Online Algorithms

Susanne Albers

University of Freiburg, Germany

1 Introduction

This book chapter reviews fundamental concepts and results in the area of online algorithms. We first address classical online problems and then study various applications of current interest.

Online algorithms represent a theoretical framework for studying problems in interactive computing. They model, in particular, that the input in an interactive system does not arrive as a batch but as a sequence of input portions and that the system must react in response to each incoming portion. Moreover, they take into account that at any point in time future input is unknown. As the name suggests, online algorithms consider the algorithmic aspects of interactive systems: We wish to design strategies that always compute good output and keep a given system in good state. No assumptions are made about the input stream. The input can even be generated by an adversary that creates new input portions based on the system's reactions to previous ones. We seek algorithms that have a provably good performance.

Formally, an online algorithm receives a *sequence of requests* $\sigma = \sigma(1), \ldots, \sigma(m)$. These requests must be served in the order of occurrence. When serving request $\sigma(t)$, an online algorithm does not know requests $\sigma(t')$ with $t' > t$. Serving requests incurs cost and the goal is to minimize the total cost paid on the entire request sequence. This process can be viewed as a *request answer game*. An adversary generates requests and an online algorithm has to serve them one at a time. The performance of online algorithms is usually evaluated using *competitive analysis* [65]. Here an online algorithm ALG is compared to an optimal offline algorithm OPT that knows the entire request sequence σ in advance and can serve it with minimum cost. Given a sequence σ, let $ALG(\sigma)$ and $OPT(\sigma)$ denote the costs incurred by ALG and OPT, respectively. Algorithm ALG is called c-competitive if there exists a constant b such that $ALG(\sigma) \leq c \cdot OPT(\sigma) + b$, for all sequences σ. The constant b must be independent of the input σ. We note that competitive analysis is a strong worst-case performance measure.

Over the past 15 years online algorithms have received tremendous research interest. Online problems have been studied in many application areas including resource management in operating systems, data structuring, scheduling, networks, and computational finance. In the following sections we first survey fundamental results. We address the paging problem, self-organizing lists, the k-server problem as well as metrical task systems. Then we review a number of new results in application areas of current interest. We focus on algorithmic problems in large networks and competitive auctions. Finally we present refinements of competitive analysis and conclude with some remarks.

2 Basic Results

Paging is an extensively studied problem and perhaps one of the oldest examples of an interactive computing problem. It arises when a CPU communicates with the underlying memory hierarchy. Paging is also an excellent problem to illustrate basic concepts in the theory of online algorithms and we therefore study it in the rest of this section.

In paging we have to maintain a two-level memory system consisting of a small fast memory and a large slow memory. The memory is partitioned into pages of equal size. The system receives a sequence of requests, where each request specifies a page in the memory system. A request can be served immediately if the referenced page is available in fast memory. If the requested page is not in fast memory, a *page fault* occurs. The missing page is then loaded from slow memory into fast memory so that the request can be served. At the same time a page is evicted from fast memory to make room for the missing one. A paging algorithm decides which page to evict on a fault. This decision must usually be made online, i.e. without knowledge of any future requests. The cost to be minimized is the number of page faults.

The two most popular online paging algorithms are LRU and FIFO.

LRU (Least Recently Used): On a fault, evict the page in fast memory that was requested least recently.

FIFO (First-In First-Out): Evict the page that has been in fast memory longest.

Sleator and Tarjan [65] analyzed the performance of the two algorithms. Let k be the number of pages that can simultaneously reside in fast memory.

Theorem 1. [65] *LRU and FIFO are k-competitive.*

There exists a more general class of algorithms that achieve a competitiveness of k.

Marking: A *Marking* strategy processes a request sequence in phases. At the beginning of each phase all pages in the memory system are unmarked. Whenever a page is requested, it is *marked*. On a fault, an arbitrary unmarked

page in fast memory is evicted. A phase ends when all pages in fast memory are marked and a page fault occurs. Then all marks are erased and a new phase is started.

It is not hard to see that LRU is in fact a *Marking* algorithm. *Marking* strategies were considered in [24, 37]. Torng [67] explicitly observed that any *Marking* strategy is k-competitive. This factor is best possible for deterministic paging algorithms.

Theorem 2. [65] *No deterministic online algorithm for the paging problem can achieve a competitive ratio smaller than k.*

An optimal offline algorithm for the paging problem was presented by Belady [17]. The algorithm is called MIN and works as follows.

MIN: On a fault, evict the page whose next request occurs furthest in the future.

Belady showed that on any sequence of requests, MIN incurs the minimum number of page faults.

In many problems, such as paging, online algorithms can achieve a better performance if they are allowed to make random choices. The competitive ratio of a randomized online algorithm ALG is defined with respect to an adversary. The adversary generates a request sequence σ and also has to serve σ. When constructing σ, the adversary always knows the description of ALG. The crucial question is: When generating requests, is the adversary allowed to see the outcome of the random choices made by A on previous requests? Oblivious adversaries do not have this ability while adaptive adversaries do. In the literature there exist three kinds of adversaries, which were introduced by Ben-David et al. [19].

Oblivious adversary: The oblivious adversary has to generate the entire request sequence in advance before any requests are served by the online algorithm. The adversary is charged the cost of the optimum offline algorithm for that sequence.

Adaptive online adversary: This adversary may observe the online algorithm and generate the next request based on the algorithm's (randomized) answers to all previous requests. The adversary must serve each request online, i.e. without knowing the random choices made by the online algorithm on the present or any future request.

Adaptive offline adversary: This adversary also generates a request sequence adaptively. However, it may serve the sequence offline and hence is charged the optimum offline cost for that sequence.

A randomized online algorithm ALG is called c-competitive against oblivious adversaries if there is a constant b such that, for all request sequences σ generated by an oblivious adversary, $E[ALG(\sigma)] \leq c \cdot OPT(\sigma) + b$. The expectation is taken over the random choices made by ALG.

Given a randomized online algorithm ALG and an adaptive online (adaptive offline) adversary ADV, let $E[ALG(\sigma)]$ and $E[ADV(\sigma)]$ denote the expected costs incurred by ALG and ADV in serving a request sequence generated by ADV. Algorithm ALG is called c-competitive against adaptive online (adaptive offline) adversaries if there is a constant b such that, for all adaptive online (adaptive offline) adversaries ADV, $E[ALG(\sigma)] \leq c \cdot E[ADV(\sigma)] + b$ where the expectation is taken over the random choices made by ALG.

Ben-David et al. [19] investigated the relative strength of the adversaries and proved the following results.

