MAX-PLANCK-INSTITUT FÜR INFORMATIK

The Influence of Lookahead in Competitive On-Line Algorithms

Susanne Albers

MPI-I-92-143 September 1992

Im Stadtwald
66123 Saarbrücken
Germany

The Influence of Lookahead in Competitive On-Line Algorithms

Susanne Albers

MPI–I–92–143 September 1992

The Influence of Lookahead in Competitive On-Line Algorithms

Susanne Albers*
Max-Planck-Institut für Informatik
and
Graduiertenkolleg Informatik
Universität des Saarlandes

September 24, 1992

Abstract

In the competitive analysis of on-line problems, an on-line algorithm is presented with a sequence of requests to be served. The on-line algorithm must satisfy each request without the knowledge of any future requests. We consider the question of lookahead in on-line problems: What improvement can be achieved in terms of competitiveness, if the on-line algorithm sees not only the present request but also some future requests? We introduce two different models of lookahead and study the "classical" on-line problems such as paging, caching, the k-server problem, the list update problem and metrical task systems using these two models. We prove that in the paging problem and the list update problem lookahead can significantly reduce the competitive factors of on-line algorithms without lookahead. In addition to lower bounds we present a number of on-line algorithms with lookahead for these two problems. However, we also show that in more general on-line problems such as caching, the k-server problem and metrical task systems lookahead cannot improve competitive factors of deterministic on-line algorithms without lookahead.

1 Introduction

Recently the competitive analysis of on-line algorithms has evoked great interest [ST85, BLS87, KMRS88, MMS88, BBKTW90]. The on-line problems studied extensively include paging, caching and the k-server problem as well as the list update problem and metrical task systems. All these problems can generally be formulated as follows. An on-line algorithm is presented with a sequence of requests which must be served in the order of occurrence. In particular, the on-line algorithm must satisfy each request without the knowledge of any future requests. The processing of requests incurs cost, and the goal is to minimize the cost incurred on the entire request sequence.

Sleator and Tarjan [ST85] have proposed a new measure for analyzing the performance of on-line algorithms. In their work on the list update problem and paging they compared on-line

*Research supported by a graduate fellowship of the Deutsche Forschungsgemeinschaft.

algorithms to an optimal off-line algorithm which knows the entire request sequence in advance and can serve it with minimum cost. Karlin *et al.* [KMRS88] used this approach to analyze on-line snoopy caching algorithms, and introduced the notion of *competitive analysis*. An on-line algorithm is called *c-competitive* if, for any request sequence, its cost does not exceed c times the cost of the optimal off-line algorithm for that sequence. More formally, let $C_A(\sigma)$ and $C_{OPT}(\sigma)$ be the cost of the on-line algorithm A and the optimal off-line algorithm OPT on request sequence σ. Then the algorithm A is c-competitive, if there exists a constant a such that

$$C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + a$$

for all request sequences σ.

The *competitive factor* of A is the infimum of all c such that A is c-competitive. If A is a randomized algorithm, then $C_A(\sigma)$ is the expected cost incurred by A on request sequence σ. In this paper we evaluate the performance of randomized on-line algorithms only against the *oblivious adversary* (see [BBKTW90] for details). The oblivious adversary specifies a request sequence in advance and pays the optimal off-line cost.

We study the problem of *lookahead* in on-line algorithms. An important question is, what improvement can be achieved in terms of competitiveness, if an on-line algorithm knows not only the present request to be served, but also some future requests. This issue is fundamental from the practical and theoretical point of view. In practical applications the requests do not necessarily arrive one after the other, but rather in blocks of possibly variable size. In paging systems there is generally a gap in speed between the memory and the faster CPU. Hence it is to be expected that some requests usually wait in line to be processed by a paging algorithm. In fact, some paging algorithms used in practice make use of lookahead [S77]. In the theoretical context a natural question is: What is it worth to know the future? We investigate the question of lookahead for on-line problems such as the list-update problem, paging, caching, the k-server problem and metrical task systems.

We introduce two different models of lookahead. Let $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ be a request sequence of length m. $\sigma(t)$ denotes the request at time t. Let R be the set of possible requests. Two requests $\sigma(t)$ and $\sigma(s)$ are called *equal* ("$\sigma(t) = \sigma(s)$"), if they represent the same element in R; otherwise they are called *distinct*. For a given set S, $card(S)$ denotes the cardinality of S. Let $l \geq 1$ be an integer.

Weak lookahead of size l: The on-line algorithm sees the present request and the next l future requests. More specifically, when answering $\sigma(t)$, the on-line algorithm already knows $\sigma(t+1), \sigma(t+2), \ldots, \sigma(t+l)$. However, requests $\sigma(s)$, with $s \geq t+l+1$, are not seen by the on-line algorithm at time t.

Strong lookahead of size l: The on-line algorithm sees the present request and a sequence of future requests. This sequence contains l pairwise distinct requests which also differ from the present request. More precisely, when serving request $\sigma(t)$, the algorithm knows requests $\sigma(t+1), \sigma(t+2), \ldots, \sigma(t')$, where $t' = \min\{s > t | card(\{\sigma(t), \sigma(t+1), \ldots, \sigma(s)\}) = l+1\}$. The requests $\sigma(s)$, with $s \geq t'+1$, are not seen by the on-line algorithm at time t.

In the following, when we investigate on-line problems with lookahead, l always denotes the size of the lookahead. We always assume that an on-line algorithm has a lookahead of fixed size $l \geq 1$. If a strong lookahead of size l is given, then for all $t \geq 1$ we define a value $\lambda(t)$. If $card(\{\sigma(t), \sigma(t+1), \ldots, \sigma(m)\}) < l+1$ then let $\lambda(t) = m$; otherwise let $\lambda(t) = \min\{t' > t | card(\{\sigma(t), \sigma(t+1), \ldots, \sigma(t')\}) = l+1\}$. Suppose an on-line algorithm has a lookahead of size l. We define *the lookahead $L(t)$ at time t*. If a weak lookahead is given, then

$$L(t) = \{\sigma(s) | s = t, t+1, \ldots, \min\{t+l, m\}\}.$$

If a strong lookahead is given, then

$$L(t) = \{\sigma(s) | s = t, t+1, \ldots, \lambda(t)\}.$$

At first sight weak lookahead seems to be the intuitive model of lookahead. However, as we shall see later, weak lookahead is usually only of minor advantage in the on-line problems we study here. The reason is that an adversary that constructs a request sequence can replicate requests in the lookahead, thereby weakening the effect of the lookahead. Strong lookahead is of special interest in the theoretical context, when we ask how significant it is to know part of the future. An adversary may replicate requests in the lookahead, but nevertheless it has to reveal some really significant information on future requests. We show that strong lookahead can sometimes significantly reduce the competitive factors of on-line algorithms without lookahead.

So far, only few on-line problems with lookahead have been studied in the literature. Specifically, previous research on lookahead has not addressed the on-line problems we consider here. Chung *et al.* [CGS89] examine dynamic location problems with lookahead. They identify graphs for which the dynamic location problem can be solved optimally using a constant lookahead. The remaining results on lookahead were obtained in the area of on-line graph problems. In these problems our models of weak and strong lookahead coincide. Irani [I90] and Halldórsson and Szegedy [HS92] study on-line coloring of inductive and general graphs. Kao and Tate [KT91] examine on-line matching of bipartite graphs. In these problems lookahead is only of advantage, if its size is relatively large and depends on the size n of the entire graph. A constant lookahead (independent of n) is basically of no help.

2 Outline of results

The two main on-line problems we address in this paper are paging and the list update problem.

The paging problem: Consider a two-level memory system. There exists a fast memory, a *cache*, consisting of k pages, and a slow memory consisting of $n-k$ pages. A sequence of requests to pages in the memory system must be served by an on-line algorithm. A request is served if the corresponding page is in cache. A *page fault* occurs if the requested page is not in the cache. Then a page from fast memory must be moved to slow memory, such that the requested page can be loaded into the cache. A paging algorithm specifies which page to evict on a page fault. The goal is to minimize the number of faults.

Section 3 and Section 4 are an in-depth study of paging with strong and weak lookahead. We show that strong lookahead significantly reduces competitive factors of deterministic and randomized on-line paging algorithms without lookahead; even a lookahead of constant size is of advantage. In addition to lower bounds we give a number of deterministic and randomized on-line paging algorithms with strong lookahead which are optimal or nearly optimal. To our knowledge, paging with strong lookahead is the first example of an on-line problem in which a lookahead of constant size helps to reduce the competitive factors of algorithms without lookahead. As opposed to strong lookahead, weak lookahead cannot improve competitive factors of deterministic and randomized on-line paging algorithms. For every on-line algorithm A with a weak lookahead of fixed size, there exists a request sequence σ such that $C_A(\sigma) \geq c \cdot C_{OPT}(\sigma)$. Here c denotes the best competitive factor of a deterministic or randomized on-line paging algorithm without lookahead. However an on-line algorithm A with weak lookahead l can satisfy $C_A(\sigma) < c \cdot C_{OPT}(\sigma)$ for all request sequences of a restricted length $m \leq \overline{m}$; the value of $\overline{m}$ depends of the size of l. We bound the value of $\overline{m}$, i.e. we evaluate which part of a given request sequence σ has to be seen by an on-line algorithm A in order to satisfy $C_A(\sigma) < c \cdot C_{OPT}(\sigma)$. We complement these results by presenting on-line algorithms with weak lookahead which are optimal or nearly optimal for request sequences of restricted length.

The list update problem: The list update problem consists of maintaining a set of items as an unsorted linear list. A list of items is given. Each request is an access to an item in the list. In order to serve an access, a list update algorithm starts at the front of the list and searches linearly through the items until the desired item is found. Accessing the ith item in the list incurs a cost of i. At any time, a list update algorithm may exchange adjacent items in the list. Immediately after an access, 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*. All other exchanges cost 1 and are called *paid exchanges*. The goal is to serve the request sequence such that the total cost is as small as possible.

Section 5 addresses the deterministic list update problem with strong and weak lookahead. We show that an on-line algorithm requires a strong lookahead of size $\Omega(n)$ in order to be better than 2-competitive. The factor of 2 is the best possible competitive factor of an on-line algorithm without lookahead. If an on-line algorithm is given a weak lookahead, the situation is worse. A lookahead of size $\Omega(n^2)$ is necessary to asymptotically beat the competitive factor of 2. We give simple on-line algorithms with strong and weak lookahead which are competitive, if the off-line algorithm uses a *static* algorithm. Furthermore, we present a more complicated algorithm which is competitive against dynamic off-line algorithms.

We also touch the following problems.

Caching problems: The structure of caching problems resembles that of paging problems. The difference is that loading different items into the cache can cause different costs. The goal is to minimize the cost incurred for loading items into the cache. An example of a caching problem is the caching of fonts in printers; see [MMS90] for a more detailed description of applications.

The k-server problem: The general version of the k-server problem was introduced by Manasse *et al.* [MMS88]. It generalizes paging and caching problems as well as some other important scheduling problems. The k-server problem is that of scheduling the motion of k mobile servers

which cover vertices in a complete directed graph G. The edge lengths are non-negative reals and satisfy the triangle inequality. A request sequence consisting of vertices in G must be served. In response to each request, a server must be moved to the requested vertex, unless a server is already present. The goal is to minimize the total distance traveled by the servers. A k-server problem is called *symmetric*, if the distance from vertex i to vertex j equals the distance from j to i for all vertices i and j; otherwise it is called *asymmetric*.

Metrical task systems: These systems were introduced by Borodin *et al.* [BLS87] and generalize a large class of on-line problems. A task system (S, d) consists of a set $S = \{1, 2, \ldots, n\}$ of n states and an $n \times n$ cost matrix d, where $d(i, j) \geq 0$ denotes the cost of changing from state i to state j. The matrix d satisfies the triangle inequality and the diagonal entries are zero. At any time the task system resides in one state of S. A request sequence equals a sequence of tasks. A task T is a vector $T = (T(1), T(2), \ldots, T(n))$, where $T(i)$ denotes the cost of processing the task while in state i. Given a task sequence $\sigma = T^1, T^2, T^3, \ldots$ and an initial state $s(0)$, an algorithm for metrical task systems must determine a schedule of states $s(1), s(2), \ldots$, 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. The goal is to process a given task sequence such that the cost is as small as possible.

Section 6 deals with more general on-line problems such as caching, the k-server problem and metrical task systems. We prove that in these more general on-line problems neither weak nor strong lookahead can generally improve the competitive factors of deterministic on-line algorithms without lookahead.

3 Competitive paging with strong lookahead

First we review the main results known for paging without lookahead. Sleator and Tarjan [ST85] have demonstrated that the well-known replacement algorithms LRU (Least Recently Used) and FIFO (First-In First-Out) are k-competitive. On a fault LRU removes the page whose most recent request was earliest, and FIFO evicts the page that has been in the cache longest. Sleator and Tarjan have also proved that no on-line paging algorithm can be better than k-competitive; hence LRU and FIFO achieve the best competitive factor. Fiat *et al.* [FKLSY91] have shown that no randomized on-line paging algorithm can be better than $H(k)$-competitive against an oblivious adversary. Here $H(k) = \sum_{i=1}^{k} 1/i$ denotes the kth harmonic number. They have also given a simple replacement algorithm, called the MARKER algorithm, which achieves a competitive factor of $2H(k)$. McGeoch and Sleator [MS91] have proposed a more complicated randomized paging algorithm which is $H(k)$-competitive.

The paging problem is isomorphic to the *uniform* k-server problem on an n-vertex graph. In the uniform k-server problem, all edge lengths equal one. The n vertices in the graph correspond to the n pages in the memory system, and the vertices covered by the k servers correspond to the pages in cache. In the sections on paging we generally use the terminology of the k-server problem rather than that of the paging problem. A fault occurs if there is a request to an uncovered vertex. If a page is brought into cache, then a server is moved to that vertex.

Belady [B66] has exhibited an optimal off-line algorithm for the paging problem, which is also called the MIN algorithm. On a fault, MIN vacates the vertex whose next request occurs farthest in the future.

Unless otherwise stated, we assume in the following that all our paging algorithms are *lazy* algorithms, i.e. they only move a server if it is necessary in order to serve a request to an uncovered vertex. It is shown in [MMS91] that any algorithm for the symmetric k-server problem can be converted into a lazy algorithm without increasing its cost.

3.1 Deterministic algorithms

We assume that an on-line paging algorithm has a strong lookahead of size l. First we consider the important case $l \leq k-2$. The on-line paging algorithms we present are extensions of the algorithm LRU to our model of strong lookahead.

Algorithm LRU(l): On a fault execute the following steps. Among the covered vertices which are not contained in the present lookahead determine the vertex whose last request occurred least recently. Choose the server covering this vertex, and move it to the request.

Theorem 1 *Let $l \leq k-2$. The algorithm LRU(l) with strong lookahead l is $(k-l)$-competitive.*

Now we prove this theorem. Let $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ be a request sequence of length m. We assume without loss of generality that LRU(l)'s and OPT's servers are initially all on vertex 1. The following proof consists of three main parts. First, we introduce the potential function we use to analyze LRU(l). In the second part, we partition the request sequence σ into a series of phases and then, in the third part, we bound LRU(l)'s cost in each phase.

1. The potential function

We introduce some basic notations. For $t = 1, 2, \ldots, \lambda(1)-1$, let $\mu(t) = 1$ and for $t = \lambda(t), \lambda(t)+1, \ldots, m$, let

$$\mu(t) = \max\{t' < t | card(\{\sigma(t'), \sigma(t'+1), \ldots, \sigma(t)\}) = l+1\}.$$

Define

$$M(t) = \{\sigma(s) | s = \mu(t), \mu(t)+1, \ldots, t\}.$$

For a given time t, the set $M(t)$ contains the last $l+1$ requested vertices.

Let $S_{LRU(l)}(t)$ be the set of vertices covered by LRU(l) after request t, and let $S_{OPT}(t)$ be the set of vertices covered by OPT after request t. For the analysis of the algorithm we assign weights to all vertices. These weights are updated after each request. Let $w(x,t)$ denote the weight of vertex x after request t. The weights are set as follows. If $x \notin S_{LRU(l)}(t)$ or $x \in L(t)$, then

$$w(x,t) = 0.$$

Let $j = card(S_{LRU(l)}(t) \setminus L(t))$. Assign integer weights from the range $[1, j]$ to the vertices in $S_{LRU(l)}(t) \setminus L(t)$ such that

$$w(x,t) < w(y,t)$$

iff the last request to x occurred earlier than the last request to y. If a vertex $x \in S_{LRU(l)}(t) \setminus L(t)$ has not been requested so far (note that this may apply only to the initially covered vertex 1), we assume that the last request to x occurred earlier than a request to any other vertex in $S_{LRU(l)}(t) \setminus L(t)$.

We now define the potential function:

$$\Phi(t) = \sum_{x \in S_{LRU(l)}(t) \setminus \{M(t) \cup S_{OPT}(t)\}} w(x,t)$$

2. The partitioning of the request sequence

We will partition the request sequence σ into $p+1$ phases, numbered from 0 to p, such that phase 0 contains at most $l+1$ distinct vertices and phase i, $i = 1, 2, \ldots, p$, satisfies the following two properties. Let t_i^b and t_i^e denote the beginning and the end of phase i, respectively.

Property 1: Phase i contains exactly $l+1$ distinct vertices, i.e.

$$card(\{\sigma(t_i^b), \sigma(t_i^b + 1), \ldots, \sigma(t_i^e)\}) = l + 1.$$

Property 2: For all $x \in S_{LRU(l)}(t_{i-1}^e) \setminus \{L(t_i^b) \cup S_{OPT}(t_{i-1}^e)\}$,

$$w(x, t_i^e) \leq k - l - 2.$$

We claim that the following algorithm generates a desired partition.

1. $i := m$;
2. $t_i^e := m$;
3. **repeat**
4. $\quad t_i^b := \mu(t_i^e)$;
5. $\quad$ Let x be the vertex in $S_{LRU(l)}(t_i^b - 1) \setminus L(t_i^b)$ that was requested most recently among vertices in $S_{LRU(l)}(t_i^b - 1) \setminus L(t_i^b)$;
6. $\quad$ **if** $t_i^b = 1$ **or** $x \in S_{OPT}(t_i^b - 1)$ **then**
7. $\quad\quad$ Let $P(i) = \sigma(t_i^b), \sigma(t_i^b + 1), \ldots, \sigma(t_i^e)$ be the ith phase, and call this phase of type 1;
8. $\quad$ **else**
9. $\quad\quad$ Determine the largest $t' < t_i^b$, such that OPT vacates vertex x at time t';
10. $\quad\quad t_i^b := t'$;
11. $\quad\quad$ Let $P(i) = \sigma(t_i^b), \sigma(t_i^b + 1), \ldots, \sigma(t_i^e)$ be the ith phase, and call this phase of type 2;
12. $\quad$ **endif**;
13. $\quad i := i - 1$;
14. $\quad t_i^e := t_{i+1}^b - 1$;
15. **until** $t_i^e = 0$;
16. Number the phases from 0 to p;

Note that phase 0 cannot be of type 2 because in this case OPT would vacate vertex 1 at the first request. Hence phase 0 is of type 1 and contains at most $l+1$ distinct vertices.

Lemma 1 *The partition generated by the above algorithm satisfies Property 1 and Property 2.*

Proof: Let $P(i)$, $1 \leq i \leq p$, be an arbitrary phase. First we prove Property 1.

If $P(i)$ is of type 1, then Property 1 clearly holds. If $P(i)$ is of type 2, then let $t = \mu(t_i^e)$ and let x be the vertex in $S_{LRU(l)}(t-1) \setminus L(t)$ that was requested most recently among vertices in $S_{LRU(l)}(t-1) \setminus L(t)$. The vertex x is not requested in $P(i)$ (see step 9 of the above algorithm). Let y be a vertex that is requested at time t', where $t_i^b \leq t' < t$. We have $x \notin L(s)$ for all $t_i^b \leq s < t$. Since x was not vacated in the interval $[t_i^b, t-1]$, the vertex y must also be in $S_{LRU(l)}(t-1)$. It follows that $y \in L(t)$. This shows that $P(i)$ contains exactly $l+1$ vertices.

Now we prove Property 2. Consider an arbitrary $x \in S_{LRU(l)}(t_{i-1}^e) \setminus \{L(t_i^b) \cup S_{OPT}(t_{i-1}^e)\}$. If $w(x, t_i^e) = 0$, then the property clearly holds. Therefore, assume $w(x, t_i^e) \geq 1$. By Property 1, $L(t_i^b)$ contains all vertices which are requested in phase i and $L(t_i^b) = M(t_i^e)$. If $P(i)$ is of type 1, then there must exist a vertex $y \in S_{LRU(l)}(t_{i-1}^e) \setminus L(t_i^b)$ which was requested more recently than x. We have $w(x, t_i^e) \geq 1$, which implies $x \notin L(t)$ for all $t_i^b \leq t \leq t_i^e$. Since x was not vacated during phase i, vertex y and all vertices in $L(t_i^b)$ must also be contained $S_{LRU(l)}(t_i^e)$. All $z \in L(t_i^b) \cup \{y\}$ satisfy $w(z, t_i^e) = 0$ or $w(z, t_i^e) > w(x, t_i^e)$. Hence $w(x, t_i^e) \leq k - l - 2$.

If the phase is of type 2, then OPT vacates a vertex $y \notin L(t_i^b)$ at time t_i^b. Note that $y \in S_{LRU(l)}(t_{i-1}^e)$ and $x \neq y$. At time t_i^b, the last request to y is more recent than the last request to x. Since $w(x, t_i^e) \geq 1$, we have $x \notin L(t)$ for all $t_i^b \leq t \leq t_i^e$. As above we conclude that vertex y and all vertices in $L(t_i^b)$ must also be contained in $S_{LRU(l)}(t_i^e)$. Hence $w(x, t_i^e) \leq k-l-2$. This completes the proof of the lemma. □

3. Bounding LRU(l)'s amortized cost

We examine the effect of LRU(l)'s and OPT's moves on the potential function Φ. We generally assume that in response to a request, OPT moves first and LRU(l) moves second. First we investigate the effect of a move made by OPT.

