Mathematical Programming manuscript No.
(will be inserted by the editor)

Susanne Albers

Online Algorithms: A Survey

Received: date / Revised version: date

Abstract. During the last 15 years online algorithms have received considerable research interest. In this survey we give an introduction to the competitive analysis of online algorithms and present important results. We study interesting application areas and identify open problems.

1. Introduction

The traditional design and analysis of algorithms assumes that an algorithm, which generates output, has complete knowledge of the entire input. However, this assumption is often unrealistic in practical applications. Many of the algorithmic problems that arise in practice are *online*. In these problems the input is only partially available because some relevant input data arrives in the future and is not accessible at present. An online algorithm must generate output without knowledge of the entire input. Online problems arise in many areas of computer science. We give some illustrating examples.

Resource management in operating systems: Paging is a classical online problem where one has to maintain a two-level memory system consisting of a small fast memory and a large slow memory. The goal is to keep actively referenced pages in fast memory without knowing which pages will be requested in the future.

Data structures: Consider a data structure such as a linear linked list or a tree. We wish to dynamically maintain this structure so that a sequence of accesses to elements can be served at low cost. Future access patterns are unknown.

Scheduling: A sequence of jobs must be scheduled on a set of machines so as to optimize a given objective function. Jobs arrive one by one and must be scheduled immediately without knowledge of future jobs.

Networks: Many online problems in this area arise in the context of data transmission. The problem can be, for instance, to dynamically maintain a set of open connections between network nodes without knowing which connections are needed in the future.

The quality of online algorithms is usually evaluated using *competitive analysis*. The idea of competitiveness is to compare the output generated by an online

Institut für Informatik, Albert-Ludwigs-Universität Freiburg Georges-Köhler-Allee 79, 79110 Freiburg, Germany. salbers@informatik.uni-freiburg.de

Mathematics Subject Classification (1991): 68W25, 68W40

algorithm to the output produced by an *optimal offline algorithm*. An optimal offline algorithm is an omniscient algorithm that knows the entire input data in advance and can compute an optimal output. The better an online algorithm approximates the optimal solution, the more competitive this algorithm is.

In the following we first present fundamental concepts used to study online algorithms. Then we study various online problems and present important results. The specific problems we consider include paging, self-organizing data structures, the k-server problem, metrical task systems, scheduling and load balancing as well as problems in large networks. Finally we address refinements of competitive analysis and conclude with some remarks.

2. Basic concepts

Formally, many online problems can be described as follows. An online algorithm A is presented with a *request sequence* $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$. The requests $\sigma(t)$, $1 \leq t \leq m$, must be served in the order of occurrence. When serving request $\sigma(t)$, algorithm A does not know any request $\sigma(t')$ with $t' > t$. Serving requests incurs cost and the goal is to minimize the total cost paid on the entire request sequence. This setting can also be regarded as a *request-answer game*: An adversary generates requests and an online algorithm has to serve them one at a time.

To illustrate this formal model we re-consider the paging problem and start with a precise definition.

The paging problem: Consider a two-level memory system that consists of a small fast memory and a large slow memory. We assume that the fast memory can simultaneously store k memory pages and that the slow memory can hold potentially infinitely many pages. Each request specifies a page in the memory system. A request is served if the corresponding page is in fast memory. If a requested page is not in fast memory, a *page fault* occurs. Then a page must be moved from fast memory to slow memory so that the requested page can be loaded into the vacated location. A paging algorithm specifies which page to evict on a fault. If the algorithm is online, then the decision which page to evict must be made without knowledge of any future requests. The cost to be minimized is the total number of page faults incurred on the request sequence.

Sleator and Tarjan [93] suggested evaluating the performance of an online algorithm using *competitive analysis*. In a competitive analysis, an online algorithm A is compared to an *optimal offline algorithm*. An optimal offline algorithm knows the entire request sequence in advance and can serve it with minimum cost. Given a request sequence σ, let $A(\sigma)$ denote the cost incurred by A and let $OPT(\sigma)$ denote the cost incurred by an optimal offline algorithm OPT. The algorithm A is called c-competitive if there exists a constant a such that $A(\sigma) \leq c \cdot C_{OPT}(\sigma) + a$ for all request sequences σ. Here we assume that A is a deterministic online algorithm. Note that competitive analysis is a strong worst case performance measure in the sense that a competitive algorithm must perform well on all inputs.

With respect to the paging problem, there are two well-known deterministic online algorithms.

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 [93] analyzed the performance of LRU and FIFO and showed that on any request sequence the number of page faults incurred by these algorithms is bounded by k times the number of faults made by OPT. They also showed that this is optimal.

Theorem 1. [93] *LRU and FIFO are k-competitive.*

Theorem 2. [93] *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 [25]. 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 achieves the minimum number of page faults.

A natural question is: Can an online algorithm achieve a better competitive ratio if it is allowed to use randomization?

The competitive ratio of a randomized online algorithm A 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 A. 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? Ben-David et al. [27] introduced three kinds of adversaries.

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 is charged the optimum offline cost for that sequence.

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

Given a randomized online algorithm A and an adaptive online (adaptive offline) adversary ADV, let $E[A(\sigma)]$ and $E[ADV(\sigma)]$ denote the expected costs incurred by A and ADV in serving a request sequence generated by ADV. A

randomized online algorithm A is called c-competitive against any adaptive online (adaptive offline) adversary if there is a constant a such that for all adaptive online (adaptive offline) adversaries ADV, $E[A(\sigma)] \leq c \cdot E[ADV(\sigma)] + a$, where the expectation is taken over the random choices made by A.

Ben-David et al. [27] investigated the relative strength of the adversaries and showed the following statements.

Theorem 3. [27] *If there is a randomized online algorithm that is c-competitive against any adaptive offline adversary, then there also exists a c-competitive deterministic online algorithm.*

Theorem 4. [27] *If A is a c-competitive randomized algorithm against any adaptive online adversary and if there is a d-competitive algorithm against any oblivious adversary, then A is $(c \cdot d)$-competitive against any adaptive offline adversary.*

Theorem 3 implies that randomization does not help against the adaptive offline adversary. An immediate consequence of the two theorems above is:

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

Against oblivious adversaries, randomized online paging algorithms can considerably improve the ratio of k shown for deterministic paging. The following algorithm was proposed by Fiat et al. [55].

Marking: The algorithm 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, a page is chosen uniformly at random from among the unmarked pages in fast memory, and that page 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.

Fiat et al. [55] analyzed the performance of the *Marking* algorithm and showed that it is $2H_k$-competitive against any oblivious adversary, where $H_k = \sum_{i=1}^{k} 1/i$ is the k-th Harmonic number. Note that H_k is roughly $\ln k$.

Fiat et al. [55] also proved that no randomized online paging algorithm against any oblivious adversary can be better than H_k-competitive. Thus the *Marking* algorithm is optimal, up to a constant factor. More complicated paging algorithms achieving an optimal competitive ratio of H_k were given in [81,1].

3. Self-organizing data structures

The *list update problem* is one of the first online problems that were studied with respect to competitiveness. The problem is to maintain a dictionary as an unsorted linear list. Consider a set of items that is represented as a linear linked list. We receive a request sequence σ, where each request is one of the following operations. (1) It can be an *access* to an item in the list, (2) it can be an *insertion* of a new item into the list, or (3) it can be a *deletion* of an item. To access an

item, a list update algorithm starts at the front of the list and searches linearly through the items until the desired item is found. To insert a new item, the algorithm first scans the entire list to verify that the item is not already present and then inserts the item at the end of the list. To delete an item, the algorithm scans the list to search for the item and then deletes it.

