Acta Cybernetica 14 (1999) 217-227.

On the Partitioning Algorithm

Béla Csaba *

Abstract

We consider the deterministic and the randomized paging problem. We show the close connection between the partitioning algorithm of McGeoch and Sleator and the *OPT* graph of the problem via a natural framework. This allows us to prove some important properties of the "deterministic" partitioning algorithm. As a consequence of these we prove, that it is a k-competitive deterministic on-line algorithm. Besides, we show an application of the *OPT* graph for a special case of the k-server problem.

1 Introduction

The *paging problem* is defined as follows. We have a two-level memory system with k pages of fast memory, and $n-k$ pages of slow memory. Repeatedly a *request* to a page appears. This request should be satisfied by moving the page to fast memory, if it is in slow memory, i.e., a page fault occurs. In this case a page has to be evicted from fast memory to make room for the new, recently requested one. The paging problem is to decide which page is to be evicted. The cost of a *request sequence* is the number of page faults. Of course, this number depends on the strategy used when deciding which page to evict.

There is a simple optimum paging algorithm, Belady's *MIN* algorithm (see [B]), if one knows the whole request sequence in advance, in the *off-line* case. It is more practical to consider the *on-line* paging problem, when the algorithm has to decide immediately after a page request, without knowing what the future requests will be.

Paging is a special case of the so called k-server problem. In this problem we are given a *finite metric space*: a set V on n points, and a distance function $d(x,y)$ on V^2 satisfying the usual requirements of distances (non-negativity, simmetry, triangle-inequality). There are k $(1 < k < n)$ mobile servers, initially residing on some points of V, no two on the same point. A number of requests, each is an element of V, appears. A request has to be satisfied immediately by moving a server to the requested point, if there is no server on it. By moving a server from the point y to the requested point x a cost, $d(x,y)$ incurs. The total cost of a

*Department of Computer Science; Rutgers, The State University of NJ; 110 Frelinghuysen Road; Piscataway, NJ 08854-8019 USA. Email: bcsaba@paul.rutgers.edu

request sequence is the sum of the costs of the composing requests. One can see, that by choosing $d \equiv 1$ (*uniform metric space*), we arrive to paging. The number of faults done by some algorithm on a request sequence σ is the same as the total distance taken by the servers during the satisfaction of σ. For the case of simplicity we use the terminology of the uniform k-server problem throughout the paper. An important remark is that it is enough to consider *lazy* algorithms, i.e., algorithms, which move a server only to a requested point (see [MMS]).

For comparing two paging algorithms the *competitive ratio* is used. This measure of performance of an on-line algorithm was introduced by Sleator and Tarjan (see [ST]). Fix any starting configuration of the pages, and denote by $opt(\sigma)$ the optimal cost of the satisfaction of the request sequence σ. The competitive ratio of the on-line algorithm $\mathcal{A}$ is c, if there is a constant M such that on every request sequence σ the cost incurred by $\mathcal{A}$, $\mathcal{A}(\sigma)$ is at most $c \cdot opt(\sigma) + M$. It was shown (see [ST]) that no on-line algorithm can have a competitive ratio less than k for the paging problem. $LRU, FIFO$ and a large number of other on-line algorithms are known to be k-competitive. On the other hand, the best competitive ratio achieved by some on-line algorithm for the k-server problem is $2k-1$ (see [KP]), while the lower bound for any metric space is k (see [MMS]), like in paging.

As it happens frequently, one may expect a better performance in the randomized case. A randomized on-line algorithm $\mathcal{R}$ is c-competitive, if there is a constant M such that on every request sequence σ, $E[\mathcal{R}(\sigma)]$ is at most $c \cdot opt(\sigma) + M$, where $E[\mathcal{R}(\sigma)]$ denotes the expected cost incurred by $\mathcal{R}$ on σ. It was proved (see [FKLMSY]) that $H_k = 3D1 + \frac{1}{2} + \cdots + \frac{1}{k}$ is a lower bound for the randomized competitiveness of an on-line paging algorithm. There is a simple, elegant algorithm, which has randomized competitive ratio $2H_k$ (see [FKLMSY]). On the other hand, the only known optimal randomized algorithm, the *partitioning algorithm* has a much more complicated description. Our goal is to show that despite it's complexity, the "deterministic" version of this algorithm is based on rather plausible ideas. This is done by the help of OPT graphs.