Lemma 2 *If OPT moves a server during phase $P(i)$, this can increase the potential in either the current phase $P(i)$ or the next phase $P(i+1)$ (if this phase exists) by at most $k-l-1$.*

Proof: Suppose OPT moves a server at time t during phase $P(i)$, and vacates a vertex x that is in $S_{LRU(l)}(t)$. Such a move can increase the potential.

If the potential does increase due to this move, then there must exist a $t' \geq t$ such that

$$x \in S_{OPT}(t'-1) \text{ or } x \in M(t'-1)$$

and

$$x \in S_{LRU(l)}(t') \setminus S_{OPT}(t') \text{ and } x \notin M(t') \text{ and } x \notin L(t')$$

and x is not requested during the interval $[t, t']$. Note that $x \notin L(s)$ for all $\mu(t') \leq s \leq t'$. Since x was not vacated by LRU(l) during $[\mu(t'), t']$, all vertices in $M(t')$ must be in $S_{LRU(l)}(t')$. At time t', all vertices in $M(t')$ have a weight of 0 or a weight which is greater than $w(x, t')$. Hence $w(x, t') \leq k - l - 1$ and the potential can increase by at most $k - l - 1$. If $t' \leq t_i^e$, then

the potential increases in the current phase. Suppose $t' > t_i^e$. Since $P(i+1)$ contains $l+1$ distinct vertices (see Property 1), the vertex x is not requested in phase $P(i+1)$. Thus we have $x \notin M(t_{i+1}^e)$. We conclude $t' \leq t_{i+1}^e$ because $x \notin L(s)$ for all $t \leq s \leq t'$. Hence the potential can only increase in either the current phase $P(i)$ or the next phase $P(i+1)$. □

Now, let $C_{OPT}(i)$ be the cost incurred by OPT during phase i, let $C'_{OPT}(i)$ be the number of moves during phase i which increase the potential in that phase and let $C''_{OPT}(i) = C_{OPT}(i) - C'_{OPT}(i)$. Furthermore, let $C_{LRU(l)}(i)$ be the cost incurred by $LRU(l)$ during the ith phase.

Lemma 3 *For $i = 1, 2, \ldots, p$,*

$$C_{LRU(l)}(i) + \Phi(t_i^e) - \Phi(t_{i-1}^e) \leq C_{OPT}(i) + (k-l-1)(C'_{OPT}(i) + C''_{OPT}(i-1)).$$

Proof: Consider any $1 \leq i \leq p$. By Lemma 2, OPT's moves can increase the potential in phase i by at most $(k-l-1)(C'_{OPT}(i) + C''_{OPT}(i-1))$. If LRU($l$) makes no move during phase i, then the lemma clearly holds. Therefore, suppose $C_{LRU(l)}(i) \geq 1$.

We examine the effect of LRU(l)'s moves on the potential function Φ. Obviously, LRU(l)'s moves cannot increase the potential. We claim that if there exists a vertex

$$x \in S_{LRU(l)}(t) \setminus \{L(t) \cup M(t) \cup S_{OPT}(t)\}$$

at time t and if LRU(l) makes a move at time $t+1$, then x's weight must decrease by a least 1. Hence the potential decreases by at least 1. The claim clearly holds if x is the vacated vertex itself or if $x \in L(t+1)$. If x is not the vacated vertex and if $x \notin L(t+1)$, then there must be a vertex

$$y \in S_{LRU(l)}(t) \setminus \{L(t) \cup M(t)\}$$

which is vacated and whose last request is longer ago than x's last request. After y is vacated x's weight decreases by 1.

Define $\tilde{C}(i) = card(S_{LRU(l)}(t_{i-1}^e) \setminus \{L(t_i^b) \cup S_{OPT}(t_{i-1}^e)\})$. We show that the following inequality holds.

$$\tilde{C}(i) + \Phi(t_i^e) - \Phi(t_{i-1}^e) \leq (k-l-1)(C'_{OPT}(i) + C''_{OPT}(i-1)) \qquad (1)$$

This inequality proves the lemma because, during phase i, LRU(l) makes at most $\tilde{C}(i)$ moves more than OPT, i.e.

$$C_{LRU(l)}(i) \leq C_{OPT}(i) + \tilde{C}(i).$$

In order to prove inequality (1), we have to balance a cost of $\tilde{C}(i)$. Consider any $x \in S_{LRU(l)}(t_{i-1}^e) \setminus \{L(t_i^b) \cup S_{OPT}(t_{i-1}^e)\}$. We distinguish two cases.

Case 1: For $t = t_{i-1}^e, t_i^b, t_i^b + 1, \ldots, t_i^e$, $x \notin S_{LRU(l)}(t) \setminus \{L(t) \cup M(t) \cup S_{OPT}(t)\}$
In this case x does not increase the potential in phase i and we charge a cost of 1 to the move at which OPT vacates x. This move is counted in $C''_{OPT}(i-1)$.

Case 2: There exists a t, $t_{i-1}^e \leq t \leq t_i^e$, such that $x \in S_{LRU(l)}(t) \setminus \{L(t) \cup M(t) \cup S_{OPT}(t)\}$
In this case let $t_{\min}$ be the smallest t with this property. If $t_{\min} = t_{i-1}^e$ then x's weight must

decrease by at least 1 before or at the first move by LRU(l) during phase i. This causes a decrease of the potential function by at least 1. If $t_{\min} \geq t_i^b$ then consider $w(x, t_{\min})$ and investigate two cases. If $w(x, t_{\min}) = k-l-1$, then x's weight must decrease by at least 1 during phase i because $w(x, t_i^e) \leq k-l-2$ (see Property 2). Hence, x causes a decrease of the potential function by at least 1. If $w(x, t_{\min}) \leq k-l-2$, we may charge an additional cost of 1 to the move at which OPT vacated x. This move is counted in $C''_{OPT}(i-1)$. This completes the proof of inequality (1). □

Using Lemma 3 a proof of Theorem 1 is very simple. Phase 0 contains at most $l+1$ vertices Thus, during this phase LRU(l) does not incur a higher cost than OPT, i.e.

$$C_{LRU(l)}(0) + \Phi(t_0^e) - \Phi(0) \leq C_{OPT}(0) + (k-l-1)C'_{OPT}(0),$$

where $\Phi(0) = 0$ is the initial potential. Summing up this inequality and the inequalities of Lemma 3 for $i = 1, 2, \ldots, p$ we obtain

$$C_{LRU(l)}(\sigma) + \Phi(m) - \Phi(0) \leq (k-l)C_{OPT}(\sigma).$$

The proof of Theorem 1 is complete.

Next we present another on-line algorithm with strong lookahead. This algorithm does not use full lookahead but rather serves the request sequence in a series of blocks.

Algorithm LRU(l)-blocked: Serve the request sequence in a series of blocks $B(1), B(2), \ldots$, where $B(1) = \sigma(1), \sigma(2), \ldots, \sigma(\lambda(1))$ and $B(i) = \sigma(t_{i-1}^e + 1), \sigma(t_{i-1}^e + 2), \ldots, \sigma(\lambda(t_{i-1}^e + 1))$ for $i \geq 2$. Here t_{i-1}^e denotes the end of block $B(i-1)$. If there occurs a fault while $B(i)$ is processed, then the following rule applies. Among the covered vertices which are not contained in $B(i)$, determine the vertex whose last request occurred least recently. The server covering this vertex is moved to the request.

Interestingly, this algorithm is only slightly weaker than LRU(l).

Theorem 2 *Let $l \leq k-2$. The algorithm LRU(l)-blocked with strong lookahead l is $(k-l+1)$-competitive.*

Proof: We assume that the request sequence consists of b blocks $B(1), B(2), \ldots, B(b)$. For $i = 1, 2, \ldots, b$, let t_i^b and t_i^e denote the beginning and the end of block $B(i)$, respectively. We assume that both LRU(l)-blocked's and OPT's servers are initially all on vertex 1.

The potential function we use to analyze LRU(l)-blocked is similar to the function we introduced in the proof of Theorem 1. For $t \geq 1$, the values $\mu(t)$ and the sets $M(t)$ are defined as in the previous proof. Let $S_L(t)$ denote the set of vertices covered by LRU(l)-blocked after request t, and let $S_{OPT}(t)$ be the set of vertices covered by OPT after request t. For $t \geq 1$, we define values $\gamma(t)$. Set $\gamma(t) = i$ iff $t_i^b \leq t \leq t_i^e$. Let $S_B(t)$ be the set of vertices that are requested during block $B(\gamma(t))$.

Again, we assign weights to all vertices. Let $w(x,t)$ denote the weight of vertex x after request t. If $x \notin S_L(t)$ or if $x \in S_B(t)$ then $w(x,t) = 0$. Let $j = card(S_L(t) \setminus S_B(t))$. Assign integer weights from the range $[1, j]$ to the vertices in $S_L(t) \setminus S_B(t)$ such that

$$w(x,t) < w(y,t)$$

iff the last request to x occurred earlier than the last request to y. Again, if a vertex $x \in S_L(t) \setminus S_B(t)$ has not been requested so far, we assume that the last request to x occurred earlier than a request to any other vertex in $S_L(t) \setminus S_B(t)$.

The potential function is defined as

$$\Phi(t) = \sum_{x \in S_L(t) \setminus \{M(t) \cup S_{OPT}(t)\}} w(x,t).$$

Using a similar analysis as in the proof of Lemma 2 we are able to show

Lemma 4 *If OPT moves a server while processing block $B(i)$, this can increase the potential in either the current block $B(i)$ or the next block $B(i+1)$ (if this block exists) by at most $k-l-1$.*

Let $C_L(i)$ be the cost incurred by LRU(l)-blocked during block i. Furthermore, let $C_{OPT}(i)$ be the cost incurred by OPT during block i, let $C'_{OPT}(i)$ be the number of moves by OPT during block i which increase the potential in that block and let $C''_{OPT}(i) = C_{OPT}(i) - C'_{OPT}(i)$. Define $t^e_0 = 0$.

Lemma 5 *For $i = 2, 3, \ldots, b$,*

$$C_L(i) + \Phi(t^e_i) - \Phi(t^e_{i-1}) \leq C_{OPT}(i) + (k-l)(C'_{OPT}(i) + C''_{OPT}(i-1)).$$

Proof: Consider any $2 \leq i \leq b$. By Lemma 4 OPT's moves can increase the potential in block i by at most $(k-l-1)(C'_{OPT}(i) + C''_{OPT}(i-1))$. Thus, if LRU($l$)-blocked does not incur any cost during block i, then the lemma clearly holds. Therefore, we assume $C_L(i) \geq 1$.

We investigate the effect of LRU(l)-blocked's moves on the potential function Φ. Obviously, LRU(l)-blocked's moves do not increase the potential. If there exists a vertex

$$x \in S_L(t) \setminus \{S_B(t) \cup M(t) \cup S_{OPT}(t)\}$$

at time t and if LRU(l)-blocked makes a move at time $t+1$, then the x's weight decreases by at least 1. Hence the potential decreases by at least 1. This claim can be proved in the same way as in the proof of Lemma 3.

Let $\tilde{C}(i) = card(S_L(t^e_{i-1}) \setminus \{S_B(t^b_i) \cup S_{OPT}(t^e_{i-1})\})$. We show that

$$\tilde{C}(i) + \Phi(t^e_i) - \Phi(t^e_{i-1}) \leq (k-l)(C'_{OPT}(i) + C''_{OPT}(i-1)). \quad (2)$$

This inequality obviously proves the lemma because $C_L(i) \leq C_{OPT}(i) + \tilde{C}(i)$. We prove inequality (2). We have to balance a cost of $\tilde{C}(i)$. Consider any $x \in S_L(t^e_{i-1}) \setminus \{S_B(t^b_i) \cup S_{OPT}(t^e_{i-1})\}$. We examine two cases.

Case 1: For $t = t^e_{i-1}, t^b_i, t^b_i + 1, \ldots, t^e_i$, $x \notin S_L(t) \setminus \{S_B(t) \cup M(t) \cup S_{OPT}(t)\}$
In this case x does not increase the potential during block i and we charge a cost of 1 to the move at which OPT vacated x. This move is counted in $C''_{OPT}(i-1)$.

Case 2: There exists a t, $t^e_{i-1} \leq t \leq t^e_i$, such that $S_L(t) \setminus \{S_B(t) \cup M(t) \cup S_{OPT}(t)\}$
In this case let $t_{\min}$ be the smallest t with this property. If $t_{\min} = t^e_{i-1}$, then x's weight decreases by 1 at the first move by LRU(l)-blocked during block i. Hence the potential function decreases by 1. If $t_{\min} \geq t^b_i$, then observe that $w(x, t_{\min}) \leq k - l - 1$. Now we charge an additional cost of 1 to the move at which OPT vacated x. This move is counted in $C''_{OPT}(i)$. This completes the proof of the inequality (2). □

Note that during block 1, LRU(l)-blocked does not incur a higher cost than OPT, i.e.

$$C_L(1) + \Phi(t^e_1) - \Phi(0) \leq C_{OPT}(1) + (k - l - 1)C'_{OPT}(1),$$

where $\Phi(0) = 0$ is the initial potential. Summing up this inequality and the inequalities of Lemma 5 for $i = 1, 2, \ldots, b$, we obtain

$$C_L(\sigma) + \Phi(m) - \Phi(0) \leq (k - l + 1)C_{OPT}(\sigma).$$

Here $C_L(\sigma)$ denotes the cost of LRU(l)-blocked on request sequence σ. This concludes the proof that LRU(l)-blocked is $(k - l + 1)$-competitive. □

The following theorem shows that no deterministic on-line paging algorithm with strong lookahead $l \leq k - 2$ can achieve a better competitive factor than LRU(l) and that LRU(l)-blocked is nearly optimal.

Theorem 3 *Let A be a deterministic on-line paging algorithm with strong lookahead l, where $l \leq k - 2$. If A is c-competitive, then $c \geq (k - l)$.*

Proof: Consider a set $S = \{x_1, x_2, \ldots, x_{k+1}\}$ of $k + 1$ vertices. We assume without loss of generality that both $A's$ and OPT's servers initially cover vertices $x_1, x_2, \ldots, x_k$. Let $SL = \{x_1, x_2, \ldots, x_l\}$.

We construct a request sequence σ consisting of a series of phases. Each phase contains $l+1$ requests to $l + 1$ distinct vertices. The first phase $P(1)$ consists of requests to the vertices in SL, followed by a request to the uncovered vertex x_{k+1}, i.e. $P(1) = x_1, x_2, \ldots, x_l, x_{k+1}$. Each of the following phases $P(i)$, $i \geq 2$, has the form $P(i) = x_1, x_2, \ldots, x_l, y_i$, where $y_i \in S \setminus SL$ is chosen as follows. Let $z_i \in S$ be the vertex uncovered by A after the last request of phase $i - 1$. If $z_i \in S \setminus SL$, then set $y_i = z_i$. Otherwise, if $z_i \in SL$, y_i is an arbitrary vertex in $S \setminus SL$.

The algorithm A incurs a cost of 1 in each phase. We show that during $k - l$ successive phases, OPT's cost is at most 1. This proves the theorem.

OPT always covers vertices $x_1, x_2, \ldots, x_l$. Note that $k - l$ successive phases contain at most k different vertices. If the last request in a given phase is uncovered by OPT, then OPT can move a server such that all vertices in the next $k - l - 1$ phases remain covered. □

So far, we have only considered the case $l \leq k - 2$, which, of course, is the interesting one. If $l = k - 1$ and $n = k + 1$ then LRU(l) achieves a competitive factor of 1 because it behaves like Belady's optimal paging algorithm MIN [B66]. If $n > k + 1$ and $n > l \geq k - 1$, then the competitive factor of an optimal on-line algorithm depends not only on l but also on n.

3.2 Randomized algorithms

Suppose a randomized paging algorithm has a strong lookahead of size l. Again, we assume $l \leq k-2$. The first algorithm we propose is a slight modification of the MARKER algorithm due to Fiat *et al.* [FKLMSY91], which we review for the sake of completeness.

Algorithm MARKER: The algorithm proceeds in a series of phases. During each phase a set of marked vertices is maintained. At the beginning of each phase all vertices are unmarked. Whenever a vertex is requested, that vertex is marked. On a fault a server is chosen uniformly at random from among the unmarked vertices, and that server is moved to the request. A phase ends immediately before a request to an uncovered vertex, when all k servers cover marked vertices. At that moment all marks are erased and a new phase is started.

The modified algorithm with strong lookahead l uses lookahead once during each phase.

Algorithm MARKER(l): At the beginning of each phase execute an initial step: Determine the set U of uncovered vertices in the present lookahead. Then choose $card(U)$ servers uniformly at random from among the servers which cover vertices not contained in the current lookahead. Move these servers to the vertices in U. After this initial step proceed with the MARKER algorithm.

Theorem 4 *Let $l \leq k-2$. The algorithm MARKER(l) with strong lookahead l is $2H(k-l)$-competitive.*

Proof: The idea of the proof is the same as the idea of the original proof of the MARKER algorithm [FKLMSY91]. During each phase we compare the cost incurred by MARKER(l) to the cost incurred by the optimal algorithm OPT.

Consider an arbitrary phase. We use the same terminology as Fiat *et al.* A vertex is called *stale* if it is unmarked but was marked in the previous phase, and *clean* if it is neither stale nor marked.

Let c be the number of clean vertices and s be the number of stale vertices requested in the phase. Fiat *et al.* prove that OPT has an amortized cost of at least $c/2$ during the phase.

We evaluate MARKER(l)'s cost during the phase. Serving c requests to clean vertices obviously costs c. It remains to bound the expected cost for serving the stale vertices. Let s_1 be the number of stale vertices contained in the lookahead at the beginning of the phase and let $s_2 = s - s_1$. Note that serving the first s_1 stale requests does not incur any cost and that we just have to evaluate MARKER(l)'s cost on the following s_2 requests to stale vertices. The expected cost of a request to such a stale vertex is $\tilde{c}/\tilde{s}$, where $\tilde{s}$ is the current number of stale vertices and $\tilde{c}$ is the number of clean vertices requested in the phase so far.

It follows that the expected cost of the requests to stale vertices is highest if all clean vertices are requested before any stale vertices and that this cost is bounded by

$$\begin{aligned} & \frac{c}{k-s_1} + \frac{c}{k-s_1-1} + \frac{c}{k-s_1-2} + \ldots + \frac{c}{k-s_1-(s-s_1)+1} \\ = \; & \frac{c}{k-s_1} + \frac{c}{k-s_1-1} + \frac{c}{k-s_1-2} + \ldots + \frac{c}{k-s+1}. \end{aligned}$$

This sum is bounded by $c(H(k-l)-1)$ since $s_2 \leq k-l-1$, and we conclude that MARKER(l)'s cost during the phase is bounded from above by $cH(k-l)$. This proves the theorem because OPT's amortized cost during the phase is at least $c/2$. □

We give a lower bound for randomized on-line paging algorithms with strong lookahead. This lower bound implies that MARKER(l)'s competitive factor is optimal, up to a constant factor. Raghavan [R89] has an elegant proof for the theorem that no randomized on-line paging algorithm (without lookahead) can be better than $H(k)$-competitive. He makes use of Yao's minimax principle [Y77] and provides a probability distribution on request sequences σ on which no deterministic on-line paging algorithm has an expected cost of less than $H(k)$ times the optimal cost; see [R89] for details.

Theorem 5 *Let $l \leq k-2$ and let A be a randomized on-line paging algorithm with strong lookahead l. If A is c-competitive, then $c \geq H(k-l)$.*

Proof: The proof is similar to Raghavan's proof [R89] and we just sketch the difference. Let $S = \{x_1, x_2, \ldots, x_{k+1}\}$ be a set of $k+1$ vertices and let $SL = \{x_1, x_2, \ldots, x_l\}$. We construct a request sequence which consists of a series of phases.

The first phase is of the form $P(1) = x_1, x_2, \ldots, x_l, y_1$, where y_1 is chosen uniformly at random from all vertices in $S \setminus SL$. The following phases $P(i)$ equal $P(i) = x_1, x_2, \ldots, x_l, y_i$, where y_i is chosen uniformly at random from $S \setminus \{SL \cup \{y_{i-1}\}\}$. It is possible to partition σ into rounds such that during each round OPT incurs a cost of 1 and any deterministic on-line algorithm with strong lookahead l incurs an expected cost of at least $H(k-l)$. □

We conclude this section by presenting another randomized algorithm, called RANDOM(l)-blocked. As the name suggests this algorithm is a variant of the algorithm RANDOM due to Raghavan and Snir [RS89]. On a fault RANDOM chooses a server uniformly at random from among the covered vertices and moves it to the request. In terms of competitiveness RANDOM(l)-blocked represents no improvement upon the previously presented algorithms with strong lookahead. However, RANDOM(l)-blocked, as the original algorithm RANDOM, is very simple and uses no information on previous requests.

Algorithm RANDOM(l)-blocked: Service the request sequence σ in a series of blocks $B(1), B(2), \ldots$, where $B(1) = \sigma(1), \sigma(2), \ldots, \sigma(\lambda(1))$ and $B(i) = \sigma(t^{\sigma}_{i-1}+1), \sigma(t^{\sigma}_{i-1}+2), \ldots, \sigma(\lambda(t^{\sigma}_{i-1}+1))$ for $i \geq 2$. Again, t^{σ}_{i-1} denotes the end of block $B(i-1)$. At the beginning of block $B(i)$ determine the set U_i of uncovered vertices in $B(i)$. Choose $card(U_i)$ servers uniformly at random from among the servers covering vertices not contained in $B(i)$, and move them to the vertices in U_i. Then serve the requests in $B(i)$.