In serving requests a list update algorithm incurs cost. If a request is an access or a delete operation, then the incurred cost is i, where i is the position of the requested item in the list. If the request is an insertion, then the cost is $n+1$, where n is the number of items in the list before the insertion. While processing a request sequence, a list update algorithm may rearrange the list. Immediately after an access or insertion, the requested item may be moved at no extra cost to any position closer to the front of the list. These exchanges are called *free exchanges*. Using free exchanges, the algorithm can lower the cost on subsequent requests. At any time two adjacent items in the list may be exchanged at a cost of 1. These exchanges are called *paid exchanges*. The goal is to serve the request sequence so that the total cost is as small as possible.

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*. More precisely, a deterministic online algorithm for list update is called c-competitive if there is a constant a such that for all size lists and all request sequences σ, $A(\sigma) \leq c \cdot OPT(\sigma) + a$.

Linear lists are one possibility for representing a set of items. Certainly, there are other data structures such as balanced search trees or hash tables that, depending on the given application, can maintain a set in a more efficient way. In general, linear lists are useful when the set is small and consists of only a few dozen items. Recently, list update techniques have been applied very successfully in the development of data compression algorithms [8,28,34].

There are three well-known deterministic online algorithms for the list update problem.

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.

The formulations of list update algorithms generally assume that a request sequence consists of accesses only. It is obvious how to extend the algorithms so that they can also handle insertions and deletions. On an insertion, the algorithm first appends the new item at the end of the list and then executes the same steps as if the item was requested for the first time. On a deletion, the algorithm first searches for the item and then just removes it.

In the following, we discuss the algorithms Move-To-Front, Transpose and Frequency-Count. We note that Move-To-Front and Transpose are *memoryless* strategies, i.e. they do not need any extra memory to decide where a requested item should be moved. Thus, from a practical point of view, they are more at-

tractive than Frequency-Count. Sleator and Tarjan [93] analyzed the competitive ratios of the three algorithms.

Theorem 5. [93] *The Move-To-Front algorithm is 2-competitive.*

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

Karp and Raghavan [72] developed a lower bound on the competitiveness that can be achieved by deterministic online algorithms. This lower bound implies that Move-To-Front has an optimal competitive ratio.

Theorem 6. [72] *Let A be a deterministic online algorithm for the list update problem. If A is c-competitive, then* $c \geq 2$.

Ambühl [10] showed that the offline variant of the list update problem is NP-hard. Thus, in contrast to the paging problem, there is no efficient algorithm for computing an optimal service schedule for a given input.

Next we address the problem of randomization in the list update problem. Against adaptive adversaries, no randomized online algorithm for list update can be better than 2-competitive, see [27, 86]. Thus we concentrate on algorithms against oblivious adversaries. Many randomized algorithms for list update have been proposed [2, 9, 64, 86]. We present the two most important algorithms. Reingold et al. [86] gave a very simple algorithm, called *Bit*.

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 7. [86] *The Bit algorithm is 1.75-competitive against any oblivious adversary.*

Reingold et al. [86] generalized the Bit algorithm and proved an upper bound of $\sqrt{3} \approx 1.73$ against oblivious adversaries. The best randomized algorithm currently known is a combination of the *Bit* algorithm and 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.

As an example, consider a list of six items being in the order $L : x_1 \to x_2 \to x_3 \to x_4 \to x_5 \to x_6$. Suppose that algorithm TS has to serve the second request to x_5 in the request sequence $\sigma = \ldots x_5, x_2, x_2, x_3, x_1, x_1, x_5$. Items x_3 and x_4 were requested at most once since the last request to x_5, whereas x_1 and x_2 were both requested twice. Thus, TS will insert x_5 immediately in front of x_3 in the list. A combination of *Bit* and *TS* was proposed by [9].

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

Theorem 8. [9] *The algorithm Combination is 1.6-competitive against any oblivious adversary.*

Ambühl et al. [11] presented a lower bound for randomized list update algorithms.

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

An interesting open problem is to determine tight bounds on the competitive ratio that can be achieved by randomized online algorithms against oblivious adversaries.

Using techniques from learning theory, Blum et al. [30] recently 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.

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 [94]. 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. This reorganization of the tree is called *splaying*.

Sleator and Tarjan [94] analyzed splay trees and proved a series of interesting results. They showed that the amortized asymptotic time of access and update operations is as good as the corresponding time of balanced trees. More formally, in an n-node splay tree, the amortized time of each operation is $O(\log n)$.

Theorem 10. [94] *Splay trees are $O(\log n)$-competitive.*

It was also shown [94] that on any sequence of accesses, a splay tree is as efficient as the optimum static search tree.

Theorem 11. [94] *Splay trees are $O(1)$-competitive against optimal static search trees.*

Moreover, Sleator and Tarjan [94] presented a series of conjectures, some of which have been resolved or partially resolved [45, 46, 95]. On the other hand, 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. Blum et al. [30] showed that there is an $O(1)$-competitive algorithm if the online algorithm is allowed to make free rotations after each request.

4. The k-server problem

The k-server problem is one of the most fundamental problems in the theory of online algorithms. In the k-server problem we are given a metric space S and k mobile servers that reside on points in S. Each request specifies a point $x \in S$. To serve a request, one of the k servers 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 x and y. The goal is to serve a sequence of requests so that the total distance traveled by all servers is as small as possible.

The k-server problem contains paging as a special case. 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 also models more general caching problems, where the cost of loading an item into fast memory depends on the size of the item. Such a situation occurs, for instance, when font files are loaded into the cache of a printer. More generally, the k-server problem can also be regarded as a vehicle routing problem.

The k-server problem was introduced in 1988 by Manasse et al. [80] who also showed a lower bound for deterministic k-server algorithms.

Theorem 12. [80] *Let A be a deterministic online k-server algorithm in an arbitrary metric space. If A is c-competitive, then $c \geq k$.*

In their seminal paper Manasse et al. [80] also conjectured that there exists a deterministic k-competitive online k-server algorithm. Seven years later Koutsoupias and Papadimitriou [78] showed that there is a $(2k-1)$-competitive algorithm and hence achieved a breakthrough. Before, k-competitive algorithms were known for special metric spaces (e.g. trees [39] and resistive spaces [47]) and special values of k ($k = 2$ and $k = n-1$, where n is the number of points in the metric space [80]). It is worthwhile to note that the greedy algorithm, which always moves the closest server to the requested point, is not competitive.

The algorithm analyzed by Koutsoupias and Papadimitriou is the *Work Function* algorithm. 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. [78] *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.

The performance of randomized online algorithms is not as well understood. In particular no randomized algorithm is known that has a competitiveness

smaller than $2k-1$ in arbitrary metric spaces. An elegant randomized strategy for moving servers was proposed by Raghavan and Snir [85].

Harmonic: Suppose that there is a new request at point r and that server s_i, $1 \leq i \leq k$, is currently at point x_i. Let $d_i = dist(x_i, r)$ be the distance between x_i and r. Move server s_i with probability $p_i = 1/(d_i \sum_{j=1}^{k} \frac{1}{d_j})$ to the request.

Intuitively, the closer a server is to the request, the higher the probability that it will be moved. Bartal and Grove [24] proved that the *Harmonic* algorithm achieves a competitive ratio of $c \leq \frac{5}{4}k \cdot 2^k - 2k$ against adaptive online adversaries. Against these adversaries no randomized online algorithm can achieve a competitive ratio smaller than k [85]. The competitiveness of *Harmonic* is not better than $k(k+1)/2$, see [85]. The algorithm has a competitive ratio of 3, for $k = 3$, and of $k(k+1)/2$ in metric spaces consisting of $k+1$ points [40,85]. Against lazy adversaries *Harmonic* achieves a competitiveness of $k(k+1)/2$ [22]. An adversary is lazy if, whenever one of its servers is located on a point not covered by the online algorithm's servers, it requests that point. It was conjectured that lazy adversaries achieve the highest possible competitive ratio against randomized memoryless online algorithms that only move one of their servers if the requested point is not already covered by a server. However, Peserico [83] disproved this conjecture.