Theorem 3. [19] *If there is a randomized online algorithm that is c-competitive against adaptive offline adversaries, then there also exists a c-competitive deterministic online algorithm.*

Theorem 4. [19] *If ALG is a c-competitive randomized algorithm against adaptive online adversaries and if there is a d-competitive algorithm against oblivious adversaries, then ALG is $(c \cdot d)$-competitive against adaptive offline adversaries.*

Theorem 3 implies that randomization does not help against adaptive offline adversaries, and we can ignore them when in search for improved competitive ratios. An immediate consequence of the two theorems above is:

Corollary 1. *If there exists a c-competitive randomized algorithm against adaptive online adversaries, then there is a c^2-competitive deterministic algorithm.*

A result by Raghavan and Snir [61] implies that against adaptive online adversaries, no randomized online paging strategy can be better than k-competitive. Hence we concentrate on oblivious adversaries and show that we can achieve an exponential improvement over the deterministic bound of k. The most popular randomized online paging algorithm is the *Randomized-Marking* strategy presented by Fiat et al. [37]. It is optimal up to a constant factor.

Randomized-Marking: The algorithm is a *Marking* strategy. On a fault, a page is chosen uniformly at random from among the unmarked pages in fast memory, and that page is evicted.

Let $H_k = \sum_{i=1}^{k} 1/i$ be the k-th Harmonic number, which is closely approximated by $\ln k$, i.e. $\ln(k+1) \leq H_k \leq \ln k + 1$.

Theorem 5. [37] *Randomized-Marking is $2H_k$-competitive against oblivious adversaries.*

Theorem 6. [37] *The competitive ratio of randomized online paging algorithms against oblivious adversaries is not smaller than H_k.*

More complicated algorithms achieving an optimal competitiveness of H_k were presented in [1, 58].

3 Self-Organizing Data Structures

Data structuring is a classical field where many online problems arise. We have to maintain a given structure not knowing which items in the structure will be accessed next. There has been a lot of research on self-organizing lists and trees.

The problem of self-organizing lists, also called the *list update problem*, consists in maintaining a set of items as an unsorted linear list. We are given an unsorted linear linked list of items. As input we receive a sequence of requests, where each request specifies an item in the list. To serve a request, we have to access the requested item. We start at the front of the list and search linearly through the items until the desired item is found. Serving a request to an item that is currently stored at position i in the list incurs a cost of i. Immediately after a request, the referenced item may be moved at no extra cost to any position closer to the front of the list. This can lower the cost of subsequent requests. However, the decision where to move an item must be made online, without knowledge of any future requests. At any time, two adjacent items in the list may be exchanged at a cost of 1. The goal is to serve the request sequence so that the total cost is as small as possible.

Self-organizing lists are useful when maintaining a small dictionary consisting of only a few dozens of items and, moreover, have interesting applications in data compression [5, 20, 27].

With respect to the list update problem we require that a c-competitive online algorithm has a performance ratio of c, for *all size lists*. There exist three very well-known deterministic algorithms.

Move-To-Front: Move the requested item to the front of the list.
Transpose: Exchange the requested item with the immediately preceding item in the list.

Frequency-Count: Maintain a frequency count for each item in the list. Whenever an item is requested, increase its count by 1. Maintain the list so that the items always occur in nonincreasing order of frequency count.

Sleator and Tarjan analyzed these three algorithms. It shows that Move-To-Front achieves an optimal competitiveness of 2 while the other strategies are not competitive at all.

Theorem 7. [65] *The Move-To-Front algorithm is 2-competitive.*

Theorem 8. [50] *The competitive ratio of any deterministic online algorithm is not smaller than 2.*

Proposition 1. *The algorithms Transpose and Frequency-Count are not c-competitive, for any constant c.*

Ambühl [8] showed that the offline variant of the list update problem is NP-hard. Thus, there is no efficient algorithm for computing an optimal service schedule.

We next consider the influence of randomization. Against adaptive online adversaries no randomized strategy can be better than 2-competitive [62]. However, against oblivious adversaries we can improve the factor of 2. A number of randomized strategies have been proposed in the literature. We mention here only the two most important ones. Reingold et al. [62] presented counter-based algorithms, which move an item to the front of the list if its counter takes a certain value. Using mod 2 counters, we obtain the elegant *Bit* algorithm.

Bit: Each item in the list maintains a bit that is complemented whenever the item is accessed. If an access causes a bit to change to 1, then the requested item is moved to the front of the list. Otherwise the list remains unchanged. The bits of the items are initialized independently and uniformly at random.

Theorem 9. [62] *The Bit algorithm is 1.75-competitive against oblivious adversaries.*

The best randomized algorithm known to date combines *Bit* with a deterministic 2-competitive online algorithm called *Timestamp* proposed in [2].

Timestamp (TS): Insert the requested item, say x, in front of the first item in the list that precedes x and that has been requested at most once since the last request to x. If there is no such item or if x has not been requested so far, then leave the position of x unchanged.

Combination: With probability 4/5 serve a request sequence using *Bit*, and with probability 1/5 serve it using *TS*.

Theorem 10. [6] *The algorithm Combination is 1.6-competitive against oblivious adversaries.*

This factor of 1.6 is close to the best lower bound known.

Theorem 11. [9] *Let A be a randomized online algorithm for the list update problem. If A is c-competitive against oblivious adversaries, then $c \geq 1.50084$.*

The latest results on the list update problem are by Blum et al. [21]. Using techniques from learning theory, they gave a randomized online algorithm that, for any $\epsilon > 0$, is $(1.6 + \epsilon)$-competitive and at the same time $(1 + \epsilon)$-competitive against an offline algorithm that is restricted to serving a request sequence with a static list. The main open problem with respect to the list update problem is to develop tight upper and lower bounds on the performance of randomized algorithms.

Many of the concepts shown for self-organizing linear lists can be extended to binary search trees. The most popular version of self-organizing binary search trees are the *splay trees* presented by Sleator and Tarjan [66]. In a splay tree, after each access to an element x in the tree, the node storing x is moved to the root of the tree using a special sequence of rotations that depends on the structure of the access path. Sleator and Tarjan [66] showed that on

any sequence of accesses a splay tree is as efficient as the optimum static search tree. The famous splay tree conjecture is still open: It is conjectured that on any sequence of accesses splay trees are as efficient as any dynamic binary search tree.

4 The *k*-Server Problem

The k-server problem is one of the most famous online problems. It has received a lot of research interest, partly because proving upper bounds on the performance of k-server algorithms is a very challenging task. The k-server problem generalizes paging as well as other caching problems. It can also be viewed as an online vehicle routing problem.

In the k-server problem we are given a metric space S and k mobile servers that reside on points in S. As usual we receive a sequence of requests, where each request specifies a point $x \in S$. In response, a server must be moved to the requested point, unless a server is already present. Moving a server from point x to point y incurs a cost equal to the distance between the two points. The goal is to minimize the total distance traveled by all servers.

It is easy to see that the k-server problem models paging: Consider a metric space in which the distance between any two points in 1. Each point in the metric space represents a page in the memory system and the pages covered by servers are those that reside in fast memory. The k-server problem was introduced in 1988 by Manasse et al. [57] who showed a lower bound for deterministic k-server algorithms.

