RC 21340 revised November 10, 1998
Computer Science/Mathematics

IBM Research Report

On-line Algorithms: How Much is it Worth to Know the Future?

Pinar Keskinocak

IBM T. J. Watson Research Center
Yorktown Heights, New York 10598
e-mail: pinar@watson.ibm.com

Abstract

In an on-line problem, the input is obtained incrementally over time. An on-line algorithm makes a sequence of decisions, where each decision is based only on the partial input obtained until that time, without knowledge of future input. Some on-line algorithms make decisions as soon as new data becomes available. But there are other on-line algorithms with *lookahead*, i.e. they obtain partial information about future input by possibly delaying their decision at some extra cost. The question is, when and how much does it help to have more information about the future? In this note, we survey some of the well known problems studied in the on-line context and summarize performance guarantees of on-line algorithms with and without lookahead. For some of the problems, we observe that a substantial amount of lookahead is required to obtain an improvement in the competitive ratio of any on-line algorithm. We have also observed that the influence of lookahead depends on the choice of the objective function and on the model of lookahead used for each problem. Finally, for certain problems, measuring the performance of on-line algorithms by alternative measures other than the competitive ratio may reflect the influence of lookahead on performance better.

Keywords: On-line algorithms, competitive ratio, lookahead.

Research Division
Almaden · Austin · T.J. Watson · Beijing · Haifa · Tokyo · Zurich

1 Introduction

In a wide range of optimization problems, the decisions must be made *on-line*, in response to a sequence of events. An on-line algorithm makes decisions based on the history of events that have already occured and incomplete knowledge of future events. One approach to such problems is to make probabilistic assumptions about future events and to evaluate decision strategies based on their expected performance under a hypothesized distribution. Another approach is to study the "worst case" performance of algorithms. In the worst case or *competitive analysis* [49], the performance of an on-line algorithm is compared to an optimal off-line algorithm. An optimal off-line algorithm knows all the events in advance and makes the optimum decision for that input sequence. The maximum deviation of the on-line solution from the optimum off-line solution over all possible input sequences is the worst case performance guarantee of an on-line algorithm. The interest in the competitive analysis of algorithms has been groving rapidly [9], especially in the situations where a good stochastic model for predicting the input is not available.

Given a problem instance I, let $z_A(I)$ denote the solution obtained by using algorithm A, and let $z^*(I)$ denote the solution obtained by an optimum off-line algorithm for instance I. We call an algorithm A *c-competitive*, if there exists a constant a, such that

$$z_A(I) \leq cz^*(I) + a$$

for any instance I. Here we assume that A is a deterministic on-line algorithm and we are solving a minimization problem. The factor c is also called the *competitive ratio* of A. If A is a randomized algorithm, then $z_A(I)$ is the expected solution obtained by A for instance I.

One can view competitive analysis as a two person game between an on-line algorithm and an adversary. The adverasary creates the worst possible input to maximize the competitive ratio, based on the knowledge of the algorithm used by the on-line player.

Intuitively, the performance on an on-line algorithm would improve if the algorithm could obtain some information about future events before making a decision about the current event. We consider two possible ways for the algorithm to obtain information about future events:
(1) Every time a new event occurs, in addition to that current event, the algorithm is presented some future events. Based on the information about the past events and the partial information about future events, the algorithm makes a decision about the current event.
(2) When a new event occurs, instead of making a decision immediately, the algorithm can wait for more events to occur. In this case, to obtain more information about future events, the algorithm employs a *waiting strategy* and delays its decision about the current event.

If the timing of the decision does not affect the objective function (e.g. as in the paging, bin packing or graph coloring problems), then these two alternative ways of looking into the future are equivalent. However, if the response time of the algorithm to the events affects the objective function, then employing a waiting strategy may imply a penalty. For example, consider the problem of scheduling jobs on a single machine to minimize the sum of job completion times. If an on-line algorithm delays the scheduling of a job and waits for more jobs to arrive, this delay increases the competion time of the job, and hence, increases the objective function value. In this case, the algorithm (possibly) obtains more information about future events by waiting, but at an extra cost.

We say that an on-line algorithm has *lookahead*, if it obtains some partial information about future events before making a decision about the current event. Based on the discussion above, the following natural models of lookahead come to mind: An algorithm has (constant) *lookahead* L, if

(M1) it sees the next L (or $L-1$) events in addition to the current event.
(M2) it sees the events in the next L periods at any time.
(M3) it can wait and see the next L events before deciding about the current event.
(M4) it can wait for L time periods before deciding about the current event.

For certain on-line problems such as paging and graph coloring, without loss of generality we may assume that one event occurs per unit time and as we mentioned before, the timing of the algorithm's response to these events does not affect the objective function. For the problems of this type, these four definitions of lookahead are equivalent. However, for other problems, such as on-line job scheduling to minimize the sum of job completion times, these four different types of lookahead are likely to have different influences on algorithmic performance. Also note that algorithms with lookahead model M1 have performance guarantees at least as good algorithms with lookahead model M3. A similar argument holds for lookahead models M2 and M4. As we discuss in the rest of this paper, the precise definition of lookahead depends on the problem instance, and for many problems, if L is finite the competitive ratio is insensitive to the the lookahead as defined above.

Off-line solution methods assume that all the information about the problem input is known up front (infinite lookahead). Probabilistic analysis assumes that we know the "structure" of the input. On-line solution methods (with zero lookahead) assume that we know nothing about the future input. The reality is in between. For many problems, it is quite likely that we won't have complete information about the input up front, or that the structure of the input may change over time. But we may have some information about the the near future. Using the two person game analogy, on-line algorithms with lookahead learn the future (partially, at some extra cost) to make better decisions. The question is, how much into the future one needs to look, to be able to make better decisions.