For randomized algorithms against oblivious adversaries the best lower bound currently known is due to Bartal et al. [20]

Theorem 14. [20] *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 $\epsilon > 0$, [20]. It is conjectured that $\Theta(\log k)$ is the true competitiveness of randomized algorithms against oblivious adversaries. Bartal et al. [21] presented an algorithm that has a competitive ratio of $O(c^6 \log^6 k)$ in metric spaces consisting of $k + c$ points. Seiden [90] 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. The main open problem in the area of the k-server problem is to develop randomized online algorithms that have a competitive ratio of $c < k$ in an arbitrary metric space.

5. Metrical task systems

Metrical task systems were introduced by Borodin et al. [33] and represent a framework for modeling a large class of on-line problems. The definition of task systems is motivated by the observation that in many computer systems there are several ways to execute a given job.

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 set S of 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, the function 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. [33] settled the competitiveness of deterministic online algorithms.

Theorem 15. [33] *There exists a deterministic online algorithm that is* $(2n-1)$*-competitive for any metrical task system with* n *states.*

Theorem 16. [33] *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.*

It is worthwhile to notice that the competitive factor of $2n-1$ for deterministic online algorithms 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, the known bounds are tight up to a logarithmic factor.

Theorem 17. [57] *There exists a randomized online algorithm that is* $O(\log^2 n / \log^2 \log n)$*-competitive against any oblivious adversary, for any metrical task system with* n *states.*

Theorem 18. [20] *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 oblivous 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 at al. [33] 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 [67] and is equal to $H_n + O(\sqrt{\log n})$.

6. Scheduling and load balancing

Scheduling is a classical and well-studied problem that still receives a lot of research interest. The general situation in *online scheduling* is as follows. We

are given a set of m machines. A sequence of jobs $\sigma = J_1, J_2, \ldots, J_n$ arrives online. Each job J_k has a processing p_k time that may or may not be known in advance. Whenever a new job arrives, it has to be scheduled immediately on one of the m machines. The goal is to optimize a given objective function. There are many problem variants: Preemption of jobs may or may not be allowed; we can study various machine types and various objective functions. A very large number of different problems have been investigated in the literature and we can only discuss a few basic scenarios in this survey.

First we consider one of the most basic problems in online scheduling. Suppose that we are given m *identical* machines. The jobs $\sigma = J_1, J_2, \ldots, J_n$ arrive one by one. Whenever the scheduler is presented with a new job, its processing time is known in advance. Preemption of jobs is not allowed. We wish to minimize the *makespan*, which is the completion time of the last job that finishes in the schedule.

Graham [63] in 1966 proposed the elegant *Greedy* algorithm and analyzed its performance.

Greedy: Assign a new job to the least loaded machine.

Theorem 19. [63] *Greedy is* $(2 - \frac{1}{m})$*-competitive.*

Graham also showed that the competitive ratio of *Greedy* is not smaller than $2 - \frac{1}{m}$. In recent years, research has focused on finding algorithms that achieve a competitive ratio asymptotically smaller than 2. In 1992, Bartal et al. [23] gave an algorithm that is 1.986-competitive. This bound was improved to 1.945, to 1.923 and finally to 1.9201, which is the best upper bound known to date [69, 3, 60]. All the algorithms are deterministic. The best lower bound currently known is due to Rudin [87]. He proved that no deterministic online algorithm can be better 1.88-competitive. An interesting open problem is to close the gap between the lower and the upper bounds.

Since the publication of the paper by Bartal et al. [23], there has also been research interest in developing randomized online algorithms for the above scheduling problem. Bartal et al. gave a randomized algorithm for 2 machines that achieves an optimal competitive ratio of 4/3. Chen et al. [36] and Sgall [91] proved that no randomized online algorithm can have a competitiveness smaller than $1/(1-(1-1/m)^m)$. This expression tends to $e/(e-1) \approx 1.58$ as $m \to \infty$. Seiden [88] presented a randomized algorithm whose competitive ratio is smaller than the best known deterministic ratio for $m \in \{3, \ldots, 7\}$. The competitiveness is also smaller than the deterministic lower bound for $m = 3, 4, 5$.

Recently, Albers [4] developed a randomized online algorithm that is 1.916-competitive, for all m, and hence gave the first algorithm that performs better than known deterministic algorithms for general m. She also showed that a performance guarantee of 1.916 cannot be proven for a deterministic online algorithm based on analysis techniques that have been used in the literature so far. An interesting feature of the new randomized algorithm, called *Rand*, is that at most two schedules have to be maintained at any time. In contrast, the algorithms by Bartal et al. [23] and by Seiden [90] have to maintain t schedules

when t jobs have arrived. The *Rand* algorithm is a combination of two deterministic algorithms A_1 and A_2. Initially, when starting the scheduling process, *Rand* chooses A_i, $i \in \{1, 2\}$, with probability $\frac{1}{2}$ and then serves the entire job sequence using the chosen algorithm. Algorithm A_1 is a conservative strategy that tries to maintain schedules with a low makespan. On the other hand, A_2 is an aggressive strategy that aims at generating schedules with a high makespan. A challenging open problem is to design randomized online algorithms that beat the deterministic lower bound, for all m.

We next consider some variants of the basic scenario studied so far.

Identical machines, restricted assignment: We have a set of m identical machines, but each job can only be assigned to one of a subset of admissible machines. Azar et al. [18] showed that the *Greedy* algorithm, which always assigns a new job to the least loaded machine among the admissible machines, achieves a competitiveness of $\lceil \log_2 m \rceil + 1$. They also proved that no deterministic online algorithm can be better than $\lceil \log_2 m \rceil$-competitive.

Related machines: Each machine i has a speed s_i, $1 \leq i \leq m$. The processing time of job J_k on machine i is equal to p_k/s_i. Aspnes et al. [13] showed that the *Greedy* algorithm, that always assigns a new job to a machine so that the load after the assignment is minimized, is $\Theta(\log m)$-competitive. They also presented an algorithm that is 8-competitive. The bound was improved to 5.828 in [30].

Unrelated machines: The processing time of job J_k on machine i is $p_{k,i}$, $1 \leq k \leq n$, $1 \leq i \leq m$. Aspnes et al. [13] showed that *Greedy* is only m-competitive. However, they also gave an algorithm that is $O(\log m)$-competitive.

In *online load balancing* we have again a set of m machines and a sequence of jobs $\sigma = J_1, J_2, \ldots, J_n$ that arrive online. Here, each job J_k has a *weight* $w(k)$ and an unknown duration. For any time t, let $l_i(t)$ denote the load of machine i, $1 \leq i \leq m$, at time t, which is the sum of the weights of the jobs present on machine i at time t. The goal is to minimize the maximum load that occurs during the processing of σ.

For the scenario with m identical machines, Azar and Epstein [16] showed that the *Greedy* algorithm is $(2 - \frac{1}{m})$-competitive. The load balancing problem becomes more complicated with *restricted assignments*, i.e. each job can only be assigned to a subset of admissible machines. Azar et al. [15] proved that *Greedy* achieves a competitive ratio of $m^{2/3}(1 + o(1))$. They also proved that no online algorithm can be better than $\Omega(\sqrt{m})$-competitive. In a subsequent paper, Azar et al. [17] gave a matching upper bound. The algorithm works as follows.

Robin Hood: Let OPT be the optimum load achieved by the offline algorithm. *Robin Hood* maintains an estimate L for OPT satisfying $L \leq OPT$. At any time t, machine i is called *rich* if $l_i(t) \geq \sqrt{m}L$; otherwise machine i is called *poor*. When a new job J_k arrives, L is updated, i.e. $L := \max\{L, w(k), \frac{1}{m}(w(k) + \sum_{i=1}^{m} l_i(t))\}$. If possible, J_k is assigned to a poor machine. Otherwise it is assigned to the rich machine that became rich most recently.