OPT graphs for on-line problems were first considered in [CL1], [CL2] when investigating the deterministic k-server problem, and in [LR] were used to analyze the randomized case. Roughly speaking, the OPT graph for the k-server problem is a finite directed graph, with which one can easily compute the optimal cost of any request sequence, and find the corresponding satisfaction.

The outline of the paper is as follows. In the second section we give the definition of the partitioning algorithm, and our framework, which is based on the OPT graph of the problem. The third section deals with the deterministic paging problem, and the connection between the OPT graph and the partitioning algorithm, and we show that the partitioning algorithm is k-competitive in the deterministic case. In the fourth section we give an application of our results for a special k-server problem, achieving $2k-2$ competitive ratio.

2 Definitions

First we give the definition of the partitioning algorithm following [MS]. The algorithm dynamically maintains a partition of the points: $V = S_\alpha \bigcup S_{\alpha+1} \bigcup \ldots \bigcup S_\beta$. Each set S_i $(i < \beta)$ is labeled with an integer k_i. Initially $\alpha = 1$ and $\beta = 2$,S_2 contains the k points that are covered by the servers, S_1 contains the unoccupied points, and $k_1 = 0$. In response to a request at a point $r \in V$ the labeled partition is updated. Let $r \in S_i$, then there are three cases.

Rule 1: $i = \beta$
Do nothing.

Rule 2: $\alpha < i < \beta$

First do the following assignements:

- $S_i \leftarrow S_i - r$
- $S_\beta \leftarrow S_\beta + r$
- $k_j \leftarrow k_j - 1$, $i \leq j < \beta$.

If some label changed from 1 to 0, find the largest number j such that $k_j = 0$. Then let

- $S_j \leftarrow S_\alpha \bigcup S_{\alpha+1} \bigcup \ldots \bigcup S_j$
- $\alpha \leftarrow j$.

Rule 3: $i = \alpha$

Do the following assignements:

- $S_\alpha \leftarrow S_\alpha - r$
- $S_{\beta+1} \leftarrow r$
- $k_\beta \leftarrow k - 1$
- $\beta \leftarrow \beta + 1$.

By induction one can easily show that the following *labeling invariant conditions* always hold:

- $k_\alpha = 0$
- $k_i > 0$ $(\alpha < i < \beta)$
- $k_i = k_{i-1} + |S_i| - 1$ $(\alpha < i < \beta)$
- $k_{\beta-1} = k - |S_\beta|$.

Now we are ready to give the partitioning algorithm itself. Denote $S_\alpha \bigcup S_{\alpha+1} \bigcup \ldots \bigcup S_i$ by S_i^*. For each set S_i^*, where $\alpha < i < \beta$, there are k_i *i-marks* occupying different points of S_i^*. An i-mark is only allowed on the points of S_i or on points with an $(i-1)$-mark - on the *i-eligible* points. The algorithm keeps a server on each element of S_β and on each point with a $(\beta-1)$-mark. By the labeling invariant conditions this means exactly k points covered by some server. Before the first request $\alpha = 1$ and $\beta = 2$, and there are no marks of any kind. Now, let the new request be r.

(1) If $r \in S_\beta$, then do nothing.

(2) If $r \in S_i\,(\alpha < i < \beta)$, move the marks around so as to achieve that there is a j-mark on r for all $j\,(i \leq j < \beta)$ in the following way. If r has a j-mark, then do nothing. Otherwise randomly choose some point w that has a j-mark. Transfer each l-mark $(l \geq j)$ from w to r. Repeat this step, if r does not have all marks up to $\beta - 1$. Then apply **Rule 2**, and erase all the marks on r. If α changes, all t-marks $(t \leq \alpha)$ are erased. If a $(\beta-1)$-mark moves to r from another point, the corresponding server moves to r.

(3) If $r \in S_\alpha$, then apply **Rule 3**. Then create $k-1$ new $(\beta-1)$-marks, and distribute them randomly amongst the k $(\beta-1)$-eligible points. We move the server to r from the point which is left without a $(\beta-1)$-mark.