In this paper, we discuss the effects of lookahead on the performance of on-line algorithms for the following optimization problems: the dynamic location problem (section 2), the paging problem (section 3), the list update problem (section 4), bin packing (section 5), graph coloring (section 6), scheduling multi-pass assembly lines (section 7) and single machine scheduling with job release times (section 8).

2 The Dynamic Location Problem

In the Dynamic Location Problem (DLP), we are given a graph $G = (V, E)$, which represents a system, and a sequence of requests to the system. The processing of each request requires the use of the same indivisible resource. The state of the system is characterized by a single node representing the current location of the resource. Requests arrive to the system at particular nodes and must be processed in the order of arrival. The cost of processing a request at node v is $d(v, w)$, where w is the location of the resource when the request is processed and $d(v, w)$ denotes the (shortest path) distance between nodes v and w in G. The objective is to find the optimal sequence of states to process the requests at minimum total cost.

Given a finite request sequence, in the off-line version of DLP an optimal state sequence can be computed using dynamic programming [45]. In the on-line version, requests are presented one at a time and the state sequence must be constructed on-line. In [12], lookahead is defined as follows: an on-line algorithm for DLP has *lookahead L*, if the algorithm is allowed to see the next L requests at any time (lookahead model M1). Equivalently, an on-line algorithm for DLP has lookahead L, if it can wait for the next $L-1$ requests to arrive, before deciding on how to serve the current request. In [12], for several graph classes, the authors study the smallest lookahead parameter L, such that on-line algorithms with lookahead L exist for DLP, that can solve the problem optimally; they call this parameter the *window index* of G and denote by $WX(G)$. They present the following interesing results.

1. $WX(G) \geq 2$ for any connected graph on at least two nodes.
2. The following classes of graphs have window index 2: trees, hypercubes, planar grids.
3. The complete graph with n nodes has window index n.
4. The following graphs have window index ∞: a cycle of 5 nodes, complete graph with 4 nodes, with an edge deleted.
5. $WX(G)$ is finite if and only if G is a weak retract (see [12] for a definition) of a product of cliques.
6. If $WX(G)$ is finite, then it is equal to the size of the maximum clique in G.

For trees, hypercubes and planar graphs, a simple optimal on-line algorithm with lookahead 2 is presented in [12]. Although on-line algorithms with constant lookahead L can produce optimum solutions for certain graph classes, there are other graph classes for which no finite lookahead would allow an on-line algorithm to solve the problem optimally.

A generalization of DLP is the K-server problem. At any time, K servers are located on the nodes of a graph $G = (V, E)$. To serve a request at node v, if no server is on v, at least one server must be moved from its present location u to v, at cost $d(u, v)$, where d is a metric defined on the nodes. On-line algorithms with lookahead for the K-server problem are considered in [6]. It is shown that for every algorithm with lookahead L, there exists an on-line algorithm achieving the same competitive ratio, i.e. no finite lookahead is sufficient for any improvement in the competitive ratio. Noting that the competitive ratio measure may not always capture the potential benefit of lookahead on the performance of on-line algorithms, an alternative measure is proposed in [6], called the Max/Max ratio. It is shown that for the K-server problem, a greedy strategy achieves Max/Max ratio $(n-1)/(n-K)$ and $(n-1)/L$ for $L \leq n-K$ and $n-K < L \leq n-1$, respectively, and that these ratios are optimal. Hence, although the competitive ratio measure is insensitive to lookahead, increasing the lookahead improves the Max/Max ratio, indicating that lookahead improves algorithmic performance.

3 The Paging Problem

Consider a two-level memory system which has a fast memory that can store k pages and a slow memory that can manage an unbounded number of pages. A sequence of requests arrive for the pages, and a request can be served if the requested page is in the fast memory; otherwise, a *page fault* occurs. In that case, a page must be evicted from the fast memory so that the requested page can be loaded into the vacated location. The objective is to serve the requests to minimize the total number of page faults, or *fault rate*.

If all the future requests are known, then the *off-line* paging problem can be solved optimally as follows: whenever a page fault occurs, evict the page in the fast memory that will not be requested for the longest time [4]. In the *on-line* paging problem, an on-line algorithm decides which page to evict on a fault without knowing the future requests. Deterministic and randomized on-line algorithms and their competitive ratios for the paging problem have been studied extensively [49], [18], [43], [9], [50].

Sleator and Tarjan [49] have shown that no on-line paging algorithm can be better than k-competitive. A natural question to ask is: Does the competitive ratio improve, if the on-line algorithm sees a fixed number, say L, of the future requests, in addition to the current request (lookahead model M1)? Since the adversary can construct a request sequence by replicating each request L times, a constant lookahead of this type does not improve the competitive ratio of the on-line algorithms. To obtain improved competitive ratios for on-line algorithms, three different lookahead models are proposed in [1], [10] and [53]. However, in all of these models, the *length of the lookahead window*, i.e. the number of future requests the algorithm needs to see, is not known upfront and can be arbitrarily long.

An alternative objective in paging is to minimize the total memory access time [50]. Each page is accessed in 1 time step, and a fault incurs a fault penalty p. If f is the number of faults incurred on a sequence of n requests, then the total memory access time is $n + pf$. Torng [50] notes that focusing on fault rate may lead to pessimistic estimates about the performance of on-line algorithms and also does not consider important problem parameters such as page size. The results in [50] show that if minimizing total memory access time is the objective rather than the fault rate, then finite lookahead may result in improved competitive ratios.

In the rest of this section, we discuss the results of Young [53], Breslauer [10], Albers [1] and Torng [50] on on-line paging algorithms with lookahead in more detail.