Theorem 6 *Let $l \leq k-2$. The algorithm RANDOM(l)-blocked with strong lookahead l is $(k-l+1)$-competitive.*

Proof: The potential function we use to analyze the algorithm is

$$\Phi(t) = (k-l) \cdot card(S_R(t) \setminus S_{OPT}(t)).$$

$S_R(t)$ denotes the set of vertices covered by RANDOM(l)-blocked after request t and $S_{OPT}(t)$ denotes the set of vertices covered by OPT after request t. We assume that RANDOM(l)-blocked and OPT start with the same initial cache.

Suppose the request sequence σ consists of b blocks $B(1), B(2), \ldots, B(b)$. We assume without loss of generality that the last block $B(b)$ contains $l+1$ distinct requests. The values t_i^b and t_i^e denote the beginning and the end of block $B(i)$, respectively. Define $t_0^e = 0$. Let $E(\Phi(t_i^e) - \Phi(t_{i-1}^e))$ be the expected change in potential between t_{i-1}^e and t_i^e. Furthermore, let $C_R(i)$ and $C_{OPT}(i)$ denote the cost incurred by RANDOM(l)-blocked and OPT during block $B(i)$. We show that for all $i = 1, 2, \ldots, b$,

$$C_R(i) + E(\Phi(t_i^e) - \Phi(t_{i-1}^e)) \leq (k - l + 1) C_{OPT}(i).$$

This inequality obviously proves the theorem.

If $C_R(i) = 0$, then the inequality clearly holds. Each time OPT moves a server during block i, it might vacate a vertex which is in $S_R(t_{i-1}^e) = S_R(t_i^e)$. Hence each move can increase the potential by $(k - l)$.

Now suppose $C_R(i) \geq 1$ and let

$$\tilde{C}(i) = card(S_R(t_{i-1}^e) \setminus \{L(t_i^b) \cup S_{OPT}(t_{i-1}^e)\}).$$

We analyze the effect of the moves by RANDOM(l)-blocked and OPT on the potential function Φ, and assume that our on-line algorithm moves first and OPT moves second.

RANDOM(l)-blocked vacates $C_R(i)$ vertices at the beginning of block $B(i)$. The expected decrease in potential is

$$(k - l) C_R(i) \frac{\tilde{C}(i)}{k - l - 1 + C_R(i)} \geq (k - l) \tilde{C}(i) \frac{1}{k - l} = \tilde{C}(i).$$

Note that a newly covered vertex might not be in $S_{OPT}(t_{i-1}^e)$, which can increase the potential by $(k - l)$ per vertex. Hence, the moves by RANDOM(l)-blocked cause an expected increase of potential of at most

$$-\tilde{C}(i) + (k - l) C_R(i).$$

We consider OPT's moves. Each time OPT makes a move and vacates a vertex, this can increase the potential by $(k - l)$. Note that a vertex x which is requested in $B(i)$ is not in $S_{OPT}(t_i^e)$ if and only if it was vacated by OPT during $B(i)$. Hence

$$C_R(i) + E(\Phi(t_i^e) - \Phi(t_{i-1}^e)) \leq C_R(i) - \tilde{C}(i) + (k - l) C_{OPT}(i).$$

Since $C_R(i) \leq C_{OPT}(i) + \tilde{C}(i)$, we obtain

$$C_R(i) + E(\Phi(t_i^e) - \Phi(t_{i-1}^e)) \leq (k - l + 1) C_{OPT}(i).$$

□

4 Competitive paging with weak lookahead

It is easy to prove that weak lookahead cannot improve the competitive factors of deterministic and randomized on-line paging algorithms, see also [I91]. If an on-line paging algorithm is given a weak lookahead of size l, an adversary can simply replicate each request l times in order to make the lookahead useless. However, we will show in the following sections that a deterministic (randomized) on-line paging algorithm with weak lookahead can satisfy $C_A(\sigma) < k \cdot C_{OPT}(\sigma)$ ($C_A(\sigma) < H(k) \cdot C_{OPT}(\sigma)$) for request sequences σ of a restricted length. We examine which part of a request sequence σ has to be seen by an on-line algorithm A such that $C_A(\sigma) < k \cdot C_{OPT}(\sigma)$ or $C_A(\sigma) < H(k) \cdot C_{OPT}(\sigma)$.

4.1 Deterministic algorithms

Throughout this section we assume that both the k servers of the given on-line paging algorithm and the k servers of the optimal off-line algorithm initially cover the same k vertices $x_1, x_2, \ldots, x_k$. The reason is the following. As mentioned before, we will show that a deterministic on-line paging algorithm A with weak lookahead can satisfy $C_A(\sigma) < k \cdot C_{OPT}(\sigma)$ for request sequences σ of a restricted length. Hence, when evaluating the performance of an on-line paging algorithm, one should not take into account the cost any paging algorithm incurs when filling up the cache on the first k faults.

Theorem 7 *Let A be a deterministic on-line paging algorithm with weak lookahead l. For all integers $m \geq 1$, there exists a request sequence σ of length m such that*

$$C_A(\sigma) \geq \min\{k, \lceil \frac{m}{l+1} \rceil\} \cdot C_{OPT}(\sigma).$$

Proof: Fix an integer $m \geq 1$. Let $S = \{x_1, x_2, \ldots, x_{k+1}\}$ be a set of $k+1$ vertices. We assume that A's and OPT's servers are initially on vertices $x_1, x_2, \ldots, x_k$. The request sequence we construct consists of $c = \min\{k, \lceil \frac{m}{l+1} \rceil\}$ phases $P(1), P(2), \ldots$, where

$$P(i) = \sigma((i-1)(l+1)+1), \sigma((i-1)(l+1)+2), \ldots, \sigma(i(l+1))$$

for $i = 1, 2, \ldots, c-1$, and

$$P(c) = \sigma((c-1)(l+1)+1), \sigma((c-1)(l+1)+2), \ldots, \sigma(m).$$

Within each phase, all requests are equal. In the first phase, we request the initially uncovered vertex x_{k+1}. Phase $P(i)$, for $i = 2, 3, \ldots, c$, then requests the vertex in S which is uncovered by A after the first request of phase $P(i-1)$.

When serving $\sigma(1)$, the algorithm OPT can vacate a vertex that will not be requested during the remaining part of the request sequence. Hence OPT's cost equals 1. The on-line algorithm incurs a cost of 1 per phase. This proves the theorem. □.

The above theorem implies that for every deterministic on-line paging algorithm A with weak lookahead l, there exists a request sequence of length $m = (k-1)(l+1)+1$ such that

$C_A(\sigma) = k \cdot C_{OPT}(\sigma)$. However, A can fulfill $C_A(\sigma) < k \cdot C_{OPT}(\sigma)$ for all request sequences of length $m \leq (k-1)(l+1)$. Thus, given a request sequence σ, a substantial lookahead, whose size depends on the length of the entire request sequence, can be necessary in order to satisfy $C_A(\sigma) < k \cdot C_{OPT}(\sigma)$. The above theorem also evaluates for which request sequences σ an on-line algorithm A with weak lookahead can satisfy $C_A(\sigma) \leq c \cdot C_{OPT}(\sigma)$, given a $c \in [1, k]$.

We present an on-line algorithm with weak lookahead l which is optimal for request sequences of length $m \leq k(l+1)$.

Algorithm ARBITRARY/MIN(l): Serve the request sequence in a series of blocks $B(1)$, $B(2), \ldots$, where $B(i) = \sigma((i-1)(l+1)+1), \sigma((i-1)(l+1)+2), \ldots, \sigma(i(l+1))$. On a fault determine the set S of covered vertices which will not be requested during the remaining part of the current block.

(1) If S is not empty, then vacate an arbitrary vertex in S and move the corresponding server to the requested vertex.

(2) Otherwise, if S is empty, vacate a vertex according to the MIN rule, i.e. choose the vertex whose next request occurs farthest in the future, and move the corresponding server to the request.

Theorem 8 *Let σ be a request sequence of length m and let $C_{AM}(\sigma)$ denote the cost incurred by ARBITRARY/MIN(l) in processing σ. Then*

$$C_{AM}(\sigma) \leq c \cdot C_{OPT}(\sigma),$$

where $c = \lceil \frac{m}{l+1} \rceil$.

Proof: First we introduce some notations. Let $S_{AM}(t)$ denote the set of vertices covered by ARBITRARY/MIN(l) (also called AM for simplicity) after request $\sigma(t)$; $S_{OPT}(t)$ denotes the set of vertices covered by OPT after request $\sigma(t)$. Furthermore, let $S(0)$ be the set of vertices covered initially. A fault that occurs when serving a request $\sigma(t)$ is called an *initial fault*, if the requested vertex is in $\{\sigma(s) | s = 1, 2, \ldots, m\} \setminus S(0)$ and $\sigma(t)$ is the first request to that vertex. Every non-initial fault at a request $\sigma(t)$ can be viewed as being *generated* by another request $\sigma(t')$, where $t' < t$: The paging algorithm vacates the vertex $x = \sigma(t)$ when serving $\sigma(t')$, and x is not requested during $\sigma(t'+1), \sigma(t'+2), \ldots, \sigma(t-1)$. A fault is called a *near* fault, if it was generated in the same block; otherwise it is called a *far* fault.

Let f be the number of initial faults made by AM and OPT on the request sequence, i.e.

$$f = card(\{\sigma(s) | s = 1, 2, \ldots, m\} \setminus S(0)).$$

We observe that during the entire request sequence, OPT incurs at least f initial faults and that in each block, AM has at most f far faults. Hence the number of far faults made by AM is $\leq c \cdot f$, since AM serves $c = \lceil \frac{m}{l+1} \rceil$ blocks. The following lemma obviously implies the theorem.

Lemma 6 *In each block $B(i)$, $i = 1, 2, \ldots, c$, the number of non-initial faults made by OPT is at least as large as the number of near faults incurred by AM.*

Proof: Let N be the number of near faults incurred by AM during a given block $B(i)$, and let $t_1 < t_2 < \ldots < t_N$ be the times at which these near faults occur. We show that there exist $s_1 < s_2 < \ldots < s_N$ such that

(1) OPT incurs a non-initial fault when processing $\sigma(s_j)$, $j = 1, 2, \ldots, N$.

(2) $t_i^b \leq s_j \leq t_i^e$ for $j = 1, 2, \ldots, N$. Here t_i^b and t_i^e denote the beginning and the end of block $B(i)$.

More specifically, we will construct sets $T_{AM} = \{t_1, t_2, \ldots, t_N\}$ and $T_{OPT} = \{s_1, s_2, \ldots, s_N\}$ such that the above conditions (1) and (2) hold. These sets are built up as follows. Starting with empty sets $T_{AM} = T_{OPT} = \emptyset$, we trace AM's moves during block $B(i)$. Each time AM generates a new near fault, which will occur at some time t', we show the existance of a time $s' \notin T_{OPT}$, $t_i^b \leq s' \leq t_i^e$, such that OPT makes a non-inital fault when serving $\sigma(s')$. We then add t' to T_{AM} and s' to T_{OPT}. At the end of the block we obtain the desired sets.

Let $T_{AM}(t-1)$ and $T_{OPT}(t-1)$ be the sets obtained after request $t-1$. We show how to construct $T_{AM}(t)$ and $T_{OPT}(t)$. Note that $T_{AM} = T_{AM}(t_i^e)$ and $T_{OPT} = T_{OPT}(t_i^e)$. If AM does not generate a near fault at time t, then $T_{AM}(t) = T_{AM}(t-1)$ and $T_{OPT}(t) = T_{OPT}(t-1)$. So suppose that at time t, the algorithm AM generates a near fault that will occur at some time t'. Notice that all vertices in $S_{AM}(t-1)$ will be requested in the remaining part of the current block. Define

$$\overline{T}_{AM}(t-1) = \{s > t | s \in T_{AM}(t-1) \setminus T_{OPT}(t-1)\}$$

$$\overline{T}_{OPT}(t-1) = \{s > t | s \in T_{OPT}(t-1) \setminus T_{AM}(t-1)\}$$

We distinguish between two cases.

Case 1: $\exists s \in \overline{T}_{AM}(t-1) : \sigma(s) \notin S_{OPT}(t)$
Then OPT incurs a non-initial fault at time s. Let $T_{AM}(t) = T_{AM}(t-1) \cup \{t'\}$ and $T_{OPT}(t) = T_{OPT}(t-1) \cup \{s\}$.

Case 2: $\forall s \in \overline{T}_{AM}(t-1) : \sigma(s) \in S_{OPT}(t)$
Note that the requests $\sigma(s)$, $s \in \overline{T}_{AM}(t-1)$, are pairwise distinct and that $\sigma(s) \notin S_{AM}(t)$ for all $s \in \overline{T}_{AM}(t-1)$.

Case 2.1: $t' \in T_{OPT}(t-1)$
Since $\sigma(s) \in S_{OPT}(t)$ for all $s \in \overline{T}_{AM}(t-1)$, there must exist $card(\overline{T}_{AM}(t-1))$ vertices in $S_{AM}(t)$ which are not contained in $S_{OPT}(t)$. The algorithm AM vacates the vertex $x = \sigma(t')$ according to the MIN rule. Thus there must be $card(\overline{T}_{AM}(t-1))$ times t'', where $t < t'' < t'$, such that $\sigma(t'') \notin S_{OPT}(t)$ and $\sigma(t'') \in S_{AM}(t)$. Since $card(\overline{T}_{OPT}(t-1)) = card(\overline{T}_{AM}(t-1))$ and $t' \in \overline{T}_{OPT}(t-1)$, there must be a time $s' \notin \overline{T}_{OPT}(t-1)$, with $t < s' < t'$, such that $\sigma(s') \notin S_{OPT}(t)$, $\sigma(s') \in S_{AM}(t)$ and $\sigma(s) \neq \sigma(s')$ for all $t < s < s'$.

Case 2.2: $t' \notin T_{OPT}(t-1)$
If $\sigma(t') \notin S_{OPT}(t)$ then let $s' = t'$. Otherwise, if $\sigma(t') \in S_{OPT}(t)$, there must exist $card(\overline{T}_{AM}(t-1)) + 1$ times t'', where $t < t'' < t'$, such that $\sigma(t'') \notin S_{OPT}(t)$ and $\sigma(t'') \in S_{AM}(t)$. Again, since $card(\overline{T}_{OPT}(t-1)) = card(\overline{T}_{AM}(t-1))$, there must exist a time $s' \notin \overline{T}_{OPT}(t-1)$, with $t < s' < t'$, such that $\sigma(s') \notin S_{OPT}(t)$, $\sigma(s') \in S_{AM}(t)$ and $\sigma(s) \neq \sigma(s')$ for all $t < s < s'$.

In both cases 2.1 and 2.2 we set $T_{AM}(t) = T_{AM}(t-1) \cup \{t'\}$ and $T_{OPT}(t) = T_{OPT}(t-1) \cup \{s'\}$. Note that $\sigma(s')$ represents a non-initial fault for OPT because $\sigma(s') \in S_{AM}(t-1)$. □

The proof of Theorem 8 is complete. □

4.2 Randomized algorithms

We examine for which request sequences σ a randomized on-line paging algorithm A can satisfy $C_A(\sigma) < H(k) \cdot C_{OPT}(\sigma)$. Again, we assume throughout this section that the k servers of the on-line algorithm and the k servers of the optimal off-line algorithm initially cover the same k vertices $x_1, x_2, \ldots, x_k$.

We introduce some definitions. For $h = 1, 2, \ldots, k-1$, let $(n_1, n_2, \ldots, n_h)$ be the h-tupel that maximizes the function

$$f_h(x_1, x_2, \ldots, x_h) = \sum_{i=1}^{h} \frac{x_i}{k - \sum_{j=1}^{i-1} x_j}$$

subject to the constraints

$$\sum_{i=1}^{h} x_i - k + 1 = 0$$

$$x_i \text{ is a non-negative integer for } i = 1, 2, \ldots, h.$$

If there exist several such h-tupel, let $(n_1, n_2, \ldots, n_h)$ be the one that is largest according to the lexicographic order. It is easy to show that $n_i > 0$ for all $i = 1, 2, \ldots, h$. Define $f_{\max}(h) = f_h(n_1, n_2, \ldots, n_h)$. The following theorem bounds the values $f_{\max}(h)$. Note that $f_{\max}(k-1)+1 = H(k)$.

Theorem 9 *a) For* $h = 2, 3, \ldots, k-1$,

$$f_{\max}(h) - f_{\max}(h-1) \leq 1.$$

b) For $h = 1, 2, \ldots, k-1$,

$$h(1 - \frac{1}{k^{1/h}}) - (1 - \frac{1}{k}) < f_{\max}(h) \leq h(1 - \frac{1}{k^{1/h}}).$$

Proof: See appendix. □

Given a value $1 \leq h \leq k-1$ and a weak lookahead $l \geq 1$, define values $m(h, l)$ as follows. If $l < k-2$, then

$$m(h, l) = h \cdot l + k + \sum_{i=1}^{h} \max\{0, \lceil (\sum_{j=1}^{i-1} n_j - l) \frac{n_i}{k - \sum_{j=1}^{i-1} n_j} \rceil\}.$$

If $l \geq k-2$, then

$$m(h, l) = h \cdot l + k.$$

Theorem 10 *Let A be a randomized on-line paging algorithm with weak lookahead l. Then, for $h = 1, 2, \ldots, k-1$, there exists a request sequence σ^h of length $\leq m(h,l)$ such that*

$$C_A(\sigma^h) \geq (f_{\max}(h) + 1) \cdot C_{OPT}(\sigma^h).$$

Proof: Fix a value $1 \leq h \leq k-1$. We will give a probability distribution on request sequences σ^h which has the following properties. On this probability distribution the expected cost of any deterministic on-line paging algorithm with weak lookahead l is at least $(f_{\max}(h)+1)$ while the optimal off-line cost is exactly 1. Moreover, any sequence σ^h that is chosen according to this distribution has a length of $m(h,l)$. Applying Yao's minimax principle [Y77] we obtain the theorem.

We show how to choose σ^h. Let $S = \{x_1, x_2, \ldots, x_{k+1}\}$ be a set of $k+1$ vertices, and let A be a deterministic on-line paging algorithm with weak lookahead l. We assume that both A's and OPT's servers initially cover vertices $x_1, x_2, \ldots, x_k$. The request sequence σ^h consists of $h+1$ phases, numbered from 0 to h. Phase 0 consists of a single request to the uncovered vertex x_{k+1}. We assume that we have already chosen phases 0 to $i-1$ and show how to choose phase i. Let M_{i-1} be the set of vertices requested during phases 0 to $i-1$, and let y_{i-1} be the vertex that is requested last in phase $i-1$. The first part of phase i consists of requests to vertices in M_{i-1}. More precisely, if $card(M_{i-1}) = 1$, then we pose l requests to the vertex in M_{i-1}. Otherwise, if $card(M_{i-1}) > 1$, we use a slightly different strategy. We choose an arbitrary subset $L_i \subseteq M_{i-1} \setminus \{y_{i-1}\}$ of cardinality $\min\{l, card(M_{i-1} \setminus \{y_{i-1}\})\}$ and request each vertex in L_i exactly once. If $card(M_{i-1} \setminus \{y_{i-1}\}) < l$, we replicate some of the requests until we obtain a total of l requests. If $card(M_{i-1} \setminus \{y_{i-1}\}) > l$, we choose $\lceil card(M_{i-1} \setminus \{L_i \cup \{y_{i-1}\}\}) \cdot \frac{n_i}{k - \sum_{j=1}^{i-1} n_j} \rceil$ vertices uniformly at random from $M_{i-1} \setminus \{L_i \cup \{y_{i-1}\}\}$ and add one request to each of these vertices. The second part of phase i consists of requests to vertices in $S \setminus M_{i-1}$. We choose n_i vertices uniformly at random from $S \setminus M_{i-1}$ and request each of these vertices exactly once. This concludes phase i.

We analyze A's and OPT's cost on request sequence σ^h. During phase 0 both A and OPT incur a cost of 1. Notice that during the remaining phases OPT does not incur any other fault because σ contains exactly k distinct vertices. We estimate A's cost on an arbitrary phase i, where $1 \leq i \leq h$. Let z_i be the vertex in S that is not covered by A at the beginning of the ith phase. Note that $z_i \neq y_{i-1}$. Since phase i contains vertex z_i with probability of at least $n_i/(k - \sum_{j=1}^{i-1} n_j)$, A's expected cost during phase i is at least $n_i/(k - \sum_{j=1}^{i-1} n_j)$. Thus A's expected cost on σ^h is at least $(f_{\max}(h)+1)$. It is easy to verify that the length of σ^h is $m(h,l)$. □

Remarks: We mention some implication of the above theorem. Let A be a randomized on-line paging algorithm with weak lookahead l. If $l \leq k-2$, then

$$m(h,l) \leq h \cdot l + k + (k-l-1) \cdot \sum_{i=1}^{h} \frac{n_i}{k - \sum_{j=1}^{i-1} n_j} + h$$

for $h = 1, 2, \ldots, k-1$. Thus, if $l < k-2$, Theorem 10 implies that there exists a request sequence of length $m \leq (k-1)l + 2k - 1 + (k-l-1)H(k)$ such that $C_A(\sigma) \geq H(k) \cdot C_{OPT}(\sigma)$.

If $l \geq k-2$, then there exists a request sequence σ of length $m \leq (k-1)l + k$ such that $C_A(\sigma) \geq H(k) \cdot C_{OPT}(\sigma)$.

We present two on-line algorithms with weak lookahead l, called MARKER2(l)/STATIC and MARKER2(l)/MIN, for processing request sequences of restriced length $m \leq m(k-1, l)$. We will use MARKER2(l)/STATIC if $l < k-2$, and MARKER2(l)/MIN if $l \geq k-2$. Both algorithms make use of the following variant of the MARKER algorithm.