Recall, that this randomized algorithm is H_k-competitive. Note, that with any deterministic rule for distributing the marks this algorithm switches to a deterministic one. Now we turn our attention to the OPT graph of the paging problem. We repeat the definition of [LR].

Definition 2.1 *An OPT graph of an on-line k-server problem is a finite directed graph with one distinguished vertex I, such that (1) each edge is labeled with a request from V and a cost, (2) for each vertex and each request r there is a unique edge out of that vertex whose request label is r, and (3) for every request sequence ϱ, opt(ϱ) equals the sum of the cost labels on the path given by ϱ starting from I.*

Every legal configuration of the servers can be viewed as a k element subset of the points. If $S \subset V$ is the set of points occupied by the servers, and r is the new request, the k-sets reachable by some algorithm from S by r are the elements of the following set system: $\mathcal{H} = \{H : H = S - s + r,\ s \in S\}$. If $r \in S$, then $S = \mathcal{H}$, otherwise $\mathcal{H}$ has k elements. Denote $\mathcal{H}_0$ the initial configuration of the servers, and let $\varrho = \varrho_1\varrho_2 \ldots \varrho_m$ be a request sequence of lenth m. If $\mathcal{H}_i = \{H : H = H' - -h' + \varrho_i,\ h' \in H' \in \mathcal{H}_{i-1}\}$, where $1 \leq i \leq m$, then every ϱ-satisfying algorithm's movements from configuration to configuration is embedded in the following sequence: $\mathcal{H}_0 \to \mathcal{H}_1 = 20 \to \ldots \to \mathcal{H}_m$. An important observation, that it is possible that in a set system $\mathcal{H}_i$ there is a k-set H for which the cost of any satisfying algorithm starting from $\mathcal{H}_0$ and getting to H is "too big" comparing to that of some other $H' \in \mathcal{H}_i$. One may think that this kind of sets can be ignored without eliminating any actions of a "good algorithm". In what follows these simple ideas will be made precise.

First, we give three rules for building a finite directed graph G. The vertices of

this graph are associated with set systems of k-sets, one distinguished vertex is I, the initial configuration. Now let $\mathcal{H}$ be a vertex of the graph, $r \in V$.

Rule A: If $r \in \bigcap_{H\in\mathcal{H}} H$, then do nothing.

Rule B: If $r \notin \bigcap_{H\in\mathcal{H}} H$, but $r \in \bigcup_{H\in\mathcal{H}} H$, then let $\mathcal{H}' = \{H' : H' \in \mathcal{H},\ r \in H'\}$. If $\mathcal{H}'$ is not present in G, then put it in as a new vertex. Draw the directed edge $(\mathcal{H}, \mathcal{H}')$ in G with label r. We call this kind of edge a decreasing edge.

Rule C: If $r \notin \bigcup_{H\in\mathcal{H}} H$, then let $\mathcal{H}' = \{H' : H' = H - h + r,\ h \in H \in \mathcal{H}\}$. If $\mathcal{H}'$ is not present in G, then put it in as a new vertex. Draw the directed edge $(\mathcal{H}, \mathcal{H}')$ in G with label r. We call this kind of edge an increasing edge.

Note, that **Rule B** stands for discarding the "expensive k-sets", and we apply **Rule C**, when we are forced to move a server. Starting from I after a finite number of applications of the rules we arrive to a graph G to which we can't put new vertices or edges. As one can see, there cannot be more than $n \cdot 2^{\binom{n}{k-1}}$ vertices of G.

3 Properties of OPT graphs and an on-line algorithm

Call a vertex of the graph G defined in the previous section a *single vertex*, if it contains only one configuration, otherwise call it a *multiple vertex*. The following two lemmas prove, that G is the OPT graph of the problem.