In [53], lookahead is defined as follows: An on-line paging algorithm has a *resource bounded lookahead* L, if it sees the present request and the maximal sequence of future requests for which it will incur L faults. Deterministic and randomized algorithms are presented, which are $\max\{2k/L, 2\}$-competitive and $2\ln(k/L)+1$-competitive, respectively. On the positive side, these results show that competitive ratio, and hence the performance of the on-line algorithm, improves with lookahead. However, a lookahead of this type may be impracticle, since the length of the lookahead window at any time depends on the past and future behaviour of the on-line algorithm, and can be arbitrarily long.

According to Albers [1], an on-line algorithm has *strong lookahead* L, if it sees, in addition to the present request, a sequence of future requests, which contains L pairwise distinct pages different from the page presently requested.

The results in [1] show that the improvement in the competitiveness of on-line algorithms for the paging problem is proportional to the strong lookahead parameter L. It is shown that a variant of the well known replacement algorithm LRU (Least Recently Used) [49] has competitive ratio $k - L$, given strong lookahead of size $L \leq k - 2$. It is also shown that this is the best possible ratio, i.e. no deterministic on-line paging algorithm with strong lookahead $L \leq k - 2$ can have competitive ratio better than $k - L$. Another variant of the algorithm LRU with strong lookahead L is shown to have competitive ratio $k - L + 1$. Competitive ratios of randomized paging algorithms are given in terms of the k-th harmonic number $H(k) = \sum_{i=1}^{k} 1/i$. A modification of the MARKING akgorithm [18] with strong lookahead $L \leq k - 2$ is shown to be $2H(k - L)$ competitive. It is also shown that no randomized on-line paging algorithm with strong lookahead $L \leq k - 2$ can be better than $H(k - L)$-competitive, i.e. the competitive ratio of the modified MARKING algorithm is within a factor of 2 of optimal. Finally, another randomized on-line algorithm with strong lookahead L is presented, which is $(k - L + 1)$-competitive. In the strong lookahead model, the length of the lookahead window is independent of the algorithm's behavior, but may still be arbitrarily long.

The definition of lookahead in [10] is similar to the definition in [1]. An on-line algorithm has *natural lookahead* L, if it sees a sequence of future requests with L distinct pages different from the current request and from the pages currently in its fast memory. Note that natural lookahead allows the algorithm to see at least as many pages as strong lookahead and resource-bounded lookahead. Breslauer [10] presents a modified LRU algorithm with natural lookahead L, which can also use c additional fast memory pages. He shows that the competitive ratio of this algorithm is $\frac{k+c+L}{c+L+1}$ and that this is the best possible competitive ratio. Since the competitive ratio depends on $k + L$, under these measures an on-line algorithm has the same benefit from using an extra page or knowing an extra bit of the future. It is argued that in practice it might be easier to add extra fast memory pages rather than predicting future requests.

Torng [50] considers the paging problem with the objective of minimizing memory access time (PMAT). He defines a class of marking algorithms, which includes LRU, and shows that the competitive ratio of these algorithms is $k(p+1)/(k+p) \approx \min\{k, p+1\}$ and that this is the best possible ratio for any deterministic paging algorithm for PMAT. He also presents a randomized algorithm, which has competitive ratios $2H(p) + 2 - 2\ln 2$ and $2H(k)$ for $2\sqrt{e} \leq p \leq 2k$ and $p > 2k$, respectively. He then considers *constant lookahead* L (the algorithm sees the next $L-1$ requests in addition to the current request) and shows that the marking algorithms with lookahead L have competitive ratios less than $2 + \sqrt{2kp/L}$. In particular, if $L \geq pk$, then the competitive ratios are at most 2.

For the paging problem, if the objective is to minimize the fault rate, any on-line algorithm needs to see an arbitrarily long sequence of future requests, in order for the competitive ratio to improve. However, if the objective is to minimize the memory access time, then even for constant lookahead, the competitive ratio improves as lookahead increases. These results indicate that the choice of the objective function is important for observing the influence of lookahead on algorithmic performance.

4 The List Update Problem

The list update problem is about maintaining a set of n items as a linear linked list. A sequence of requests arrive where each request specifies one of the n items. To serve a request, a list update algorithm must access the requested item. Accessing the i-th item in the list incurs a cost of i. While processing a request sequence, a list update algorithm may rearrange the list. Immediately after a request, the requested item may be moved at no extra cost to any position closer to the front of the list (*free exchanges*). At any time, adjacent items in the list may be exchanged at a cost of 1 (*paid exchanges*). The objective is to minimize the total cost of serving the request sequence. The list update problem studied extensively, both due to its structural simplicity and its practical applications in data compression and in the commonly used linked list data structures [9].

A list update algorithm is on-line, if it serves each request without the knowledge of future requests. The Move-To-Front algorithm, which simply moves an item to the front of the list each time it is requested, is 2-competitive [49] and this is the best possible competitive ratio for any deterministic on-line algorithm for the list update problem [31]. Results and additional references on randomized algorithms for the list update problem can be found in [27], [47].

List update algorithms with lookahead are studied in [2]. Two types of lookahead are proposed. An on-line algorithm has *(weak) lookahead* L, if it sees the next L requests in addition to the current request (lookahead model M1). An on-line algorithm has *strong lookahead* L, if it sees, in addition to the current request, a sequence of requests which contains L pairwise distinct items. This definition of strong lookahead for the list update problem is the same as the definition given in [1] for the paging problem. In terms of lower bounds, it is shown that an on-line algorithm with strong lookahead $L \leq n-1$ cannot be better than $2 - \frac{L-2}{n+1}$ competitive. For weak lookahead $L = o(n^2)$ and $(L+1) = Kn^2$, for constant $K > 0$, the minimum attainable competitive ratios are 2 and $2 - 2\sqrt{4K^2 + 2K} + 4K$, respectively. Based on these results, one might conclude that the positive influence of weak lookahead on algorithmic performance is negligible, and dissepears quickly as n increases. However, note that a small K value, such as $K = \frac{1}{100}$ decreases the second bound to 1.75 and the bound goes to 1 as K goes to infinity. Also, the lookahead parameter $L = \frac{1}{100}n^2 - 1$ is small, e.g. is equal to 1 and 5, for small lists of size 12 and 24, respectively. On-line algorithms with strong and weak lookahead are presented in [2], which have competitive ratios $2 - \frac{2}{3}\frac{L+2}{2n-L}$ and $2 - \frac{2}{3}(\sqrt{K^2 + 2K} - K)$ (as n tends to infinity), respectively. The bounds on competitive ratios of the algorithms are proved against *static* off-line algorithms, which initially arrange the list in some order and don't make any other changes while processing the request sequence. The lower bounds hold against any type of adversary.