Algorithm MARKER2(l): Serve the request sequence in a series of phases. These phases have almost the same structure as those in the original MARKER algorithm. During each phase we maintain a set of marked vertices. Whenever a vertex is requested, that vertex is marked. A phase ends immediately before a fault, when there are k marked vertices. At that moment all marks are erased and a new phase is started. Each phase is served in a sequence of blocks. Each block consists of $l+1$ successive requests. More precisely, let $\sigma(t)$ be the first request in a given phase. Then the ith block, $i \geq 1$, in that phase consists of $\sigma(t+(i-1)(l+1)), \sigma(t+(i-1)(l+1)+1), \ldots, \sigma(t+i(l+1)-1)$. On a fault, determine the set S of unmarked covered vertices which are not contained in the current block. If S is non-empty, then choose a vertex uniformly at random in S and move the corresponding server to the request. If S is empty, then vacate an unmarked covered vertex according to Belady's MIN algorithm. If a phase ends before the current block is completely served, then that block is stopped and declared finished.

In the following we assume that the length m of the given request sequence is known to our on-line algorithms.

Case 1: $l < k-2$

Algorithm MARKER2(l)/STATIC: Determine the maximum h such that $m(h, l) \leq m$. On the first $h(l+1)$ requests execute the MARKER2(l) algorithm. After the $h(l+1)$st request, stop the MARKER2(l) algorithm, even if the current phase is not finished. Serve the remaining requests using the algorithm STATIC which works as follows. Let S be the set of vertices which were covered initially or which were requested during the first $h(l+1)$ requests. First, the algorithm STATIC chooses k vertices uniformly at random from S and covers them by servers. While STATIC serves the remaining requests, the set S is occasionally updated such that it always includes the vertices which have been requested so far. Suppose there occurs a fault when STATIC serves a request to a vertex which is in the current set S. In this case the algorithm moves an arbitrary server to the request, serves the request, and then moves the server back to the original vertex. If there occurs a fault while serving a request to a vertex not in S, the algorithm first adds this new vertex to S. Then it chooses a server uniformly at random and moves this server to the request. With probability $k/card(S)$, the server remains on this vertex, and with probability $1 - k/card(S)$ it moves back.

Case 2: $l \geq k-2$

Algorithm MARKER2(l)/MIN: Compute the minimum h such that $m(h, l) \geq m$. Serve the first $h(l+1)$ requests using the algorithm MARKER2(l). Then switch to Belady's MIN algorithm (note that $m - h(l+1) \leq l+1$).

Theorem 11 *Let $l \geq 1$ and let σ be a request sequence of length $m \leq m(k-1, l)$.*

a) If $l < k-2$, then let h be the largest integer such that $m(h,l) \leq m$. Let $C_{MS}(\sigma)$ be the cost incurred by MARKER2(l)/STATIC in processing σ. Then

$$C_{MS}(\sigma) \leq c \cdot C_{OPT}(\sigma),$$

where $c = 4(f_{\max}(h)+1)+5+\frac{2}{k}$.

b) If $l \geq k-2$, then let h be the smallest integer such that $m(h,l) \geq m$. Let $C_{MM}(\sigma)$ be the cost incurred by MARKER2(l)/MIN in processing σ. Then

$$C_{MM}(\sigma) \leq c \cdot C_{OPT}(\sigma),$$

where $c = 2(f_{\max}(h)+1)+2$.

Proof: Let σ be a request sequence of length m. If $l < k-2$, then let h be the largest integer such that $m(h,l) \leq m$. If $l \geq k-2$, then let h be the smallest integer such that $m(h,l) \geq m$. First we analyze the cost incurred on the first $h(l+1)$ requests. Suppose MARKER2(l) executes p phases (the last phase might be incomplete). We use the same terminology as Fiat *et al.* [FKLMSY91]. Consider an arbitrary phase. We call a vertex *stale* if it is unmarked but was marked in the previous phase, and *clean* if it is neither stale nor marked. Let c_i be the number of clean vertices in phase i, $1 \leq i \leq p$. Let d_i^b be the number of vertices that are covered by MARKER2(l) servers but not by OPT's servers at the beginning of the ith phase, and let d_i^e be this value at the end of the ith phase. Using the same analysis as Fiat *et al.* we are able to lower bound OPT's cost during each phase. During phase i, where $1 \leq i \leq p-1$, OPT incurs a cost of at least

$$\frac{1}{2}(c_i - d_i^b + d_i^e).$$

Note that $d_1^b = 0$. During phase p, OPT incurs a cost of at least

$$\frac{1}{2}(c_p - d_p^b).$$

Summing up these lower bounds, we obtain that OPT incurs a cost of at least

$$\frac{1}{2}\sum_{i=1}^{p} c_i$$

during the first $h(l+1)$ requests.

We study the cost of MARKER2(l). Consider an arbitrary phase i and suppose it consists of b_i blocks. Note that $1 \leq b_i \leq h$. At each request to a clean vertex, MARKER2(l) incurs a cost of 1. We estimate the cost incurred for serving stale vertices. For $1 \leq j \leq b_i$, let s_j^i be the number of stale vertices requested during the jth block of phase i. After block $j-1$ is served, the previously clean vertices are uniformly distributed among

$$k - \sum_{\nu=1}^{j-1} s_\nu^i$$

servers which covered stale vertices at the beginning of the phase. Thus MARKER2(l)'s expected cost of serving s_j^i stale vertices in block j is at most

$$s_j^i \cdot \frac{c_i}{k - \sum_{\nu=1}^{j-1} s_\nu^i}.$$

By calculating the sum over all blocks, we obtain that MARKER2(l)'s expected cost during one phase is bounded from above by

$$c_i \cdot (\sum_{j=1}^{b_i} \frac{s_j^i}{k - \sum_{\nu=1}^{j-1} s_\nu^i} + 1) \leq c_i \cdot (f_{\max}(h) + 1).$$

Hence MARKER2(l)'s expected cost during the first $h(l+1)$ requests is at most

$$\sum_{i=1}^{p} c_i (f_{\max}(h) + 1).$$

Now we prove part a) of the theorem. Our task is to evaluate the cost incurred by the algorithm STATIC on the last $m - h(l+1)$ requests. We first bound the value $m - h(l+1)$. We show

$$m - h(l+1) \leq 2 \cdot k + 1 + (k-l) \cdot f_{\max}(h). \tag{3}$$

We have

$$\begin{aligned} m(h,l) &\leq h \cdot l + k + \sum_{i=1}^{h} \lceil (k-l) \frac{n_i}{k - \sum_{j=1}^{i-1} n_j} \rceil \\ &\leq h \cdot l + k + (k-l) f_{\max}(h) + h \\ &= h \cdot (l+1) + k + (k-l) f_{\max}(h) \end{aligned}$$

If $h = k-1$, then $m = m(h,l)$ and hence

$$m - h(l+1) \leq k + (k-l) f_{\max}(h).$$

If $h < k-1$, then

$$m(h+1,l) \leq (h+1)(l+1) + k + (k-l) f_{\max}(h+1)$$

and hence

$$\begin{aligned} m - h(l+1) &\leq m(h+1) - h(l+1) \\ &\leq l + 1 + k + (k-l) f_{\max}(h+1) \\ &\leq l + 1 + k + (k-l) f_{\max}(h) + k - l \\ &= 2 \cdot k + 1 + (k-l) f_{\max}(h). \end{aligned}$$

The last inequality follows because $f_{\max}(h+1) - f_{\max}(h) \leq 1$ by Theorem 9. This completes the proof of the inequality (3).

For $t \geq 1$ define $S(t)$ as the set of vertices which were covered at the beginning of the request sequence or which were requested during the first t requests. Let $r = card(S(m)) - k$.

Before processing $\sigma(h(l+1)+1)$, the algorithm STATIC chooses k vertices uniformly at random from the set $S(h(l+1))$ and covers these vertices by servers. This step incurs a cost of at most $card(S(h(l+1)))-k$, which is at most r. It is easy to verify that when STATIC serves $\sigma(t)$, where $t > h(l+1)$, each vertex in $S(t-1)$ is covered by a server with probability of $k/card(S(t-1))$. Thus, if $\sigma(t) \in S(t-1)$, then STATIC's expected cost on $\sigma(t)$ is

$$2 \cdot \frac{card(S(t-1)) - k}{card(S(t-1))} \leq \frac{2 \cdot r}{k}.$$

If $\sigma(t) \notin S(t-1)$, then this request incurs a cost of at most 2. Hence STATIC's expected cost on the last $m - h(l+1)$ requests of σ is bounded by

$$\begin{aligned} & r + 2 \cdot r + \frac{2r}{k}(2 \cdot k + 1 + (k-l)f_{\max}(h)) \\ \leq \; & r(2f_{\max}(h) + 7 + \frac{2}{k}). \end{aligned}$$

We obtain

$$C_{MS}(\sigma) \leq \sum_{i=1}^{p} c_i(f_{\max}(h)+1) + r(2f_{\max}(h) + 7 + \frac{2}{k}).$$

Since

$$C_{OPT}(\sigma) \geq \max\{\frac{1}{2}\sum_{i=1}^{p} c_i, r\}$$

we conclude

$$C_{MS}(\sigma) \leq c \cdot C_{OPT}(\sigma),$$

where $c \leq 4(f_{\max}(h)+1) + 5 + \frac{2}{k}$.

It remains to prove part b) of the theorem. For $t \geq 1$ define sets $S(t)$ as above and let $r = card(S(m)) - k$. After MARKER2(l) has served the first $h(l+1)$ requests, there can be at most r vertices which are covered by OPT but not by MARKER2(l)/MIN. Let $\tilde{C}$ be the cost incurred by OPT during the last $m - h(l+1)$ requests. Then the cost incurred by MARKER2(l)/MIN during the last $m - h(l+1)$ requests is at most $\tilde{C} + r$. This implies

$$C_{MM}(\sigma) \leq \sum_{i=1}^{p} c_i(f_{\max}(h)+1) + \tilde{C} + r$$

and

$$C_{OPT}(\sigma) \geq \max\{\frac{1}{2}\sum_{i=1}^{p} c_i, \tilde{C}, r\}.$$

Thus

$$C_{MM}(\sigma) \leq c \cdot C_{OPT}(\sigma),$$

where $c \leq 2(f_{\max}(h)+1) + 2$. □

5 The list update problem with lookahead

The list update problem without lookahead has been studied extensively [R76, BM85, ST85, IRWS90]. We just mention the important results relevant to our work. Sleator and Tarjan [ST85] have shown that the move-to-front algorithm, which simply moves an item to the front of the list each time it is requested, is 2-competitive. Karp and Raghavan [KR90] have observed that no deterministic on-line algorithm for the list update problem can be better than 2-competitive. Thus the move-to-front algorithm achieves the best competitive factor.

We consider the deterministic list update problem with strong and weak lookahead. We assume that the given list consists of n items $x_1, x_2, \ldots, x_n$ and that $n \geq 2$. Furthermore, we generally assume that the size l of the given lookahead is constant of a function of n. First we derive lower bound for list update with strong and weak lookahead. Then we are concerned with the development of on-line alorithms with lookahead. The algorithms we propose are essentially adaptations of the move-to-front algorithm (MTF) to the model of lookahead.

5.1 Lower bounds

We show that a deterministic on-line algorithm with strong lookahead $l \leq n-1$ can only be better than 2-competitive if l is linear in n.

Theorem 12 *Let A be a deterministic on-line algorithm with strong lookahead $l \leq n-1$ for the list update problem. If A is c-competitive, then*

$$c \geq 2 - \frac{l+2}{n+1}.$$

Proof: We construct a request sequence $\sigma = \sigma(1), \sigma(2), \ldots$ using the following algorithm.

Algorithm LIST-REQUEST: The first $l+1$ requests $\sigma(1), \sigma(2), \ldots, \sigma(l+1)$ are requests to the last $l+1$ items in the initial list. For $t \geq l+2$ the request $\sigma(t)$ is constructed as follows. After A has served $\sigma(t-l-1)$, determine the item x which has the highest position in the current list among items not contained in $\{\sigma(t-l), \sigma(t-l+1), \ldots, \sigma(t-1)\}$. Set $\sigma(t) = x$.

Given this request sequence σ, we compare the cost incurred by A to the cost incurred by the optimal algorithm OPT. It is not hard to see that OPT can process each request sequence such that its amortized cost on each request is at most $(n+1)/2$. OPT can simply use the optimal static algorithm which initially sorts the items in non-increasing order of request frequencies and then makes no exchanges while processing the request sequence. Hence OPT's amortized cost during $l+1$ successive requests in σ is at most $(l+1)(n+1)/2$.

We evaluate A's cost on request sequence σ. For simplicity, we handle paid exchanges made by A in the following way. Whenever A moves an item x closer to the front of the list using paid exchanges, we charge the cost of these paid exchanges to the next request to x. The following lemma proves that on any $l+1$ successive requests, A incurs a cost of at least $\sum_{i=0}^{l}(n-i)$. This implies that if A is c-competitive, then

$$c \geq \frac{\sum_{i=0}^{l}(n-i)}{(l+1)(n+1)/2} = \frac{(l+1)n - (l+1)l/2}{(l+1)(n+1)/2} = \frac{2n-l}{n+1} = 2 - \frac{l+2}{n+1}. \square$$

Lemma 7 *Let $C_A(\tilde{\sigma}(t))$ be the cost incurred by A when processing the subsequence $\tilde{\sigma}(t) = \sigma(t), \sigma(t+1), \ldots, \sigma(t+l)$. Then*

$$C_A(\tilde{\sigma}(t)) \geq \sum_{i=0}^{l}(n-i) \quad \textit{for all } t \geq 1.$$

Proof: For $t \geq 1$ let $S(t) = \{t, t+1, \ldots, t+l\}$ and let $C_A(\sigma(t))$ be the cost incurred by A when processing request $\sigma(t)$. We prove by induction on t that for all $t \geq 1$ and for all k, where $n - l \leq k \leq n$, the inequality

$$card(\{s \in S(t) | C_A(\sigma(s)) \geq k\}) \geq n - k + 1 \tag{4}$$

holds. This obviously implies the lemma.

We introduce some more notations. For an item x and $t \geq 1$, let $pos(x,t)$ denote x's position in the list immediately after A has served $\sigma(t-1)$. We assume that paid exchanges at time t are executed before $\sigma(t)$ is served. Furthermore, for $t_1 \leq t_2$ let $p(x,[t_1,t_2])$ be the number of paid exchanges which move item x closer to the front of the list during the interval $[t_1,t_2]$.

Note that $l+1$ successive requests are pairwise distinct. Thus for any $s \in S(t)$, $s \neq t+l$,

$$C_A(\sigma(s)) \geq pos(\sigma(s),t) + p(\sigma(s),[s-l,t-1])$$

and

$$C_A(\sigma(t+l)) \geq pos(\sigma(t+l),t).$$

We proceed with the inductive proof. The inequality (4) obviously holds at time $t = 1$. By induction hypothesis it holds at time $t-1$. We show that the inequality is also satisfied at time t. We distinguish between two cases.

Case 1: $pos(\sigma(t+l),t) \geq C_A(\sigma(t-1))$
In this case the inequality (4) obviously holds for all $n - l \leq k \leq n$.

Case 2: $pos(\sigma(t+l),t) < C_A(\sigma(t-1))$
After A has served $\sigma(t-1)$, the items $\sigma(s)$ with $s \in S(t) \setminus \{(t+l)\}$ occupy all positions $pos(\sigma(t+l),t)+1, pos(\sigma(t+l),t)+2, \ldots, n$. We observe that for $k > pos(\sigma(t+l),t)$

$$card(\{s \in S(t) \setminus \{(t+l)\} | C_A(\sigma(s)) \geq k\}) \geq n - k + 1.$$

By induction hypothesis, we have, for $n - l \leq k \leq pos(\sigma(t+l),t)$,

$$card(\{s \in S(t) \setminus \{(t+l)\} | C_A(\sigma(s)) \geq k\}) \geq n - k.$$

We conclude that the inequality (4) must hold for all $n - l \leq k \leq n$. □

Theorem 13 *Let A be a deterministic on-line algorithm with weak lookahead l for the list update problem.*

a) If $l = o(n^2)$ and A is c-competitive, then $c \geq 2$.

b) If $(l+1) = Kn^2$ and A is c-competitve, then

$$c \geq 2 - 2\sqrt{4K^2 + 2K} + 4K.$$

Proof: For integers j with $1 \leq j \leq \min\{l+1, n-1\}$ we construct request sequences σ^j. If $j = l+1$ then we generate a request sequence using the algorithm LIST-REQUEST proposed in the proof of Theorem 12.

If $j < l+1$ we use a slightly different algorithm. Let x_1 be the first item in the initial list. The request sequence σ^j which we are constructing consists of a series of phases each of which contains exactly $l+1$ requests. In each phase, the first j requests are made to items $x \neq x_1$, while the remaining $l+1-j$ requests are made to item x_1. More precisely, the first j requests in the first phase are requests to the last j items in the initial list. If $\sigma(t)$ belongs to the first j requests in a given phase i, where $i \geq 2$, then $\sigma(t)$ is constructed as follows. After A has served $\sigma(t-l-1)$, determine the item x which is at the highest position in the current list among items not contained in $\{\sigma(t-l), \sigma(t-l+1), \ldots, \sigma(t-1)\}$. Set $\sigma(t) = x$.

We analyze A's and OPT's cost incurred on a given sequence σ^j. Again, whenever A moves an item x closer to the front of the list using paid exchanges, we charge the cost of these paid exchanges to the next request to x. We claim that in each phase, A incurs a cost of at least

$$jn - \frac{j(j-1)}{2} + (l+1-j).$$

The claim clearly holds if $j = l+1$ or if $j < l+1$ and A always stores x_1 at the first position of the list. In these two cases we can use the same analysis as in the proof of Lemma 7. If $j < l+1$ and if A does not always store x_1 at the front of the list, then consider the following algorithm A'. The algorithm A' always maintains the items $x \neq x_1$ in the same order as A, but always stores x_1 at the first position of the list. It is easy to verify that in each phase, A' does not incur a higher cost than A. Again, using the same analysis as in the proof of Lemma 7, we can show that on any $l+1$ successive requests, A' incurs a cost of at least $jn - \frac{1}{2}j(j-1) + (l+1-j)$.

We show that in each phase, OPT's amortized cost is at most

$$j\left(\frac{n(n+1)}{2(n-1)} - \frac{1}{n-1}\right) + (l+1-j).$$

This bound obviously holds true if $j = l+1$ because

$$\frac{n(n+1)}{2(n-1)} - \frac{1}{n-1} \geq \frac{n+1}{2}.$$

If $j < l+1$ then OPT can apply the following static algorithm. Initially, the list is rearranged such that item x_1 occupies position 1 in the list and such that the remaining items are sorted in

order of non-increasing request frequencies. While processing σ^j no exchanges are made. Using this static algorithm, OPT's amortized cost on a request to an item $x \neq x_1$ is at most

$$\frac{1}{n-1}(\sum_{k=2}^{n} k) = \frac{n(n+1)}{2(n-1)} - \frac{1}{n-1}.$$

Hence A's competitive factor c satisfies $c \geq \max_j C_n(j)$, where

$$C_n(j) = \frac{jn - \frac{j(j-1)}{2} + (l+1-j)}{j(\frac{n(n+1)}{2(n-1)} - \frac{1}{n-1}) + (l+1-j)} = \frac{2jn - j^2 - j + 2l + 2}{jn + 2l + 2}. \tag{5}$$

Now we prove the two parts of the theorem.

Part a): If $l = O(1)$ then, for all positive integers n, set $j_n = 1$. It is easy to see that $C_n(j_n)$ converges to 2 as n tends to infinity.

Now suppose $l = \omega(1)$ and $l = O(n^2)$. We maximize the function

$$C_n(j) = \frac{2jn - j^2 - j + 2l + 2}{jn + 2l + 2} \tag{6}$$

subject to the constraint $0 < j \leq \min\{l+1, n-1\}$. Here we are also interested in possibly non-integral solutions for j. We determine j_n such that $\frac{dC_n(j_n)}{dj} = 0$.

$\frac{dC_n(j_n)}{dj} = 0$ is equivalent to

$$\begin{aligned}
& (2n - 2j_n - 1)(j_n n + 2l + 2) - (2j_n n - j_n^2 - j_n + 2l + 2)n &= 0 \\
\Leftrightarrow \quad & 2j_n n^2 + 4nl + 4n - 2j_n^2 n - 4j_n l - 4j_n - j_n n - 2l - 2 & \\
& -(2j_n n^2 - j_n^2 n - j_n n + 2nl + 2n) &= 0 \\
\Leftrightarrow \quad & j_n^2 n + 4j_n l + 4j_n - 2nl - 2n + 2l + 2 &= 0
\end{aligned}$$

This implies

$$(j_n + 2\frac{(l+1)}{n})^2 = 4\frac{(l+1)^2}{n^2} + 2(l+1) - 2\frac{(l+1)}{n}.$$

Since we require $j_n > 0$, only

$$j_n = \frac{1}{n}(\sqrt{4(l+1)^2 + 2(l+1)n(n-1)} - 2(l+1))$$

can be a solution to our maximization problem.

Defining $D = 4(l+1)^2 + 2(l+1)n(n-1)$, we have

$$\begin{aligned}
C_n(j_n) &= \frac{1}{\sqrt{D}}(2\sqrt{D} - 4(l+1) - \frac{1}{n^2}(D - 4\sqrt{D}(l+1) + 4(l+1)^2) \\
&\quad -\frac{1}{n}(\sqrt{D} - 2(l+1)) + 2(l+1)) \\
&= \frac{1}{\sqrt{D}}(2\sqrt{D} - \frac{2}{n^2}D + \frac{1}{n^2}\sqrt{D}(4(l+1) - n)) \\
&= 2 - \frac{2}{n^2}\sqrt{D} + \frac{1}{n^2}(4(l+1) - n).
\end{aligned}$$

Hence

$$C_n(j_n) = 2 - \frac{2}{n^2}\sqrt{4(l+1)^2 + 2(l+1)n(n-1)} + \frac{1}{n^2}(4(l+1) - n). \tag{7}$$

It is easy to verify that $C_n(j_n)$ is in fact a maximum of the function $C_n(j)$ and that $0 < j_n \leq \min\{l+1, n-1\}$.