Theorem 20. [17] *Robin Hood is $O(\sqrt{m})$-competitive.*

For related machines an upper bound of 20 and a lower bound of $3 - o(1)$ on the competitive ratio are known [17]. Recently, Armon [12] settled the complexity for unrelated machines. They proved a lower bound of $\Omega(m/\log m)$ on the competitiveness of any deterministic online algorithm, almost matching the trivial upper bound of $O(m)$ of the *Greedy* algorithm. We refer the reader to [14,92] for excellent surveys on online scheduling and load balancing.

7. Large networks

With the advent of the world-wide web, researchers have started investigating algorithmic problems that arise in large networks. Many of these problems are online and we discuss some selected problems.

7.1. Generalized caching

We consider the caching of web documents. Caches can be maintained by web clients or servers. Storing actively accessed documents in local caches can substantially reduce user response times as well as the network congestion because requested documents do not have to be transmitted repeatedly over the web. Web caching problems differ from standard paging problems in that documents have varying sizes and incur varying costs when downloaded into a local cache. The loading cost depends, for instance, on the size of the documents and on the current congestion in the network.

In *generalized caching* we have again a two-level memory system consisting of a fast and a slow memory. In the network setting, the fast memory is a local cache; the slow memory is the memory of the remaining network, i.e. the universe of all documents accessible in the network. We assume that the fast memory has a capacity of K. For any document d, let $size(d)$ be the size and $cost(d)$ be the cost of d. The total size of the pages in fast memory may never exceed K. If a requested document is not in cache, the incurred cost is $cost(d)$. The goal is to serve a sequence of requests so that the total loading cost is as small as possible. Various cost models have been proposed in the literature.

1. *The Bit Model* [65]: For each document d, we have $cost(d) = size(d)$. (The delay in bringing the document into fast memory depends only upon its size.)
2. *The Fault Model* [65]: For each document d, we have $cost(d) = 1$ while the sizes can be arbitrary.
3. *The Cost Model*: For each document d, we have $size(d) = 1$ while the costs can be arbitrary.
4. *The General Model*: For each document d, both the cost and size can be arbitrary.

Note that generalized caching is a problem that arises in networks but the network topology is not directly part of a problem instance. It is captured only implicitly in the cost of downloading a document.

For the Bit and the Fault models, the LRU strategy is $(k+1)$-competitive [52], where k is the ratio of K to the size of the smallest document ever requested. This bound holds in a relaxed caching scenario where the requested document does not necessarily have to be brought into fast memory, which is an option in web applications. The performance ratio of $k+1$ is optimal for deterministic algorithms. For the Bit and the Fault Model, Irani presented randomized $O(\log^2 k)$-competitive online algorithms. Caching in the Cost Model is also known as *weighted caching*, which is a special instance of the k-server problem. Young [98] gave a K-competitive online algorithm for the General Model.

Landlord: For each d in fast memory, the algorithm maintains a variable $credit(d)$ that takes values between 0 and $cost(d)$. If a requested document d is already in fast memory, then $credit(d)$ is reset to any value between its current value and $cost(d)$. If the requested page is not in fast memory, then the following two steps are executed until there is enough room to load d. (1) For each document d' in fast memory, decrease $credit(d')$ by $\Delta \cdot size(d')$, where $\Delta = \min_{d' \in F} credit(d')/size(d')$ and F is the set of documents in fast memory. (2) Evict any document d' from fast memory with $credit(d') = 0$. When there is enough room, load d and set $credit(d)$ to $cost(d)$.

Theorem 21. [98] *Landlord is K-competitive in the General Model.*

The above bound is optimal. An interesting problem is to develop randomized online algorithms for generalized caching. For the Bit and the Fault Model it would be nice to design algorithms with improved competitive ratios. In the General Model we are interested in $o(K)$-competitive randomized algorithms. This is a challenging problem as it involves finding $o(k)$-competitive algorithms for the k-server problem.

7.2. Maintaining TCP connections

We study two algorithmic problems that arise in the context of maintaining open TCP connections.

Cohen et al. [43] initiated the theoretical study of *connection caching* in the world-wide web. Communication between clients and servers in the web is performed using HTTP (Hyper Text Transfer Protocol), which in turn uses TCP (Transmission Control Protocol) to transmit data. The current protocol HTTP/1.1 works with *persistent connections*, i.e. once a TCP connection is established it may be kept open and used for transmission until the connection is explicitly closed by one of the endpoints. Of course, each network node can simultaneously maintain only a limited number of open TCP connections. If a connection is closed, there is a mechanism by which one endpoint can signal the close to the other endpoint [59].

Formally, in connection caching, we are given a network modeled as an undirected graph G. The nodes of the graph represent the nodes in the network. The edges represent the possible connections. Each node has a cache in which

it can maintain information on *open* connections. A connection $c = (u, v)$ is open if information on c is stored in the caches of both u and v. For a node v, let $k(v)$ denote the number of open connections that v can maintain simultaneously. Let k be the size of the largest cache in the network. For a connection $c = (u, v)$, let $cost(c)$ be the *establishment cost* of c that is incurred when c is opened. An algorithm for connection caching is presented with a request sequence $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$, where each request $\sigma(t)$ specifies a connection $c_t = (u_t, v_t)$, $1 \leq t \leq m$. If the requested connection c_t is already open, then the request can be served at cost 0; otherwise the connection has to be opened at a cost of $cost(c_t)$. The goal is to serve the request sequence σ so that the total cost is as small as possible.

An important feature of this problem is that local cache configurations are not independent of each other. When one endpoint of an open connection decides to close the connection, then the other endpoint also cannot use that connection anymore.

Cohen et al. [43] investigated *uniform connection caching* where the connection establishment cost is uniform for all the connections. They first showed that any c-competitive algorithm for standard paging can be transformed into a $2c$-competitive algorithm for uniform connection caching. Each local node simply executes a paging strategy ignoring notifications of connections that were closed by other nodes. Using LRU or FIFO, we obtain $2k$-competitive algorithms. Cohen et al. [44] also considered deterministic *Marking* strategies, which work in the same way as their randomized counterparts except that on a fault an *arbitrary* unmarked page may be evicted.

Theorem 22. [44] *Deterministic Marking strategies can be implemented in uniform connection caching such that a competitive ratio of k is achieved. For each request, at most 1 bit of extra communication is exchanged between the two corresponding network nodes.*

Obviously, the above performance is optimal since the lower bound of k for deterministic standard paging carries over to uniform connection caching. Cohen et al. [44] also investigated randomized *Marking* strategies and showed that they are $4H_k$-competitive against oblivious adversaries.

In [5] Albers investigated *generalized connection caching* where the connection establishment cost can be different for the various connections. She showed that the *Landlord* algorithm known for generalized caching can be adapted so that it achieves an optimal competitiveness. The implementation is as follows.

Landlord: For each cached connection c, the algorithm maintains a credit value $credit(c)$ that takes values between 0 and $cost(c)$. Whenever a connection is opened, $credit(c)$ is set to $cost(c)$. If a requested connection (u, v) is not already open, then each node $w \in \{u, v\}$ that currently has $k(w)$ open connections executes the following steps. Let $\delta = \min_{c \text{ open at } w} credit(c)$. Close a connection c_w at w with $credit(c_w) = \delta$ and decrease the credit of all the other open connections at w by δ.

Theorem 23. [5] *Landlord is k-competitive for generalized connection caching.*

Ideally, we implement *Landlord* in a distributed fashion such that, for each open connection $c = (u, v)$, both endpoints u and v keep their copies of $credit(c)$. If one endpoint, say u, reduces the credit by δ, then this change has to be communicated to v so that v can update its $credit(c)$ value accordingly. The amount of extra communication for an open connection can be large if the repeated δ reductions are small. It is possible to reduce the amount of extra communication at the expense of increasing slightly the competitiveness of the algorithm. For any $0 < \epsilon \leq 1$, *Landlord* can be modified so that it is $(1 + \epsilon)k$-competitive and uses at most $\lceil \frac{1}{\epsilon} \rceil - 1$ bits of extra communication for each open connection [5]. Setting $\epsilon = 1$, we obtain a $2k$-competitive algorithm that does not use any extra communication. For $\epsilon = 1/2$, the resulting algorithm is $\frac{3}{2}k$-competitive and uses only one bit of extra communication.

