The Delayed k-Server Problem

Wolfgang W. Bein[1], Kazuo Iwama[2], Lawrence L. Larmore[1], and John Noga[3]

[1] School of Computer Science, University of Nevada
Las Vegas, Nevada 89154, USA. **
bein@cs.unlv.edu larmore@cs.unlv.edu

[2] School of Informatics, Kyoto University
Kyoto 606-8501, Japan.
iwama@kuis.kyoto-u.ac.jp

[3] Department of Computer Science, California State University Northridge
Northridge, CA 91330, USA
jnoga@csun.edu

Abstract. We introduce a new version of the server problem: the *delayed server problem*. In this problem, once a server moves to serve a request, it must wait for one round to move again, but could serve a repeated request to the same point. We show that the delayed k-server problem is equivalent to the $(k-1)$-server problem in the uniform case, but not in general.

Keywords: Design and analysis of algorithms; approximation and randomized algorithms.

1 Introduction

The *k-server problem* is defined as follows: We are given $k \geq 2$ mobile servers that reside in a metric space M. A sequence of requests is issued, where each request is specified by a point $r \in M$. To *service* this request, one of the servers must be moved to r, at a cost equal to the distance moved. The goal is to minimize the total cost over all requests. $\mathcal{A}$ is said to be *online* if it must decide which server, or servers, to move without the knowledge of future requests.

We say that an online algorithm $\mathcal{A}$ for any online problem is *C-competitive* if the cost incurred by $\mathcal{A}$ for any input sequence is at most C times the optimal (offline) cost for that same input sequence, plus possibly an additive constant independent of the input. The *competitive ratio of $\mathcal{A}$* is the smallest C for which $\mathcal{A}$ is C-competitive. The *competitiveness* of any online problem is then defined to be the smallest competitive ratio of any online algorithm for that problem.

The competitive ratio is frequently used to study the performance of online algorithms for the k-server problem, as well as other optimization problems. We refer the reader to the book of Borodin and El-Yaniv [2] for a comprehensive discussion of competitive analysis.

** Research of these authors supported by NSF grant CCR-0312093.

The k-server problem is originally given by Manasse, McGeoch and Sleator [11], who prove that no online algorithm for the k server problem in a metric space M has competitive ratio smaller than k, if M has at least $k+1$ points. They also present an algorithm for the 2-server problem which is 2-competitive, and thus optimal, for any metric space. They furthermore state the *k-server conjecture*, namely that for each k, there exists an online algorithm for the k server problem which is k-competitive in any metric space. For $k > 2$, this conjecture has been settled only in a number of special cases, including trees and spaces with at most $k+2$ points [4, 5, 10]. Bein *et al.* [1] have shown that the work function algorithm for the 3-server problem is 3-competitive in the Manhattan plane, while Koutsoupias and Papadimitriou have shown that the work function algorithm for the k-server problem is $(2k-1)$-competitive in arbitrary metric spaces [8, 9].

We study here a modified problem, the *delayed k-server problem*. Informally, in the delayed server problem, once a server serves a request, it must wait for one round. This problem is motivated by applications where there are latencies to be considered or service periods to be scheduled.

More precisely, we consider the following two versions of the problem. Let $r^1, r^2, \ldots,$ be a given request sequence.

(a): If a server serves the request at time t, it must stay at that request point until time $t+2$. However, if $r^{t+1} = r^t$, the server may serve the request at time $t+1$.

(b): If a server is used to serve the request at time t, it cannot be used to serve the request at time $t+1$.

In practice, the difference between these two versions is that, in Version (b), it may be necessary to have two servers at the same point, while in Version (a) that is never necessary.

We refer to the server which served the last request as *frozen*. We will assume an initial configuration of servers, one of which will be initially designated to be frozen.

Lemma 1.1. *Let $C_{a,M,k}$ and $C_{b,M,k}$ to be the competitiveness of the delayed k-server problem in a metric space M, for Versions (a) and (b), respectively. Then $C_{a,M,k} \leq C_{b,M,k}$.*

Proof. Let $\mathcal{A}$ be a C-competitive online algorithm for Version (b). We can then construct a C-competitive online algorithm $\mathcal{A}'$ for Version (a), as follows. For any request sequence ϱ, let ϱ' be the request sequence obtained from ϱ by deleting consecutive duplicate requests. Then $\mathcal{A}'$ services ϱ by emulating $\mathcal{A}$ on the requests of ϱ', except that $\mathcal{A}'$ services any consecutive duplicate request at zero cost and then forgets that it happened.

