May 1997.

[4] Susanne Albers, Bernhard von Stengel, and Ralph Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. Technical Report TR-95-016, International Computer Science Institute, Berkeley, CA, April 1995.

[5] Y. Azar, A. Z. Broder, and A. R. Karlin. On-line load balancing. *Theoretical Computer Science*, 130(1):73–84, August 1994.

[6] Y. Azar, J. Naor, and R. Rom. The competitiveness of online assignments. In *Proc. 3rd ACM-SIAM Symp. Discrete Algorithms*, pages 203–210, 1992.

[7] A. Bar-Noy, R. Motwani, and J. Naor. The greedy algorithm is optimal for online edge coloring. *Information Processing Letters*, 44:251–253, 1992.

[8] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science*, pages 184–193, Burlington, Vermont, 14–16 October 1996. IEEE.

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

[10] Yair Bartal, Amos Fiat, Howard Karloff, and Rakesh Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51(3):359–366, December 1995.

[11] Yair Bartal, Amos Fiat, and Stefano Leonardi. Lower bounds for on-line graph problems with application to on-line circuit and optical routing. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 531–540, Philadelphia, Pennsylvania, 22–24 May 1996.

[12] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[13] S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11(1):73–91, 1994.

[14] S. Ben-David, A. Borodin, G. Tardos R. Karp, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11:2–14, 1994.

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

[16] A. Blum, H. Karloff, Y. Rabani, and M. Saks. A decomposition theorem and bounds for randomized server problems. In IEEE, editor, *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 197–207, Pittsburgh, PN, October 1992. IEEE Computer Society Press.

[17] Avrim Blum, Prabhakar Raghavan, and Baruch Schieber. Navigating in unfamiliar geometric terrain (preliminary version). In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 494–504, New Orleans, Louisiana, 6–8 May 1991.

[18] Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50(2):244–258, April 1995.

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

[20] Bo Chen, André van Vliet, and Gerhard J. Woeginger. An optimal algorithm for preemptive on-line scheduling. In Jan van Leeuwen, editor, *Algorithms—ESA '94, Second Annual European Symposium*, volume 855 of *Lecture Notes in Computer Science*, pages 300–306, Utrecht, The Netherlands, 26–28 September 1994. Springer.

[21] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. In David Johnson, editor, *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, pages 291–300, San Francisco, CA, USA, January 1990. SIAM.

[22] Marek Chrobak and Lawrence L. Larmore. On fast algorithms for two servers. In Branislav Rovan, editor, *Mathematical Foundations of Computer Science 1990*, volume 452 of *lncs*, pages 202–208, Banská Bystrica, Czechoslovakia, 27–31 August 1990. Springer.

[23] Marek Chrobak and Lawrence L. Larmore. The server problem and on-line games. In Lyle A. McGeoch and Daniel D. Sleator, editors, *On-line Algorithms*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 11–64. AMS/ACM, February 1991.

[24] Marek Chrobak, Lawrence L. Larmore, Carsten Lund, and Nick Reingold. A better lower bound on the competitive ratio of the randomized 2-server problem. *Information Processing Letters*, 63(2):79–83, 28 July 1997.

[25] Marek Chrobak and John Noga. LRU is better than FIFO. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 78–81, San Francisco, California, 25–27 January 1998.

[26] Don Coppersmith, Peter Doyle, Prabhakar Raghavan, and Marc Snir. Random walks on weighted graphs and applications to on-line algorithms. *Journal of the ACM*, 40(3):421–453, July 1993.

[27] F. D'Amore and V. Liberatore. The list update problem and the retrieval of sets. *Theoretical Computer Science*, 130(1):101–123, August 1994.

[28] F. D'Amore, A. Marchetti-Spaccamela, and U. Nanni. The weighted list update problem and the lazy adversary. *Theoretical Computer Science*, 108(2):371–384, February 1993.

[29] Jeff Edmonds, Donald D. Chinn, Tim Brecht, and Xiaotie Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 120–129, El Paso, Texas, 4–6 May 1997.

[30] A. Feldmann, J. Sgall, and Shang-Hua Teng. Dynamic scheduling on parallel machines. *Theoretical Computer Science*, 130(1):49–72, August 1994.