Interestingly no extra communication is necessary if we are willing to use randomization. It is possible to implement the *Harmonic* algorithm for the k-server problem in such a way that it does not need any extra communication between network nodes. The implementation achieves a competitiveness of k against adaptive online adversaries [5].

Secondly in this section we study a *dynamic TCP acknowledgement problem*. Consider an open TCP connection between two network nodes that wish to exchange data. The data is partitioned into segments or *packets* that are sent across the connection. A node receiving data must acknowledge the arrival of each incoming packet so that the sending node is notified that the transmission was successful; lost packets must be retransmitted. However, data packets do not have to be acknowledged individually. Instead, most TCP implementations employ some delay mechanism that allows the TCP to acknowledge multiple incoming packets with a single acknowledgement and, possibly, to piggyback the acknowledgement on an outgoing data segment. Reducing the number of acknowledgements has several advantages, e.g. the overhead incurred at the network nodes for sending and receiving acknowledgements is reduced and, more importantly, the network congestion is reduced. On the other hand, by reducing the number of acknowledgements, one adds latency to a TCP connection, which is not desirable. The goal is to balance the reduction in the number of acknowledgements with the increase in latency.

Motivated by the fact that TCP supports dynamic acknowledgement mechanisms, Dooly et al. [51] formulated the following problem. A network node receives a sequence of n data packets. Let a_i denote the arrival time of packet i, $1 \leq i \leq n$. At time a_i, the arrival times a_j, $j > i$, are not known. We have to partition the sequence $\sigma = (a_1, \ldots, a_n)$ of packet arrival times into m subsequences $\sigma_1, \ldots, \sigma_m$, for some $m \geq 1$, 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. [51] 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 $f = m + \sum_{i=1}^{m} \sum_{a_j \in \sigma_i} (t_i - a_j)$. They analyzed the following algorithm.

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

Theorem 24. [51] *The Greedy algorithm is 2-competitive and this is the best competitive ratio a deterministic online algorithm can achieve.*

Karlin et al. [70] studied randomized algorithms and proved the following result.

Theorem 25. [70] *There exists a randomized online strategy that achieves a competitiveness of* $e/(e-1) \approx 1.58$ *against oblivious adversaries.*

Noga [82] and independently Seiden [89] showed that no randomized algorithm can do better.

Dooly et al. [51] also studied the minimization of a second objective function $f' = m + \sum_{i=1}^{m} \max_{a_j \in \sigma_i} (t_i - a_j)$ where one considers the sum of the maximum delays incurred in subsequences σ_i in addition to the number of acknowledgements sent. They showed that the best competitive ratio of a deterministic online algorithm is equal to 2.

In [6] Albers and Bals investigate a new family of objective functions that penalize long acknowledgement delays of individual data packets more heavily. In applications where a TCP connection is used for interactive data transfer, long delays are not desirable as they are noticeable to a user. Hence [6] studies the objective function that minimizes the number of acknowledgements and the maximum delay incurred for any of the data packets. Given an input σ, consider a partitioning $\sigma_1, \ldots, \sigma_m$. Let $d_i = \max_{a_j \in \sigma_i} (t_i - a_j)$ be the maximum delay of any packet in σ_i, $1 \leq i \leq m$. We wish to minimize the function $g = m + \max_{1 \leq i \leq m} d_i$. 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 26. [6] *For any* z *with* $z \geq 1/2$, *Linear-Delay(z) is* c*-competitive, where* $c = \max\{1+z, (1+z)/(2+z-\pi^2/6)\}$. *Setting* $z = \pi^2/6 - 1$ *the resulting algorithm achieves a competitive ratio of* $\pi^2/6 \approx 1.644$.

It is well known that $\pi^2/6 = \sum_{i=1}^{\infty} 1/i^2$. This performance ratio cannot be improved.

Theorem 27. [6] *No deterministic online algorithm can achieve a competitive ratio smaller than* $\pi^2/6$.

Additionally, Albers and Bals [6] investigate a generalization of the objective function g where delays are taken to the p-th power and hence are penalized even more heavily. Again, they present tight upper and lower bounds on the best possible competitiveness of deterministic algorithms. 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 $p \to \infty$. An interesting open problem is to develop randomized online algorithms for the objective functions g and its generalization. Some initial lower bounds were given in [6].

Frederiksen and Larsen [61] studied a modified version 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.

7.3. Routers and switches

Routers and switches handle the data traffic in networks and ensure that data packets sent over connections reach their correct destination. Typically, traffic is *bursty*, i.e. the number of packets that reach a buffer or switch during a certain time interval exceeds the number of packets that can be processed during that interval. This leads to packet loss, which is not desirable as the corresponding packets have to be resent. To reduce packet loss, routers and switches are equipped with buffers in which packets can be stored temporarily until they are forwarded. We study two algorithmic problems related to the maintenance of such buffers.

Bar-Noy et al. [19] and independently Koga [77] address the question how large buffers should be in order to avoid packet loss. Consider n data streams that share a common output channel at a router. The data is partitioned into packets of equal size. At time t, $N(t,i)$ packets of stream i arrive, $1 \leq t \leq m$ and $1 \leq i \leq n$. Associated with each data stream is a FIFO queue of potentially infinite capacity, in which the packets of the stream can be stored. In each time step a scheduling algorithm in the router can select one of the queues and send the packet at the head over the output channel. The goal is to minimize the maximum queue length that ever occurs at any of the queues.

Bar-Noy et al. [19] and Koga [77] gave tight lower and upper bounds on the best possible competitiveness.

Theorem 28. [19,77] *Any deterministic online algorithm has a competitive ratio of* $\Omega(\log n)$.

Koga showed that the popular *Round Robin* algorithm is not better than n-competitive. A natural greedy algorithm works as follows.

Longest Queue First: Always serve the longest queue, ties can be broken arbitrarily.

Theorem 29. [19,77] *Longest Queue First is* $O(\log n)$*-competitive.*

Thus the greedy algorithm achieves an optimal competitive ratio. The *Longest Queue First* algorithm was proposed and analyzed by Koga. Bar-Noy et al. considered a slight variant of that algorithm. Koga also showed that randomization does not help in this problem; the competitiveness of any randomized strategy is

still $\Omega(\log n)$. Additionally, Koga proposed a second objective function that aims at balancing the packet delays among the n queues. Let the flow time of a data packet be the length of the time interval when the packet resides in one of the queues. Koga suggested to sum up, for each queue, the flow times of the packets. The goal is to minimize the maximum sum. Koga proved that no deterministic online algorithm is better than $\Omega(\log n)$-competitive. An interesting problem is to develop upper bounds for this second objective function.

The second problem we study considers scenarios where buffers or queues have bounded capacity. In this case packet loss cannot be avoided and the goal is to transmit the packets of highest value. Kesselman et al. [74] investigated the following problem in the context of managing the output buffer of a router or switch. At time t, a set $N(t)$ of new data packets arrives. Each packet p has a value $v(p)$, which is a positive real number. There is a buffer in which the data packets can be stored temporarily. In each time step t an algorithm can transmit one of the packets that are available in the buffer or in the set $N(t)$. The goal is to maximize the value of the transmitted packets. Kesselman et al. investigate two types of buffers. In a *FIFO buffer* the packet transmission times must be consistent with the arrival times. More precisely, if packet p is transmitted before p', then p must not have arrived later than p'. Moreover, the buffer can simultaneously hold only B packets. An algorithm has to decide which packets to drop so as to obey this buffer capacity. In a *bounded-delay buffer* each packet p has an associated slack time $sl(p)$. If the packet arrives at time t, then it must be transmitted or dropped by time $t + sl(p)$. There is no explicit bound on the buffer size and packets may be re-ordered.