For the paging problem, the advantage of weak lookahead (lookahead model M1) on on-line algorithms' performance (measured by the competitive ratio) is limited. In order to obtain substantial improvement in the compatitive ratios, the adversary must reveal significant information about future requests, and hence the number of future requests an on-line paging algorithm needs to see may be arbitrarily large (strong lookahead).

5 Bin Packing

In the one-dimensional *bin packing* problem, we are given a sequence of items of sizes $a_1, a_2, \ldots, a_n$, $0 < a_i < 1$, and unit capacity bins. The goal is to assign these items to the bins, such that the total size of the items assigned to each bin does not exceed the bin capacity and the number of bins used is minimum.

The bin-packing problem is NP-complete [32]. For many years, researchers tried various approaches to find approximate solutions for the bin packing problem [13]. The *off-line* bin packing problem, where all the items in the list are given up front, can be solved within $1 + \epsilon$ in linear time [16], [30]. In the *on-line* bin packing problem, items are presented one at a time, and an on-line algorithm has to decide immediately to which bin each newly arrived item must be assigned. Among on-line bin-packing algorithms, Next Fit (NF) [41], First Fit (FF) [26] and Refined First Fit (RFF) [52] have competitive ratios 2, 17/10 and 5/3, respectively. Assuming that the bins are numbered as $1, 2, \ldots, n$, FF assigns each item to the lowest number bin, whereas NF assigns each item to the highest number bin that has enough remaining capacity to hold that item. In RFF, the items are divided into four types according to their sizes and the bins are divided into four classes. Items are packed sequentially into bins as in FF, but for each item, a particular class of bins is chosen depending on the type of the item and on the number of previously packed items of that type.

In all of these algorithms (FF, NF and RFF), it is possible that some bins stay "open", i.e. new items are added to them, for an arbitrarily long period of time. If we think of the bin packing problem as loading items to trucks at a loading dock, for certain instances, these algorithms could require an infinite number of trucks. An on-line bin packing algorithm, which guarantees that the number of open bins at any time is bounded, is called *bounded space*. A modified version of the First Fit algorithm, called Next-K-Fit (NFK), is a bounded space on line algorithm [14], [15]. NFK assigns items to the bins as in FF, but closes the smallest numbered bin whenever there are $K+1$ open bins. It is shown that NFK has competitive ratio 1.7 [41]. Another example of bounded space algorithms is the Harmonic algorithm (HK) [37]. HK separates items into K disjoint sets of sizes. The algorithm keeps a bin open for each of the following ranges: $[0, \frac{1}{K}], \ldots, (\frac{1}{m+1}, \frac{1}{m}], \ldots, (\frac{1}{2}, 1]$, $1 \leq m < K$. If an item does not fit into the bin corresponding to its range, the bin is closed and a new bin is opened. The competitive ratio of HK tends to 1.691, as K tends to infinity. It is also shown that this is the best possible competitive ratio of any on-line bounded space bin packing algorithm [37].

Now let us consider on-line algorithms with lookahead for the bin packing packing problem. These algorithms are allowed to see some of the future items, before assigning a newly arrived item to a bin. In other words, they can accumulate some items, before assigning them to the bins. First note that if an algorithm is allowed to see only a fixed number of future items (lookahead models M1 and M3), the competitive ratio cannot improve, because in that case the adversary can present many items of very small size. Therefore, an alternative definition of lookahead is given in [23]: an on-line bin packing algorithm has *lookahead* L, if at any time it sees a sequence of future items with total "cost" of processing at most L. If item i will be packed into a bin, such that the total size of the items already assigned to that bin is S, then the cost of processing item i is defined as a_i/S. Note that the total cost of the items is equal to the total number of bins used. A bounded space on-line algorithm, called the Modified Warehouse Algorithm ($\mathrm{M5W}_W$) is proposed in [23]. At any time, the total size of the unassigned items seen by $\mathrm{M5W}_W$ is at most W, where W is a sufficiently large constant. It is shown that the total cost of items processed between receiving and assigning an item to a bin is at most $L \leq 6W(2W+6)$, i.e. the algorithm has lookahead L. Furthermore, the algorithm has competitive ratio 1.691, i.e. it is an optimal bounded space algorithm.

As in the paging and list update problems, lookahead models M1 and M3 do not improve the competitive ratio of on-line bin packing algorithms. The lookahead model given in [23] improves the competitive ratio of (bounded space) on-line algorithms, however one should note that the length of the lookahead window in that model depends on the behavior of the algorithm and can be arbitrarily large.

6 Graph Coloring

In graph coloring, given a graph $G = (V, E)$ and k colors, each node must be assigned a color, different from the colors of its neighbors. The smallest such k is called the *chromatic number* of G, denoted by $\chi(G)$.

Coloring a graph with the minimum number of colors is NP-complete [32]. Hence, many researchers focused on finding approximation algorithms, in that the number of colors used by the algorithm is compared to the chromatic number of the graph. In terms of lower bounds, it is shown that for any $\epsilon > 0$, the chromatic number cannot be approximated to a ratio better than $n^{1/7-\epsilon}$, unless P=NP [5]. Approximation algorithms and their competitive ratios for 3-colorable and k-colorable graphs are studied in [29], [40], [7], [8].