Lemma 3.1 *Let ϱ be a request sequence, and assume that starting from I we arrive to the vertex $\mathcal{H}$ following ϱ in G, and $H \in \mathcal{H}$ is a configuration. Then there exists a satisfaction of ϱ with endconfiguration H and its cost is the number of increasing edges in the above walk.*

Proof. Let's go backwards from H on the walk determined by ϱ. From the definition of decreasing edges it follows that until we reach an increasing edge, we don't have to change configuration. When an increasing edge is coming, it is enough to move a server, and we can get to the configuration, from which this increasing edge is going out. Hence, for the decreasing edges on the walk there is no incurring cost, and in the case of increasing edges the cost is 1. □

Lemma 3.2 *Let ϱ be a request sequence, and assume that starting from I we arrive to the vertex $\mathcal{H}$ in G. Then the set system $\mathcal{H}$ contains exactly the optimally reachable configurations, and the optimal cost is the number of increasing edges traversed when getting to $\mathcal{H}$.*

Proof. We proceed by induction on the length of the request sequence. If $|\varrho| \leq 1$, then the lemma trivially holds. Let's suppose that it is true for every request sequence with length at most t, and $|\varrho| = t$.

First we prove, that for every new request r, if the edge $(\mathcal{H}, \mathcal{H}')$ is labeled r, then for $H' \in \mathcal{H}'$ the optimal cost of arriving to H' is the number of increasing edges.

Let's assume that there is a satisfaction of ϱr which arrives to H' with cost less than the number of increasing edges. Denote the jth configuration of this satisfaction by Q_j. If $Q_t \notin \mathcal{H}$, then by the induction hypothesis we know that the cost of getting to Q_t by ϱ is bigger than the number of increasing edges, because every optimally reachable configuration is the element of $\mathcal{H}$. Hence, after a new request the cost of getting to $Q_{t+1} = H'$ is at least as big as the number of increasing edges. If $Q_t \in \mathcal{H}$, then the statement trivially holds, either the edge $(\mathcal{H}, \mathcal{H}')$ is an increasing or a decreasing edge.

So far we have proved that the optimal cost for $\mathcal{H}'$ is the number of increasing edges. Let's assume that there is a satisfaction $\mathcal{S}$ of ϱr such that S_{t+1}, the last configuration of it is not in $\mathcal{H}'$, but it has optimal cost. If the cost of $\mathcal{S}$ were smaller, than the number of increasing edges, then because up to S_t the cost can't be smaller than the cost of $\mathcal{H}$ (by the induction hypothesis), $S_t \in \mathcal{H}$. One can easily see, that r has to be an increasing request, but then the cost of $\mathcal{S}$ cannot be smaller than the cost of $\mathcal{H}'$. Hence, the only case is when $S_{t+1} \notin \mathcal{H}'$, and has the same cost. Denote the directed walk from I by $\mathcal{H}_0(= I) \to \mathcal{H}_1 \to \ldots \to \mathcal{H}_t(= \mathcal{H}) \to \mathcal{H}_{t+1}(= \mathcal{H}')$. There is a last configuration of $\mathcal{S}$ which is the element of the corresponding vertex of the graph. Denote it by S_m, and let $r_1, r_2, \ldots r_l$ be the last $l = t + 1 - m$ requests. Thus, $S_m \in \mathcal{H}_m$, but for $j \geq 1$ $S_{m+j} \notin \mathcal{H}_{m+j}$. By our assumption the cost incurred on $\mathcal{S}$ before the last request is greater than that incurred on the ϱ-path in G. Hence, the last edge, $(\mathcal{H}_t, \mathcal{H}_{t+1})$ is an increasing edge, and $S_t = S_{t+1}$. Denote R_i the closest set in $\mathcal{H}_i$ to S_i, i.e., $|R_i \bigcap S_i| = max\{|H \bigcap S_i| : H \in \mathcal{H}_i\}$. Observe, that $r_1 \notin S_m$, and the $(\mathcal{H}_m, \mathcal{H}_{m+1})$ edge is a decreasing edge, otherwise $S_1 \in \mathcal{H}_{m+1}$. From this follows, that $|R_1 \bigcap S_1| = k - 1$. There are five cases to consider when satisfying the last l requests. In the following $1 \leq i \leq l - 1$.

(1) $r_{i+1} \in S_i \bigcap R_i \Longrightarrow |S_i \bigcap R_i| = |S_{i+1} \bigcap R_{i+1}|$, and the difference of the costs doesn't change (lazy satisfactions).

(2) $(\mathcal{H}_i, \mathcal{H}_{i+1})$ is an increasing edge, and $r_{i+1} \in S_i \Longrightarrow$ the difference of the costs is decreased by 1, and $|S_i \bigcap R_i| + 1 = |S_{i+1} \bigcap R_{i+1}|$. This equality easily follows from the definition of G (**Rule C**).