First consider the FIFO model. Kesselman et al. [74] analyzed the following algorithm.

Greedy: If there is a buffer overflow, discard the packets with the smallest values; ties are broken arbitrarily.

Theorem 30. [74] *Greedy achieves a competitive ratio of* $2 - \frac{1}{B+1}$. *This ratio is tight for that algorithm.*

Kesselman et al. also showed that Greedy has a competitiveness of $2-\frac{2}{\alpha+1}$, where α is the ratio of the maximum to minimum packet value. Zhu [99] recently gave a lower bound.

Theorem 31. [99] *In the FIFO model no deterministic online algorithm can achieve a competitive ratio smaller than* $\sqrt{2}$.

A challenging problem is to close the gap between the lower and the upper bounds. For the special case $B = 2$, Zhu showed tight bounds of $(5 + \sqrt{13})/6 \approx 1.434$.

Next we examine the bounded-delay model. Again Kesselman et al. [74] proposed a Greedy strategy.

Greedy: In each step, send the packet with the highest value.

Theorem 32. *The Greedy algorithm achieves a competitive ratio of* 2 *and this is tight for that algorithm.*

If there are only two packet values (cheap and expensive), then Greedy has a competitiveness of exactly $1 + 1/\alpha$, where α is the ratio of the expensive to the cheap value. Zhu [99] gave a lower bound of 1.366. This bound even holds in a restricted model where the slack time of each packet is equal to 2. For this special scenario, Zhu also showed an upper bound of $\sqrt{2}$. Finally tight upper and lower bounds of $(1 + \sqrt{5})/2 \approx 1.618$ are known for the case that the slack time of each packet is at most 2, see [74,99]. The major open problem is to determine tight bounds for the general bounded-delay model.

8. Refinements of competitive analysis

Competitive analysis is a strong worst-case performance measure. For some online problems, such as paging, the competitive ratios of online algorithms are much higher than the corresponding performance ratios observed in practice. The reason is typically that in a competitive analysis we have to consider arbitrary request sequences whereas in practice only restricted classes of inputs occur. Therefore, a line of research has analyzed online algorithms on restricted request sequences and proposed other measures for evaluating online algorithms.

We consider the paging problem in more detail. As discussed in Section 2 the best competitive ratio of deterministic online algorithms is equal to k, where k is the number of pages in fast memory, and both LRU and FIFO achieve this competitiveness. From a practical point of view this bound is not very meaningful as fast memories can often store several hundreds or thousands of pages. In fact, the ratio of k is much higher than the algorithms' performance in practice. In an experimental study presented by Young [97], LRU achieved competitive ratios between 1 and 2. Also, in practice, the performance of LRU is much better than that of FIFO. This is not evident in the competitive analysis.

In the paging problem standard competitive analysis ignores the fact that request sequences generated by real programs have a special structure, i.e. they exhibit *locality of reference*: Whenever a page is requested, the next request is usually to a page that comes from a very small set of associated pages. Borodin et al. [32] proposed *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 can only be to a page that is adjacent to p in the access graph. Formally, let $G = (V, E)$ be an undirected graph. V represents the set of memory pages and E is a set of edges. A request sequence $\sigma = \sigma(1), \ldots, \sigma(m)$, is *consistent with* G if $(\sigma(t), \sigma(t+1)) \in E$ for all $t = 1, \ldots, m-1$. We say that an online algorithm A is c-competitive on G if there exists a constant a such that $A(\sigma) \leq c \cdot OPT(\sigma) + a$ for all σ consistent with G. The competitive ratio of A on G, denoted by $R(A, G)$, is the infimum of all c such that A is c-competitive on G. Let $R(G) = \min_A R(A, G)$ be the best competitive ratio achievable on G.

Borodin et al. [32] showed that LRU achieves the best possible competitive ratio on access graphs that are trees. Trees represent the access graphs for many data structures. Borodin et al. also analyzed $R(LRU, G)$ on arbitrary graphs. In particular they showed that there exist graphs for which the competitive ratio

of FIFO is much higher than that of LRU. Another important result, due to Chrobak and Noga [41], is that LRU is never worse than FIFO on access graphs.

Theorem 33. [41] *For any graph G, $R(LRU, G) \leq R(FIFO, G)$.*

Borodin et al. [32] also presented an optimal online algorithm for any access graph.

FAR: The algorithm is a marking strategy. If there is a fault at a request to a page p, then FAR evicts an unmarked page from fast memory that has the largest distance to a marked page in the access graph.

Irani et al. [66] showed that this algorithm achieves the best possible competitive ratio, up to a constant factor, for all access graphs.

Theorem 34. [66] *For any graph G, $R(FAR, G) = O(R(G))$.*

Fiat and Karlin [54] presented randomized online paging algorithms for access graphs that achieve an optimal competitive ratio. A disadvantage of FAR and the randomized algorithms by Fiat and Karlin [54] is that the access graph has to be known in advance. Fiat and Mendel [57] presented deterministic and randomized online algorithms that do not have to know the access graph but still achieve the best possible competitive ratios.

So far we have addressed undirected access graphs. An initial investigation of directed access graph was presented by Irani et al. [66], who considered structured program graphs. A fundamental open problem is to develop online paging algorithms for general directed access graphs.

As an alternative to access graphs, Karlin et al. [71] modeled locality of reference by assuming that request sequences are generated by a Markov chain. They analyzed paging algorithm in terms of their *fault rate* which is the performance measure used in practice. In particular, they developed an algorithm that achieves an optimal fault rate, for any Markov chain. Torng [96] 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. [7] proposed another framework for modeling locality of reference that goes back again to the working set concept by Denning [49,50]. 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 whose general behavior is depicted in Figure 1. The function is increasing and concave.

Inspired by this simple and natural model, [7] devises two ways of modeling locality of reference. In both models, it is assumed that an application is characterized by a concave function f; the application generates request sequences that are *consistent with f*. In the Max-Model a request sequence is consistent with f if the maximum number of distinct pages referenced in a window of size n is at most $f(n)$, for any $n \in \mathbb{N}$. In the Average-Model a request sequence is consistent with f if the average number of distinct pages referenced in a window of

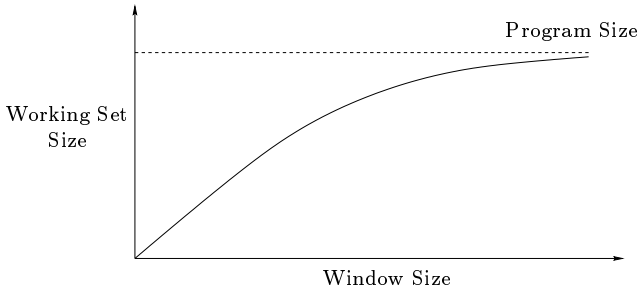

Fig. 1. Working set size as a function of the window size.

size n is at most $f(n)$, for any $n \in \mathbb{N}$. Albers et al. performed extensive experiments with traces from standard corpora, analyzing maximum/average working set sizes in windows of size n. In all of the cases, the functions have an overall concave shape. The authors use again the page fault rate to evaluate the quality of paging algorithms, and develop tight or nearly tight bounds on the fault rates achieved by LRU, FIFO, deterministic Marking strategies and MIN. It shows that LRU is an optimal online algorithm, whereas FIFO and Marking strategies are not optimal in general. Finally [7] presents an experimental study comparing the page fault rates proven in the analyses to the page fault rates observed in practice. The gap between the theoretical and observed bounds is considerably smaller than the corresponding gap in competitive analysis.

Further refinements of competitive analysis include extra resource analyses, see e.g. [68, 93], statistical adversaries [37, 84], accomodating functions [35] and the max/max ratio [26]. With respect to arbitrary online problems, Koutsoupias and Papadimitriou [79] proposed 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

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

Koutsoupias and Papadimitriou show that LRU is optimal against diffuse adversaries. Secondly, Koutsoupias and Papadimitriou [79] introduced *comparative analysis*, which compares the performance of online algorithms from given classes of algorithms.