In the on-line graph coloring problem, the nodes of the graph are presented one at a time and an on-line algorithm must color each node without the knowledge of future nodes. For general graphs, the best possible competitive ratio for an on-line graph coloring algorithm is $\Omega(n/(\log\ n)^2)$ [28]. On-line graph coloring algorithms with competitive ratios $O(n/\log^* n)$ and $O(n/\sqrt{\log\ n})$ (randomized) are presented in [39] and [51], respectively. Researchers have also studied on-line coloring algorithms for special classes of graphs, such as interval graphs [35], [34], split graphs, complements of bipartite graphs and complements of chordal graphs [24].

On-line coloring (with lookahead), for a special class of graphs, called d-inductive graphs is studied by Irani [28]. A graph is *d-inductive*, if its nodes can be numbered so that each node has at most d edges to higher-numbered nodes. In particular, planar graphs are 5-inductive and chordal graphs are $\chi(G)$-inductive. Note that $d+1$ is an upper bound on the chromatic number of any d-inductive graph. In [28], it is shown that the best possible competitive ratio for coloring d-inductive graphs on-line is $\Omega(d\ \log\ n)$, and that the on-line First Fit algorithm, which assigns each node the lowest numbered color possible, is $O(d\ \log\ |V|)$-competitive for any d-inductive graph. This implies a competitive ratio of $O(\log\ n)$ for planar graphs and $O(\chi(G)\ \log\ n)$ for chordal graphs.

An on-line graph coloring algorithm has *lookahead L*, if it sees a sequence of L future nodes, in addition to the current node [28] (lookahead model M1). For on-line coloring of d-inductive graphs with lookahead, a hybrid algorithm is proposed in [28]. If $d\ \log\ n < (d+1)(n/L)$, the algorithm colors the nodes as in First Fit, ignoring the lookahead. Otherwise, the nodes are divided into n/L groups of L consecutive nodes. Note that the graph induced by each group of nodes is d-inductive and can be colored using at most $d+1$ colors. Hence, the total number of colors used for all the nodes is at most $(n/l)(d+1)$. From these results it follows that the competitive ratio of this hybrid algorithm is $O(\min\{d\ \log\ n, dn/L\})$. It is also shown that this is (asymptotically) the best possible competitive ratio.

It is interesting to see that until the lookahead reaches the treshhold $L = ((d+1)n)/(d\log n)$, it does not seem to improve the competitive ratio. But once that treshhold is passed, the improvement in the competitive ratio becomes proportional to the length of the lookahead window.

7 Scheduling Multipass Assembly Lines

In a multipass manufacturing environment, each job requires between 1 to m equivalent processing tasks, or passes, and has to be reoriented after each pass [44]. An assembly line consists of m processing and m orienting devices. Based on the number of tasks they require, the jobs may be divided into m classes, such that class i jobs require exactly i tasks. A job is completed when its last task is completed. The input to a scheduling algorithm is a sequence of jobs and the output is a sequence of tasks, including the *null* task, which indicates an idle cycle. The following constraints should be considered:

FCFS Constraint: Jobs need to be completed in the order of arrival.

Return Constraint: Each job takes exactly k processor cycles to return to the processor. Therefore the $(i+1)$-st pass of a job through the processor must be scheduled exactly k cycles after its i-th pass.

Orientation Constraint: Jobs must be reoriented after each pass.

Possible scheduling objectives include minimizing the *makespan*, i.e. the length of the schedule, and minimizing the number of null tasks.

In [44], on-line scheduling (with lookahead) of multipass assembly lines is considered for the case of $m = 2$. The motivation for this problem came from networked reprographic machine (combination printer-copier-fax machines servicing a network) scheduling.

In this context, an on-line scheduling algorithm with *lookahead* L can see the sequence of L future jobs at any time (lookahead model M1). As soon as the algorithm schedules a job, a new job arrives. The lookahead parameter L can also be thought of as a buffer size. A greedy algorithm with lookahead L, called L-GREEDY, is proposed in [44], which optimally schedules L jobs at a time. When scheduling a block of L jobs, L-GREEDY computes an optimal extension of the partial schedule of earlier jobs. A special case of L-GREEDY is the simple 1-GREEDY algorithm, which schedules each job as early as possible.

For the objective of minimizing makespan, it is shown that the competitive ratio of L-GREEDY is between $1 + \frac{k-2}{L+3}$ and $1 + \frac{k-1}{L+1}$, for $k \geq 3$ and $L \geq k$. In terms of lower bounds, the competitive ratio of any deterministic (randomized) on-line algorithm is at least $1 + \frac{k-1}{4L+4K+4}$ $(1 + \frac{k}{8L+8K})$. Hence, the competitive ratio of L-GREEDY is almost optimal, and improves from approximately 2 to 1, as L increases.

For the objective of minimizing null tasks, it is shown that the best possible competitive ratios are at least $\lceil \frac{k+1}{2} \rceil$ and $k/4$ for deterministic and randomized on-line algorithms (with lookahead L), respectively. It is also shown that for $l \geq 2$ and $k \geq 3$, the competitive ratio of L-GREEDY is between $k-1$ and k.

As in the paging problem, these results once more indicate the importance of the choice of objective function, in terms of reflecting the influence of lookahead on the performance of on-line algorithms. Although the two objectives we considered, minimizing makespan and minimizing the number of null tasks, are equivalent in an off-line setting, they result in significantly different competitive ratios in an on-line setting. Competitive ratios clearly reflect the influence of lookahead with respect to the makespan objective, however they are insensitive to lookahead with respect to the null task objective.