[31] Anja Feldmann, Ming-Yang Kao, Jiri Sgall, and Shang-Hua Teng. Optimal online scheduling of parallel jobs with dependencies. Technical Report CS-92-189, Carnegie Mellon University, School of Computer Science, September 1992.

[32] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, December 1991.

[33] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48(3):410–428, June 1994.

[34] P. A. Franaszek and T. J. Wagner. Some distribution-free aspects of paging algorithm performance. *Journal of the ACM*, 21(1):31–39, January 1974.

[35] T. Garefalakis. A new family of randomized algorithms for list accessing. In *Algorithms—ESA '97, Fifth Annual European Symposium*, pages 200–216, 15–17 September 1997.

[36] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[37] E. F. Grove. The harmonic online K-server algorithm is competitive. In Lyle A. McGeoch and Daniel D. Sleator, editors, *On-line Algorithms*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 65–77. AMS/ACM, February 1991.

[38] András Gyárfás and Jenö Lehel. On-line and first fit colorings of graphs. *Journal of Graph Theory*, 12(2):217–227, 1988.

[39] M. M. Halldorsson and M. Szegedy. Lower bounds for on-line graph coloring. *Theoretical Computer Science*, 130(1):163–174, August 1994.

[40] S. Irani. Coloring inductive graphs on-line. *Algorithmica*, 11:53–72, 1994. Also in Proc. 31st IEEE Symposium on Foundations of Computer Science, 1990, 470-479.

[41] Sandy Irani. Two results on the list update problem. *Information Processing Letters*, 38(6):301–306, June 1991.

[42] Sandy Irani and Steve Seiden. Randomized algorithms for metrical task systems. *Theoretical Computer Science*, 194(1–2):163–182, 10 March 1998.

[43] B. Kalyanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, May 1993.

[44] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 214–223, Los Alamitos, October 1995. IEEE Computer Society Press.

[45] David R. Karger, Steven J. Phillips, and Eric Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20(2):400–430, March 1996.

[46] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542–571, June 1994.

[47] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(?):79–119, 1988.

[48] Howard Karloff, Yuval Rabani, and Yiftach Ravid. Lower bounds for randomized k-server and motion-planning algorithms. *SIAM Journal on Computing*, 23(2):293–312, April 1994.

[49] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, pages 352–358, Baltimore, MD, May 1990. An on-line algorithm receives a sequence of requests and must respond to each request as soon as it is received. In contrast, an off-line algorithm may wait until all requests have been received before determining its responses. The authors give a simple, randomized, optimal, on-line algorithm for bipartite matching.

[50] H. A. Kierstead. The linearity of first-fit coloring of interval graphs. *SIAM Journal on Discrete Mathematics*, 1(4):526–530, ???? 1988.

[51] Elias Koutsoupias and Christos Papadimitriou. On the k-server conjecture. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 507–511, Montréal, Québec, Canada, 23–25 May 1994.

[52] László Lovász, Michael Saks, and W. T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75(1-3):319–325, 1989.

[53] Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive algorithms for on-line problems. In Richard Cole, editor, *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 322–333, Chicago, IL, May 1988. ACM Press.

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

[55] Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6:816–825, 1991.

[56] P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. *IBM Journal of Research and Development*, 38(6):683–707, November 1994.

[57] Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms (extended abstract). In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium*, volume 372 of *Lecture Notes in Computer Science*, pages 687–703, Stresa, Italy, 11–15 July 1989. Springer-Verlag.

[58] Nick Reingold, Jeffery Westbrook, and Daniel Dominic Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, January 1994.

[59] David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6):1313–1331, December 1995.

[60] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, February 1985.

[61] K. So and R. N. Rechtschaffen. Cache operations by mru change. *IEEE Transactions on Computers*, 37(6):700–709, June 1988.

[62] Boris Teia. Lower bound for randomized list update algorithms. *Information Processing Letters*, 47(1):5–9, August 1993.

[63] Ying Teh Tsai and Chuan Yi Tang. The competitiveness of randomized algorithms for on-line Steiner tree and on-line spanning tree problems. *Information Processing Letters*, 48(4):177–182, November 1993.

[64] S. Vishwanathan. Randomized on-line graph coloring. *J. Algorithms*, 13:657–669, 1992. Also in Proc. 31st IEEE Symposium on Foundations of Computer Science, 1990, 464–469.

[65] N. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, June 1994.