9. Conclusions

In this paper we gave an introduction to competitive online algorithms and presented a number of important results. An excellent text book on the subject was written by Borodin and El-Yaniv [31]. The book [58] contains many survey articles on various online problems. Of course, there are many application areas that we have not addressed here. Bin packing is a classical problem that is still

actively investigated, see e.g. [42,48] and references therein. 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. We refer the reader to [75,76,73] for some basic literature. Recently, there has been research interest in competitive auctions, see e.g. [53, 62], a fresh field that deserves further investigations. In summary there is no doubt that online algorithms continue to be an interesting research area and that competitive analysis will be a powerful tool to analyze their performance.

References

1. D. Achlioptas, M. Chrobak and J. Noga. Competitive analysis of randomized paging algorithms. Theoretical Computer Science **234,** (2000) 203–218.
2. S. Albers. Improved randomized on-line algorithms for the list update problem. SIAM Journal on Computing. **27,** (1998) 670–681.
3. S. Albers. Better bounds for online scheduling. SIAM Journal on Computing **29,** (1999) 459-473.
4. S. Albers. On randomized online scheduling. Proc. 34th ACM Symposium on Theory of Computing, (2002) 134-143.
5. S. Albers. Generalized connection caching. Theory of Computing Systems **35,** (2002) 251-267.
6. S. Albers and H. Bals. Dynamic TCP acknowledgement: Penalizing long delays. Proc. 14th ACM-SIAM Symposium on Theory of Computing, 2003.
7. S. Albers, L.M. Favrholdt and O. Giel. On paging with locality of reference. Proc. 34th ACM Symposium on Theory of Computing, (2002) 258–268.
8. S. Albers and M. Mitzenmacher. Average case analyses of list update algorithms, with applications to data compression. Algorithmica **21,** (1998) 312–329.
9. S. Albers, B. von Stengel and R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. Information Processing Letters **56,** (1995) 135–139.
10. Christoph Ambühl. Offline list update is NP-hard. Proc. 8th Annual European Symposium on Algorithms, Springer LNCS 1879, (2001) 42–51.
11. C. Ambühl, B. Gärtner and B. von Stengel. Towards new lower bounds for the list update problem. Theoretical Computer Science **268,** (2001) 3–16.
12. A. Armon, Y. Azar, L. Epstein and O. Regev. Temporary tasks assignment resolved. Proc. 13th Annual Symposium on Discrete Algorithms, (2002) 116–124.
13. J. Aspnes, Y. Azar A. Fiat, S. Plotkin and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. Journal of the ACM **44,** (1997) 486–504.
14. Y. Azar. On-line load balancing. In A. Fiat and G. Woeginger, *Online Algorithms: The State of the Art,* Springer LNCS 1442, (1998) 178–195.
15. Y. Azar, A. Broder and A. Karlin. On-line load balancing. Theoretical Computer Science **130,** (1994) 73–84.
16. Y. Azar and L. Epstein. On-line load balancing of temporary tasks on identical machines Proc. 5th Israeli Symposium on Foundations of Computer Science, (1997) 119–125.
17. Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs and O. Waarts. Online load balancing of temporary tasks. Journal of Algorithms **22,** (1997) 93–110.
18. Y. Azar, J. Naor and R. Rom. The competitiveness of on-line assignments. Journal of Algorithms **18,** (1995) 221–237.
19. A. Bar-Noy, A. Freund, S. Landa, and J.(S.) Naor. Competitive on-line switching policies. Proc. 13th ACM-SIAM Symposium on Discrete Algorithms, (2002) 525–534.
20. Y. Bartal, B. Bollobás and M. Mendel. A Ramsey-type theorem for metric spaces and its applications for metrical task systems and related problems. Proc. 42nd IEEE Annual Symposium on Foundations of Computer Science, (2001) 396–405.
21. Y. Bartal, A. Blum, C. Burch and A. Tomkins. A polylog(n)-competitive algorithm for metrical task systems. Proc. 29th Annual ACM Symposium on Theory of Computing, (1997) 711–719.

22. Y. Bartal, M. Chrobak, J. Noga and P. Raghavan. More on random walks, electrical networks and the Harmonic k-server algorithm. Information Processing Letters **84,** (2002) 271–276.
23. Y. Bartal, A. Fiat, H. Karloff and R. Vohra. New algorithms for an ancient scheduling problem. Journal of Computer and System Sciences **51,** (1995) 359–366.
24. Y. Bartal and E.F. Grove. The Harmonic online k-server algorithm is competitive. Journal of the ACM **47**, (2000) 1–15.
25. L.A. Belady. A study of replacement algorithms for virtual storage computers. IBM Systems Journal **5**, (1966) 78–101.
26. S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. Algorithmica **11,** (1994) 73–91.
27. S. Ben-David, A. Borodin, R.M. Karp, G. Tardos and A. Wigderson. On the power of randomization in on-line algorithms. Algorithmica **11,** (1994) 2–14.
28. J.L. Bentley, D.S. Sleator, R.E. Tarjan and V.K. Wei. A locally adaptive data compression scheme. Communication of the ACM **29,** (1986) 320–330.
29. P. Berman, M. Charikar and M. Karpinski. On-line load balancing for related machines. Journal of Algorithms **35,** (2000) 108–121.
30. A. Blum, S. Chawla and A. Kalai. Static optimality and dynamic search-optimality in lists and trees. Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, (2002) 1–8.
31. A. Borodin and R. El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.
32. A. Borodin, S. Irani, P. Raghavan and B. Schieber. Competitive paging with locality of reference. Journal on Computer and System Sciences **50,** (1995) 244–258.
33. A. Borodin, N. Linial and M. Saks. An optimal online algorithm for metrical task systems. Journal of the ACM **39,** (1992) 745–763.
34. M. Burrows and D.J. Wheeler. A block-sorting lossless data compression algorithm. DEC SRC Research Report 124, 1994.
35. J. Boyar, K.S. Larsen and M.N. Nielsen. The accommodating function: A generalization of the competitive ratio. SIAM Journal on Computing **31,** (2001), 233–258.
36. B. Chen, A. van Vliet and G.J. Woeginger. A lower bound for randomized on-line scheduling algorithms. Information Processing Letters **51,** (1994) 219–222.
37. A. Chou, J. Cooperstock, R. El Yaniv, M. Klugerman and T. Leighton. The statistical adversary allows optimal money-making trading strategies. Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms, (1995) 467–476.
38. M. Chrobak, H. Karloff, T. Paye and S. Vishwanathan. New results on the server problem. SIAM Journal on Discrete Mathematics **4,** (1991) 172–181.
39. M. Chrobak and L.L. Larmore. An optimal online algorithm for k servers on trees. SIAM Journal on Computing **20,** (1991) 144–148.
40. M. Chrobak and L.L. Larmore. Harmonic is 3-competitive for two servers. Theoretical Computer Science **98,** (1992) 339–346.
41. M. Chrobak and J. Noga. LRU is better than FIFO. Algorithmica **23,** (1999) 180–185.
42. E.G. Coffman Jr., M.R. Garey and D.S. Johnson. Approximation algorithms for bin packing: A survey. *Approximation Algorithms for NP-Hard Problems*, D. Hochbaum (editor), PWS Publishing, (1997), 46–93.
43. E. Cohen, H. Kaplan and U. Zwick. Connection caching. Proc. of the 31st Annual ACM Symposium on Theory of Computing, (1999) 612-621.
44. E. Cohen, H. Kaplan and U. Zwick. Connection caching under various models of communiation. Proc. 12th Annual ACM Symposium on Parallel Algorithms and Architectures, (2000) 54–63.
45. R. Cole. On the dynamic finger conjecture for splay trees. Part 2: The proof. SIAM Journal on Computing **30,** (2000) 44–85.
46. R. Cole, B. Mishra, J. Schmidt, A. Siegel. On the dynamic finger conjecture for splay trees. Part I: Splay sorting $\log n$-block sequences. SIAM Journal on Computing **30**, (2000) 1–43.
47. D. Coppersmith, P. Doyle, P. Raghavan and M. Snir. Random walks on weighted graphs, and applications to on-line algorithms. Journal of the ACM **40,** (1993) 421–453.
48. J. Csirik and G.J. Woeginger. On-line packing and covering problems. In A. Fiat and G. Woeginger, *Online Algorithms: The State of the Art,* Springer LNCS 1442, (1998) 147–177.
49. P.J. Denning. The working set model of program behavior. Communications of the ACM **11,** (1968) 323–333.
50. P.J. Denning. Working sets past and present. IEEE Transactions on Software Engineering **6,** (1980) 64–84.