8 Paying for the Lookahead

In the problems we have considered so far, the lookahead models M1 and M3 (similarly, M2 and M4) were equivalent, since the timing of the decision about an event did not affect the objective function (except that the algorithms are supposed to respond to the events on a first come first serve basis).

In most scheduling problems however, the response time of an algorithm to an event has an impact on the objective function. To obtain more information about future events, an algorithm may delay its decision about an event by employing a *waiting strategy* (lookahead models M3 and M4), but delaying the decision may increase the objective function value. For example, when the objective is to minimize the sum of job completion times in a scheduling problem, objective function value increases as the response time of the algorithm (the time the algorithm schedules an order for processing) to an event (the arrival of a new order) increases. Therefore, in such problems the algorithm should consider the tradeoff between obtaining more information about future events and increasing in the objective function value as a result of employing a waiting strategy (lookahead models M3 and M4).

In this section, we discuss the influence of lookahead model M4 on the performance of on-line algorithms for two single machine scheduling problems.

8.1 Single Machine Scheduling with Job Release Times

In the single machine scheduling problem with job release times (SMR), we are given n jobs, where each job j has a release time (or arrival time) r_j and a processing time p_j. A job's processing cannot start before its release time, and once started, the processing of a job cannot be interrupted (*preemption* is not allowed). The objective is to schedule the jobs, such that the sum of the completion times is minimized. Using the three field notation introduced in [22], this problem can be represented as $1|r_j|\sum C_j$. In the weighted version of SMR (WSMR), each job j also has a weight w_j and the objective is to minimize the sum of weighted completion times.

SMR is NP-hard [38], implying that WSMR is also NP-hard. The preemptive version of SMR is polynomially solvable by a simple extension of Smith's rule [3], whereas the preemptive version of WSMR is strongly NP-hard [36]. Several integer programming formulations are presented in [17], and lower bounds for different linear programming relaxations are compared. Both off-line and on-line versions of SMR and WSMR have received much attention within the context of competitive analysis [25], [21], [46], [11]. In the remaining part of this section, we concentrate on on-line algorithms (with lookahead) for SMR and WSMR.

First, let us consider on-line algorithms with zero lookahead for WSMR, i.e. an on-line algorithm must decide about the start time of the processing of a job, as soon as the job arrives.

Lemma 1 *Any on-line algorithm with zero lookahead has competitive ratio at least 2.618 for WSMR.*

Proof Consider the following instance. At time zero, a job with processing time p and weight 1 arrives. Suppose that the on-line algorithm schedules this job to time $t \geq 0$. Depending on t, one of the following events occur: (1) no more jobs arrive, (2) another job with processing time t and weight w arrives at time ϵ. In the first case, the first job would be scheduled at time zero in the optimum solution, and the competitive ratio is at least $(p+t)/p = 1 + t/p$. In the second case, in the optimum solution the second job would be scheduled at time ϵ followed immediately by the firts job, resulting in competitive ratio no less than $(p + t + w(p + 2t))/(wt + p + t) = 2 + p/t$ for large w and small ϵ. To achieve the best competitive ratio for both cases, the algorithm must choose t such that $t/p = 1 + p/t$, i.e. $\frac{t}{p} = \frac{1+\sqrt{5}}{2}$=2.618. □

From the proof of Lemma 1 it follows that lookahead models M1 and M3 cannot improve the competitive ratio of an algorithm, since the adversary may present L small jobs with processing time and weight ϵ/L at time zero, in addition to the first job.

To the best of our knowledge, all the approximation algorithms discussed in the literature for SMR and WSMR employ a *waiting strategy* (lookahead model M4) [25], [21], [46]. After a job, say job j, arrives, these algorithms wait for a while before scheduling this job , and if some other jobs arrive during this waiting period, the scheduling of job j can further be delayed. In fact, the waiting period or lookahead L as defined in lookahead model M4 is not bounded in these algorithms, i.e. an algorithm may choose to delay scheduling a job for an arbitrarily long period of time. However, one should keep in mind that for every unit of time a job j waits in the system, there is a penalty w_j. Therefore, having an unbounded lookahead L in model M4 may be justified. Recently, a zero loookahead model for SMR is discussed in [19], where it is shown that a competitive ratio better than $\log n$ is not possible, if very small non-integer processing times such as $1/n \log n$ are allowed.

A deterministic on-line algorithm with lookahead model M4 for WSMR is presented in [21] with competitive ratio 2.415. This results shows that lookahead model M4 improves the competitive ratio of (deterministic) on-line algorithms from at least 2.618 to 2.415.

8.2 Single Machine Scheduling with Job Availability Intervals (SMAI)

SMAI is very similar to SMR, with the addition that for each job j, there is also a deadline d_j. Each job's processing can start only within the interval $[r_j, d_j)$. In particular, if $d_j = r_j + 1$, then job j must be processed immediately at its arrival, or it is lost. The decisions to be made are which jobs to process and when. The objective is to minimize $\sum_j w_j(s_j - r_j)$, i.e. the sum of weighted *waiting* times for all the jobs, or equivalently to maximize the sum of early response premiums $\sum_j (d_j - s_j)$. Waiting time of a job is the difference between the start time s_j of the processing of a job and the job's arrival time. If a job is not processed, that job's waiting time is equal to the length of it availability window $l_j = d_j - r_j$.

SMAI with the maximization objective is considered in [33]. It is shown that for the most general version of the problem with arbitrary weights and processing times, it is not possible to have on-line algorithms (with lookahead model M4) with finite competitive ratio. Given this negative result, several special cases are studied, including the case with only a single type of orders (SMAI-ST), where all the processing times, weigths and availability intervals are equal.