(3) $(\mathcal{H}_i, \mathcal{H}_{i+1})$ is an increasing edge, and $r_{i+1} \notin S_i \Longrightarrow$ the difference of the costs doesn't change, and $|S_i \bigcap R_i| \leq |S_{i+1} \bigcap R_{i+1}|$.

(4) $r_{i+1} \notin S_i$, and $r_{i+1} \in R_i \Longrightarrow$ the difference of the costs is increased by 1, and $|S_i \bigcap R_i| \leq |S_{i+1} \bigcap R_{i+1}|$.

(5)$r_{i+1} \notin S_i$, $(\mathcal{H}_i, \mathcal{H}_{i+1})$ is a decreasing edge, but $R_i \notin \mathcal{H}_{)+\infty} \Longrightarrow$ the difference of the costs is increased by 1, and $|S_i \bigcap R_i| \leq |S_{i+1} \bigcap R_{i+1}| + 1$.

Let us suppose first, that at the jth stage $(1 \leq j \leq l)$ the difference of the costs is $\geq k$. Then by moving at most $k - 1$ servers from R_j we can reach S_j (the intersection $S_i \bigcap R_i$ always contains the most recently requested point), and this has cost at most $k - 1$. Hence, there is a ϱ-satisfying configuration sequence to S_{t+1} with smaller cost, and this contradicts with the optimality of $\mathcal{S}$. Let us consider now the quantity $D_i = k - |S_i \bigcap R_i|$. We claim, that D_i is a lower bound for the difference of the costs in the ith stage. For $i = 1$ this is obviously true. Assume, that D_i is a lower bound up to the ith step, and a new request, r_{i+1} is

coming. We check the possible five cases. In case (1) $D_i(= D_{i+1})$ certainly remains a lower bound. In case (2) the drawback of $\mathcal{S}$ is decreased, but the intersection size increases by one. In case (3) the drawback doesn't change, and the intersection size may increase. In cases (4)-(5) the drawback increases, but the size of the intersection doesn't decrease by more than one. Thus, in all cases D_{i+1} is a lower bound. By our assumptions, $D_{l-1} \leq 1$, and the last request should be a request considered in case (2). By the definition of G, $S_{t+1} \in \mathcal{H}_{t+1}$. We get, that $\mathcal{H}_{t+1}$ consists of exactly the optimally reachable configurations. □

Definition 3.3 *Consider the partition $S_\alpha \bigcup S_{\alpha+1} \bigcup \ldots \bigcup S_\beta$ given by the partitioning algorithm. Let $\mathcal{P} = \{P : |P| = k - |S_\beta|$, P does not contain more than k_i points from $S_i^*\}$. The set system given by the partition is $\mathcal{S} = \{S : S = S_\beta \bigcup P, \in \mathcal{P}\}$.*

Lemma 3.4 *Let $\mathcal{S}$ be the set system determined by the partitioning algorithm. Then $S_\beta = \bigcap_{S \in \mathcal{S}} S$.*

Proof. $S_\beta \subseteq \bigcap_{S \in \mathcal{S}} S$ trivially holds. Let's suppose that for some $v \in V$ $v \in \bigcap_{S \in \mathcal{S}} S$, and $v \in S_i$, $\alpha < i < \beta$. By the labeling invariant conditions $k_i = k_{i-1} + |S_i| - 1$, i.e., $|S_i| = k_i - k_{i-1} + 1$. If $v \in \bigcap_{S \in \mathcal{S}} S$, then every point of S_i is in the intersection, because they have the same role. Thus, from S_{i-1}^* we could choose only $k_{i-1} - 1$ points. On the other hand we are allowed to choose k_{i-1} points — we arrived at a contradiction. □

Lemma 3.5 *Assume, that starting from I we follow the edges of G determined by the request sequence ϱ, and we arrive to the vertex $\mathcal{H}$. Let the set system given by the actual partition (after ϱ) is $\mathcal{S}$. Then $\mathcal{H} = \mathcal{S}$.*

Proof. We prove the statement by comparing the maintaining rules for the partitioning algorithm and the build-up rules for G, by induction on the length of ϱ. If $|\varrho| = 0$, the lemma trivially holds. Let's suppose now that the lemma is true for ϱ, a new request r is coming, the edge $(\mathcal{H}, \mathcal{F})$ is labeled r in G, and the new set system given by the partition is $\mathcal{Q}$. We distinct three cases depending on the rule we use to maintain the partition.