Let OPT be the optimal offline algorithm for Version (b), and let OPT$'$ be the optimal offline algorithm for Version (a). Note that the optimal cost to service a request sequence with no consecutive duplicates is the same for both versions.

We know that there exists a constant K such that $cost_{\mathcal{A}} \leq C \cdot cost_{\text{OPT}} + K$ for every request sequence. Thus, for every request sequence ϱ,

$$\begin{gathered} cost_{\mathcal{A}'}(\varrho) - C \cdot cost_{\text{OPT}'}(\varrho) = \\ cost_{\mathcal{A}'}(\varrho') - C \cdot cost_{\text{OPT}'}(\varrho') = \\ cost_{\mathcal{A}}(\varrho') - C \cdot cost_{\text{OPT}}(\varrho') \leq K \end{gathered}$$

We remark that, trivially, the delayed k-server problem is only defined for $k \geq 2$ and that it is 1-competitive if $k = 2$ in all situations.

In the *cache problem*, we consider a two-level memory system, consisting of fast memory (the *cache*), which can hold k memory units (commonly called *pages*) and an area of slow memory capable of holding a much larger number of pages. For fixed k, we refer to the cache problem as the *k-cache problem* if the cache size is k.

In the most basic model, if a page in slow memory is needed in fast memory this is called a *page request*. Such a request causes a *hit* if the page is already in the cache at the time of the request. But in the case of a *fault*, *i.e.* when the page is not in the cache, the requested page must be brought into the cache – we assume unit cost for such a move – while a page in the cache must be evicted to make room for the new page. An online paging algorithm must make decisions about such evictions as the request sequence is presented to the algorithm. The k-cache problem is equivalent to the k-server problem in a uniform metric space and thus the definition of the delayed server problem implies the following two versions of the *delayed k-cache problem*:

(a): If a page in the cache is read at time t, it cannot be ejected at time $t + 1$.
(b): If a page in the cache is read at time t, it can neither be read nor ejected at time $t + 1$. (That implies that duplication of pages in the cache must be allowed.)

We remark that Version (a) of the delayed k-cache problem is equivalent to Version (a) of the delayed k-server problem in uniform spaces, while Version (b) of the delayed k-cache problem is equivalent to Version (b) of the delayed k-server problem in uniform spaces.

We refer to the cache location which was read in the previous step as *frozen*. We will assume an initial cache configuration, and one of the cache locations will be initially frozen.

Henceforth, we shall consider only Version (a) of the delayed k-server problem. We claim that it makes more sense for applications, such as the cache problem described above, but we have included a short section (Section 5, at the end of the paper) which discusses Version (b). In the Section 2 we show that the classic online algorithm LRU can be adapted to the delayed k-cache problem, and is $(k-1)$-competitive. More generally, we show that the (randomized or deterministic) delayed k-cache problem is equivalent to the (randomized or deterministic) $(k-1)$-cache problem. This implies that the (randomized or deterministic) delayed k-server problem in uniform spaces is equivalent to the (randomized or deterministic) $(k-1)$-server problem in uniform spaces. This result might prompt the conjecture that the delayed k-server problem is equivalent to the $(k-1)$-server problem in all metric spaces. This is however not the case, as we show in Section 3. In Section 4, we give a k-competitive algorithm for the delayed k-server problem in a tree. Finally, we discuss future work in Section 6.

2 The Delayed k-Cache Problem

2.1 LRU is $(k-1)$-Competitive

It is well-known that *least recently used* (LRU) a deterministic algorithm for the k-cache problem (and hence for the k-server problem in a uniform space) is k-competitive. At each fault, the least recently used page is ejected. (In the terminology of the server problem this means that at each step the least recently used server is moved to serve the request, if the request is at a point which does not already have a server.)

Theorem 2.1. *The algorithm LRU is $(k-1)$-competitive for the delayed k-cache problem.*

Proof. Let a request sequence $\varrho = r^1 \dots r^n$ be given. Let OPT be an optimal offline algorithm. The most recently used page is the frozen page. We insist that LRU eject all other initial pages before it ejects the initially frozen page.

We partition ϱ into *phases* $\sigma^0, \sigma^1, \dots$ We will show that OPT faults once during the phase σ^t for each $t > 0$, and that LRU faults at most $k-1$ times during any phase. The result follows.

Define σ^0 to be the (possibly empty) sequence of requests that precedes the first fault of OPT. For $t > 0$, let σ^t consists of all requests starting with the t^{th} fault of OPT, up to but not including the $(t+1)^{\text{st}}$ fault of OPT.