Note that j_n might not be an integer. However, since $l = \omega(1)$, the sequence j_n, $j = 1, 2, 3, \ldots$, is $\omega(1)$. Thus, using equation (6), we can easily prove that the sequences $C_n(j_n)$ and $C_n(\lfloor j_n \rfloor)$ have the same limit as n tends to infinity. Taking the limit of the sequence $C_n(j_n)$, we obtain that A's competitive factor cannot be asymptotically better than 2, if $l = o(n^2)$. This proves part a) of the theorem.

Part b): If $(l+1) = Kn^2$, then by equation (7)

$$\begin{aligned} C_n(j_n) &= 2 - \frac{2}{n^2}\sqrt{4K^2n^4 + 2Kn^4 - 2Kn^3} + 4K - \frac{1}{n} \\ &\geq 2 - 2\sqrt{4K^2 + 2K} + 4K - \frac{1}{n} \end{aligned}$$

and this expression converges to

$$2 - 2\sqrt{4K^2 + 2K} + 4K$$

as n tends to infinity. □

Remarks: (1) Note that the bounds given in Theorem 13 hold asymptotically. For small values of n one can derive more precise bounds. (2) At first sight Theorem 13 seems to imply that it would not be worthwhile to consider weak lookahead in the list update problem. This might not be true, as the following example shows. Part b) of the above theorem implies that an on-line algorithm needs a weak lookahead of at least $l = \frac{1}{100}n^2 - 1$ in order to be 1.75-competitive. Recall that the list update problem is of practical interest for small lists consisting of only a few dozen items. For $n_1 = 12$ and $n_2 = 24$ we obtain a lookahead of size $l_1 = 1$ and $l_2 = 5$, respectively. Assuming that our lower bounds are relatively tight, a 1.75-competitive algorithm working on small lists would require a weak lookahead of reasonable size.

5.2 Upper bounds against static adversaries

In this section we present deterministic on-line algorithms with lookahead for the list update problem. These algorithms are competitive, if the off-line algorithm uses a *static* algorithm. Static algorithms initially arrange the list in some order and make no other exchanges while processing the request sequence. Static algorithms are less powerful than dynamic off-line algorithms which may rearrange the list at each request. In the following STAT denotes the optimal static off-line algorithm. Given a request sequence σ, STAT arranges the item in the list in order of non-increasing request frequencies.

Early work on the list update problem has evaluated the performance of on-line algorithms against static adversaries, e.g. Bentley and McGeoch [BM85] have shown that the move-to-front algorithm is 2-competitive against static off-line algorithms. Recently, d'Amore *et al.* [DMN91] have discussed a variant of the list update problem, called the weighted list update problem,

with respect to static adversaries. Static off-line algorithms are interesting from the practical point of view, since they can compute an optimal ordering of the list in $O(m)$ time, where m is the length of the request sequence. The best known dynamic off-line algorithm due to Reingold and Westbrook [RW90] takes $O(2^n n! m)$ time.

In the following we consider strong and weak lookahead in parallel because the algorithms and analyses are very similar for both kinds of lookahead. Note that if a strong lookahead l is provided, then $l \leq n-1$.

Algorithm FREQUENCY-COUNT(l): Serve the request sequence in a series of blocks $B(i)$. If a strong lookahead l is given, then $B(1) = \sigma(1), \sigma(2), \ldots, \sigma(\lambda(1))$ and $B(i) = \sigma(t^s_{i-1} + 1), \sigma(t^s_{i-1}+2), \ldots, \sigma(\lambda(t^s_{i-1}+1))$ for $i \geq 2$. Here t^s_{i-1} denotes the end of block $B(i-1)$. If a weak lookahead l is provided, then $B(i) = \sigma((i-1)(l+1)+1), \sigma((i-1)(l+1)+2), \ldots, \sigma(i(l+1))$ for $i \geq 1$. Each block is processed as follows. At the beginning of each block, sort the items in the list such that they are in non-increasing order of request frequencies with respect to the current block. Execute this step using as few exchanges as possible. (This restriction ensures that items with the same request frequency are not exchanged.) After this rearrangement, serve the requests in the current block without making any further exchanges. Note that the sorting of the items can be implemented as follows. First determine the items with the highest request frequency in the current block, and move these items in an order preserving way to the front of the list. Then determine the items with the next lower request frequency and move these items (in an order preserving way) as close to the front of the list as possible, but without passing the items with the highest request frequency. Repeat this process for the other request frequencies.

Theorem 14 *Let $l \leq n-1$. The algorithm FREQUENCY-COUNT(l) with strong lookahead l is c-competitive against static off-line algorithms, where*

$$c \leq 2 - \frac{2}{3} \cdot \frac{l+2}{2n-l}.$$

Theorem 15 *Let $K > 0$ be a real constant. If a weak lookahead l is given with $(l+1) = Kn^2$, then FREQUENCY-COUNT(l) is c-competitive against static off-line algorithms, where*

$$c \leq 2 - \frac{1}{3}\left(\frac{2}{(n-1)^2}\sqrt{K^2n^4 + Kn^2(2n-1)(n-1)} - \frac{2Kn^2}{(n-1)^2} - \frac{1}{n-1}\right).$$

Notice that FREQUENCY-COUNT(l) can be (4/3)-competitive, if a large lookahead is given. Table 1 compares, for various values of a weak lookahead l and various n, the performance of FREQUENCY-COUNT(l) to the lower bounds derived in Section 5.1.

In order to prove the two theorems, we start with a general analysis of the algorithm FREQUENCY-COUNT(l) (also called FC) that applies to strong and weak lookahead. We use a potential function Φ to analyse the performance of our on-line algorithm. Φ is the number of inversions in FC's list with respect to STAT's list. Given two lists containing the same items, an inversion is an unordered pair of items $\{x, y\}$ such that x occurs before y in one list while x occurs after y in the other list. We assume that FC and STAT start with the same list, such that the initial potential is zero.

$l+1$	Competitive Factors		Value of $(l+1)$ for			
	Lower Bound	Upper Bound for $n \in [15, 30]$	$n = 15$	$n = 20$	$n = 25$	$n = 30$
$\frac{1}{500}n^2$	1.88	1.98	0.45	0.8	1.25	1.8
$\frac{1}{200}n^2$	1.82	1.95	1.125	2	3.125	4.5
$\frac{1}{100}n^2$	1.75	1.93	2.25	4	6.25	9
$\frac{1}{50}n^2$	1.67	1.89	4.5	8	12.5	18
$\frac{1}{20}n^2$	1.54	1.83	11.25	20	31.25	45
$\frac{1}{10}n^2$	1.42	1.77	22.5	40	62.5	90

Table 1: Competitive factors for list update with weak lookahead

There is given a request sequence σ. Initially, STAT rearranges the items in the list using paid exchanges. Each paid exchange incurs a cost of 1 and can increase the potential by 1. In the following we bound FC's amortized cost in each block of σ. We consider an arbitrary block B. Let $C_{FC}(B)$ be the actual cost FC incurs in processing B and let $\Delta\Phi$ be the change in the potential function between the beginning and the end of the given block. The sum $C_{FC}(B)+\Delta\Phi$ is FC's amortized cost in block B. Furthermore, let S be the set of items in the list, and let S_B be the set of items requested in block B. For an item $x \in S_B$ and $A \in \{\text{FC, STAT}\}$, let $C_A(x)$ be the cost that algorithm A incurs when serving a request to item x in block B. $f_B(x)$ denotes the request frequency of item x in block B, i.e. $f_B(x)$ is the number of times item x is requested in B. Finally, let $j = card(S_B)$ be the number of different items requested in B. Note that $j = l+1$ if we deal with strong lookahead.

Lemma 8

$$C_{FC}(B) + \Delta\Phi \leq 2 \sum_{x \in S_B} C_{STAT}(x) + \frac{4}{3} \sum_{x \in S_B} (f_B(x) - 1) C_{STAT}(x) - \frac{1}{3} j(j+1)$$

Proof: For a subset $M \subseteq S$ we introduce the following definitions.

1. For $A \in \{\text{FC, STAT}\}$ and $x \in S_B$ let

$$C_A(x, M) = card(\{y \in M |\ y = x \text{ or item } y \text{ precedes item } x \text{ in } A\text{'s list when } A \text{ serves a request to } x \text{ in block } B\}).$$

   $C_A(x, M)$ can be regarded as the cost caused by set M when A serves a request to item x.

2. Let $\Delta\Phi^+(M)$ be the number of inversions $\{x, y\}$ created between items $x \in S_B$ and $y \in M$ when B is served, and let $\Delta\Phi^-(M)$ be the number of inversions $\{x, y\}$ removed between items $x \in S_B$ and $y \in M$. Set $\Delta\Phi(M) = \Delta\Phi^+(M) - \Delta\Phi^-(M)$.

3. Let $p(M)$ be the number of paid exchanges FC incurs when swapping an item $x \in S_B$ with an item $y \in M$ at the beginning of the block.

Notice that for any $x \in S_B$ and $A \in \{\text{FC, STAT}\}$

$$C_A(x) = C_A(x, S_B) + C_A(x, S \setminus S_B)$$

and

$$\Delta\Phi = \Delta\Phi(S_B) + \Delta\Phi(S \setminus S_B).$$

FC's amortized cost in block B satisfies

$$\begin{aligned} C_{FC}(B) + \Delta\Phi &= \sum_{x \in S_B} f_B(x) C_{FC}(x) + p(S_B) + p(S \setminus S_B) + \Delta\Phi(S_B) + \Delta\Phi(S \setminus S_B) \\ &= \sum_{x \in S_B} f_B(x) C_{FC}(x, S_B) + p(S_B) + p(S \setminus S_B) + \Delta\Phi(S_B) + \Delta\Phi(S \setminus S_B). \end{aligned}$$

The last equation follows because $C_{FC}(x, S \setminus S_B) = 0$ for all $x \in S_B$.

Claim 1

$$p(S \setminus S_B) + \Delta\Phi(S \setminus S_B) \leq 2 \sum_{x \in S_B} C_{STAT}(x, S \setminus S_B)$$

Proof of Claim 1: We have

$$\sum_{x \in S_B} C_{STAT}(x, S \setminus S_B) = \sum_{x \in S_B} \sum_{y \in S \setminus S_B} C_{STAT}(x, \{y\}).$$

Suppose FC moves an item $x \in S_B$ closer to the front of the list using paid exchanges and swaps x with an item $y \in S \setminus S_B$. If an inversion is removed, then the potential decreases by 1. If an inversion is created, then the pair $\{x, y\}$ incurs a cost of 2 on the left hand side of the inequality in the claim. But $C_{STAT}(x, \{y\}) = 1$. This proves the claim. □

Claim 2

$$\sum_{x \in S_B} f_B(x) C_{FC}(x, S_B) + p(S_B) - \Delta\Phi^-(S_B) \leq \sum_{x \in S_B} f_B(x) C_{STAT}(x, S_B)$$

Proof of Claim 2: We have

$$\sum_{x \in S_B} f_B(x) C_{FC}(x, S_B) = \sum_{x \in S_B} \sum_{y \in S_B} f_B(x) C_{FC}(x, \{y\})$$

and

$$\sum_{x \in S_B} f_B(x) C_{STAT}(x, S_B) = \sum_{x \in S_B} \sum_{y \in S_B} f_B(x) C_{STAT}(x, \{y\}).$$

This implies that the inequality in the claim is equivalent to

$$\sum_{x \in S_B} \sum_{\substack{y \in S_B \\ y \neq x}} f_B(x) C_{FC}(x, \{y\}) + p(S_B) - \Delta\Phi^-(S_B) \leq \sum_{x \in S_B} \sum_{\substack{y \in S_B \\ y \neq x}} f_B(x) C_{STAT}(x, \{y\}). \quad (8)$$

Consider any pair $\{x, y\}$ with $x, y \in S_B$ and $x \neq y$. Suppose y is before x in FC's list after the rearrangement of the items in S_B.

Case 1: If FC does not swap x and y at the beginning of the block, then

$$f_B(x)C_{FC}(x,\{y\}) + f_B(y)C_{FC}(y,\{x\}) \leq f_B(x)C_{STAT}(x,\{y\}) + f_B(y)C_{STAT}(y,\{x\}).$$

Case 2: If FC swaps x and y and the potential decreases, then

$$f_B(x)C_{FC}(x,\{y\}) + f_B(y)C_{FC}(y,\{x\}) + 1 - 1 \leq f_B(x)C_{STAT}(x,\{y\}) + f_B(y)C_{STAT}(y,\{x\}).$$

Case 3: If FC swaps x and y and the potential increases, then

$$f_B(x)C_{FC}(x,\{y\}) + f_B(y)C_{FC}(y,\{x\}) + 1 \leq f_B(x)C_{STAT}(x,\{y\}) + f_B(y)C_{STAT}(y,\{x\}),$$

because $f_B(y) > f_B(x)$.

Adding the appropriate inequalities for all such pairs, we obtain inequality (8). □

Claim 3

$$\Delta\Phi^+(S_B) \leq \frac{1}{3}\sum_{x \in S_B} f_B(x)C_{STAT}(x, S_B)$$

Proof of Claim 3: Suppose FC moves an item x closer to the front of the list and creates an inversion with an item $y \in S_B$. Notice that x must be requested at least twice in block B and that $C_{STAT}(x,\{y\}) = 1$. f x is requested three times, then we may charge a cost of $1/3$ to each of these $f_B(x)$ requests.

We estimate the number J of inversions created between items requested twice and items requested once in B. Let S_B^1 be the set of items requested exactly once in B and let S_B^2 be the set of items requested exactly twice in B. Define $j_1 = card(S_B^1)$ and $j_2 = card(S_B^2)$. We prove

$$J \leq \frac{1}{3}\sum_{x \in S_B^1 \cup S_B^2} C_{STAT}(x, S_B^1 \cup S_B^2) + \frac{1}{3}\sum_{x \in S_B^2} C_{STAT}(x, S_B^1 \cup S_B^2). \tag{9}$$

This obviously implies the claim. We have $\sum_{x \in S_B^1 \cup S_B^2} C_{STAT}(x, S_B^1 \cup S_B^2) = \frac{1}{2}(j_1+j_2)(j_1+j_2+1)$. First suppose that each of the j_2 items in S_B^2 causes j_1 new inversions. Then $J = j_1 j_2$ and $\sum_{x \in S_B^2} C_{STAT}(x, S_B^1 \cup S_B^2) = \frac{1}{2}((j_1+j_2)(j_1+j_2+1) - j_1(j_1+1))$. Now suppose that an item $x \in S_B^2$ causes only $j_1 - k_x$ inversions. Then, $J = j_1 j_2 - \sum_{x \in S_B^2} k_x$ and

$$\frac{1}{2}((j_1+j_2)(j_1+j_2+1) - j_1(j_1+1)) - \sum_{x \in S_B^2} k_x \leq \sum_{x \in S_B^2} C_{STAT}(x, S_B^1 \cup S_B^2).$$

It is easy to prove that

$$j_1 j_2 \leq \frac{1}{3}(\frac{1}{2}(j_1+j_2)(j_1+j_2+1) + \frac{1}{2}((j_1+j_2)(j_1+j_2+1) - j_1(j_1+1))).$$

Using the last two inequalities, we can easily derive inequality (9). □

Summing up the inequalities in Claim 1, Claim 2 and Claim 3 we obtain, as desired,

$$\begin{aligned} C_{FC}(B) + \Delta\Phi &\leq 2\sum_{x\in S_B} C_{STAT}(x, S \setminus S_B) + \frac{4}{3}\sum_{x\in S_B} f_B(x) C_{STAT}(x, S_B) \\ &\leq 2\sum_{x\in S_B} C_{STAT}(x) + \frac{4}{3}\sum_{x\in S_B}(f_B(x)-1)C_{STAT}(x) - \frac{2}{3}\cdot\frac{j(j+1)}{2} \end{aligned}$$

□

Proof of Theorem 14 Suppose the request sequence consists of b blocks $B(1), B(2), \ldots, B(b)$. By Lemma 8,

$$\frac{C_{FC}(\sigma)}{C_{STAT}(\sigma)} \leq \frac{\sum_{i=1}^{b}(2\sum_{x\in S_{B(i)}} C_{STAT}(x) + \frac{4}{3}\sum_{x\in S_{B(i)}}(f_{B(i)}(x)-1)C_{STAT}(x) - \frac{(l+1)(l+2)}{3})}{C_{STAT}(\sigma)}$$

Here we assume without loss of generality that the last block $B(b)$ contains $l+1$ distinct requests. If $B(b)$ contains less than $l+1$ distinct requests, then we can simply add $\frac{1}{3}\frac{(l+1)(l+2)}{C_{STAT}(\sigma)}$ on the right-hand side of the above inequality. This does not affect FC's competitive factor. Hence,

$$\frac{C_{FC}(\sigma)}{C_{STAT}(\sigma)} \leq 2 - \frac{2}{3}\cdot\frac{\frac{b(l+1)(l+2)}{2} + \sum_{i=1}^{b}\sum_{x\in S_{B(i)}}(f_{B(i)}(x)-1)C_{STAT}(x)}{\sum_{i=1}^{b}\sum_{x\in S_{B(i)}} f_{B(i)}(x) C_{STAT}(x)}$$

We have

$$\sum_{i=1}^{b}\sum_{x\in S_B} C_{STAT}(x) \geq b\frac{(l+1)(l+2)}{2}.$$

Thus

$$\begin{aligned} \frac{C_{FC}(\sigma)}{C_{STAT}(\sigma)} &\leq 2 - \frac{2}{3}\cdot\frac{\frac{b(l+1)(l+2)}{2}}{\sum_{i=1}^{b}\sum_{x\in S_{B(i)}} C_{STAT}(x)} \\ &\leq 2 - \frac{2}{3}\cdot\frac{(l+2)/2}{n-l/2}. \end{aligned}$$

The last line follows because $\sum_{i=1}^{b}\sum_{x\in S_{B(i)}} C_{STAT}(x) \leq b\sum_{k=0}^{l}(n-k) = b((l+1)n - l(l+1)/2)$. We conclude that FC is c-competitive, where

$$c \leq 2 - \frac{2}{3}\cdot\frac{l+2}{2n-l}.$$

□

Proof of Theorem 15: Again, we assume that the request sequence σ consists of b blocks $B(1), B(2), \ldots, B(b)$. Furthermore, we assume without loss of generality that the last block $B(b)$ contains $l+1$ requests. Let j_i be the number of different items requested in block $B(i)$. By Lemma 8

$$\frac{C_{FC}(\sigma)}{C_{STAT}(\sigma)} \leq \frac{\sum_{i=1}^{b}(2\sum_{x\in S_{B(i)}} C_{STAT}(x) + \frac{4}{3}\sum_{x\in S_{B(i)}}(f_{B(i)}(x)-1)C_{STAT}(x) - \frac{j_i(j_i+1)}{3})}{\sum_{i=1}^{b}(\sum_{x\in S_{B(i)}} C_{STAT}(x) + \sum_{x\in S_{B(i)}}(f_{B(i)}(x)-1)C_{STAT}(x))}$$

Note that $\sum_{x \in S_{B(i)}} C_{STAT}(x) \leq j_i n$ and that

$$\sum_{i=1}^{b} j_i(j_i+1) \geq bj(j+1),$$

where $j = \frac{1}{b}\sum_{i=1}^{b} j_i$. Hence,

$$\frac{C_{FC}(\sigma)}{C_{STAT}(\sigma)} \leq \frac{\sum_{i=1}^{b}(2jn - \frac{j(j+1)}{3} + \frac{4}{3}\sum_{x \in S_{B(i)}}(f_{B(i)}(x)-1)C_{STAT}(x))}{\sum_{i=1}^{b}(jn + \sum_{x \in S_{B(i)}}(f_{B(i)}(x)-1)C_{STAT}(x))}.$$

It is easy to verify that

$$\frac{2jn - \frac{1}{3}j(j+1)}{jn} \geq \frac{4}{3}.$$

Hence

$$\begin{aligned}\frac{C_{FC}(\sigma)}{C_{STAT}(\sigma)} &\leq \frac{\sum_{i=1}^{b}(2jn - \frac{1}{3}j(j+1) + \frac{4}{3}(l+1-j))}{\sum_{i=1}^{b}(jn + (l+1-j))} \\ &= \frac{2jn - \frac{1}{3}j^2 - \frac{5}{3}j + \frac{4}{3}(l+1)}{jn - j + (l+1)}.\end{aligned}$$

We have $(l+1) = Kn^2$. We maximize the function

$$C_n(j) = \frac{2jn - \frac{1}{3}j^2 - \frac{5}{3}j + \frac{4}{3}Kn^2}{jn - j + Kn^2}$$

subject to the constraint $0 < j \leq \min\{Kn^2, n\}$. We determine j_n such that $\frac{dC_n(j_n)}{dj} = 0$.