Theorem 12. [57] *Let A be a deterministic online k-server algorithm in an arbitrary metric space. If A is c-competitive, then $c \geq k$.*

Manasse et al. also conjectured that there exist k-competitive deterministic online algorithms. This conjecture essentially is still open. In 1995, however, Koutsoupias and Papadimitriou [53] achieved a breakthrough. They showed that the *Work Function* algorithm is $(2k-1)$-competitive. Before, k-competitive algorithms were known only for special metric spaces (e.g. trees [29] and resistive spaces [31]) and special values of k ($k = 2$ and $k = n - 1$, where n is the number of points in the metric space [57]).

The *Work Function* algorithm tries to mimic the optimal offline algorithm and at the same time incorporates aspects of the *Greedy* strategy. Let X be a configuration of the servers. Given a request sequence $\sigma = \sigma(1), \ldots, \sigma(t)$, the *work function* $w(X)$ is the minimal cost of serving σ and ending in configuration X. For any two points x and y in the metric space, let $dist(x, y)$ be the distance between x and y.

Work Function: Suppose that the algorithm has served $\sigma = \sigma(1), \ldots, \sigma(t-1)$ and that a new request $r = \sigma(t)$ arrives. Let X be the current configuration of the servers and let x_i be the point where server s_i, $1 \leq i \leq k$, is located.

Serve the request by moving the server s_i that minimizes $w(X_i) + dist(x_i, r)$, where $X_i = X - \{x_i\} + \{r\}$.

Theorem 13. [53] *The Work Function algorithm is* $(2k-1)$*-competitive in an arbitrary metric space.*

An interesting open problem is to show that the *Work Function* algorithm is indeed k-competitive or to develop an other deterministic online k-server algorithm that achieves a competitive ratio of k.

Next we turn to randomized k-server algorithms. Against adaptive online adversaries, no randomized strategy can be better than k-competitive. Against oblivious adversaries the best lower bound currently known is due to Bartal et al. [15].

Theorem 14. [15] *The competitive ratio of a randomized online algorithm in an arbitrary metric space is* $\Omega(\log k / \log^2 \log k)$ *against oblivious adversaries.*

The bound can be improved to $\Omega(\log k)$ if the metric space consists of at least $k^{\log^\epsilon k}$ points, for any $\epsilon > 0$, [15]. It is conjectured that $\Theta(\log k)$ is the true competitiveness of randomized algorithms against oblivious adversaries. Bartal et al. [14] presented an algorithm that has a competitive ratio of $O(c^6 \log^6 k)$ in metric spaces consisting of $k + c$ points. Seiden [64] gave an algorithm that achieves a competitive ratio polylogarithmic in k for metric spaces that can be decomposed into a small number of widely separated subspaces. A very challenging open problem is to develop randomized online algorithms that have a competitive ratio of $c < k$ in an arbitrary metric space.

5 Metrical Task Systems

So far we have presented a number of online problems and related results. A natural question is if there exists a more general framework for studying online algorithms. Borodin et al. [25] developed *metrical task systems* that can model a very large class of online problems.

A metrical task system is defined by a metric space (S, d) and an associated set $\mathcal{T}$ of tasks. The space (S, d) consists of a finite set S of, say, n states and a distance function $d : S \times S \longrightarrow \mathbb{R}_0^+$, where $d(i, j) \geq 0$ denotes the cost of changing from state i to state j. Since the space is metric, d is symmetric, satisfies the triangle inequality and $d(i, i) = 0$, for all states i. The set $\mathcal{T}$ is the set of allowable tasks. A task $T \in \mathcal{T}$ is a vector $T = (T(1), T(2), \ldots, T(n))$, where $T(i) \in \mathbb{R}_0^+ \cup \{\infty\}$ denotes the cost of processing the task while in state i. A request sequence is a sequence of tasks $\sigma = T^1, T^2, T^3, \ldots, T^m$ that must be served starting from some initial state $s(0)$. When receiving a new task, an algorithm may serve the task in the current state or may change states at a cost. Thus the algorithm must determine a schedule of states $s(1), s(2), \ldots, s(m)$, such that task T^i is processed in state $s(i)$. The cost of

serving a task sequence is the sum of all state transition costs and all task processing costs: $\sum_{i=1}^{m} d(s(i-1), s(i)) + \sum_{i=1}^{m} T^i(s(i))$. The goal is to process a given task sequence so that the cost is as small as possible.

Borodin et al. [25] settled the competitiveness of deterministic online algorithms. Interestingly, the best competitiveness is achieved by a *Work Function* algorithm. Given a request sequence $\sigma = \sigma(1), \ldots, \sigma(t)$, let the *work function* $w_t(s)$ be the minimum cost to process σ starting from $s(0)$ and ending in state s.

Work Function: Suppose that the algorithm has served the first t requests $\sigma(1), \ldots, \sigma(t)$ of a request sequence and that it is currently in state s_t. To process the next task T^{t+1}, move to state the $s_{t+1} = s$ that minimizes $w_{t+1}(s) + d(s_t, s)$.

Theorem 15. [23, 25] *The Work Function algorithm is* $(2n-1)$*-competitive for any metrical task system with* n *states.*

Theorem 16. [25] *Any deterministic online algorithm for the metrical task systems problem has a competitive ratio of at least* $2n-1$*, where* n *is the number of task system states.*

Unfortunately, the competitive factor of $2n-1$ often does not provide meaningful bounds when special online problems are investigated. Consider the list update problem. Here the given list can be in $n!$ states. Hence, we obtain a bound of $(2n!-1)$ on the competitive factor of a deterministic online algorithm for the list update problem. However, *Move-To-Front* achieves a competitive factor of 2.

For randomized algorithms against oblivious adversaries, the known bounds are tight up to a logarithmic factor.

Theorem 17. [39] *There exists a randomized online algorithm that is* $O(\log^2 n / \log^2 \log n)$*-competitive against oblivious adversaries, for any metrical task system with* n *states.*

Theorem 18. [15] *Any randomized online algorithm for the metrical task systems problem has a competitive ratio of at least* $\Omega(\log n / \log^2 \log n)$ *against oblivious adversaries, where* n *is the number of task system states.*

Better bounds hold for uniform metrical task systems, where the cost $d(i, j)$ of changing states is equal to 1, for all $i \neq j$. Borodin et al. [25] gave a lower bound of H_n, where H_n is the n-th Harmonic number. The best upper bound currently known was presented by Irani and Seiden [46] and is equal to $H_n + O(\sqrt{\log n})$.

6 Application Areas

In the previous sections we presented a selection of important results for classical online problems. In this section we study two application areas that

have received a lot of research interest recently, namely large networks and competitive auctions.

6.1 Large Networks

With the advent of the Internet, researchers started investigating algorithmic problems that arise in large networks. There exists a host of interesting online problems addressing, e.g., the construction of networks, the maintenance of TCP connections or the management of local caches and buffers. Due to space limitations we only address a few recent problems here.