Clearly, OPT faults at most once during each phase. We need to prove that LRU faults at most $k-1$ times during a phase.

Initially, mark all pages in the cache. If a page p is requested, and if OPT does not fault on that request, mark p, unless it is already marked. If OPT faults on that that request, unmark all pages except the frozen page, and then mark p.

We observe that OPT never ejects a marked page, because the only time OPT faults is when a phase begins, and at that time all pages are unmarked except for the frozen page, which cannot be ejected. Thus, at any given step, all marked pages are in OPT's cache. LRU also never ejects a marked page. If there are unmarked pages in the cache, each of the marked pages has been used more recently than any of the unmarked pages, and if all of LRU's cache pages are marked and LRU faults, then OPT must have those same pages in the cache, so OPT must fault also, ending the phase, and unmarking the least recently used page before it is ejected. Thus, at any given step, all marked pages are in LRU's cache.

During the phase σ^t, for $t > 0$, each time LRU faults, the number of marks increases by 1. Since σ^t begins with one page marked, LRU faults at most $k-1$ times during that phase.

2.2 Equivalence of the Delayed $(k-1)$-Cache Problem and the k-Cache Problem

We now generalize Theorem 2.1 by showing that the delayed k-cache problem and the $(k-1)$-cache problem are equivalent in a very strong sense, and thus have the same competitiveness. This equivalence is valid for both the deterministic and the randomized cases. This result implies that the delayed k-server problem for uniform spaces is equivalent to the $(k-1)$-server problem for uniform spaces. In particular, given any algorithm for the k-cache problem we construct an algorithm for the $(k-1)$-cache problem, and vice versa.

To formally construct the equivalence, it helps to assign standard names to all pages. Let $\{p_1, p_2, \ldots\}$ be the set of pages for the $(k-1)$-cache problem, and let $Q = \{q_0, q_1, q_2, \ldots\}$ be the set of pages for the delayed k-cache problem. To simplify our construction below, it helps if both caches are the same size. Thus, we introduce a fictitious page p_0 for the $(k-1)$-cache problem, which is never requested, and which is always in a fictitious cache location which can never be used. Write $P = \{p_0, p_1, p_2, \ldots\}$. We will designate q_0 to be the initially frozen page.

Without loss of generality, a request sequence for the delayed k-cache problem has no duplicate consecutive request, since such a request can be serviced at zero cost by doing nothing. Let $\mathcal{R}$ be the set of all request sequences for the $(k-1)$-cache problem, and $\mathcal{S}$ the set of all request sequences with no consecutive duplicate requests for the delayed k-cache problem. We will first construct a 1-1 onto mapping $f : \mathcal{S} \to \mathcal{R}$. Let $s^0 = p_0$. Given any $\varsigma = s^1 s^2 \ldots s^n \in \mathcal{S}$, we inductively construct a sequence of one-to-one and onto functions $f^t : Q \to P$, for $0 \le t < n$, as follows:

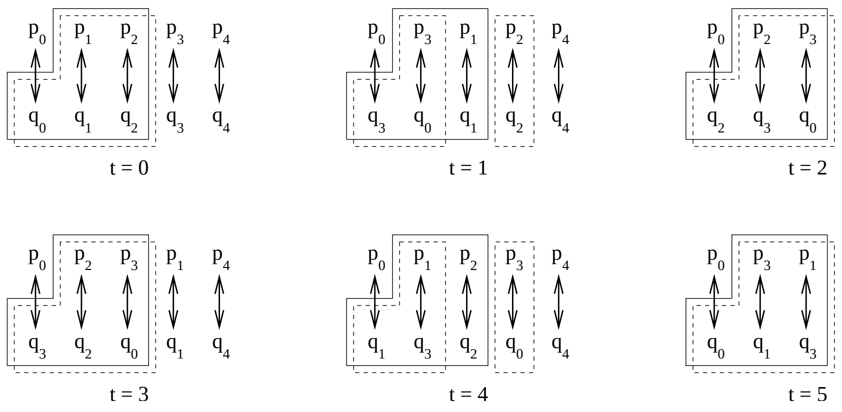
1. $f^0(q_i) = p_i$ for all $i \ge 0$.
2. If $t > 0$, then $f^t(q_i) = \begin{cases} p_0 \text{ if } q_i = s^t \\ f^{t-1}(s^t) \text{ if } q_i = s^{t-1} \\ f^{t-1}(q_i) \text{ otherwise} \end{cases}$

Now, define $f(\varsigma) = \varrho = r^1 r^2 \ldots r^n$, where $r^t = f^{t-1}(s^t)$. Note that ϱ can be defined online, *i.e.*, r^t is determined by $s^1 \ldots s^u$.