If $r \in S_\beta$: There is nothing to prove, $\mathcal{F} = \mathcal{H}$ and $\mathcal{Q} = \mathcal{S}$ (**Rule 1** $\Leftrightarrow$ **Rule A**).

If $r \in S_i\,(\alpha < i < \beta)$: By **Rule 2**, $Q_\beta = S_\beta + r$, and $Q_i = S_i - r$ and $k_j \leftarrow k_j - 1$ for $j : i \leq j < \beta$. If for some j, k_j has become 0, this means, that previously k_j was 1, and hence, from S_j^* we could choose only one point to some $S \in \mathcal{S}$. By $Q_\alpha = S_\alpha \bigcup S_{j_0}^*$, where j_0 is the biggest such j, we discard all the sets $S \in \mathcal{S}$ which doesn't contain r. Thus, the set system Q contains exactly those sets $S \in \mathcal{S}$, for which $r \in S$. Using the induction hypothesis and the definition of **Rule B**, we get, that $\mathcal{Q} = \mathcal{F}$.

The only possibility left is that $r \in S_\alpha$. By **Rule 3**, $Q_\beta = r$, thus, $r \in Q$ for every $Q \in \mathcal{Q}$, and this is the only element of the intersection of the sets of $\mathcal{Q}$. Also, $k_\beta \leftarrow k_\beta - 1$ and $\beta \leftarrow \beta + 1$, hence $\mathcal{Q}$ contains exactly the sets which has r and other $k - 1$ points from some $S \in \mathcal{S}$. But this is the set system $\mathcal{F}$ we get from $\mathcal{H}$ by applying **Rule C**. □

Definition 3.6 *Let H and F be two configurations. We say, that F is achievable from H (and vice versa), if $|H \bigcap F| = k-1$, i.e., moving one server is enough to reach one from the other.*

Lemma 3.7 *Let $\mathcal{H}$ be a vertex in G, $r \in V$, and $H \in \mathcal{H}$. If $(\mathcal{H}, \mathcal{F})$ is the outgoing edge from $\mathcal{H}$ labeled r, then there is an achievable $F \in \mathcal{F}$ from H.*

Proof. If $\mathcal{H} = \mathcal{F}$ (**Rule A**), then there is nothing to prove
If $(\mathcal{H}, \mathcal{F})$ is an increasing edge, then there are k sets in $\mathcal{F}$ which are achievable from H: all the sets of the form $F = H - h + r$, $h \in H$.

Let's suppose now, that $(\mathcal{H}, \mathcal{F})$ is a decreasing edge. If $r \in H$, then $H \in \mathcal{F}$, so, let $r \notin H$. By Lemma 3.5 (and using the notation of it), $r \in S_i$ for some $i : \alpha < i < \beta$.

There are two possibilities.

(1) When composing H, we did not pick any points from S_i^*. But then there is a largest t ($t \geq 0$), such that we did not choose any point from S_{i+t}^*, because we had the necessary number of points. Hence, we picked a point $v \in S_{i+t+1}^*$. Substituting this v by r ($r \in S_{i+t+1}^*$) we arrive to a set H', for which $|H \bigcap H'| = k-1$, and $r \in H'$. Thus, $H' \in \mathcal{F}$, and it is achievable from H.

(2) We picked another point w from S_i^*. Substituting w by r, we again get a set of $\mathcal{F}$, which is achievable from H. □

Lemma 3.8 *There cannot be more than $k-1$ consecutive decreasing edges in any walk on G.*

Proof. Notice, that for any vertex $\mathcal{H}$ in G, $r \in \bigcap_{H \in \mathcal{H}} H$ if r is the label of an ingoing edge. After each application of **Rule B** this intersection size increases. Thus, after $k-1$ decreasing edges we arrive to a vertex which is represented by a single k-set. From such vertices every outgoing edge is an increasing edge: no k consecutive decreasing edges are possible. □

Now we are ready to discuss our on-line algorithm. Roughly speaking, we do a walk on the OPT graph step by step according to the incoming requests. We introduce some notation: r is the new request, S is the configuration of the servers, and $\mathcal{H}, \mathcal{H}'$ are vertices of the OPT graph.