Network Switches

The performance of high-speed networks critically depends on switches that route data packets arriving at the input ports to the appropriate output ports so that the packets can reach their correct destinations in the network. To reduce packet loss when the traffic is bursty, ports are equipped with buffers where packets can be stored temporarily. However the buffers are of limited capacity so that effective buffer management strategies are important to maximize the throughput at a switch. As a result there has recently been considerable research interest in various single and multi-buffer management problems.

We first study single buffer problems, which arise e.g. when maintaining an output port queue. Consider a buffer that can simultaneously store up to B data packets. Packets arrive online and can be buffered if space permits. More specifically, at any time step t let $Q(t)$ be the set of packets currently stored in the buffer and let $A(t)$ be the set of newly arriving packets. Each packet p has a value $v(p)$ that represents a QoS parameter. If $|Q(t)| + |A(t)| \leq B$, then all new packets can be admitted to the buffer; otherwise $|Q(t)| + |A(t)| - B$ packets from $Q(t) \cup A(t)$ must be dropped. In the time step we can select one packet from the buffer and transmit it through the output port. We assume that the packet arrival step precedes the transmission step. The goal is to maximize the total value of the transmitted packets.

Several problem variants are of interest. In a FIFO model packets must be transmitted in the order they arrive. If packet p is transmitted before p', then p must not have arrived later than p'. In a non-FIFO model there is no such restriction. In a preemptive model we may drop packets from the buffer, while in a non-preemptive model this is not allowed.

Kesselman et al. [51] analyzed a natural *Greedy* algorithm in the preemptive FIFO model and proved that it is 2-competitive.

Greedy: In the event of buffer overflow, drop the packets with the smallest values.

In the following let α be the ratio of the largest to smallest packet value.

Theorem 19. [51] *Greedy achieves a competitive ratio of* $\min\{2 - \frac{1}{B+1}, 2 - \frac{2}{\alpha+1}\}$.

Recently Bansal et al. [13] gave an algorithm that achieves an improved competitiveness of 1.75. Kesselman et al. [52] showed a lower bound of 1.419.

Aiello et al. [7] investigated non-preemptive single buffer problems. In this case the buffer can simply be maintained as a FIFO queue. Andelman et al. [10] gave asymptotically tight bounds for this scenario. They analyzed the following algorithm. Suppose that the packet values are in the range $[1, \alpha]$.

Exponential-Interval-Round-Robin: Divide the buffer into k partitions of size B/k, where $k = \lceil \ln \alpha \rceil$. Split the interval $[1, \alpha]$ into k subintervals $[\alpha_0, \alpha_1), [\alpha_1, \alpha_2), \ldots, [\alpha_{k-1}, \alpha_k)$, where $\alpha_j = \alpha^{j/k}$. Each partition of the buffer is associated with one of the subintervals, accepting in a greedy manner packets from that subinterval. The partitions take turn in sending packets. If a partition is empty, its turn is passed to the next partition.

Theorem 20. [10] *Exponential-Interval-Round-Robin achieves a competitive ratio of* $e\lceil \ln \alpha \rceil$.

Theorem 21. [10] *No online algorithm can achieve a competitive ratio smaller than* $1 + \ln \alpha$ *in the non-preemptive model.*

Kesselman et al. [51] also introduced a *bounded delay model* where packets have deadlines. A packet that has not been transmitted by its deadline is lost. There is no bound on the buffer size and packets may be reordered. Kesselman et al. analyzed a *Greedy* strategy which at any time transmits the packet of highest value among those with unexpired deadlines. This strategy is 2-competitive.

Azar and Richter [12] extended many of the results mentioned so far to multi-buffer problems. Consider a switch with m input ports, each of which is equipped with a buffer that can simultaneously store up to B packets. These buffers serve a common output port. At any time t, let $Q_i(t)$ be the set of packets stored in buffer i and let $A_i(t)$ be the set of packets arriving at that buffer. If $|Q_i(t)| + |A_i(t)| \leq B$, then all arriving packets can be admitted to buffer i; otherwise $|Q_i(t)| + |A_i(t)| - B$ packets must be dropped. At any time, the switch can select one non-empty buffer and transmit the packet at the head through the output port. The goal is to maximize the total value of the transmitted packets.

Azar and Richter presented a general technique that transforms a buffer management strategy for a single queue (for both the preemptive and non-preemptive models) into an algorithm for m queues. The technique is based on the algorithm *Transmit-Largest* that works in the preemptive non-FIFO model.

Transmit-Largest (TL):

1. Admission control: Use *Greedy* for admission control in any of the m buffers. More precisely, enqueue a packet arriving at buffer i if buffer i

is not full or if the packet with the smallest value in the buffer has a lower value than the new packet. In the latter case the packet with the smallest value is dropped.
2. Transmission: In each time step transmit the packet with the largest value among all packets in the m queues.

Using this algorithm, Azar and Richter designed a technique *Generic-Switch* that takes a single buffer management algorithm A as input parameter. We are interested in the preemptive FIFO and the non-preemptive models. Here packets are always transmitted in the order they arrive (w.l.o.g., in the non-preemptive model) and only A's admission control strategy is relevant to us.

Generic-Switch:

1. Admission control: Apply admission control strategy A to any of the m buffers.
2. Transmission: Run a simulation of TL (in the preemptive non-FIFO model) with online paket arrival sequence σ. In each time step transmit the packet from the head of the queue served by TL.

The main result by Azar and Richter is as follows.

Theorem 22. [12] *If A is a c-competitive algorithm, then Generic-Switch is $2c$-competitive.*

Using this statement, one can derive a number of results for multi-queue problems. In the preemptive FIFO model *Greedy* achieves a competitiveness of $\min\{4-\frac{2}{B+1}, 4-\frac{4}{\alpha+1}\}$. The improved algorithm by Bansal et al. [13] gives a 3.5-competitive strategy. In the non-preemptive setting we obtain a $2e\lceil \ln \alpha \rceil$-competitive strategy.

TCP Acknowledgement

In large networks data transmission is performed using the Transmission Control Protocol (TCP). If two network nodes wish to exchange data, then there has to exist an open TCP connection between these two nodes. The data is partitioned into packets which are then sent across the connection. A node receiving data must acknowledge the receipt of each incoming packet so that the sending node is aware that the transmission was successful. In most TCP implementations today data packets do not have to be acknowledged individually. Instead, there is some delay mechanism which allows the TCP to acknowledge multiple packets with a single acknowledgement and, possibly, to piggyback the acknowledgement on an outgoing data packet. This reduces the number of acknowledgements sent and hence the network congestion as well as the overhead at the network nodes for sending and receiving acknowledgements. On the other hand, by reducing the number of acknowledgements, we add latency to the TCP connection, which is not desirable. Thus, the goal

is to balance the reduction in the number of acknowledgements with the increase in latency.