If $\mathcal{A}$ is an algorithm for the $(k-1)$-cache problem, we construct an algorithm $\mathcal{B} = F(\mathcal{A})$ for the delayed k-cache problem, as follows. Given a request sequence $\varsigma \in \mathcal{S}$ for the delayed k-cache problem, let $\varrho = f(\varsigma)$ and $f^t : \mathcal{S} \to \mathcal{R}$ be defined as above. For each step t, let $r^t = f^{t-1}(s^t)$. If $\mathcal{A}$ ejects p_i at step t with request r^t, then $\mathcal{B}$ ejects $f^{t-1}(p_i)$ at step t with request s^t. The following two lemmas can be verified inductively:

Lemma 2.2. *Let $\mathcal{A}$ be an algorithm for the $(k-1)$-cache problem, and let $\mathcal{B} = F(\mathcal{A})$. Then*

1. *for any $i \ge 0$ and any $t \ge 0$, q_i is in $\mathcal{B}$'s cache after t steps if and only if either $f^t(q_i) = p_0$ or $f^t(q_i)$ is in $\mathcal{A}$'s cache after t steps,*
2. *for any $t \ge 0$, $cost^t_{\mathcal{B}}(\varsigma) = cost^t_{\mathcal{A}}(\varrho)$.*



p_0 p_1 p_2 p_3 p_4
q_0 q_1 q_2 q_3 q_4
t = 0
p_0 p_3 p_1 p_2 p_4
q_3 q_0 q_1 q_2 q_4
t = 1
p_0 p_2 p_3 p_1 p_4
q_2 q_3 q_0 q_1 q_4
t = 2
p_0 p_2 p_3 p_1 p_4
q_3 q_2 q_0 q_1 q_4
t = 3
p_0 p_1 p_2 p_3 p_4
q_1 q_3 q_2 q_0 q_4
t = 4
p_0 p_3 p_1 p_2 p_4
q_0 q_1 q_3 q_2 q_4
t = 5

Fig. 1. Equivalence of 2-Cache and Delayed 3-Cache

Lemma 2.3. *If $\mathcal{B}$ is an algorithm for the delayed k-cache problem, then there is a unique algorithm $\mathcal{A}$ for the $(k-1)$-cache problem such that $F(\mathcal{A}) = \mathcal{B}$.*

The following theorem reduces the delayed cache problem to the cache problem.

Theorem 2.4. *Let $k \geq 2$. There is a C-competitive online algorithm, deterministic or randomized, for the $(k-1)$-cache problem, if and only if there is a C-competitive online algorithm, deterministic or randomized, respectively, for the delayed k-cache problem.*

Proof. We will only give the proof for the deterministic case, and in only one direction, as the converse has a similar proof, and the randomized case has the same proof where cost is replaced by expected value of cost.

Suppose that $\mathcal{A}$ is a C-competitive online algorithm for the $(k-1)$-cache problem. Let OPT be an optimal offline algorithm for the $(k-1)$-cache problem, and let OPT$'$ be an optimal offline algorithm for the delayed k-cache problem. Let $\mathcal{B} = F(\mathcal{A})$. Note that $\mathcal{B}$ is online. By Lemma 2.3, there exists an offline algorithm $\mathcal{D}$ for the $(k-1)$-cache problem such that $F(\mathcal{D}) = \text{OPT}'$.

We know there is some constant K such that $cost_{\mathcal{A}}(\varrho) \leq C \cdot cost_{\text{OPT}}(\varrho) + K$ for any $\varrho \in \mathcal{R}$. If $\varsigma \in \mathcal{S}$, let $\varrho = f(\varsigma)$. Then, by Lemma 2.2,

$$\begin{gathered} cost_{\mathcal{B}}(\varsigma) - C \cdot cost_{\text{OPT}'}(\varsigma) = \\ cost_{\mathcal{A}}(\varrho) - C \cdot cost_{\mathcal{D}}(\varrho) \leq \\ cost_{\mathcal{A}}(\varrho) - C \cdot cost_{\text{OPT}}(\varrho) \leq K \end{gathered}$$

2.3 An Example

Figure 1 illustrates the equivalence of the delayed 3-cache problem and the 2-cache problem in an example. Suppose that $\varsigma = q_3q_2q_3q_1q_0$. Then $\varrho = f(\varsigma) =$

$p_3p_2p_2p_1p_3$. The vertical arrows show the one-to-one correspondence f^t for each $0 \le t \le 5$.