$\frac{dC_n(j_n)}{dj} = 0$ is equivalent to

$$\begin{aligned}&(2n - \tfrac{2}{3}j_n - \tfrac{5}{3})(j_n n - j_n + Kn^2) - (2j_n n - \tfrac{1}{3}j_n^2 - \tfrac{5}{3}j_n + \tfrac{4}{3}Kn^2)(n-1) = 0 \\ \Leftrightarrow\ &2j_n n^2 - 2j_n n + 2Kn^3 - \tfrac{2}{3}j_n^2 n + \tfrac{2}{3}j_n^2 - \tfrac{2}{3}Kj_n n^2 - \tfrac{5}{3}j_n n + \tfrac{5}{3}j_n - \tfrac{5}{3}Kn^2 \\ &-(2j_n n^2 - 2j_n n - \tfrac{1}{3}j_n^2 n + \tfrac{1}{3}j_n^2 - \tfrac{5}{3}j_n n + \tfrac{5}{3}j_n + \tfrac{4}{3}Kn^3 - \tfrac{4}{3}Kn^2) = 0 \\ \Leftrightarrow\ &\tfrac{1}{3}j_n^2 - \tfrac{1}{3}j_n^2 n - \tfrac{2}{3}Kj_n n^2 + \tfrac{2}{3}Kn^3 - \tfrac{1}{3}Kn^2 = 0 \\ \Leftrightarrow\ &j_n^2(n-1) + 2Kj_n n^2 - (2n-1)Kn^2 = 0\end{aligned}$$

This is equivalent to

$$(j_n + \frac{Kn^2}{n-1})^2 = (\frac{Kn^2}{n-1})^2 + Kn^2(\frac{2n-1}{n-1}).$$

Since we require $j_n > 0$, only

$$j_n = \frac{1}{n-1}(\sqrt{K^2n^4 + Kn^2(2n-1)(n-1)} - Kn^2)$$

can be a solution to our maximization problem.

We have

$$C_n(j_n) = 2 - \frac{1}{3}\cdot\frac{j_n^2 - j_n + 2Kn^2}{j_n n - j_n + Kn^2}$$

Define $D = K^2n^4 + Kn^2(2n-1)(n-1)$. Then

$$\begin{aligned}\frac{j_n^2 - j_n + 2Kn^2}{j_n n - j_n + Kn^2} &= \frac{1}{\sqrt{D}}\left(\frac{1}{(n-1)^2}(D - 2Kn^2\sqrt{D} + K^2n^4) - \frac{1}{n-1}(\sqrt{D} - Kn^2) + 2Kn^2\right) \\ &= \frac{2}{(n-1)^2}\sqrt{D} - \frac{2Kn^2}{(n-1)^2} - \frac{1}{n-1}.\end{aligned}$$

Hence

$$C_n(j_n) = 2 - \frac{1}{3}\left(\frac{2}{(n-1)^2}\sqrt{K^2n^4 + Kn^2(2n-1)(n-1)} - \frac{2Kn^2}{(n-1)^2} - \frac{1}{n-1}\right).$$

It is easy to see that $C_n(j_n)$ is in fact a maximum of $C_n(j)$ and that $0 < j_n \leq \min\{Kn^2, n\}$.

We remark that it is possible to derive more precise but also more complicated bounds on the competitive factor, if one takes into account that $\sum_{x \in S_{B(i)}} C_{STAT}(x) \leq j_i n - \frac{j_i(j_i-1)}{2}$. □

Now we give a lower bound on the performance of FC. We show that FC with a strong lookahead $1 \leq l \leq n-1$ is not better than c-competitive, where

$$c \geq \max\{2 - \frac{l+2}{n+1}, \frac{4(n+1) - \lfloor\frac{(l+1)}{2}\rfloor - 1}{3(n+1) + \frac{2n+2}{l}}\}.$$

This implies that FC is not better than $(7/6)$-competitive if a strong lookahead $l = n-1$ is given. The bound of $c \geq 2 - \frac{l+2}{n+1}$ is obvious. We prove the bound

$$c \geq \frac{4(n+1) - \lfloor\frac{(l+1)}{2}\rfloor - 1}{3(n+1) + \frac{2n+2}{l}}.$$

We construct a request sequence consisting of a series of blocks $B(i)$. Each block contains $l+1$ distinct requests. Let L_1 be the list given initially, and let L_i be the configuration of the list after FC has served block $B(i-1)$, where $i \geq 2$. Given L_i, we show how to contruct $B(i)$. First we request the last $\lfloor\frac{l+1}{2}\rfloor$ items stored in L_i; each of these items is requested exactly twice. Then we add one request to each of the items stored at positions

$$n-l, n-l+1, \ldots, n - \lfloor\frac{l+1}{2}\rfloor$$

in L_i.

The algorithm OPT can serve the request sequence such that its amortized cost in one block is less than or equal to

$$(l+1+\lfloor\frac{l+1}{2}\rfloor)\frac{n+1}{2}$$

which is at most

$$3\lfloor\frac{l+1}{2}\rfloor\frac{n+1}{2} + \frac{n+1}{2}.$$

We analyze FC's cost in an arbitrary block $B(i)$. The algorithm moves the items requested in $B(i)$ to the front of the list. In particular, all items requested twice are exchanged with items requested only once. FC incurs a cost of

$$(l+1)(n-(l+1)) + \lfloor\frac{n+1}{2}\rfloor(l+1-\lfloor\frac{l+1}{2}\rfloor)$$

for paid exchanges. After the rearrangement, the requests in the current block can be processed using a cost of

$$\frac{1}{2}(l+1)(l+2) + \frac{1}{2}\lfloor\frac{l+1}{2}\rfloor(\lfloor\frac{l+1}{2}\rfloor+1).$$

Hence FC's cost in block $B(i)$ satisfies

$$C_{FC}(B(i)) = n(l+1) - \frac{1}{2}(l+1)^2 + \frac{1}{2}(l+1) + \lfloor\frac{l+1}{2}\rfloor\lceil\frac{l+1}{2}\rceil + \frac{1}{2}\lfloor\frac{l+1}{2}\rfloor(\lfloor\frac{l+1}{2}\rfloor+1).$$

We show

$$n(l+1) - \frac{1}{2}(l+1)^2 \geq 2n\lfloor\frac{l+1}{2}\rfloor - 2\lfloor\frac{l+1}{2}\rfloor\lfloor\frac{l+1}{2}\rfloor.$$

This inequality clearly holds if $\lfloor\frac{l+1}{2}\rfloor = \frac{l+1}{2}$. If $\lfloor\frac{l+1}{2}\rfloor = \frac{l+1}{2} - \frac{1}{2}$, then

$$\begin{aligned} n(l+1) - \frac{1}{2}(l+1)^2 &= 2n\lfloor\frac{l+1}{2}\rfloor + n - 2(\lfloor\frac{l+1}{2}\rfloor + \frac{1}{2})^2 \\ &= 2n\lfloor\frac{l+1}{2}\rfloor + n - 2\lfloor\frac{l+1}{2}\rfloor\lfloor\frac{l+1}{2}\rfloor - 2\lfloor\frac{l+1}{2}\rfloor - \frac{1}{2} \\ &\geq 2n\lfloor\frac{l+1}{2}\rfloor - 2\lfloor\frac{l+1}{2}\rfloor\lfloor\frac{l+1}{2}\rfloor \end{aligned}$$

Thus

$$\begin{aligned} C_{FC}(B(i)) &\geq 2n\lfloor\frac{l+1}{2}\rfloor - \lfloor\frac{l+1}{2}\rfloor\lfloor\frac{l+1}{2}\rfloor + \frac{1}{2}(l+1) + \frac{1}{2}\lfloor\frac{l+1}{2}\rfloor(\lfloor\frac{l+1}{2}\rfloor+1) \\ &\geq 2n\lfloor\frac{l+1}{2}\rfloor - \lfloor\frac{l+1}{2}\rfloor(\lfloor\frac{l+1}{2}\rfloor-1) + \frac{1}{2}\lfloor\frac{l+1}{2}\rfloor(\lfloor\frac{l+1}{2}\rfloor+1) \\ &= 2n\lfloor\frac{l+1}{2}\rfloor - \lfloor\frac{l+1}{2}\rfloor(\frac{1}{2}\lfloor\frac{l+1}{2}\rfloor - \frac{3}{2}) \end{aligned}$$

We conclude that FC's competitive factor c satisfies

$$c \geq \frac{2n - \frac{1}{2}\lfloor\frac{l+2}{2}\rfloor + \frac{3}{2}}{3\frac{n+1}{2} + \frac{n+1}{2}\cdot(\frac{1}{\lfloor(l+1)/2\rfloor})} \geq \frac{4(n+1) - \lfloor\frac{l+1}{2}\rfloor - 1}{3(n+1) + \frac{2n+2}{l}}.$$

5.3 Upper bounds against dynamic adversaries

We present an on-line algorithm with lookahead which is competitive against dynamic off-line algorithms.

Algorithm MTF/OPT(l): Serve the request sequence in a series of blocks $B(i)$. If a strong lookahead l is given then $B(1) = \sigma(1), \sigma(2), \ldots, \sigma(\lambda(1))$ and $B(i) = \sigma(t^e_{i-1}+1), \sigma(t^e_{i-1}+2), \ldots, \sigma(\lambda(t^e_{i-1}+1))$ for $i \geq 2$. Here t^e_{i-1} denotes the end of block $B(i-1)$. If a weak lookahead l is provided then $B(i) = \sigma((i-1)(l+1)+1), \sigma((i-1)(l+1)+2), \ldots, \sigma(i(l+1))$ for $i \geq 1$. At the beginning of each block, rearrange the list as follows.

(1) Determine the items that are requested in the current block and move them in an order preserving way to the front of the list.

(2) Then determine the items that are requested at least twice in the current block and move them in an order preserving way to the front of the list.

After this rearrangement, serve the requests in the current block such that the incurred cost is as small as possible. Whenever an item x is requested for the last time in the current block, determine the items y that precede x in the present list and that are still requested at least once in the current block. Move item x together with the items y in an order preserving way to the front of the list.

We are able to show that MTF/OPT(l) has a competitive factor of less than 2, if a large strong lookahead is provided (see Theorem 16 below). However, the competitive factor we can prove is quite weak. If $l = n-1$, then our theorem implies that MTF/OPT(l) is basically (15/8)-competitive. We conjecture that the above algorithm without step (2) is $(2-\frac{1}{2} \cdot \frac{l+2}{n+1})$-competitive if a strong lookahead is given and that this variant is optimal among on-line algorithms with strong lookahead which serve the request sequence in a series of blocks. It is possible to show that any on-line algorithm with strong lookahead $l = n-1$ which serves the request sequence in blocks cannot be better than 1.5-competitive.

Theorem 16 *If a strong lookahead $l < \lfloor \frac{n}{2} \rfloor$ is given, then the algorithm MTF/OPT(l) is 2-competitive. If a strong lookahead l, $\lfloor \frac{n}{2} \rfloor \leq l \leq n-1$, is given, then the algorithm MTF/OPT(l) is c-competitive, where*

$$c \leq \min\{2, 2 - \frac{\frac{1}{4}(2(l+1)-n)^2 + \frac{1}{4}(2(l+1)-n) - \frac{1}{8}(l+1)^2}{(l+1)n}\}.$$

Now we prove this theorem. We use a potential function Φ to analyze the algorithm MTF/OPT(l) (also called MO for simplicity). Again, Φ is the number of inversions in MO's list with respect to OPT's list. Recall that an inversion is an unordered pair $\{x, y\}$ of items such that x occurs before y in one list while x occurs after y in the other list. $\Phi(t)$ is the number of inversions after request $\sigma(t)$ has been served. The algorithms MO and OPT start with the same list such that the initial potential $\Phi(0)$ is 0.

We assume that the request sequence σ consists of b blocks $B(1), B(2), \ldots, B(b)$. For $i = 1, 2, \ldots, b$, the values t^b_i and t^e_i denote the beginning and the end of the ith block. We assume

that the last block $B(b)$ contains $l+1$ distinct items, too. As we shall see later, this assumption represents no restriction. We introduce the following notations. Let S be the set of items in the list and let $S_{B(i)}$ be the set of items which are requested during block $B(i)$. We partition $S_{B(i)}$ into two set $S^1_{B(i)}$ and $S^2_{B(i)}$. The set $S^1_{B(i)}$ contains all items which are requested exactly once in $B(i)$ while $S^2_{B(i)} = S_{B(i)} \setminus S^1_{B(i)}$ contains all items which are requested at least twice in $B(i)$. For $x \in S_{B(i)}$, the value $f_{B(i)}(x)$ denotes the request frequency of item x in block $B(i)$, i.e. $f_{B(i)}(x)$ is the number of times item x is requested during $B(i)$.

Now consider an arbitrary block $B(i)$. For an algorithm $A \in \{\text{MO, OPT}\}$ we introduce the following definitions. Let $C_A(B(i))$ be the cost incurred by algorithm A in processing block $B(i)$. Furthermore, for $x \in S_{B(i)}$ and $j = 1, 2, \ldots, f_{B(i)}(x)$, let $C^j_A(x,i)$ be the cost incurred by A when serving the jth request to item x in block $B(i)$. Set

$$C_A(x,i) = \sum_{j=1}^{f_{B(i)}(x)} C^j_A(x,i).$$

We split the cost $C_A(x,i)$ into two summands $C'_A(x,i)$ and $C''_A(x,i)$. Here $C'_A(x,i) = C^1_A(x,i)$ is the cost incurred by A when processing the first request to item x in $B(i)$. The difference $C''_A(x,i) = C_A(x,i) - C'_A(x,i)$ is the cost of processing the other requests to x in $B(i)$. Note that $C''_A(x,i) = 0$ if $x \in S^1_{B(i)}$.

If the algorithm A moves an item $x \in S_{B(i)}$ closer to the front of the list using paid exchanges, then we charge the cost of these paid exchanges to the next request to item x in block $B(i)$. More precisely, if A exchanges x with a preceding item y between the $(j-1)$st and the jth request to x, then we assume that y precedes x in A's list at the jth request to x, i.e. y is contained in the set

$$M = \{z \in S \mid z = x \text{ or item } z \text{ precedes item } x \text{ in } A\text{'s list when } A \text{ serves the } j\text{th request to } x \text{ in block } B(i)\}$$

and

$$C^j_A(x,i) = card(M).$$

Note that the algorithm MO does not swap an item $x \in S_{B(i)}$ with a preceding item in the list after the last request to item x in the present block has been served. For simplicity we assume that the same property also holds for the algorithm OPT. If OPT exchanges x with preceding items after its last request in a given block $B(i)$, then each of these paid exchanges can increase the potential by 1. But OPT incurs a cost of 1 for each paid exchange. Therefore, these paid exchanges cannot increase MO's competitive factor. The same argument holds if OPT uses paid exchanges to move an item $x \notin S_{B(i)}$ closer to the front of the list. Hence, in the following, we also assume that these paid exchanges do not occur.

Furthermore, for $i = 1, 2, \ldots, b$, we introduce sets $P_{B(i)}$. The set $P_{B(i)}$ contains

(1) all sets $\{x\}$ with $x \in S_{B(i)}$ and

(2) all sets $\{x, y\}$ with $x, y \in S_{B(i)}$, $x \neq y$ such that $\{x, y\}$ represents no inversion at the end of block $B(i)$.

In order to simplify the following proof, we define a set $S_{B(0)}$. $S_{B(0)}$ contains $l+1$ arbitrary

items in the initial list. The set $P_{B(0)}$ contains all sets $\{x, y\}$ with $x, y \in S_{B(0)}$.

We present a lemma which is crucial for the proof of Theorem 16. Define

$$D = \frac{1}{4}(2(l+1)-n)^2 + \frac{1}{4}(2(l+1)-n) - \frac{1}{8}(l+1)^2.$$

Note that D is the numerator of the fraction in Theorem 16. Finally, let $t_0^e = 0$.

Lemma 9 *a) If* $1 \leq l \leq n-1$, *then for* $i = 1, 2, \ldots, b$

$$C_{MO}(B(i)) + \Phi(t_i^e) - \Phi(t_{i-1}^e) \leq 2 \cdot C_{OPT}(B(i)).$$

b) If $l \geq \lfloor \frac{n}{2} \rfloor$, *then for* $i = 1, 2, \ldots, b$

$$\begin{aligned} C_{MO}(B(i)) + \Phi(t_i^e) - \Phi(t_{i-1}^e) \;\leq\; & 2 \sum_{x \in S_{B(i)}} C'_{OPT}(x,i) + \frac{3}{2} \sum_{x \in S_{B(i)}} C''_{OPT}(x,i) - D \\ & + \frac{1}{2}(card(P_{B(i-1)}) - card(P_{B(i)})). \end{aligned}$$

First we finish the proof of Theorem 16 and then show Lemma 9. We sum up the inequalities in Lemma 9. For arbitrary $1 \leq l \leq n-1$ we obtain

$$C_{MO}(\sigma) + \Phi(t_b^e) - \Phi(0) \leq 2 \cdot C_{OPT}(\sigma)$$

which implies that MO is 2-competitive.

Now let $l \geq \lfloor \frac{n}{2} \rfloor$ and $D \geq 0$. We obtain

$$\begin{aligned} \sum_{i=1}^{b} C_{MO}(B(i)) + \Phi(t_b^e) - \Phi(0) \;\leq\; & \sum_{i=1}^{b} (2 \sum_{x \in S_{B(i)}} C'_{OPT}(x,i) + \frac{3}{2} \sum_{x \in S_{B(i)}} C''_{OPT}(x,i)) - b \cdot D \\ & + \frac{1}{2} card(P_{B(0)}). \end{aligned}$$

Note that $card(P_{B(0)}) = (l+1)(l+2)/2$. Hence MO's competitive factor c satisfies

$$c \;\leq\; \frac{2 \cdot \sum_{i=1}^{b} \sum_{x \in S_{B(i)}} C'_{OPT}(x,i) - b \cdot D + \frac{3}{2} \cdot \sum_{i=1}^{b} \sum_{x \in S_{B(i)}} C''_{OPT}(x,i)}{\sum_{i=1}^{b} \sum_{x \in S_{B(i)}} C'_{OPT}(x,i) + \sum_{i=1}^{b} \sum_{x \in S_{B(i)}} C''_{OPT}(x,i)}$$

Taking into account that $\sum_{x \in S_{B(i)}} C'_{OPT}(x,i) \geq (l+1)(l+2)/2$, it is easy to prove that

$$\frac{2 \cdot \sum_{i=1}^{b} \sum_{x \in S_{B(i)}} C'_{OPT}(x,i) - b \cdot D}{\sum_{i=1}^{b} \sum_{x \in S_{B(i)}} C'_{OPT}(x,i)} \geq \frac{3}{2}.$$

Thus

$$\begin{aligned} c \;&\leq\; \frac{2 \cdot \sum_{i=1}^{b} \sum_{x \in S_{B(i)}} C'_{OPT}(x,i) - b \cdot D}{\sum_{i=1}^{b} \sum_{x \in S_{B(i)}} C'_{OPT}(x,i)} \\ &\leq\; 2 - \frac{b \cdot D}{b(l+1)n} \\ &=\; 2 - \frac{\frac{1}{4}(2(l+1)-n)^2 + \frac{1}{4}(2(l+1)-n) - \frac{1}{8}(l+1)^2}{(l+1)n} \end{aligned}$$

This proves the other desired bound.

Proof of Lemma 9: First we prove part b) of the lemma. Then we show that a simple modification of the proof yields part a).

We need a few more notations. Let $1 \leq i \leq b$. Consider a subset $M \subseteq S$ and an algorithm $A \in \{\text{MO, OPT}\}$. For $x \in S_{B(i)}$ and $j = 1, 2, \ldots, f_{B(i)}(x)$, let $C_A^j(x, i, M)$ be the cost caused by the set M when A serves the jth request to item x in block $B(i)$, i.e.

$$C_A^j(x, i, M) = card(\{y \in M |\ y = x \text{ or item } y \text{ precedes item } x \text{ in } A\text{'s list when } A \text{ serves the } j\text{th request to } x \text{ in block } B(i)\}).$$

Define

$$C_A(x, i, M) = \sum_{j=1}^{f_{B(i)}(x)} C_A^j(x, i, M),$$

$$C'_A(x, i, M) = C_A^1(x, i, M)$$

and

$$C''_A(x, i, M) = C_A(x, i, M) - C'_A(x, i, M).$$

For a subset $M \subseteq S$ let

$$P_{B(i)}(M) = \{N \in P_{B(i)} | N \subseteq M\}.$$

Furthermore, for subsets $M_1 \subseteq S$ and $M_2 \subseteq S$, let $\Phi(t, M_1, M_2)$ be the number of inversions $\{x, y\}$ which exist after request $\sigma(t)$ and which satisfy $x \in M_1$ and $y \in M_2$.

We proceed with the proof of part b) of the lemma. In the following we assume $l \geq \lfloor \frac{n}{2} \rfloor$. Consider a fixed $1 \leq i \leq b$.

Claim 4

$$\sum_{x \in S_{B(i)}} C_{MO}(x, i, S \setminus S_{B(i)}) + \Phi(t_i^e, S_{B(i)}, S \setminus S_{B(i)}) - \Phi(t_{i-1}^e, S_{B(i)}, S \setminus S_{B(i)})$$
$$\leq 2 \sum_{x \in S_{B(i)}} C'_{OPT}(x, i, S \setminus S_{B(i)})$$

Claim 5

$$\sum_{x \in S_{B(i)}} C_{MO}(x, i, S_{B(i)}) + \Phi(t_i^e, S_{B(i)}, S_{B(i)}) - \Phi(t_{i-1}^e, S_{B(i)}, S_{B(i)})$$
$$-\frac{1}{2} card(P_{B(i-1)}(S_{B(i)})) + \frac{1}{2} card(P_{B(i)}) - \frac{1}{2} card(\{\{x, y\} | x \in S_{B(i)}^1 \text{ and } y \in S_{B(i)}^2\})$$
$$\leq 2 \sum_{x \in S_{B(i)}} C'_{OPT}(x, i, S_{B(i)}) - \frac{1}{2} \sum_{x \in S_{B(i-1)} \cap S_{B(i)}} C'_{OPT}(x, i, S_{B(i-1)} \cap S_{B(i)})$$
$$+ \frac{3}{2} \sum_{x \in S_{B(i)}} C''_{OPT}(x, i, S_{B(i)}) \qquad (10)$$

Before we prove the two claims, we show that they imply part b) of Lemma 9. Summing up the two inequalities and taking into account that $P_{B(i-1)}(S_{B(i)}) \subseteq P_{B(i-1)}$, we obtain

$$\begin{aligned}
&C_{MO}(B(i)) + \Phi(t_i^e) - \Phi(t_{i-1}^e) \leq \\
&\quad 2 \sum_{x \in S_{B(i)}} C'_{OPT}(x,i) - \frac{1}{2} \sum_{x \in S_{B(i-1)} \cap S_{B(i)}} C'_{OPT}(x, i, S_{B(i-1)} \cap S_{B(i)}) \\
&\quad + \frac{3}{2} \sum_{x \in S_{B(i)}} C''_{OPT}(x,i) \\
&\quad + \frac{1}{2}(card(P_{B(i-1)}) - card(P_{B(i)})) + \frac{1}{2} card(\{\{x,y\} | x \in S^1_{B(i)} \text{ and } y \in S^2_{B(i)}\})
\end{aligned}$$

We show that

$$\frac{1}{2}\Big(\sum_{x \in S_{B(i-1)} \cap S_{B(i)}} C'_{OPT}(x, i, S_{B(i-1)} \cap S_{B(i)}) - card(\{\{x,y\} | x \in S^1_{B(i)} \text{ and } y \in S^2_{B(i)}\})\Big) \geq D.$$

This implies part b) of the lemma.