Dooly et al. [34] formulated the following TCP acknowledgement problem. A network node receives a sequence of m data packets. Let a_i denote the arrival time of packet i, $1 \leq i \leq m$. At time a_i, the arrival times a_j, $j > i$, are not known. We have to partition the sequence $\sigma = (a_1, \ldots, a_m)$ of packet arrival times into n subsequences $\sigma_1, \ldots, \sigma_n$, for some $n \geq 1$, such that each subsequence ends with an acknowledgement. We use σ_i to denote the set of arrivals in the partition. Let t_i be the time when the acknowledgement for σ_i is sent. We require $t_i \geq a_j$, for all $a_j \in \sigma_i$. If data packets are not acknowledged immediately, there are acknowledgement delays. Dooley et al. [34] considered the objective function that minimizes the number of acknowledgements and the sum of the delays incurred for all of the packets, i.e. we wish to minimize $f = n + \sum_{i=1}^{n} \sum_{a_j \in \sigma_i} (t_i - a_j)$. It turns out that a simple *Greedy* strategy is optimal for this problem.

Greedy: Send an acknowledgement whenever the total delay of the unacknowledged packets is equal to 1, i.e. equal to the cost of an acknowledgement.

Theorem 23. [34] *The Greedy algorithm is 2-competitive and no deterministic online algorithm can achieve a smaller competitive ratio.*

Noga [59] and independently Seiden [63] showed that no randomized algorithm can achieve a competitive ratio smaller than $e/(e-1) \approx 1.58$ against oblivious adversaries. Karlin et al. [48] presented a randomized strategy that achieves this factor. Let $P(t, t')$ be the set of packets that arrive after time t but up to (and including) time t. The following algorithm works for positive real numbers between 0 and 1. It sends an acknowledgement when, in hindsight, z time units of latency could have been saved by sending an earlier acknowledgement.

Save(z): Let t be the time when the last acknowledgement was sent. Send the next acknowledgement at the first time $t' > t$ such that there is a time τ with $t \leq \tau \leq t'$ and $P(t, t')(t' - \tau) = z$.

Theorem 24. [48] *If z is chosen according to the probability density function $p(z) = e^z/(e-1)$, Save(z) achieves a competitive ratio of $e/(e-1)$.*

Albers and Bals [3] investigate another family of objective functions that penalize long acknowledgement delays of individual data packets more heavily. When TCP is used for interactive data transfer, long delays are not desirable as they are noticeable to a user. Hence we wish to minimize the function $g = n + \max_{1 \leq i \leq n} d_i$, where $d_i = \max_{a_j \in \sigma_i} (t_i - a_j)$ is the maximum delay of any packet in σ_i. The following family of algorithms is defined for any positive real z.

Linear-Delay(z): Initially, set $d = z$ and send the first acknowledgement at time $a_1 + d$. In general, suppose that the i-th acknowledgement has just been sent and that j packets have been processed so far. Set $d = (i+1)z$ and send the $(i+1)$-st acknowledgement at time $a_{j+1} + d$.

Theorem 25. [3] *Setting $z = \pi^2/6 - 1$, Linear-Delay(z) achieves a competitive ratio of $\pi^2/6 \approx 1.644$ and no deterministic strategy can achieve a smaller competitiveness.*

It is well known that $\pi^2/6 = \sum_{i=1}^{\infty} 1/i^2$. Additionally, Albers and Bals [3] investigate a generalization of the objective function g where delays are taken to the p-th power and hence are penalized even more heavily. They proved that the best competitive ratio is an alternating sum of Riemann's zeta function. The ratio is decreasing in p and tends to 1.5 as $p \to \infty$. Frederiksen and Larsen [41] studied a variant of the TCP acknowledgement problem, where it is required that there is some minimum delay between sending two acknowledgements to reflect the physical properties of the network.

6.2 Competitive Auctions

In electronic markets goods are often sold using protocols that resemble classical auctions. The goods available for distribution are not physical but digital and may include e.g. electronic books, software and digital copies of music or movies. The players who are interested in buying such goods send bids to an auctioneer, who then decides which bidders receive goods at which price. The mechanisms by which resources are transferred should be *truthful* and *competitive*, i.e. players should place bids which reflect their true valuations of the goods and the revenue of the auction should be close to the optimal one. There has recently been considerable research interest in designing truthful competitive auctions [22, 35, 42, 43, 44, 55, 56] and we consider two basic settings here.

Lavi and Nisan [56] were among the first who studied truthful auction mechanisms. In their model k identical invisible goods are to be sold. The players arrive online. When player i arrives he has valuations for buying various quantities of the good. More precisely, let $v_i(q)$ be the additional benefit gained from a q-th item of the good. The total valuation from receiving q goods is $\sum_{j=1}^{q} v_i(j)$. We assume $v_i(q+1) \leq v_i(q)$, which is a common assumption in economics. The valuations are only known to the player himself. To buy goods the player sends bids $b_i(q), q = 1, \ldots, k$, where $b_i(q)$ is the bid made for receiving a q-th item. The auctioneer then determines a quantity q_i to be sold to the player as well as a price p_i. The utility of player i is $U_i(q_i, p_i) = \sum_{j=1}^{q_i} v_i(j) - p_i$. As mentioned already before, we are interested in mechanisms where bidders declare their true valuations. More formally a bidding strategy $b_i(q)$ of player i is *dominant* if $U_i(q_i, p_i) \geq U_i(q_i', p_i')$, for any other strategy $b_i'(q)$ that results in quantity q_i' and price p_i'. Using this definition, an auction is called *truthful* if, for each player, declaring true valuations $b_i(q) = v_i(q)$ is a dominant strategy.

Lavi and Nisan give an exact characterization of truthful auctions in the setting under consideration. An auction is based on *supply curves* if before receiving the i-th bids $b_i(q)$, the auctioneer fixes prices $P_i(q)$. The quantity

q_i sold to player i is the value q that maximizes $\sum_{j=1}^{q}(b_i(j) - P_i(j))$ and the prize to be paid is $\sum_{i=1}^{q_i} P_i(j)$.

Theorem 26. [56] *An auction is truthful if and only if it is based on supply curves.*

Lavi and Nisan consider two performance measures of an auction, namely *revenue* and *social efficiency*. Suppose that the valuations are in the range $[p_{\min}, p_{\max}]$. For any auction A and valuation sequence σ, the revenue $R_A(\sigma)$ to the auctioneer is defined as $R_A(\sigma) = \sum_i p_i + p_{\min}(k - \sum_i q_i)$, i.e. we sum up the prices paid by the players and the minimum value of the unsold items. The social efficiency is $E_A(\sigma) = \sum_i \sum_{j=1}^{q_i} v_i(j) + p_{\min}(k - \sum_i q_i)$, i.e. we sum up the valuations of all players and the auctioneer. Lavi and Nisan compare an auction to the k-item Vickrey auction. This offline truthful auction sells the k items to the k highest bids at the price of the $(k+1)$-st highest bid. An online auction A is c-competitive with respect to revenue if, for every valuation sequence σ, $R_A(\sigma) \geq R_{VIC}(\sigma)/c$. Similarly, A is c-competitive with respect to social efficiency if, for every σ, $E_A(\sigma) \geq E_{VIC}(\sigma)/c$.