Let $\mathcal{A}$ be LRU for the 2-cache problem, and OPT an optimal offline algorithm for the 2-cache problem. Then the contents of $\mathcal{A}$'s cache and $F(\mathcal{A})$'s cache, after each number of steps, are enclosed in solid lines. The contents of OPT's cache and $F(\text{OPT})$'s cache are enclosed in dotted lines. Note that $cost_{\mathcal{A}}(\varrho) = cost_{F(\mathcal{A})}(\varsigma) = 4$ and $cost_{\text{OPT}}(\varrho) = cost_{F(\text{OPT})}(\varsigma) = 2$.

3 Lower Bounds for the Delayed k-Server Problem

In this section we show lower bounds for the delayed k-server problem. Our first theorem shows that a lower bound of $(k-1)$ holds for arbitrary metric spaces of $k+1$ or more points.

Theorem 3.1. *In any metric space M of at least $k+1$ points, the competitiveness of the delayed k-server problem is at least $k-1$.*

Proof. Pick a set $X = \{x_1, \ldots, x_{k+1}\} \subseteq M$. Initially, all servers are in X, and, without loss of generality, there is a server at x_{k+1}. We then consider the following adversary:

- For each odd t, $r^t = x_{k+1}$.
- For each even t, r^t is the point in X where the algorithm does not have a server.

Note that the server which was initially at x_{k+1} will serve all odd requests, but will never move, yielding a cost of zero for all requests at odd-numbered steps, for both the online algorithm and the optimal algorithm. Thus, we can ignore all requests for odd t, and consider the sequence $r^2, r^4, \ldots, r^{2t}, \ldots$ in the metric space $M^* = M - x_{k+1}$, which is the sequence created by the cruel adversary for the $(k-1)$-server problem in M^*. The remainder of the proof is the same as the classic proof that the cruel adversary gives a lower bound of $k-1$ for the $(k-1)$-server problem in any space with at least k points. (See, for example, [11].)

From this, the results of Section 2, and from the well-known k-server conjecture [11], we might be tempted to conjecture that the competitiveness of the delayed k-server problem is $k-1$. That conjecture can be immediately shown to be false, however. In fact, we give a proof of a lower bound of 3 for the delayed 3-server problem.

Let M be the metric space with four points, $\{a, b, c, d\}$, where $ab = bc = cd = ad = 1$ and $ac = bd = 2$. We call this metric space the *square*.

Theorem 3.2. *The competitiveness of the delayed 3-server problem in the square is at least 3.*

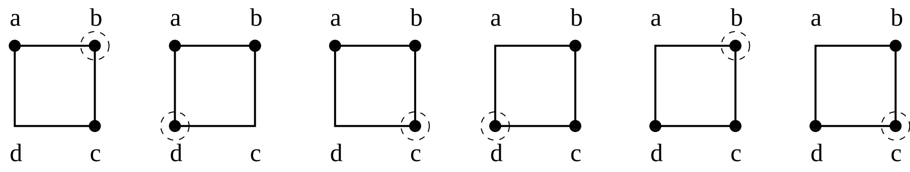


Fig. 2. One Phase Costs 3 for the Square, Optimal Cost is 1

Proof. Let $\mathcal{A}$ be an online algorithm for the delayed 3-server problem in the square. We show that the adversary can construct a request sequence consisting of *phases*, where in each phase, $\mathcal{A}$ is forced to pay 3, while the optimal offline algorithm pays 1.

At the beginning and end of each phase, the servers are at three points, and the middle point is frozen. Without loss of generality, the servers are at $\{a, b, c\}$ and the server at b is frozen.

The adversary then constructs the phase as follows:

A1. The adversary requests d.
B1. Without loss of generality, $\mathcal{A}$ serves from c. $\mathcal{A}$'s servers are now at $\{a, b, d\}$.
A2. The adversary requests $c(dbc)^N$ for sufficiently large N.
B2. The algorithm responds. No possible response costs less than 2. The phase ends when the algorithm's servers are at $\{b, c, d\}$, if ever.

We now analyze the cost. The optimal cost to serve the phase is 1, since the optimal offline algorithm moves a server from a to d at A1. The optimal algorithm's servers will now be at $\{b, c, d\}$, and move A2 is free.