- (1) Initially $S = \mathcal{H} = I$.
- (2) If $r \in S$, then no server moves. If r is a loop edge label of $\mathcal{H}$ (the actual vertex of the graph), then we stay there. If $(\mathcal{H}, \mathcal{H}')$ is a decreasing edge labeled r, then $\mathcal{H}'$ will be the new actual vertex.
- (3) If $r \notin S$, then $(\mathcal{H}, \mathcal{H}')$ is either a decreasing edge or an increasing edge labeled r. Choose any configuration $H' \in \mathcal{H}'$, which is reachable from S. We know, that there is always at least one such configuration (Lemma 3.7). Move the server on $S - H$ to r.

Note, that in (3) there is no deterministic rule how to choose the server to move, when there is a multiple choice. We will see, that one can use any kind of rule in these cases.

Theorem 3.9 *The algorithm described above is a k-competitive on-line algorithm.*

Proof. From the description it is obvious, that the algorithm is well defined, and in an on-line fashion we never have to move more than one server for a request. We decomp ose the walk defined by the request sequence ϱ in the OPT graph into s everal *phases*. The first phase starts from I, consists of the first consecutive increasing edges, and the conse cutive decreasing edges coming right after them. This phase ends, when a new increasing edge is to be traversed. =20 Then a new phase starts, with the same structure: consecutive increasing edges, and then consecutive decreasing edges. Observe, that from Lemma 3.1 and 3.2 it follows, that the optimal cost of ϱ is the number of increasing edges traversed while satisfying ϱ. In a phase by definition, there is at least one increasing edge, and not more, than $k-1$ decreasing edges. This is the consequence of Lemma 3.8. As mentioned above, the algorithm never moves more than one server for a request. Hence, in every phase the cost of the optimal satisfaction is the number of increasing edges, and the cost of our on-line algorithm is at most the sum of the umber of the increasing and decreasing edges (at most, because the algorithm not necessarily moves a server for a decreasing request). It is easy to see, that the fraction of these two quantities is always at most k. From this the theorem follows. □

Remark: Say, that there are i increasing edges in a phase. Then the compe titive ratio for that phase is at most $\frac{k-1+i}{i} < \frac{k}{i} + 1$. When n is large enough comparing to k, then we expect more than one increasing edge and less than $k-1$ decreasing edges in an "average phase". Hence, this algorithm works well in these cases. Unfortunately, either storing the OPT graph in memory or dynamically computing the next vertex needs a lot of resources. Thus, this algorithm is undesirable in practice, but possibly of theoretical interest. It suggests, that for "random sequences" the competitive ratio of an on-line algorithm can be much smaller, than k.

4 Application for a k-server problem

In this section we use the OPT graph of the paging problem to define an on-line algorithm for a special case of the k-server problem. Let us call a finite metric space *multipartite*, if the points can be distributed into several classes, where the distance between two points is 1, when they correspond to different classes, and any number in the [1,2] interval otherwise. One can easily show, that these are valid metric spaces, that is, non-negativity and symmetry of the distances, and the triangle inequality are satisfied.

Theorem 4.1 *If in a multipartite metric space no class has more than $k-1$ elements, then there is an on-line algorithm for the k-server problem of this metric space with competitive ratio $2k-2$.*

Proof. Our algorithm is almost the same, which was considered in the paging problem, but now we have less freedom in the multiple choices. If the algorithm is forced to move, then we try to move distance 1 if it is possible. Otherwise, we move any server consistently with the actual OPT graph vertex. Observe, that if the new request is an increasing request, then we can choose a server which move s distance 1. There are at most $k-1$ points in one class, hence, at least two servers are not in the class of the new request. Another important case, when a phase starts from a single vertex H, and that phase has only one increasing edge. That edge goes to a vertex $\mathcal{H}'$, which contains exactly the union of the $k-1$ element subsets of H and r, the new request. Whichever server we have moved to r, there is always another server from another class in another configuration of $\mathcal{H}'$. Hence, for the first decreasing edge we either don't move a server at all, or there is a server, which has to move only 1. Again, this is a simple consequence of the class sizes. Thus, in such phases the optimal cost is at least 1, while our on-line cost is at most $1+1+2(k-2)$, from which we have the $2k-2$ bound for the competitive ratio of these phases.