Based on these definitions Lavi and Nisan present a truthful competitive auction for selling k identical invisible goods. We only have to specify the supply curve.

Discrete-Online-Auction: Let $\Phi = p_{\max}/p_{\min}$. Use the supply curve $P(j) = p_{\min}\Phi^{\frac{j}{k+1}}$.

Theorem 27. [56] *The Discrete-Online-Auction achieves a competitive ratio of* $k\Phi^{\frac{1}{k+1}}$ *with respect to revenue and social efficiency.*

Theorem 28. [56] *The competitive ratio of any truthful online auction with respect to revenue and social efficiency is at least* $\max\{\Phi^{\frac{1}{k+1}}, c\}$*, where* c *is the solution of the equation* $c = \ln(\frac{\Phi-1}{c-1})$.

The second scenario we study here are single-round, sealed-bid competitive auctions as introduces by Goldberg et al. [43]. We first consider the offline problem, which is interesting and instructive in itself. Then we discuss the online variant. There are n players, each of whom is interested in buying one item of a given good. An auctioneer has available n items so that each player can potentially receive one copy. Player $i, 1 \leq i \leq n$, submits a bid b_i representing the maximum amount that he is willing to pay for an item. Given the vector B of bids, the auctioneer computes an allocation $X = (x_1, \ldots, x_n) \in \{0,1\}^n$ and prices $P = (p_1, \ldots, p_n)$. If $x_i = 1$, then player i receives an item, i.e. he *wins*, and pays a cost of p_i. We assume $0 \leq p_i \leq b_i$. If $x_i = 0$, then the player does not receive an item, i.e. he loses, and $p_i = 0$. The utility of player i is $v_i x_i - p_i$. The profit of the auction is $\sum_i p_i$. An auction is *truthful* if, for each player i and any choice of bid values of the other players, the utility of the i-th player is maximized by setting $b_i = v_i$.

Given a bid vector B and an auction A, let $A(B)$ be the profit of A on input B. If A is randomized, then $A(B)$ is a random variable. Goldberg et al. [43] define competitiveness with respect to the *optimal single price omniscient auction* F, which is defined as follows. In a bid vector B, let l_i be i-th largest bid. Auction F determines the largest k such that $k\ell_k$ is maximized. All players with $b_i \geq l_k$ win; the remaining players lose. The profit of F on B is $F(B) = \max_{1 \leq i \leq n} i\ell_i$. A truthful auction is called c-competitive against F if, for all bid vectors B, the expected profit of A on B satisfies $E[A(B)] \geq F(B)/c$.

Goldberg et al. give an exact characterization of truthful auctions based on the notion of *bid-independence*. Let $f_i, 1 \leq i \leq n$, be a family of functions from bid vectors to prices. The deterministic bid-independent auction defined by functions f_i has the following property for each player i.

> Let $p_i = f_i(B_{-i})$, where $B_{-i} = (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n)$. If $b_i \geq p_i$, player i wins at a price of p_i; otherwise player i loses.

Theorem 29. [43] *An auction is truthful if and only if it is bid-independent.*

Goldberg et al. presented and elegant randomized 4-competitive truthful auction which is based on the following cost-sharing mechanism.

Cost-Share(C): Given bid vector B, find the largest k such that the highest k bidders can equally share the cost of C. Charge each C/k.

The actual auction then works as follows.

Sampling-Cost-Sharing:

1. Partition B uniformly at random into two sets, resulting in bid vectors B' and B''.
2. Compute $F' = F(B')$ and $F'' = F(B'')$.
3. Compute the auction results by running *Cost-Share(F')* on B'' and *Cost-Share(F'')* on B'.

Theorem 30. [43] *Sampling-Cost-Sharing is a truthful 4-competitive auction.*

Recently Goldberg and Hartline [42] presented a randomized auction that achieves a competitiveness of 3.39 and uses only two random bits.

Bar-Yossef et al. [16] investigated the online variant of the above problem setting where players arrive one by one. A player has access to all prior bids in determining his own bid. When player i has submitted his bid, the auctioneer must fix a price p_i before any other player arrives. If $p_i \leq b_i$, player i wins; otherwise he loses. In the online scenario an auction A is called bid-independent if the price for player i depends only on the previous bids and not on b_i. That is, for any sequence of bids $b_1, \ldots, b_{i-1}$ and for any two choices of the i-th bid b_i and b_i', $f_i(b_1, \ldots, b_{i-1}, b_i) = f_i(b_1, \ldots, b_{i-1}, b_i')$. Bar-Yossef et al. show that an online auction is truthful if and only if it is bid-independent.

Assume that all bids are in the range $[1, h]$. Furthermore, let $B_{<i} = (b_1, \ldots, b_{i-1})$ be the bids up to player i. Bar-Yossef et al. presented the following randomized auction. The parameter d will be determined later.

Weighted-Interval-Auction(d): Partition the range $[1, h]$ into $l = \lfloor \log h \rfloor + 1$ subintervals $I_0, \ldots, I_{l-1}$ with $I_k = [2^k, 2^{k+1})$. When player i arrives, determine the set of previous players with bids in I_k, for any k. More precisely, let $S_k = \{j \mid j \leq i-1,\ b_j \in I_k\}$ and compute the weight $w_k(B_{<i}) = \sum_{j \in S_k} b_j$. Choose the price $p_i = 2^k$ with probability

$$\text{Prob}[p_i = 2^k] = \frac{w_k(B_{<i})^d}{\sum_{r=0}^{l-1} w_r(B_{<i})^d}.$$

Theorem 31. [16] *Weighted-Interval-Auction(d) is a truthful auction. Restricting to bidding sequences with $F(B) \geq 9h$ and setting $d = \sqrt{\log \log h}$, the competitive ratio is $O(\exp(\sqrt{\log \log h}))$.*

Using methods from learning theory, Blum et al [22] developed a constant competitive truthful auction.

7 Refinements of Competitive Analysis

Competitive analysis is a worst-case performance measure. Unfortunately, for some online problems, the competitive ratios of online algorithms are much higher than the performance ratios observed in practice. The reason is, typically, that a competitive algorithm considers arbitrary request sequences whereas in practice only restricted classes of input occur.