If $\mathcal{A}$ ever moves its servers to $\{b, c, d\}$, it pays 1 for B1, and at least 2 for B2; the configuration is symmetric to the initial configuration, and the next phase begins. If $\mathcal{A}$ never moves its servers to $\{b, c, d\}$, the phase never ends, but $\mathcal{A}$ pays unbounded cost.

Figure 2 shows one phase where $\mathcal{A}$ pays 3. The frozen server is enclosed by a dotted circle at each step.

We note that M cannot be embedded in a Euclidean space. However, if one models M with an ordinary square in the Euclidean plane, where the diagonal distance is $\sqrt{2}$ instead of 2, the above request sequence gives a lower bound of $1 + \sqrt{2}$ for the competitiveness of the delayed 3-server problem in the Euclidean plane.

4 *k*-Competitiveness for Trees

In this section we prove that the deterministic competitiveness of the delayed k-server problem is at most k for all continuous trees. The proof is similar to that for the regular k-server problem given in [5].

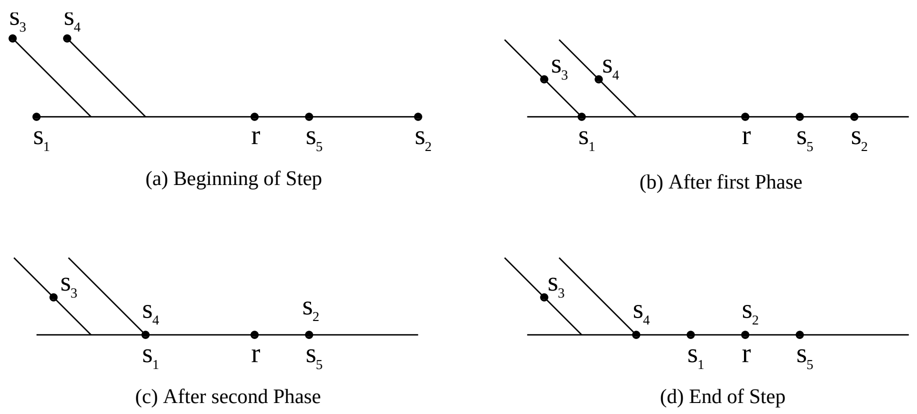


Fig. 3. One Step of Tree Algorithm Showing Three Phases

We define a *continuous tree* to be a metric space where there is exactly one continuous simple path between any two points. For example, the line is a continuous tree. We note that a continuous tree is called simply a *tree* in [5].

Let T be a continuous tree. We define an online algorithm, $\mathcal{A}$, which we call the *tree algorithm*, for the delayed k-server problem in T. $\mathcal{A}$ is very similar to the algorithm given in [5].

Suppose that $s_1, \ldots, s_k$ are the positions of $\mathcal{A}$'s servers in T, and r is the request point. Without loss of generality, s_k is the frozen page.

In response to a request, $\mathcal{A}$ moves servers continuously toward r according to a protocol given below, stopping when one of them reaches r. We call this movement a *service step*. The movement consists of *phases*, where during each phase, the number of servers moving remains constant, and all of the moving servers move toward r. At the end of each phase, one or more servers stop. Once a server stops moving, it does not start up again during that service step.

We say that a server s_i is *blocked* by a server s_j if s_j is unfrozen and s_j lies on the simple path from s_i to r. (In the special case that s_i and s_j are at the same point, we say that s_j blocks s_i if $i > j$, but not if $i < j$.) During each phase, all unblocked servers move toward r at the same speed. If one of the moving servers becomes blocked, the phase ends.

Figure 3 shows an example of a step of the tree algorithm. In the figure, $k = 5$, and initially, in (a), s_5 is frozen, and none of the other servers are blocked. During the first phase, s_1, s_2, s_3, and s_4 move toward r. When s_1 blocks s_3, as shown in (b), the second phase begins, during which s_1, s_2, and s_4 move toward r. When s_1 and s_4 reach the same point, s_1 blocks s_4, as shown in (c), starting the third phase, during which s_1 and s_2 move towards r. Finally, s_2 reaches r and services the request, ending the step, as shown in (d).