For SMAI-ST, it is shown that an on-line algorithm with zero lookahead has competitive ratio 1.618. The best possible competitive ratios are 1.5 and $\sqrt{2}$ for algorithms with zero and positive lookahead (M4), respectively. For SMAI-ST, with the additional restriction that all the processing times are one unit (SMAI-ST1), an on-line algorithm is presented with lookahead $L = l_j = l$ for each job j, which finds the optimum solution. In addition, an online algorithm without lookahead is presented with competitive ratio 1.618. These results show that for SMAI-ST1, if an on-line algorithm is allowed to delay its decision about scheduling an order as long as the order is available, then it can find the optimum solution. Finally, it is shown that if the lookahead window is between zero and l, i.e. $L = (1 - \delta)l$, then there exists and algorithm with competitive ratio $\min\{1.618, \frac{1}{1-\delta^2}\}$. This result indicates that after a treshold for L is passed, the improvement in the competitive ratio is proportional to the length of the lookahead. Also note that the same results hold if the algorithms use the stronger lookahead model M2, instead of lookahead model M4.

9 Conclusion

In practice (and intuitively), the ability to use lookahead in on-line decision making seems very powerful. Hence, the influence of lookahead on the performance of on-line algorithms is receiving increasingly more attention in the literature.

Three factors seem to play an important role in theoretically observing the positive influence of lookahead on algorithmic performance: (1) the lookahead model, (2) the choice of the objective function, (3) the measure of performance.

It is very common to use competitive analysis to measure the performance of on-line algorithms. However, when measured by competitive ratio, the performance of many on-line algorithms seems to be insensitive to (constant) lookahead. For example, in the paging problem although no finite lookahead is sufficient for any improvement in the competitive ratio, an alternative measure called the Max/Max ratio indicates that the improvement in the performance of on-line algorithms is proportional to lookahead L. Therefore, one has to be very careful before reaching conclusions such as "finite lookahead does not improve performance of on-line algorithms for problem P".

The competitive ratio and how it reflects the influence of lookahead are also very sensitive to the choice of the objective function. Two different objective functions, which are equivalent in an off-line setting could result in significantly different competitive ratios for the same on-line algorithm, and give different messages about the influence of lookahead. For example, in scheduling the multipass assembly lines, minimizing makespan and minimizing the number of null tasks are equivalent objectives in an off-line setting. However, in an on-line setting competitive ratios clearly reflect the influence of lookahead with respect to the first objective, whereas they are insensitive to lookahead with respect to the second objective.

Finally, the objective function and the measure of performance being the same, the influence of lookahead depends on the lookahead model an on-line algorithm follows. We hope that the different models reviewed in this paper will be useful in examining the power of lookahead in other on-line problems.

References

[1] S. Albers (1997), "On the Influence of Lookahead in Competitive Paging Algorithms", *Algorithmica* 18, 283-305.

[2] S. Albers (1998), "A Competitive Analysis of the List Update Problem with Lookahead", *Theoretical Computer Science* 197, 95-109.

[3] K.R.Baker (1974), *Introduction to Sequencing and Scheduling*, Wiley, New York.

[4] L.A. Belady (1966), "A Study of Replacement Algorithms for a Virtual Storage Computer", *IBM Systems Journal* 5(2), 78-101.

[5] M. Bellare, O. Goldreich, M. Sudan (1995), "Free Bits, PCP and Non-approximability - Towards Tight Results", Proc. 36th Ann. IEEE Symp. on Foundations of Computer Science, 422-431.

[6] S. Ben-David, A. Borodin (1994), "A New Measure for the Study of On-line Algorithms", *Algorithmica* 11, 73-91.

[7] A. Blum (1994), “New Approximation Algorithms for Graph Coloring”, *Journal of the ACM* 41(3), 470-516.

[8] A. Blum, D. Karger (1997), “An $O(n^{3/14})$-Coloring Algorithm for 3-Colorable Graphs”, *Information Processing Letters* 61(1), 49-53.

[9] A. Borodin, R. El-Yaniv (1998), Online Computation and Competitive Analysis, Cambridge University Press.

[10] D. Breslauer (1996), “On Competitive On-line Paging with Lookahead”, Proceedings of 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS), 593-603.

[11] C. Chekuri, R. Motwani, B. Natarajan, C. Stein (1997), “Approximation Techniques for Average Completion Time Scheduling”, in *Proceedings of SODA'97* .

[12] F.R.K. Chung, R.L. Graham, M.E. Saks (1989), “ A Dynamic Location Problem for Graphs”, Combinatorica 9, 111-131.

[13] E.G. Coffman, M.R. Garey, D.S. Johnson (1984), “Approximation Algorithms for Bin Packing - An Updated Survey”, in Algorithm Design for Computer System Design, G. Ausiello, M. Lucertini, P. Serafini eds., Springer-Verlag, 49-106.

[14] J. Csirik, B. Imreh (1986), “On the Worst Case Performance of the NkF Bin Packing Heuristic”, *Acta Cybernetica* 9, 89-105.

[15] J. Csirik, D.S. Johnson (1991), “Bounded Space On-line Bin Packing: Best is Better Than First”, Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, 309-319.

[16] W.F. de la Vega, G.S. Lueker (1981), “Bin Packing Can Be Solved Within $1+\epsilon$ in Linear Time”, *Combinatorica* 1(4), 349-355.

[17] M.E. Dyer, L.A. Wolsey (1990), “Formulating the single machine sequencing problem with release dates as a mixed integer program”, *Discrete Applied Mathematics*, 26, 255-270.

[18] A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.D. Sleator, N.E. Young (1991), “Competitive Paging Algorithms”, *Journal of Algorithms* 12, 685-699.

[19] A. Fiat, G.J. Woeginger (1997), “On-Line Scheduling on a Single Machine: Minimizing the Total Completion Time”, TU Graz Report Woe-04.