If there are more than one increasing edge in a phase, then the competitive ratio of such a phase is at most $\frac{i+2(k-1)}{i} \leq 1+k-1=k$, where i is the number of increasing edges. When there are less than $k-1$ decreasing edges, the competitive ratio of the phase is at most $\frac{i+2(k-2)}{i} \leq 2k-3$.

There is one case left: phases with one increasing edge and $k-1$ decreasing edges, starting from a multiple vertex. If a phase starts from a multiple vertex, then the previous phase has at most $k-2$ decreasing edges. We compute the competitive ratio of these two consecutive phases. It is at most $\frac{1+2(k-2)+1+2(k-1)}{2} = 2k-2$. □

This result was an illustration, the careful reader may notice that by decreasing the class sizes, a more thorough analysis gives smaller competitive ratios. On the other hand, we cannot expect a k-competitive algorithm for non-uniform metric spaces by just using the OPT graph of the paging problem. OPT graphs for non-uniform spaces may prove to be useful, but up to this time these graphs were investigated only for very special cases. Notice, that for non-uniform problems we may lose the symmetry of the graph, that can make the analysis hard.

Let us discuss a little bit more on the connection of paging and the general k-server problem. In a finite metric space divide every distance with the length of the smallest distance in it. This way every distance will be in the [1,D] interval for some D. Let ϱ be a request sequence, and denote $opt_p(\varrho)$ the optimal cost of ϱ in the uniform metric space, while $opt(\varrho)$ denotes the optimal cost of satisfaction in the original one. Then $opt_p(\varrho) \leq opt(\varrho)$ and $opt(\varrho) \leq D \cdot opt(\varrho)$, obviously. If $\mathcal{A}$ is any k-competitive paging algorithm, then $\mathcal{A}(\varrho) \leq k \cdot opt_p(\varrho) + M$ for some constant M, and thus $\mathcal{A}(\varrho) \leq D \cdot k \cdot opt(\varrho) + M$. Thus, reaching the $2 \cdot k$-competitivity for multipartite metric spaces is easy, any k-competitive paging algorithm achieves it.

5 Summary

In this paper we investigated the paging problem, and a special k-server problem. We used OPT graphs to have a better insight to paging. Our results suggests, that the partitioning algorithm in practice may perform well, considering only the competitivity as a measure. Then we proved a new nontrivial upper bound for the multipartite k-server problem. We did it by the help of the OPT graph of the paging problem. While our opinion is that one cannot expect much more with our technique, we think, that a better understanding of the structure of OPT graphs for non-uniform spaces may result in better upper bounds.

Acknowledgement The author is grateful Péter Hajnal and Endre Szemerédi for listening to earlier versions of this paper, and to Tibor Széles for his valuable help in the proofreading.

References

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

[CL1] Chrobak, M., Larmore, L., *The Server Problem and On- -line Games*, Proceedings of the DIMACS Workshop on On-line Algorithms, American Mathematical Society, February 1991

[CL2] Chrobak, M., Larmore, L., *Generosity helps, or an 1 1-competitive algorithm for three servers*, Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, 1992

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

[KP] Koutsoupias, E. and Papadimitrou, C., *On the k-Server Conjecture*, STOC 94, pp. 507-511.

[LR] Lund, C., Reingold, N., *Linear Programs for Randomized On-line Algorithms*, Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, 382-391, 1994

[MMS] Manasse, M. S., McGeoch L. A. and Sleator, D. D., *Competitive Algorithms for Server Problems*, Journal of Algorithms 11 (1990), pp. 208-230.

[MS] McGeoch L. A. and Sleator, D. D., *A Strongly Competitive Randomized Paging Algorithm*, Algorithmica (1991), pp. 816-825.

[ST] Sleator, D. D., Tarjan, R. E., *Amortized Efficiency of List Update and Paging Rules*, Comm. of the ACM, February 1985, pp. 202-208.