Theorem 4.1. *The tree algorithm is k-competitive for the delayed k-server problem in a tree T.*

Proof. Let OPT be an optimal offline algorithm. We need to prove that, for some constant K,

$$cost_{\mathcal{A}}(\varrho) \leq k \cdot cost_{\text{OPT}}(\varrho) + K \tag{1}$$

Just as in [5], we use the *Coppersmith-Doyle-Raghavan-Snir* potential [7] to prove k-competitiveness. If $X = \{x_1, \ldots x_k\}$ is any multiset of size k of points in T, define $\Sigma X = \sum_{1 \leq i < j \leq k} x_i x_j$. If $Y = \{y_1, \ldots y_k\}$ is another multiset of size k, define $||X, Y||$ to be the minimum matching distance between X and Y, *i.e.*, the smallest possible value of $\sum_{i=1}^{k} x_i y_{\pi(i)}$, over all permutations π of $\{1, \ldots, k\}$. If S and O are multisets of size k, we define

$$\Phi(S, O) = \Sigma S + k||S, O||$$

Let S^0 be the initial configuration of the servers, a multiset of points in T of size k. Let r^0 be the initial position of the initially frozen server. If $\varrho = r^1 r^2 \ldots r^n$ is a request sequence, let S^t be the configuration of $\mathcal{A}$'s servers after servicing $r^0 \ldots r^t$, and let O^t be the multiset of positions of OPT's servers after t steps, and let $\Phi^t = \Phi(S^t, O^t)$, the potential after t steps. Let $cost_{\mathcal{A}}^t$ be the cost of $\mathcal{A}$ during Step t, and let $cost_{\text{OPT}}^t$ be the cost of OPT during Step t. Then $cost_{\mathcal{A}}(\varrho) = \sum_{t=1}^{n} cost_{\mathcal{A}}^t$, and $cost_{\text{OPT}}(\varrho) = \sum_{t=1}^{n} cost_{\text{OPT}}^t$.

Let $S^{t,i}$ be the multiset of positions of $\mathcal{A}$'s servers after i phases of Step t; for example, $S^{t,0} = S^{t-1}$ and $S^{t,m_t} = S^t$, where m_t is the number of phases of Step t. Let $p_{t,i}$ be the number of moving servers during the i^{th} phase of Step t. and let $\ell_{t,i}$ be the distance that each of those servers moves. We verify the following sequence of equalities and inequalities.

$$\left|\left|S^{t,i-1}, S^{t,i}\right|\right| = p_{t,i}\ell_{t,i} \quad \text{for all } t \text{ and all } 1 \leq i \leq m_t \tag{2}$$

$$\Sigma S^{t,i} - \Sigma S^{t,i-1} \leq (2k - (1+k)p_{t,i})\ell_{t,i} \quad \text{for all } t \text{ and all } 1 \leq i \leq m_t \tag{3}$$

$$\left|\left|S^{t,i}, O^t\right|\right| - \left|\left|S^{t,i-1}, O^t\right|\right| \leq (p_{t,i} - 2)\ell_{t,i} \quad \text{for all } t \text{ and all } 1 \leq i \leq m_t \tag{4}$$

$$\left|\left|S^{t,i-1}, S^{t,i}\right|\right| + \Phi(S^{t,i}, O^t) \leq \Phi(S^{t,i-1}, O^t) \quad \text{for all } t \text{ and all } 1 \leq i \leq m_t \tag{5}$$

$$\Phi(S^{t-1}, O^t) \leq k \cdot \left|\left|O^{t-1}, O^t\right|\right| + \Phi(S^{t-1}, O^{t-1}) \quad \text{for all } t \tag{6}$$

$$\left|\left|S^{t-1}, S^t\right|\right| + \Phi(S^t, O^t) \leq \Phi(S^{t-1}, O^t) \quad \text{for all } t \tag{7}$$

$$\begin{aligned} cost_{\mathcal{A}}^t + \Phi^t &= \\ \left|\left|S^{t-1}, S^t\right|\right| + \Phi(S^t, O^t) &\leq k \cdot \left|\left|O^{t-1}, O^t\right|\right| + \Phi(S^{t-1}, O^{t-1}) \\ &= k \cdot cost_{\text{OPT}}^t + \Phi^{t-1} \quad \text{for all } t \end{aligned} \tag{8}$$

(2) follows from the fact that $p_{t,i}$ servers move a distance of $\ell_{t,i}$ each.

During Phase i of Step t, each stationary server s_j gets farther away from at most one server, namely the moving server, if any, that blocks it; and s_j gets closer to each other moving server. Furthermore, any two moving servers get closer to each other. (3) follows from routine calculation.