Since $l \geq \lfloor \frac{n}{2} \rfloor$, there exist $2(l+1) - n$ items in $S_{B(i)}$ which are also contained in $S_{B(i-1)}$. Hence

$$\begin{aligned}
\sum_{x \in S_{B(i-1)} \cap S_{B(i)}} C'_{OPT}(x, i, S_{B(i-1)} \cap S_{B(i)}) &\geq \frac{1}{2}(2(l+1) - n)(2(l+1) - n + 1) \\
&= \frac{1}{2}(2(l+1) - n)^2 + \frac{1}{2}(2(l+1) - n)
\end{aligned}$$

Note that

$$\begin{aligned}
card(\{\{x,y\} | x \in S^1_{B(i)} \text{ and } y \in S^2_{B(i)}\}) &= card(S^1_{B(i)}) \cdot card(S^2_{B(i)}) \\
&\leq \frac{l+1}{2} \cdot \frac{l+1}{2} \\
&= \frac{1}{4}(l+1)^2.
\end{aligned}$$

Using these two inequalities we obtain the desired bound for D.

It remains to prove the two claims. Recall that if an algorithm $A \in \{\text{MO}, \text{OPT}\}$ moves an item $x \in S_{B(i)}$ closer to the front of the list using paid exchanges, then we charge the cost of these paid exchanges to the next request to item x in $B(i)$. As mentioned before, we also assume that OPT does not use paid exchanges to move an item $x \in S_{B(i)}$ closer to the front of the list, after x has been requested for the last time in block $B(i)$. Furthermore, we assume the OPT does not swap items $x \in S \setminus S_{B(i)}$ with preceding items in the list.

Proof of Claim 4: Consider any two items $x \in S_{B(i)}$ and $y \in S \setminus S_{B(i)}$. We prove that

$$C_{MO}(x, i, \{y\}) + \Phi(t_i^e, \{x\}, \{y\}) - \Phi(t_{i-1}^e, \{x\}, \{y\}) \leq 2C'_{OPT}(x, i, \{y\}) \tag{11}$$

Summing up this inequality for all such pairs, we obtain Claim 4.

Note that $C_{MO}(x, i, \{y\}) \leq 1$ and that item x precedes item y in MO's list at the end of block $B(i)$.

First suppose $C_{MO}(x,i,\{y\}) = 0$. Then we have $\Phi(t_i^e,\{x\},\{y\}) - \Phi(t_{i-1}^e,\{x\},\{y\}) \leq 0$ and the inequality (11) clearly holds. The case $C_{MO}(x,i,\{y\}) = 0$ and $\Phi(t_i^e,\{x\},\{y\}) - \Phi(t_{i-1}^e,\{x\},\{y\}) = 1$ cannot occur because we assume that OPT does not exchange y with preceding items in the list during block $B(i)$.

Now let $C_{MO}(x,i,\{y\}) = 1$. If $\Phi(t_i^e,\{x\},\{y\}) - \Phi(t_{i-1}^e,\{x\},\{y\}) = -1$, then the inequality is obviously satisfied. If $\Phi(t_i^e,\{x\},\{y\}) - \Phi(t_{i-1}^e,\{x\},\{y\}) = 0$ or if $\Phi(t_i^e,\{x\},\{y\}) - \Phi(t_{i-1}^e,\{x\},\{y\}) = 1$, then $C'_{OPT}(x,i,\{y\}) = 1$. In each of the two cases the inequality (11) holds. □

Proof of Claim 5: Let I be the set of pairs $\{x,y\}$ satisfying the following properties:

(1) ($x \in S^1_{B(i)}$ and $y \in S^2_{B(i)}$) or ($x \in S^2_{B(i)}$ and $y \in S^2_{B(i)}$)

(2) The pair $\{x,y\}$ represents an inversion at the beginning of the given block or after MO has arranged the items in $S^1_{B(i)}$ and $S^2_{B(i)}$ at the front of the list (see steps (1) and (2) of the algorithm).

First we prove that

$$\sum_{x \in S_{B(i)}} C_{MO}(x,i,S_{B(i)}) \leq \sum_{x \in S_{B(i)}} C_{OPT}(x,i,S_{B(i)}) + card(I). \tag{12}$$

After MO has arranged the items in $S^1_{B(i)}$ and $S^2_{B(i)}$ at the front of the list, it serves the requests in the current block such that a minimum cost is incurred. Hence, MO never exchanges an item $x \in S^1_{B(i)}$ with a preceding item y, when the last request to y in the current block has not been processed. This implies that an inversion $\{x,y\}$ with $x,y \in S^1_{B(i)}$ cannot increase MO's cost. It is easy to see that each inversion $\{x,y\} \in I$ with $x \in S^1_{B(i)}$ and $y \in S^2_{B(i)}$ can cause an extra cost of 1. We show that each of the remaining inversions $\{x,y\} \in I$ with $x,y \in S^2_{B(i)}$ can cause MO to have an additional cost of 1. MO serves the requests to items $x \in S^2_{B(i)}$ such that the incurred cost is as small as possible. Among all strategies to serve those requests, MO could choose the following rule. Before it serves the requests in the current block, it could remove all inversions $\{x,y\}$ with $x,y \in S^2_{B(i)}$, thereby incurring a cost of $\Phi(t_{i-1}^e, S^2_{B(i)}, S^2_{B(i)}) = card(\{\{x,y\} \in I | x,y \in S^2_{B(i)}\})$, and could then maintain the items in $S^2_{B(i)}$ in the same order as OPT. In this case we would have

$$\sum_{x \in S^2_{B(i)}} C_{MO}(x,i,S^2_{B(i)}) \leq \sum_{x \in S^2_{B(i)}} C_{OPT}(x,i,S^2_{B(i)}) + card(\{\{x,y\} \in I | x,y \in S^2_{B(i)}\}).$$

If MO chooses a different rule, then its actual cost $\sum_{x \in S^2_{B(i)}} C_{MO}(x,i,S^2_{B(i)})$ must satisfy the above inequality, too. The inequality (12) must hold.

In the following we show how to amortize MO's extra cost caused by the inversions in I. We also amortize the change in potential $\Phi(t_i^e, S_{B(i)}, S_{B(i)}) - \Phi(t_{i-1}^e, S_{B(i)}, S_{B(i)})$. We sketch the main idea of the remaining part of the proof. Note that the sums in Claim 5 can be written as follows.

$$\sum_{x \in S_{B(i)}} C_{MO}(x,i,S_{B(i)}) = \sum_{x \in S_{B(i)}} \sum_{y \in S_{B(i)}} C_{MO}(x,i,\{y\}),$$

$$\sum_{x \in S_{B(i)}} C'_{OPT}(x, i, S_{B(i)}) = \sum_{x \in S_{B(i)}} \sum_{y \in S_{B(i)}} C'_{OPT}(x, i, \{y\}),$$

$$\sum_{x \in S_{B(i-1)} \cap S_{B(i)}} C'_{OPT}(x, i, S_{B(i-1)} \cap S_{B(i)}) = \sum_{x \in S_{B(i)} \cap S_{B(i)}} \sum_{y \in S_{B(i-1)} \cap S_{B(i)}} C'_{OPT}(x, i, \{y\})$$

and

$$\sum_{x \in S_{B(i)}} C''_{OPT}(x, i, S_{B(i)}) = \sum_{x \in S_{B(i)}} \sum_{y \in S_{B(i)}} C''_{OPT}(x, i, \{y\}).$$

In order to balance MO's extra cost and a possible increase in potential, we will charge additional costs to elements $C^j_{OPT}(x, i, \{y\})$ with $x, y \in S_{B(i)}$, $1 \leq j \leq f_{B(i)}(x)$ and $C^j_{OPT}(x, i, \{y\}) = 1$. If $j = 1$ and if $x \notin S_{B(i-1)}$ or $y \notin S_{B(i-1)}$, then we will charge an additional cost of at most 1. If $j > 1$ or if $x, y \in S_{B(i-1)}$, then we will charge a cost of at most $\frac{1}{2}$. In the same way we will balance the cost incurred by the term $\frac{1}{2} card(P_{B(i)})$ on the left-hand side of inequality (10). Some of the cost will also be cancelled by the negative terms $-\frac{1}{2} card(P_{B(i-1)}(S_{B(i)}))$ and $-\frac{1}{2} card(\{\{x, y\} | x \in S^1_{B(i)} \text{ and } y \in S^2_{B(i)}\})$. Note that if we balance the cost in the described manner, then the inequality (10) must be satisfied.

For simplicity we define

$$M = \{\{x, y\} | x \in S^1_{B(i)} \text{ and } y \in S^2_{B(i)}\}.$$

Consider any two items $x, y \in S_{B(i)}$.

If $x = y$ then $\{x\}$ is contained in $P_{B(i)}$ and causes a cost of $\frac{1}{2}$ on the left-hand side of inequality (10). Note that $C'_{MO}(x, i, \{x\}) = C'_{OPT}(x, i, \{x\}) = 1$ and we can charge a cost of $\frac{1}{2}$ to $C'_{OPT}(x, i, \{x\})$.

If $x \neq y$, then we distinguish between three main cases.

Case 1: $\{x, y\}$ represents an inversion at the beginning of the block.
This inversion may create an extra cost of 1 when MO processes the current block.

Case 1.1: $\{x, y\}$ represents no inversion at the end of the block.
In this case the extra cost of 1 cancels out with the decrease in potential. But the pair $\{x, y\}$ is now contained in $P_{B(i)}$. Note that $C'_{OPT}(x, i, \{y\}) + C'_{OPT}(y, i, \{x\}) \geq 1$. Hence we can charge a cost of $\frac{1}{2}$ to either $C'_{OPT}(x, i, \{y\})$ or $C'_{OPT}(y, i, \{x\})$ (depending on which value is 1).

Case 1.2: $\{x, y\}$ represents an inversion at the end of the block.
The inversion at the beginning of the block can only cause an additional cost of 1 if $x \in S^2_{B(i)}$ or $y \in S^2_{B(i)}$. Note that $C'_{OPT}(x, i, \{y\}) = 1$ or $C'_{OPT}(y, i, \{x\}) = 1$. If $x \notin S_{B(i-1)}$ or $y \notin S_{B(i-1)}$, we charge an additional cost of 1 to either $C'_{OPT}(x, i, \{y\})$ or $C'_{OPT}(y, i, \{x\})$. If $x \in S_{B(i-1)}$ and $y \in S_{B(i-1)}$, we use a slightly more complicated strategy. First suppose $x \in S^1_{B(i)}$ and $y \in S^2_{B(i)}$. In this case $\{x, y\} \in M$. Now $\frac{1}{2}$ of MO's extra cost is cancelled out by the term $-\frac{1}{2} card(M)$. In order to balance the other $\frac{1}{2}$ of MO's extra cost, we charge an additional cost of $\frac{1}{2}$ to either $C'_{OPT}(x, i, \{y\})$ or $C'_{OPT}(y, i, \{x\})$. Now suppose $x, y \in S^2_{B(i)}$. Without loss of generality we assume that the last request to item x in block $B(i)$ occurs later than the last request to item y in block $B(i)$. Hence, x precedes item y in MO's list at the end of the block. Since y precedes item x in OPT's list, we have $C''_{OPT}(x, i, \{y\}) \geq 1$. In order to balance MO's extra cost of 1, we charge a cost of $\frac{1}{2}$ to $C'_{OPT}(x, i, \{y\})$ or $C'_{OPT}(y, i, \{x\})$ and a cost of $\frac{1}{2}$ to $C''_{OPT}(x, i, \{y\})$.

Case 2: $\{x, y\}$ represents no inversion at the beginning of the block but immediately after MO has arranged the items in $S^1_{B(i)}$ and $S^2_{B(i)}$ at the front of the list.
In this case suppose $x \in S^1_{B(i)}$ and $y \in S^2_{B(i)}$. The new inversion can cause MO to have an extra cost of 1.

Case 2.1: $\{x, y\}$ represents no inversion at the end of the block.
If $x \notin S_{B(i-1)}$ or $y \notin S_{B(i-1)}$, we balance MO's additional cost by charging a cost of 1 to either $C'_{OPT}(x, i, \{y\})$ or $C'_{OPT}(y, i, \{x\})$. Note that $\{x, y\} \in M$ and $\{x, y\} \in P_{B(i)}$. Hence, $\{x, y\}$'s contributions in $-\frac{1}{2}card(M)$ and $\frac{1}{2}card(P_{B(i)})$ cancel out. If $x \in S_{B(i-1)}$ and $y \in S_{B(i-1)}$, then we only charge $\frac{1}{2}$ to either $C'_{OPT}(x, i, \{y\})$ or $C'_{OPT}(y, i, \{x\})$. The other $\frac{1}{2}$ of MO's additional cost is cancelled by $\{x, y\}$'s contribution in $-\frac{1}{2}card(P_{B(i-1)}(S_{B(i)}))$. Again, $\{x, y\}$'s contributions in $-\frac{1}{2}card(M)$ and $\frac{1}{2}card(P_{B(i)})$ cancel out.

Case 2.2: $\{x, y\}$ represents an inversion at the end of the block.
We have to balance MO's extra cost of 1 and the increase in potential of 1. Note that $C'_{OPT}(y, i, \{x\}) = 1$. Suppose the request to x occurs later than the last request to y in the current block. Then x precedes item y in MO's list at the end of the block. Since $\{x, y\}$ represents an inversion we have $C'_{OPT}(x, i, \{y\}) = 1$. If the last request to y occurs later than the request to x, then item y precedes item x in MO's list at the end of the block. Since $\{x, y\}$ represents an inversion at the end of the block we have $C''_{OPT}(y, i, \{x\}) \geq 1$. Suppose $x \notin S_{B(i-1)}$ or $y \notin S_{B(i-1)}$. Then we balance MO's extra cost of 1 by charging an additional cost of 1 to $C'_{OPT}(y, i, \{x\})$. Furthermore, $\frac{1}{2}$ of the increase in potential is balanced by charging a cost of $\frac{1}{2}$ to either $C'_{OPT}(x, i, \{y\})$ or $C''_{OPT}(y, i, \{x\})$. The other $\frac{1}{2}$ of the increase in potential is cancelled by $\{x, y\}$'s contribution in $-\frac{1}{2}card(M)$. If $x \in S_{B(i-1)}$ and $y \in S_{B(i-1)}$, we balance MO's additional cost and the increase in potential in almost the same way as before. The difference is that we only charge a cost of $\frac{1}{2}$ to $C'_{OPT}(y, i, \{x\})$. The other $\frac{1}{2}$ of the extra cost caused by the inversion is cancelled out by $\{x, y\}$'s contribution in $\frac{1}{2}card(P_{B(i-1)}(S_{B(i)}))$.

Case 3: $\{x, y\}$ represents an inversion neither at the beginning of the block nor after MO has arranged the items in $S^1_{B(i)}$ and $S^2_{B(i)}$ at the front of the list.

Case 3.1: $\{x, y\}$ represents no inversion at the end of the block.
In this case $\{x, y\}$'s contribution in $card(P_{B(i)})$ is balanced by charging a cost of $\frac{1}{2}$ to either $C'_{OPT}(x, i, \{y\})$ or $C'_{OPT}(y, i, \{x\})$.

Case 3.2: $\{x, y\}$ represents an inversion at the end of the block.
If $x \notin S_{B(i-1)}$ or $y \notin S_{B(i-1)}$, we balance the increase in potential by charging an additional cost of 1 to either $C'_{OPT}(x, i, \{y\})$ or $C'_{OPT}(y, i, \{x\})$. If $x \in S_{B(i-1)}$ and $y \in S_{B(i-1)}$, we only charge a cost of $\frac{1}{2}$ to $C'_{OPT}(x, i, \{y\})$ or $C'_{OPT}(y, i, \{x\})$, because $\frac{1}{2}$ of the increase in potential is cancelled out by $\{x, y\}$ contribution in $-\frac{1}{2}card(P_{B(i-1)}(S_{B(i)}))$.

The proof of Claim 5 is complete. □

So far, we have shown part b) of the lemma. It is an easy exercise to modify the above proof in such a way that we can prove the inequalities of part a) for all $1 \leq l \leq n-1$. Claim 4 remains the same. Claim 5 changes to the following statement.

Claim 6

$$\sum_{x \in S_{B(i)}} C_{MO}(x, i, S_{B(i)}) + \Phi(t_i^e, S_{B(i)}, S_{B(i)}) - \Phi(t_{i-1}^e, S_{B(i)}, S_{B(i)}) \leq 2 \sum_{x \in S_{B(i)}} C_{OPT}(x, i, S_{B(i)})$$

This claim can be proved in a similar way as Claim 5. However, the proof becomes much simpler, since we can neglect the cases $x, y \in S_{B(i-1)}$ and $\{x, y\} \in P_{B(i)}$. Adding the inequalities of Claim 4 and Claim 6 we obtain part a) of the lemma. □

We finish the proof of Theorem 16 by adding one remark. In the beginning of the proof we assumed that the last block $B(b)$ contains $l+1$ items. If the last block contains requests to less than $l+1$ items, then MO's cost on that block is at most $\frac{1}{2}(n-1)n$ higher than OPT's cost. (The value $\frac{1}{2}(n-1)n$ is the number of possible inversions in MO's list.) Hence the assumption represents no restriction when evaluating MO's competitiveness. This completes the proof of Theorem 16.

We conclude this section by presenting another lower bound for list update with lookahead. Consider an on-line algorithm A with strong lookahead $1 \leq l \leq n-1$ which serves the request sequence σ in a series of blocks. The algorithm partitions σ into a sequence of blocks $B(1), B(2), \ldots, B(b)$ where $B(1) = \sigma(1), \sigma(2), \ldots, \sigma(\lambda(1))$ and $B(i) = \sigma(t_{i-1}^e + 1), \sigma(t_{i-1}^e + 2), \ldots, \sigma(\lambda(t_{i-1}^e + 1))$ for $i \geq 2$. Again, t_{i-1}^e denotes the end of block $B(i-1)$. While A serves requests in block $B(i)$, it does not use information on requests in block $B(j)$, where $j > i$. In the beginning of this subsection we claimed that A cannot be better than 1.5-competitive if $l = n-1$. Now we prove this statement.

We let an adversary generate a request sequence. The adversary has to serve the request sequence, too. The request sequence consists of a series of blocks $B(1), B(2), \ldots$, where each block contains n distinct requests. Let L_1 be the list given initially. First we show how $B(1)$ and $B(2)$ are constructed. The adversary generates the first block $B(1)$ by scanning the list L_1 from the front; each item is requested exactly once. Let L_2^A be the configuration of the list after A has served $B(1)$, and let L_2^{ADV} be the configuration of the list after the adversary has served $B(1)$. While processing $B(1)$, the adversary rearranges the items in its list such that L_2^{ADV} is in the exact reverse order of L_2^A. Since $B(1)$ requests the items in L_1 in increasing order, this rearrangement can be accomplished using free exchanges only. The second block $B(2)$ is generated by scanning the list L_2^{ADV} from the front. The first $n-1$ items are requested twice, whereas the last item is requested only once. While processing the requests in $B(2)$, the adversary moves an item to the front of its list each time it is accessed. The following blocks $B(i)$ with $i \geq 3$ are constructed in the same way as $B(1)$ and $B(2)$. Let L_i^A be the configuration of the list after A has served $B(i-1)$, and let L_i^{ADV} be the configuration of the list after the adversary has served $B(i-1)$. Based on L_i^{ADV}, each odd numbered block is generated in the same way as $B(1)$, and each even numbered block is constructed in the same way as $B(2)$.

In any two successive blocks, the adversary incurs a cost of

$$2 \cdot \frac{1}{2} \cdot n(n+1) + n - 1.$$

Obviously, in any odd numbered block the algorithm A incurs a cost of at least $\frac{1}{2}n(n+1)$. It is

not hard to see that in any even numbered block A has a cost of at least

$$n^2 + n - 1.$$

This implies that if A is c-competitive, then

$$\begin{aligned} c &\geq \frac{\frac{1}{2}n(n+1) + n^2 + n - 1}{n(n+1) + n - 1} \\ &= \frac{\frac{3}{2}n^2 + \frac{3}{2}n - 1}{n^2 + 2n - 1} \end{aligned}$$

This fraction converges to 1.5 as n tends to infinity.

6 The limits of strong lookahead

In the previous sections on paging and the list update problem we have shown that strong lookahead can be a powerful means to improve the competitive factors of on-line algorithms without lookahead. In particular, on-line algorithms with strong lookahead can perform much better than on-line algorithms with weak lookahead, for which the knowledge of future requests is only of minor advantage when an adversary replicates requests in the lookahead.

In this section we prove that these statements do not hold for more general on-line problems such as caching, the k-server problem and metrical task systems. Except for a few special cases, strong lookahead is basically not more powerful than weak lookahead and cannot reduce the competitive factors of deterministic on-line algorithms without lookahead. The reason is that in these more general problems, an adversary can simulate the effect of replication by posing very inexpensive requests in the lookahead.