We consider the paging problem in more detail. In Section 2 we saw that the best competitiveness of deterministic online algorithms is equal to k, where k is the number of pages that can be stored in fast memory. Both LRU and FIFO achieve this bound. From a practical point of view the bound of k is not very meaningful as a fast memory can usually store several hundreds or thousands of pages. On the other hand, the performance ratios of LRU and FIFO in practice are much lower. An experimental study by Young [68] reports ratios in the range between 1.5 and 4. Moreover, in practice, LRU performs better than FIFO. This is not evident in competitive analysis, either. In the paging problem standard competitive analysis ignores the fact that request sequences generated by real programs exhibit locality of reference: Whenever a page is requested, the next request is to an associated page.

Borodin et al. [24] introduced *access graphs* for modeling locality of reference. In an access graph the nodes represent the memory pages. Whenever a page p is requested, the next request must be to a page that is is adjacent to p in the access graph. A number of results have been developed in this model [24, 30, 36, 38, 45]. It has been shown that, for any access graph, LRU is never worse than FIFO. For access graphs that are trees, LRU is in fact an

optimal algorithm. Moreover, a number of improved paging algorithms have been proposed that take into account the structure of the access graph.

Karlin et al. [49] modeled locality of reference by assuming that request sequences are generated by a Markov chain. They evaluate paging algorithms in terms of their *fault rate* which is the performance measure preferred by practitioners. In particular, they developed an algorithm that achieves an optimal fault rate, for any Markov chain. Torng [67] analyzed the *total access time* of paging algorithms. He assumes that the service of a request to a page in fast memory costs 1, whereas a fault incurs a penalty of p, $p > 1$. In his model a request sequence exhibits locality of reference if the average length of a subsequence containing requests to m distinct pages is much larger than m.

Recently, Albers et al. [4] proposed another framework for modeling locality of reference that goes back to the working set concept by Denning [32, 33]. In practice, during any phase of execution, a process references only a relatively small fraction of its pages. The set of pages that a process is currently using is called the *working set*. Determining the working set size in a window of size n at any point in a request sequence, one obtains, for variable n, a function that is increasing and concave. Albers et al. restrict the input to request sequences in which the maximum or the average number of distinct pages referenced in windows of size n is bounded by $f(n)$, f being a concave function. They give tight upper and lower bounds on the page fault rates achieved by popular paging algorithms. It shows that LRU is an optimal online algorithm whereas other algorithms, such as FIFO, are not optimal in general.

With respect to arbitrary online problems, other refinements of competitive analysis include extra resource analyses, see e.g. [47, 65], statistical adversaries [28, 60], accomodating functions [26] and the max/max ratio [18]. Koutsoupias and Papadimitriou [54] introduced the *diffuse adversary model*. An adversary must generate an input according to a probability distribution D that belongs to a class Δ of possible distributions known to the online algorithm. We wish to determine, for the given class Δ of distributions, the performance ratio

$$R(\Delta) = \min_{A} \max_{D \in \Delta} \frac{\mathrm{E}_D[A(\sigma)]}{\mathrm{E}_D[OPT(\sigma)]}.$$

Secondly, Koutsoupias and Papadimitriou [54] introduced *comparative analysis*, which compares the performance of online algorithms from given classes of algorithms.

8 Concluding Remarks

In this book chapter we have presented a number of fundamental results in the area of online algorithms and studied some applications that have received a lot of research attention lately. There are several important application

areas that we have not addressed here. Online bin packing is a fundamental problem where we have to pack a sequence of items into bins so that the number of bins is minimized. Problems in online scheduling are still actively investigated. Here a sequence of jobs has to be scheduled on a number of machines so that a given objective function is optimized. Online coloring and online matching are two classical online problems related to graph theory. In these problems, the vertices of a graph arrive online and must be colored resp. matched immediately. The book by Fiat and Woeginger [40] contains a collection of survey articles on these and many other topics. More generally, an excellent text book on online algorithms was written by Borodin and El-Yaniv [23].

References

1. Achlioptas D, Chrobak M, Noga J (2000) Competitive analysis of randomized paging algorithms. Theoretical Computer Science, 234:203–218
2. Albers S (1998) Improved randomized on-line algorithms for the list update problem. SIAM Journal on Computing 27:670–681
3. Albers S, Bals H (2003) Dynamic TCP acknowledgement: Penalizing long delays. In: Proc. 14th ACM-SIAM Symposium on Theory of Computing, 47–55
4. Albers S, Favrholdt LM, Giel O (2002) On paging with locality of reference. In: Proc. 34th ACM Symposium on Theory of Computing, 258–268
5. Albers S, Mitzenmacher M (1998) Average case analyses of list update algorithms, with applications to data compression. Algorithmica 21:312–329
6. Albers S, von Stengel B, Werchner R (1995) A combined BIT and TIMESTAMP algorithm for the list update problem. Information Processing Letters 56:135–139
7. Aiello W, Mansour Y, Rajagopolan S, Rosén A (2000) Competitive queue policies for differentiated services. In: Proc. INFOCOM, 431–440
8. Ambühl C (2001) Offline list update is NP-hard. In: Proc. 8th Annual European Symposium on Algorithms, Springer LNCS 1879, 42–51
9. Ambühl C, Gärtner B, von Stengel B (2001) Towards new lower bounds for the list update problem. Theoretical Computer Science 268:3–16
10. Andelman N, Mansour Y, Zhu A (2003) Competitive queueing policies in QoS switches. In: Proc. 14th ACM-SIAM Symposium on Discrete Algorithms, 761–770
11. Archer A, Papadimitriou C, Talwar K, E. Tardos E (2003) An approximate truthful mechanism for combinatorial auctions with single parameter agents. In: Proc. 14th ACM-SIAM Symposium on Discrete Algorithms, 205–214
12. Azar Y, Richter Y (2003) Management of multi-queue switches in QoS networks. In: Proc. 35th Annual ACM Symposium on Theory of Computing, 82–89
13. Bansal N, Fleischer L, Kimbrel T, Mahdian M, Schieber B, Sviridenko M (2004) Further improvements in competitive guarantees for QoS buffering. In: Proc. 31st International Colloquium on Automata, Languages and Programming, Springer LNCS 3142, 196-207