During Phase i of Step t, in the minimum matching of O^t with S, as S varies from $S^{t,i-1}$ to $S^{t,i}$, the server of OPT which served r^{t-1} can be matched with

s_k, and the server of OPT which served r^t can be matched with one moving server, say s_j. Since s_j gets closer to its partner during the phase, and since, in the worst case, the other $p_{t,i} - 1$ moving servers get farther away from their partners, (4) holds. Then, (5) follows from Inequalities (2), (3) and (4).

By the triangle inequality for minimum matching,

$$k \cdot \left|\left|O^{t-1}, O^t\right|\right| + \Phi(S^{t-1}, O^{t-1}) - \Phi(S^{t-1}, O^t) =$$
$$k \cdot \left|\left|O^{t-1}, O^t\right|\right| + k \cdot \left|\left|S^{t-1}, O^{t-1}\right|\right| - k \cdot \left|\left|S^{t-1}, O^t\right|\right| \geq 0$$

which verifies (6). Then (7) follows from (5) for each phase, while (8) follows from (6) and (7). Note that $\Phi^n \geq 0$; summing (8) over all t and letting $K = \Phi^0$, we obtain (1).

We give the following lemma without proof:

Lemma 4.2. *If $M_1 \subset M_2$, then $C_{a,M_1,k} \leq C_{a,M_2,k}$ and $C_{b,M_1,k} \leq C_{b,M_2,k}$.*

From this we have:

Theorem 4.3. *If a metric space M can be embedded into a continuous tree T, then the delayed k-server problem is k-competitive in M.*

5 Version (b)

We remark that Theorem 2.1 holds if we use Version (b) of the delayed k-cache problem. In this case, we must be sure to define LRU properly; LRU keeps track of when each cache location was used, and in case of a fault, ejects the page in the cache that was least recently read. The proof of $(k-1)$-competitiveness is very similar to that for Version (a). In the proof, if a page is read, the cache location of that page is marked. If there are two copies of the page in the cache, it is important not to mark both of them unless forced to do so. More precisely, if a page p is requested, and there is one copy of p in the cache, that copy is read and its location marked, unless it is frozen, in which case another copy is moved into the cache and its location marked. But if p is requested and there are already two copies in the cache and their locations are not both marked, only the location of the copy that was most recently moved into the cache is marked.

We remark that, using essentially the same proof as that of Theorem 4.3, we can show that the tree algorithm is k-competitive for Version (b).

6 Open Problems

We conjecture that there is a lower bound of k for the delayed k-server problem in general spaces. In fact, it could well be that a lower bound of k could be provable for tree metric spaces. We also conjecture that $C_{a,M,k} \leq C_{b,M,k} \leq C_{M,k}$ for any metric space M, where $C_{M,k}$ is the competitiveness of the k-server problem in M.

Further work is necessary to give a competitive algorithm for the delayed server problem in general spaces. We conjecture that a modification of the Work Function Algorithm (see, for example, [3, 6, 9]) could yield such an algorithm.

7 Acknowledgement

Wolfgang Bein thanks Kazuo Iwama for the generous support he received while he visited Kyoto University to work on this project during December 2004 and January 2005.

References

1. Wolfgang Bein, Marek Chrobak, and Lawrence L. Larmore. The 3-server problem in the plane. *Theoretical Computer Science*, 287(1):387–391, 2002.
2. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
3. William R. Burley. Traversing layered graphs using the work function algorithm. *Journal of Algorithms*, 20:479–511, 1996.
4. Marek Chrobak, Howard Karloff, Tom H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4:172–181, 1991.
5. Marek Chrobak and Lawrence L. Larmore. An optimal online algorithm for k servers on trees. *SIAM Journal on Computing*, 20:144–148, 1991.
6. Marek Chrobak and Lawrence L. Larmore. Metrical task systems, the server problem, and the work function algorithm. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms: The State of the Art*, pages 74–94. Springer, 1998.
7. Don Coppersmith, Peter G. Doyle, Prabhakar Raghavan, and Marc Snir. Random walks on weighted graphs and applications to online algorithms. In *Proc. 22nd Symp. Theory of Computing (STOC)*, pages 369–378. ACM, 1990.
8. Elias Koutsoupias and Christos Papadimitriou. On the k-server conjecture. In *Proc. 26th Symp. Theory of Computing (STOC)*, pages 507–511. ACM, 1994.
9. Elias Koutsoupias and Christos Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42:971–983, 1995.
10. Elias Koutsoupias and Christos Papadimitriou. The 2-evader problem. *Information Processing Letters*, 57:249–252, 1996.
11. Mark Manasse, Lyle A. McGeoch, and Daniel Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.