Manasse *et al.* [MMS88] have demonstrated that no deterministic on-line algorithm for the symmetric k-server problem can be better than k-competitive. The same lower bound holds for caching problems. They have also conjectured that for every k there exists a k-competitive on-line k-server algorithm. This conjecture is still open; see for instance [G91] for a comprehensive summary of the latest results on the k-server problem. We show that no deterministic on-line caching or k-server algorithm with strong or weak lookahead can generally be better than k-competitive. Thus lookahead is of no advantage. This negative result carries over to metrical task systems. Borodin *et al.* [BLS87] have presented a lower bound and a matching upper bound of $(2n-1)$ on the competitive factor of any deterministic on-line algorithm for metrical task systems. We show that these bounds cannot be improved using strong or weak lookahead.

It is an easy exercise to show that weak lookahead cannot reduce the competitive factors of deterministic on-line algorithms for caching, the k-server problem and metrical task systems. In the following we prove theorems for strong lookahead. When discussing caching problems, we use the terminology of the k-server problem.

Theorem 17 *Let s be a non-negative integer and let $0 < \epsilon < 1$. There exists a graph consisting of $k + s + 1$ vertices, for which no deterministic on-line caching algorithm with strong lookahead $l \leq k + s - 1$ can achieve a competitive factor of less than $\min\{k, (k - l + s)\} - \epsilon$.*

Proof: Let $S = X \cup Y$ be the set of vertices of the graph. $X = \{x_1, x_2, \ldots, x_k\}$ where $w(x_i) = 1$ for $i = 1, 2, \ldots, k$, and $Y = \{y_1, y_2, \ldots, y_{s+1}\}$ where $w(y_i) = \delta$ for $i = 1, 2, \ldots, s+1$. Here δ is a positive real. The weights $w(x_i)$ and $w(y_i)$ denote the cost of moving a server to vertex x_i or y_i, i.e. the weights equal the cost that is incurred when loading an item x_i or y_i into the cache.

let A be an on-line caching algorithm with strong lookahead $l \leq k + s - 1$. We assume that A's and OPT's servers initially cover vertices $x_1, x_2, \ldots, x_k$. The request sequence which we are constructing consists of a series of phases, each of which contains $l + 2$ requests to at least $l + 1$ distinct vertices. Each phase has the following form. If $s \geq l$, then the first $l + 1$ requests equal $y_1, y_2, \ldots, y_{l+1}$; otherwise, if $s < l$, then the first $l + 1$ requests equal $y_1, y_2, \ldots, y_{s+1}, x_1, x_2, \ldots, x_{l-s}$. The $(l + 2)$nd request is made to a vertex in X that is vacated by A after the first request of the current phase.

During each phase, the on-line algorithm A incurs a cost of at least 1. We estimate OPT's cost. OPT's strategy is to cover always at least $k - 1$ vertices in X. In particular, OPT always covers $x_1, x_2, \ldots, x_{l-s}$, if $s < l$. If there occurs a fault on a request to a vertex in X, then OPT vacates a vertex in Y and moves the corresponding server to the request. OPT then chooses a server covering a vertex that will not be requested during the next $(k - 1)$ phases (if $s \geq l$) or the next $k - l + s - 1$ phases (if $s < l$). During these phases, all requests to vertices in Y will be served with that server.

We conclude that if $s \geq l$, then A's competitive factor cannot be less than

$$c = \frac{k}{1 + k(l+1)\delta}.$$

If $s < l$, then A's competitive factor cannot be less than

$$c = \frac{(k - l + s)}{1 + (k - l + s)(l+1)\delta}.$$

Choosing δ sufficiently small, we obtain the theorem. □

Caching problems are examples of asymmetric k-server problems in which the edges into a particular vertex all have the same length. Raghavan and Snir [RS89] have observed that these problems can be converted into instances of the symmetric k-server problem. Therefore, an immediate consequence of the above theorem is

Corollary 1 *A statement analogous to Theorem 17 also holds for the symmetric k-server problem.*

The two above theorems imply that a strong lookahead of size l can only reduce the competitive factors of on-line caching and k-server algorithms, if the given graph contains less than $(k + l + 1)$ vertices. Thus, in graphs consisting of a potentially infinite number of vertices, lookahead is of no advantage.

Now we turn to metrical task systems.

Theorem 18 *Let A be a deterministic on-line algorithm with strong lookahead l for solving scheduling problems in metrical task systems. Then A's competitive factor cannot be less than* $(2n - 1)$.

Proof: The request sequence we will present is an augmentation of the request sequence given by Borodin *et al.* [BLS87]. In the following we briefly sketch their proof.

Borodin *et al.* construct an infinite request sequence $\sigma = T^1, T^2, T^3, \ldots$, where T^t is the requested task at time t. Let $s(t)$ denote the state of the metrical task system at time t, and let $\epsilon > 0$ be a real number. Task T^t is defined as

$$\begin{aligned} T^t(s(t-1)) &= \epsilon \\ \text{and} \qquad T^t(s) &= 0 \text{ for all } s \neq s(t-1). \end{aligned}$$

Borodin *et al.* show that on this request sequence any deterministic on-line algorithm incurs a cost of at least c times the optimal cost, where

$$c = (2n-1)/(1 + \frac{\epsilon}{\min_{i \neq j} d(i,j)}). \tag{13}$$

Choosing ϵ arbitrarily small, they obtain the desired lower bound. The equation (13) is derived as follows.

Let $s_1, s_2, s_3, \ldots$ and $t_1 < t_2 < t_3 < \ldots$ denote the state transitions and transition times prescribed by the on-line algorithm. At times $t'_k = t_k + 1$ the adversary changes tasks. Let $C_A(k)$ be the cost incurred by the on-line algorithm during the interval $[1, t'_k)$, and let $C_{OPT}(k)$ be the cost incurred by optimal algorithm during the interval $[1, t'_k]$. Furthermore, let $C_{OPT}(k, s)$ denote the cost incurred by OPT during the interval $[1, t'_k]$, given that the task system is in state s at time t'_k.

Borodin *et al.* prove the following equations and inequalities.

1. $$C_A(k) = D(k) + P(k),$$

   where $D(k) = \sum_{i=1}^{k} d(s_{i-1}, s_i)$ is the cost incurred by A for state transition and $P(k) = \epsilon \cdot (t_k - k)$ is the task processing cost during $[1, t'_k)$.

2. $$\epsilon \cdot t_k \leq D(k) \frac{\epsilon}{\min_{i \neq j} d(i,j)} + P(k)$$

3. For all $k \geq 1$,

   $$2(\sum_{s \neq s_k} C_{OPT}(k,s)) + C_{OPT}(k, s_k) \leq D(k) + \epsilon \cdot t_k + (2n-2) \max d(i,j)$$

   and

   $$\min_{s \in S} C_{OPT}(k,s) \leq \frac{1}{2n-1} \cdot (1 + \frac{\epsilon}{\min_{i \neq j} d(i,j)}) \cdot (D(k) + P(k) + (2n-2) \max d(i,j)).$$

Then, they conclude

$$\begin{aligned} \frac{C_A(k)}{C_{OPT}(k)} &= \frac{C_A(k)}{\min_{s \in S} C_{OPT}(k,s)} \\ &\geq (2n-1) \cdot (\frac{1}{1 + \epsilon / \min_{i \neq j} d(i,j)}) \cdot (\frac{D(k) + P(k)}{D(k) + P(k) + (2n-2) \max d(i,j)}). \end{aligned}$$

Taking the lim sup as k tends to infinity, they obtain the desired lower bound of

$$c = (2n-1) \cdot (1 + \frac{\epsilon}{\min_{i \neq j} d(i,j)}).$$

Our modified request sequence for on-line algorithms with strong lookahead is as follows. We assume, without loss of generality, that $\min_{i \neq j} d(i,j) > 0$. Let $0 < \epsilon \leq \min_{i \neq j} d(i,j)$. After each task T^t we insert a block of l different lookahead tasks $T_1, T_2, \ldots, T_l$, where

$$T_i(s) = \frac{1}{K(2n-1)} \cdot (\frac{1}{2})^i \cdot \epsilon \quad \text{for all states } s.$$

$K > 1$ is an arbitrary constant. These tasks cause the lookahead to be of no advantage, since it makes no difference in which state they are processed.

When analyzing our request sequence, we assume for simplicity that a complete block of tasks

$$T^t, T_1, T_2, \ldots, T_l$$

is processed at time t. Furthermore, we assume without loss of generality that the task system only changes states immediately preceding a non-lookahead task T^t. We now proceed along the lines of the proof by Borodin *et al.* Let C_l denote the cost incurred in processing a block of lookahead tasks $T_1, T_2, \ldots, T_l$. Note that $C_l < \epsilon/(K(2n-1))$.

It is easy to show that the following equations and inequalities hold.

1.
$$C_A(k) = D(k) + P(k),$$
where $D(k) = \sum_{i=1}^{k} d(s_{i-1}, s_i)$ and $P(k) = \epsilon \cdot (t_k - k) + C_l t_k$.

2.
$$\epsilon \cdot t_k \leq D(k) \frac{\epsilon}{\min_{i \neq j} d(i,j)} + P(k) - C_l t_k$$

3. For all $k \geq 1$,

$$2(\sum_{s \neq s_k} C_{OPT}(k,s)) + C_{OPT}(k,s_k) \leq D(k) + \epsilon \cdot t_k + (2n-1) C_l t'_k + (2n-2) \max d(i,j))$$

and

$$\begin{aligned} \min_{s \in S} C_{OPT}(k,s) \quad \leq \quad & \frac{1}{2n-1} \cdot (1 + \frac{\epsilon}{\min_{i \neq j} d(i,j)}) \cdot (D(k) + P(k) + \\ & (2n-1) C_l t'_k + (2n-2) \max d(i,j)). \end{aligned}$$

It follows that A cannot be better than c-competitive, where

$$\begin{aligned} c &= \frac{C_A(k)}{C_{OPT}(k)} \\ &= \frac{C_A(k)}{\min_{s \in S} C_{OPT}(k,s)} \\ &\geq (2n-1) \cdot (\frac{1}{1 + \epsilon / \min_{i \neq j} d(i,j)}) \cdot (\frac{D(k) + P(k)}{D(k) + P(k) + (2n-1) C_l t'_k + (2n-2) \max d(i,j)}). \end{aligned}$$

Note that $D(k) + P(k) \geq \epsilon \cdot t_k$ and that $(2n-1)C_l t'_k \leq (1/K)(t_k + 1) \cdot \epsilon$. Taking the lim sup as k tend to infinity, we obtain

$$c \geq (2n-1) \cdot \left(\frac{1}{1 + \epsilon / \min_{i \neq j} d(i,j)}\right) \cdot \left(\frac{1}{1 + 1/K}\right).$$

Choosing ϵ arbitrarily small and K arbitrarily large, we obtain the theorem. □

7 Open problems

One open problem is to prove that the proposed variant of the algorithm MTF/OPT(l) is in fact $(2 - \frac{1}{2} \cdot \frac{l+2}{n+1})$-competitive if a strong lookahead $1 \leq l \leq n-1$ is given, or to present another competitive on-line algorithm with lookahead for the list update problem.

Another important problem is to develop general properties of on-line alorithms with lookahead. We have the impression that this problem is quite difficult because the on-line problems considered in this paper behave completely differently when lookahead is added. One approach might be to study metrical task systems, which generalize a large class of on-line problems, under an even stronger model of lookahead. Borodin *et al.* [BIRS91] have considered paging problems with locality of reference. They have introduced the access graph model: each time a page is requested, the next page to be requested comes from a restricted set that is determined by the access graph. The access graph can be viewed as some kind of lookahead. A stronger lookahead for metrical task systems can be constructed by combining our model of strong lookahead with the idea of locality of reference.

Acknowledgement

The author thanks Kurt Mehlhorn for useful discussions and comments.

References

[B66] L.A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, **5**:78-101, 1966.

[BBKTW90] S. Ben-David, A. Borodin, R.M. Karp, G. Tárdos and A. Wigderson. On the power of randomization in on-line algorithms. In *Proc. 22nd Annual ACM Symposium on Theory of Computing*, pages 379-386, 1990. To appear in *Algorithmica*.

[BM85] J.L. Bentley and C.C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Communications of the ACM*, **28**(4):404-411, 1985.

[BIRS91] A. Borodin, S. Irani, P. Raghavan and B. Schieber. Competitive paging with locality of reference. In *Proc. 23rd Annual ACM Symposium on Theory of Computing*, pages 249-259, 1991.

[BLS87] A. Borodin, N. Linial and M. Saks. An optimal online algorithm for metrical task systems. In *Proc. 19th Annual ACM Symposium on Theory of Computing*, pages 373-382, 1987.

[CGS89] F.K. Chung, R. Graham and M.E. Saks. A dynamic location problem for graphs. *Combinatorica*, **9**(2):111-131, 1989.

[DMN91] F. d'Amore, A. Marchetti-Spaccamela and U. Nanni. Competitive algorithms for the weighted list update problem. Technical Report, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1991.

[FKLMSY91] A. Fiat, R.M. Karp, L.A. McGeoch, D.D. Sleator and N.E. Young. Competitive paging algorithms. *Journal of Algorithms*, **12**:685-699, 1991.

[G91] E. Grove. The harmonic online k-server algorithm is competitive. In *Proc. 23nd Symposium on Theory of Computing*, pages 260-266, 1991.

[HS92] M.M. Halldórsson and M. Szegedy. Lower bounds for on-line graph coloring. In *Proc. 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 211-216, 1992.

[I90] S. Irani. Coloring inductive graphs on-line. In *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 470-479, 1990.

[I91] S. Irani. *Competitive Algorithms for On-Line Paging and Graph Coloring.* Ph.D. thesis, University of California, Berkeley, 1991.

[IRWS91] S. Irani, N. Reingold, J. Westbrook and D.D. Sleator. Randomized competitive algorithms for the list update problem. In *Proc. 2nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 251-260, 1991.

[KT91] M.-Y. Kao and S.R. Tate. Online matching with blocked input. *Information Processing Letters*, **38**:113-116, May 1991.

[KMRS88] A.R. Karlin, M.S. Manasse, L. Rudolph and D.D. Sleator. Competitive snoopy caching. *Algorithmica*, **3**(1):79-119, 1988.

[KR90] R. Karp and P. Raghavan. Personal communication, transmitted through [IRWS91].

[MMS88] M.S. Manasse, L.A. McGeoch and D.D. Sleator. Competitive algorithms for on-line problems. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 322-333, 1988.

[MMS90] M.S. Manasse, L.A. McGeoch and D.D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, **11**:208-230, 1990.

[MS91] L.A. McGeoch and D.D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, **6**:816-825, 1991.

[R89] P. Raghavan. Lecture notes on randomized algorithms. IBM Research Report No. RC 15340 (# 68237) , Yorktown Heights, 1989.

[RS89] P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. In *Proc. 16th International Colloquium on Automata, Languages and Programming*, Springer Lecture Notes in Computer Science, Vol. 372, pages 687-703, 1989.

[RW90] N. Reingold and J. Westbrook. Optimum off-line algorithms for the list update problem. Technical Report YALEU/DCS/TR-805, August 1990.

[R76] R. Rivest. On self-organizing sequential search heuristics. *Communications of the ACM*, **19**(2):63-67, 1976.

[ST85] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communication of the ACM*, **28**:202-208, 1985.

[S77] J.R. Spirn. *Programm Behavior: Models and Measurements.* Elsevier, New York, 1977.

[Y77] A.C-C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 17th Annual IEEE Symposium on Foundations of Computer Science*, pages 222-227, 1977.

Appendix

Proof of Theorem 9: Part a): For a fixed $2 \leq h \leq k-1$ consider $(n_1, n_2, \ldots, n_h)$. Note that

$$\frac{n_h}{k - \sum_{j=1}^{h-1} n_j} \leq 1.$$

Thus

$$f_{\max}(h) - f_{\max}(h-1) \leq f_{\max}(h) - f_{h-1}(n_1, n_2, \ldots, n_{h-2}, n_{h-1} + n_h) \leq 1.$$

Part b): Let $1 \leq h \leq k-1$. First we prove the inequality

$$f_{\max}(h) \leq h(1 - \frac{1}{k^{1/h}}). \tag{14}$$

Let $(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_h) \in \mathbb{R}^h$ be the h-tupel which maximizes the fuction

$$f_h(x_1, x_2, \ldots, x_h) = \sum_{i=1}^{h} \frac{x_i}{k - \sum_{j=1}^{i-1} x_j}$$

subject to the constraints

$$\sum_{i=1}^{h} x_i - k + 1 = 0$$

$$x_i \text{ is a non-negative real for } i = 1, 2, \ldots, h.$$

In the following we determine $(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_h)$ and show $f(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_h) = h(1 - \frac{1}{k^{1/h}})$. This proves inequality (14).

Define $g(x_1, x_2, \ldots, x_h) = \sum_{i=1}^{h} x_i - k + 1$. The Lagrange multiplier rule implies the existence of a constant λ such that

$$\frac{\partial f}{\partial x_i}(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_h) - \lambda \frac{\partial g}{\partial x_i}(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_h) = 0$$

for $i = 1, 2, \ldots, h$. Thus, the h-tupel $(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_h)$ satisfies the following equations:

$$\begin{aligned}
\frac{1}{k} + \sum_{j=2}^{h} \frac{\overline{x}_j}{(k - \sum_{\nu=1}^{j-1} \overline{x}_\nu)^2} - \lambda &= 0 \\
\frac{1}{k - \sum_{\nu=1}^{i-1} \overline{x}_\nu} + \sum_{j=i+1}^{h} \frac{\overline{x}_j}{(k - \sum_{\nu=1}^{j-1} \overline{x}_\nu)^2} - \lambda &= 0 \quad \text{for } i = 2, 3, \ldots, h-1 \\
\frac{1}{k - \sum_{\nu=1}^{h-1} \overline{x}_\nu} - \lambda &= 0
\end{aligned}$$

Hence $\lambda = 1/(k - \sum_{\nu=1}^{h-1} \overline{x}_\nu)$. Furthermore, subtracting the $(i+1)$st equation from the ith equation we obtain

$$\overline{x}_i = k - \sum_{\nu=1}^{i-1} \overline{x}_\nu - \frac{(k - \sum_{\nu=1}^{i-1} \overline{x}_\nu)^2}{k - \sum_{\nu=1}^{i-2} \overline{x}_\nu}$$

for $i = 2, 3, \ldots, h$.

This implies

$$\overline{x}_i = \frac{(k-\overline{x}_1)^{i-1}}{k^{i-2}} - \frac{(k-\overline{x}_1)^i}{k^{i-1}}$$

for $i = 2, 3, \ldots, h$. Since

$$k - 1 - \overline{x}_1 = \sum_{i=2}^{h} \overline{x}_i = k - \overline{x}_1 - \frac{(k-\overline{x}_1)^h}{k^{h-1}}$$

we have

$$\overline{x}_1 = k - k^{\frac{h-1}{h}}.$$

Thus

$$\overline{x}_i = k^{\frac{h-(i-1)}{h}} - k^{\frac{h-i}{h}}$$

for $i = 1, 2, \ldots, h$, and

$$f_h(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_h) = h(1 - \frac{1}{k^{1/h}}).$$

Obviously, $\overline{x}_i \geq 0$ for all $i = 1, 2, \ldots, h$.

We now prove the second inequality

$$h(1 - \frac{1}{k^{1/h}}) - (1 - \frac{1}{k}) < f_{\max}(h). \tag{15}$$

Let $(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_h)$ be the h-tupel computed above. We show how to construct an h-tupel $(\overline{n}_1, \overline{n}_2, \ldots, \overline{n}_h)$, where each $\overline{n}_i$ is a non-negative integer, such that

$$\sum_{i=1}^{h} \overline{n}_i - k + 1 = 0$$

and

$$f_h(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_h) - f_h(\overline{n}_1, \overline{n}_2, \ldots, \overline{n}_h) < 1 - \frac{1}{k}.$$

This implies inequality (15).

The h-tupel $(\overline{n}_1, \overline{n}_2, \ldots, \overline{n}_h)$ is constructed as follows. Set $r_{h+1} = 0$. Then for $i = h, h-1, \ldots, 2, 1$ set

$$\begin{aligned} \overline{n}_i &= \lfloor \overline{x}_i + r_{i+1} \rfloor \\ r_i &= \overline{x}_i + r_{i+1} - \overline{n}_i. \end{aligned}$$

Roughly speaking, starting with $\overline{x}_h$, we round each component in $(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_h)$ down and add the difference to $\overline{x}_{i-1}$. Note that $r_1 = 0$, which implies $\sum_{i=1}^{h} \overline{n}_i - k + 1 = 0$.

For $i = 1, 2, \ldots, h$, define

$$f_h^i = f_h(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_i + r_{i+1}, \overline{n}_{i+1}, \overline{n}_{i+2}, \ldots, \overline{n}_h).$$

Then, for $i = 2, 3, \ldots, h$, we have

$$f_h^i - f_h^{i-1} \leq \frac{r_i}{k - \sum_{j=1}^{i-1} \overline{x}_j} - \frac{r_i}{k - \sum_{j=1}^{i-2} \overline{x}_j} < \frac{1}{k - \sum_{j=1}^{i-1} \overline{x}_j} - \frac{1}{k - \sum_{j=1}^{i-2} \overline{x}_j}.$$

Summing up these inequalities for $i = 2, 3, \ldots, h$ we obtain

$$\begin{aligned} f_h(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_h) - f_h(\overline{n}_1, \overline{n}_2, \ldots, \overline{n}_h) &= f_h^h - f_h^1 \\ &< \frac{1}{k - \sum_{j=1}^{h-1} \overline{x}_j} - \frac{1}{k} \\ &\leq 1 - \frac{1}{k}. \end{aligned}$$

This completes the proof of part b) of the theorem. □