[20] M.X. Goemans (1996), “A Supermodular Relaxation for Scheduling with Release Dates”, in *Proceedings of the 5th Integer Programming and Combinatorial Optimization Conference.*

[21] M.X. Goemans (1997), “Improved Approximation Algorithms for Scheduling with Release Dates”, in *Proceedings of the ??th ACM-SIAM Symposium on Discrete Algorithms*, 591-598.

[22] R.L.Graham, E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan (1979), "Optimiztion and approximation in deterministic sequencing and scheduling: A survey", *Annals of Discrete Mathematics*, No. 5, 287-326.

[23] E.F. Grove (1985), "Online Bin Packing with Lookahead", Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, 430-436.

[24] A. Gyarfas, J. Lehel (1988), "On-line and First Fit Coloring of Graphs", *Journal of Graph Theory* 12(2), 217-227.

[25] J.A.Hoogeveen, A.P.A.Vestjens (1996), "Optimal On-Line Algorithms for Single Machine Scheduling", in *Proceedings of the Fifth Conference on Integer Programming and Combinatorial Optimization*, 404-414.

[26] D.S. Johnson (1974), "Fast Algorithms for Bin-Packing", *Journal of Computer and System Sciences* 8, 274-314.

[27] S. Irani, N. Reingold, J. Westbrook, D.D. Sleator (1991), "Randomized Competitive Algorithms for the List Update Problem", Proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms, 251-260.

[28] S. Irani (1994), "Coloring Inductive Graphs On-line", Algorithmica 11, 53-72.

[29] D. Karger, R. Motwani, M. Sudan (1994), "Approximate Graph Coloring by Semidefinite Programming", Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, 2-13.

[30] N. Karmarkar, M. Karp (1982), "An Efficient Approximation Scheme for the One-dimensional Bin Packing Problem", Proceedings of the 23rd Annual Symposium on Foundations of Computer Science.

[31] R. Karp, P. Raghavan, Personal communication, transmitted through [2].

[32] R.M. Karp (1972), "Reducability Among Combinatorial Problems", in Complexity of Computer Computations, R.E. Miller and J.W. Thatcher eds., Plenum Press, NY, 85-103.

[33] P. Keskinocak (1997), "Satisfying Customer Due Dates Effectively", Ph.D. Thesis, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.

[34] H.A. Kierstead (1988), "The Linearity of First Fit Coloring of Interval Graphs", *SIAM Journal of Discrete Mathematics* 1(4), 526-530.

[35] H.A. Kierstead, W.A. Trotter (1981), "An Extremal Problem in Recursive Combinatorics", *Congress. Numer.* 33, 143-153.

[36] J.Labetoulle, E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan (1984), "Preemptive scheduling of uniform machines subject to release dates", in *Progress in Combinatorial Optimization*, W.R.Pulleyblank editor, 245-261.

[37] C.C. Lee, D.T. Lee (1985), "A Simple On-line Packing Algorithm", *Journal of the ACM* 32, 562-572.

[38] J.K.Lenstra, A.H.G.Rinnooy Kan, P.Brucker (1977), "Complexity of Machine Scheduling Problems", *Annals of Discrete Mathematics*, Vol. 1, 343-362.

[39] L. Lovasz, M.E. Saks, W.A. Trotter (1988), "An On-line Graph Coloring Algorithm with Sublinear Preformance Ratio", *Discrete Mathematics*, Special Volume on Graph Theory and Combinatorics, 319-326.

[40] S. Mahajan, J. Ramesh (1995), "Derandomizing Semidefinite Programming Based Approximation Algorithms", Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, 162-169.

[41] W. Mao (1993), "Tight Worst Case Performance Bounds for Next-k-fit Bin Packing", *Siam Journal on Computing* 22(1), 46-56.

[42] W.Mao, R.K.Kincaid, A.Rifkin (1995), "On-line algorithms for single machine scheduling problem", in *The impact of emerging technologies on computer science and operations research*, S.G.Nash and A.Sofer eds., Kluwer Academic Press, 157-173.

[43] L.A. McGeoch, D.D. Sleator (1991), "A Strongly Competitive Randomized Paging Algorithm", Algorithmica 6, 816-825.

[44] R. Motwani, V. Saraswat, E. Torng (1998), "Online Scheduling with Lookahead: Multipass Assembly Lines", *Informs Journal on Computing*, to appear.

[45] C. Papadimitriou, K. Steiglitz (1982), Combinatorial Optimization, Prentice Hall.

[46] C.Phillips, C.Stein, J.Wein (1995), "Minimizing Average Completion Time in the Presence of Release Dates", to appear in *Mathematical Programming*, earlier version appeared in *Algorithms and Data Structures: 4th International Workshop, WADS'95, Kingston, Canada, August 1995, Proceedings*, 86-97.

[47] N. Reingold, J. Westbrook, D.D. Sleator (1994), "Randomized Competitive Algorithms for the List Update Problem", *Algorithmica* 11, 15-32.

[48] A.S. Schulz, "Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds", in *Proceedings of the 5th Integer Programming and Combinatorial Optimization Conference*, 301-315.

[49] D.D. Sleator, R.E. Tarjan (1985), "Amortized Efficiency of List Update and Paging Rules", *Communication of the ACM* 28, 202-208.

[50] E. Torng (1998), "A Unified Analysis of Paging and Caching", *Algorithmica* 20, 175-200.

[51] S. Vishwanathan (1992), “Randomized Online Graph Coloring”, *Journal of Algorithms* 13(4), 657-669.

[52] A.C. Yao (1980), “New Algorithms for Bin Packing”, *Journal of the ACM* 2, 207-227.

[53] N. Young (1991), Competitive Paging and Dual-Guided On-Line Weighted Caching and Matching Algorithms, Ph.D. Thesis, Princeton University.