51. D.R. Dooly, S.A. Goldman and S.D. Scott. On-line analysis of the TCP acknowledgment delay problem. Journal of the ACM **48,** (2001) 243–273.
52. A. Feldmann, A. Karlin, S. Irani and S. Phillips. Private communication cited in [65].
53. A. Fiat, A. Goldberg, J. Hartline and A. Karlin. Competitive generalized auctions. Proc. 34th ACM Symposium on Theory of Computing, (2002) 72–81.
54. A. Fiat and A. Karlin. Randomized and multipointer paging with locality of reference. Proc. 27th Annual ACM Symposium on Theory of Computing, (1995) 626–634.
55. A. Fiat, R.M. Karp, L.A. McGeoch, D.D. Sleator and N.E. Young. Competitive paging algorithms. Journal of Algorithms **12,** (1991) 685–699.
56. A. Fiat and M. Mendel. Truly online paging with locality of reference. Proc. 38th Annual Symposium on Foundations of Computer Science, (1997) 326–335.
57. A. Fiat and M. Mendel. Better algorithms for unfair metrical task systems and applications. Proc. 32nd Annual ACM Symposium on Theory of Computing, (2000) 725–734.
58. A. Fiat and G. Woeginger, *Online Algorithms: The State of the Art,* Springer LNCS 1442, 1998.
59. R. Fielding, J. Getty, J. Mogul, H. Frystyk and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. `http://www.cis.ohio-state.edu/htbin/rfc/rfc2068.html`
60. R. Fleischer and M. Wahl. Online scheduling revisited. Journal of Scheduling **3,** (2000) 343–353.
61. J.S. Frederiksen and K.S. Larsen. Packet bundling. Proc. 8th Scandinavian Workshop on Algorithm Theory. Springer LNCS 2368, (2002) 328–337.
62. A.V. Goldberg, J.D. Hartline and A. Wright. Competitive auctions and digital goods. Proc. 12th ACM-SIAM Symposium on Discrete Algorithms, (2001) 735–744.
63. R.L. Graham. Bounds for certain multiprocessor anomalies. Bell System Technical Journal **45,** (1966) 1563–1581.
64. S. Irani. Two results on the list update problem. *Information Processing Letters* **38,** (1991) 301–306.
65. S. Irani. Page replacement with multi-size pages and applications to Web caching. Algorithmica **33,** (2002) 384–409.
66. S. Irani, A.R. Karlin and S. Phillips. Strongly competitive algorithms for paging with locality of reference. SIAM Journal on Computing **25,** (1996) 477–497.
67. S. Irani and D.S. Seiden. Randomized algorithms for metrical task systems. Theoretical Computer Science **194**, (1998) 163–182.
68. B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. Journal of the ACM **47,** (2000) 617–643.
69. D.R. Karger, S.J. Phillips and E. Torng. Journal of Algorithms **20,** (1996) 400–430.
70. A.R. Karlin, C. Kenyon and D. Randall. Dynamic TCP acknowledgement and other stories about $e/(e-1)$. Proc. 31st ACM Symposium on Theory of Computing, (2001) 502–509.
71. A. Karlin, S. Phillips und P. Raghavan. Markov paging. SIAM Journal on Computing **30,** (2000) 906–922.
72. R. Karp and P. Raghavan. From a personal communication cited in [86].
73. R. Karp, U. Vazirani and V. Vazirani. An optimal algorithm for online bipartite matching. Proc. 22nd ACM Symposium on Theory of Computing, (1990) 352–358.
74. A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber and M. Sviridenko. Buffer overflow management in QoS switches. Proc. 33rd Annual ACM Symposium on Theory of Computing, (2001) 520–529.
75. S. Khuller, S.G. Mitchell and V.V. Vazirani. On-line weighted bipartite matching. Proc. 18th International Colloquium on Automata, Languages and Programming (ICALP), Springer LNCS 510, (1991) 728–738.
76. H. Kierstead. Coloring graphs on-line. In A. Fiat and G. Woeginger, *Online Algorithms: The State of the Art,* Springer LNCS 1442, (1998) 281–305.
77. H. Koga. Balanced scheduling toward loss-free Packet queuing and delay fairness. Proc. 12th International Symposium on Algorithms and Computation (ISAAC), Springer LNCS 2223, (2001) 61–73.
78. E. Koutsoupias and C.H. Papadimitriou. On the k-server conjecture. Journal of the ACM **42,** (1995) 971–983.
79. E. Koutsoupias and C.H. Papadimitriou. Beyond competitive analysis. SIAM Journal on Computing **30,** (2000) 300–317.
80. M.S. Manasse, L.A. McGeoch and D.D. Sleator. Competitive algorithms for on-line problems. Proc. 20th Annual ACM Symposium on Theory of Computing, (1988) 322–333.
81. L.A. McGeoch and D.D. Sleator. A strongly competitive randomized paging algorithm. Algorithmica **6,** (1991), 816–825.

82. J. Noga. Private communication, 2001.
83. E. Peserico. The lazy adversary conjecture fails. Proc. 14th Annual Symposium on Parallel Algorithms and Architectures, (2002) 143–144.
84. P. Raghavan. A statistical adversary for on-line algorithms. *On-Line Algorithms*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, (1991) 79–83.
85. P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. IBM Journal of Research and Development **38,** (1994) 683–708.
86. N. Reingold, J. Westbrook and D.D. Sleator. Randomized competitive algorithms for the list update problem. Algorithmica **11,** (1994) 15–32.
87. J.F. Rudin III. Improved bounds for the on-line scheduling problem. Ph.D. Thesis. The University of Texas at Dallas, May 2001.
88. S.S. Seiden. Online randomized multiprocessor scheduling. Algorithmica, **28,** (2000) 173–216.
89. S.S. Seiden. A guessing game and randomized online algorithms. Proc. 32nd ACM Symposium on Theory of Computing, (2000) 592–601.
90. S.S. Seiden. A general decomposition theorem for the k-server problem. Proc. 9th Annual Symposium on Algorithms, Springer LNCS 2161, (2001) 86–97.
91. J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. Information Processing Letters **63,** (1997) 51–55.
92. J. Sgall. On-line scheduling. In A. Fiat and G. Woeginger, *Online Algorithms: The State of the Art,* Springer LNCS 1442, (1998) 196–231.
93. D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. Communications of the ACM **28,** (1985) 202–208.
94. D.D. Sleator and R.E. Tarjan. Self-adjusting binary search trees. Journal of the ACM **32,** (1985) 652–686.
95. R.E. Tarjan. Sequential access in splay trees takes linear time. Combinatorica, **5** (1985) 367–378.
96. E. Torng. A unified analysis of paging and caching. Algorithmica **20,** (1998) 175–200.
97. N. Young. The k-server dual and loose competitiveness for paging. Algorithmica **11,** (1994) 525–541.
98. N.E. Young. Online file caching. Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms, (1998) 82–86.
99. A. Zhu. Analysis of queueing policies in QoS switches. Proc. 14th ACM-SIAM Symposium on Theory of Computing, 2003.