14. Bartal Y, A. Blum A, Burch C, Tomkins A (1997) A polylog(n)-competitive algorithm for metrical task systems. In: Proc. 29th Annual ACM Symposium on Theory of Computing, 711–719
15. Bartal Y, Bollobás B, Mendel M (2001) A Ramsey-type theorem for metric spaces and its applications for metrical task systems and related problems. In: Proc. 42nd IEEE Annual Symposium on Foundations of Computer Science, 396–405
16. Bar-Yossef Z, Hildrum K, Wu F (2002) Incentive-compatible online auctions for digital goods. In: Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 964–970
17. Belady LA (1966) A study of replacement algorithms for virtual storage computers. IBM Systems Journal 5:78–101
18. Ben-David S, Borodin A (1994) A new measure for the study of on-line algorithms. Algorithmica 11:73–91
19. Ben-David S, Borodin A, Karp RM, Tardos G, Wigderson A (1994) On the power of randomization in on-line algorithms. Algorithmica 11:2–14
20. Bentley JL, Sleator DS, Tarjan RE, Wei VK (1986) A locally adaptive data compression scheme. Communication of the ACM 29:320–330
21. Blum A, Chawla S, A. Kalai A (2003) Static optimality and dynamic search-optimality in lists and trees. Algorithmica 36:249–260
22. Blum A, Kumar V, Rudra A, Wu F (2003) Online learning in online auctions. In: Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms, 202–204
23. Borodin A, El-Yaniv R (1998) Online computation and competitive analysis. Cambridge University Press, Cambridge
24. Borodin A, Irani S, Raghavan P, Schieber B (1995) Competitive paging with locality of reference. Journal of Computer and System Sciences 50:244–258
25. Borodin A, Linial N, Saks M (1992) An optimal online algorithm for metrical task systems. Journal of the ACM 39:745–763
26. Boyar J, Larsen KS, Nielsen MN (2001) The accommodating function: A generalization of the competitive ratio. SIAM Journal on Computing 31:233–258
27. Burrows M, Wheeler DJ (1994) A block-sorting lossless data compression algorithm. DEC SRC Research Report 124
28. Chou A, Cooperstock J, El Yaniv R, Klugerman M, Leighton T (1995) The statistical adversary allows optimal money-making trading strategies. In: Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms, 467–476
29. Chrobak M, Larmore LL (1991) An optimal online algorithm for k servers on trees. SIAM Journal on Computing 20:144–148
30. Chrobak M, Noga J (1999) LRU is better than FIFO. Algorithmica 23:180–185
31. Coppersmith D, Doyle P, Raghavan P, Snir M (1993) Random walks on weighted graphs, and applications to on-line algorithms. Journal of the ACM 40:421–453
32. Denning PJ (1968) The working set model of program behavior. Communications of the ACM 11:323–333
33. Denning PJ (1980) Working sets past and present. IEEE Transactions on Software Engineering 6:64–84
34. Dooly DR, Goldman SA, Scott DS (2001) On-line analysis of the TCP acknowledgment delay problem. Journal of the ACM 48:243–273
35. Fiat A, Goldberg A, Hartline J, Karlin A (2002) Competitive generalized auctions. In: Proc. 34th Annual ACM Symposium on Theory of Computing, 72–81

36. Fiat A, Karlin A (1995) Randomized and multipointer paging with locality of reference. In: Proc. 27th Annual ACM Symposium on Theory of Computing, 626–634
37. Fiat A, Karp RM, McGeoch LA, Sleator DD, Young NE (1991) Competitive paging algorithms. Journal of Algorithms 12:685–699
38. Fiat A, Mendel M (1997) Truly online paging with locality of reference. In: Proc. 38th Annual Symposium on Foundations of Computer Science, 326–335
39. Fiat A, Mendel M (2000) Better algorithms for unfair metrical task systems and applications In: Proc. 32nd Annual ACM Symposium on Theory of Computing, 725–734
40. Fiat A, Woeginger G (1998) *Online Algorithms: The State of the Art,* Springer LNCS 1442
41. Frederiksen JS, Larsen KS (2002) Packet bundling. In: Proc. 8th Scandinavian Workshop on Algorithm Theory, Springer LNCS 2368, 328–337
42. Goldberg A, Hartline J (2003) Competitiveness via consensus. In: Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms, 215–222
43. Goldberg AV, Hartline DS, Karlin A, Wright A (2001) Competitive auctions. Extended version of [44].
44. Goldberg AV, Hartline DS, Wright A (2001) Competitive auctions of digital goods. In: Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms, 735–744
45. Irani S, Karlin AR, Phillips S (1996) Strongly competitive algorithms for paging with locality of reference. SIAM Journal on Computing 25:477–497
46. Irani S, Seiden DS (1998). Randomized algorithms for metrical task systems. Theoretical Computer Science 194:163–182
47. Kalyanasundaram B, Pruhs K (2000) Speed is as powerful as clairvoyance. Journal of the ACM 47:617–643
48. Karlin AR, Kenyon C, Randall D (2003). Dynamic TCP acknowledgement and other stories about $e/(e-1)$. Algorithmica 36:209–224
49. Karlin A, Phillips S, Raghavan P (2000). Markov paging. SIAM Journal on Computing 30:906–922
50. Karp R, Raghavan P (1990) From a personal communication cited in [62].
51. Kesselman A, Lotker Z, Mansour Y, Patt-Shamir B, Schieber B, Sviridenko M (2001) Buffer overflow management in QoS switches. In: Proc. 33rd Annual ACM Symposium on Theory of Computing, 520–529
52. Kesselman A, Mansour Y, van Stee R (2003) Improved competitive guarantees for QoS buffering. In: Proc. 11th European Symposium on Algorithms, Springer LNCS 2832, 361–372
53. Koutsoupias E, Papadimitriou CH (1995) On the k-server conjecture. Journal of the ACM 42:971–983
54. Koutsoupias E, Papadimitriou CH (2000) Beyond competitive analysis. SIAM Journal on Computing 30:300–317
55. Lavi R, Mu'alem A, Nisan N (2003) Towards a characterization of truthful combinatorial auctions. In: Proc. 44th Annual IEEE Symposium on Foundations of Computer Science, 574-583
56. Lavi R, Nisan N (2000) Competitive analysis of incentive compatible on-line auctions. In: Proc. 2nd ACM Conference on Electronic Commerce
57. Manasse MS, McGeoch LA, Sleator DD (1988) Competitive algorithms for on-line problems. In: Proc. 20th Annual ACM Symposium on Theory of Computing, 322–333

58. McGeoch LA, Sleator DD (1991) A strongly competitive randomized paging algorithm. Algorithmica 6:816–825
59. Noga J (2001) Private communication
60. Raghavan P (1991) A statistical adversary for on-line algorithms. In: On-Line Algorithms, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 79–83
61. Raghavan P, Snir M (1994) Memory versus randomization in on-line algorithms. IBM Journal of Research and Development 38:683–708
62. Reingold N, Westbrook J, Sleator DD (1994) Randomized competitive algorithms for the list update problem. Algorithmica 11:15–32
63. Seiden SS (2000) A guessing game and randomized online algorithms. In: Proc. 32nd Annual ACM Symposium on Theory of Computing, 592–601
64. Seiden SS (2001) A general decomposition theorem for the k-server problem. In: Proc. 9th Annual Symposium on Algorithms, Springer LNCS 2161, 86–97
65. Sleator DD, Tarjan RE (1985) Amortized efficiency of list update and paging rules. Communications of the ACM 28:202–208
66. Sleator DD, Tarjan RE. Self-adjusting binary search trees. Journal of the ACM 32:652–686
67. Torng E (1998) A unified analysis of paging and caching. Algorithmica 20:175–200
68. Young N (1994) The k-server dual and loose competitiveness for paging. Algorithmica 11:525–541