Page Migration Algorithms Using Work Functions

Marek Chrobak[*] *Lawrence L. Larmore*[†] *Nick Reingold*[‡] *Jeffery Westbrook*[§]

September 26, 1994

Abstract

The *page migration* problem occurs in managing a globally addressed shared memory in a multiprocessor system. Each physical page of memory is located at a given processor, and memory references to that page by other processors are charged a cost equal to the network distance. At times the page may migrate between processors, at a cost equal to the distance times a page size factor, D. The problem is to schedule movements on-line so as to minimize the total cost of memory references. Page migration can also be viewed as a restriction of the *1-server with excursions* problem.

This paper presents a collection of algorithms and lower bounds for the page migration problem in various settings. Competitive analysis is used. The competitiveness of an on-line algorithm is the worst-case ratio of its cost to the optimum cost on any sequence of requests. Randomized $(2 + \frac{1}{2D})$-competitive on-line algorithms are given for trees and products of trees, including the mesh and the hypercube, and for uniform spaces when $D = 1, 2$. These algorithms are shown to be optimal. A lower bound of $\frac{85}{27}$ on the competitiveness of any deterministic algorithm (in arbitrary spaces) with $D = 1$ is shown, disproving a conjecture by Black and Sleator. Deterministic $(2 + \frac{1}{2D})$-competitive algorithms are given for products of continuous trees under the $\mathcal{L}^1$ metric, such as $\mathbb{R}^n$ with the "Manhattan" metric. A deterministic algorithm for $\mathbb{R}^n$ under any norm is presented, and shown to have competitive ratio tending to $1 + \phi = 2.618\ldots$

The paper makes extensive use of the concept of a *work function*, a tool which provides a systematic approach to many competitive analysis problems.

Marek : Thu Aug 4 16:56:15 PDT 1994
Marek : Sat Aug 6 11:38:15 PDT 1994
Larry : Sat Sep 3 20:51:15 PDT 1994
Marek : Mon Sep 5 11:13:24 PDT 1994
Larry : Tue Sep 13 16:24:46 PDT 1994

[*]Department of Computer Science, University of California, Riverside, CA 92521–0304. Research partially supported by National Science Foundation grant CCR-9112067.

[†]Department of Computer Science, University of Nevada, Las Vegas, NV 89154-4019. Research partially supported by National Science Foundation grant CCR-9112067, while the author was in residence at the University of California, Riverside.

[‡]AT&T Bell Laboratories, Room 2C-320, P.O. Box 636, Murray Hill, NJ 07974-0636. Research partially supported by National Science Foundation grant CCR-8958528.

[§]Department of Computer Science, Yale University, New Haven, CT 06520-2158. Research partially supported by the National Science Foundation grant CCR-9009753.

Marek: Tue Sep 13 19:26:13 PDT 1994
Jeff: Mon Sep 19 20:06:38 EDT 1994

1 Introduction

The *page migration problem* arises as a memory management problem for a globally addressed shared memory in a multiprocessor system [5, 6, 22]. A common design for such a system is a network of processors, each of which has its own local memory [10, 19, 24]. In such a design, a programming abstraction of a single global memory address space is supported by a virtual memory system that distributes one or more copies of each physical page of memory among the processors' local memories. If processor p contains a copy of page b, then it can directly read and write memory addresses on b. If p does not have a copy, however, then it must satisfy memory accesses by transmitting a request through the network to some processor q that does have a copy of b and by waiting for an answer from q. The communication cost of this request depends upon the network, and may vary depending on the choice of p and q. Essentially the same situation arises in distributed databases, where multiple copies of a data file may be stored at various local nodes.

Having a given virtual page stored at multiple processors reduces communication overhead during reads, but introduces the problem of maintaining consistency among the multiple copies during writes, something for which most multiprocessors do not provide mechanisms [6]. Therefore, various network designers have proposed restricting each writable page to a single copy [5, 6, 22], and allowing the operating system to move the page between processors in response to changes in access patterns. The *page migration* problem is to decide, on-line, a sequence of page movements that minimizes the total cost of communication over all requests.

We formalize the page migration problem as follows: An instance of the problem consists of a metric space M (representing the network) and an integer constant $D \geq 1$ (the page size factor). Let $s \in M$ denote the current position of the page, which we think of as a "server." A sequence ϱ of requests arrives on-line. Each request is a point $r \in M$ (representing a processor which needs to access the page). Once a request r arrives, it must be immediately served at cost d_{sr}, the distance from s to r, which we also call the *remote service cost.* We are then allowed to move the server from s to any other point x at cost $D \cdot d_{sx}$. This is called the *movement cost.* The total cost associated with servicing r is thus $d_{sr} + D \cdot d_{sx}$. We assume that before the first request the server may be moved from its initial position s to an arbitrary point x at cost $D \cdot d_{sx}$.

Among on-line optimization problems, the *k-server problem*, proposed by Manasse, McGeoch, and Sleator [18] is perhaps the best known and most widely studied [4, 7, 8, 9, 12, 14, 15, 18, 20]. In their original paper, Manasse *et al.* also define the *k-server with excursions* problem, in which k mobile servers move through a metric space, paying costs for movements and for remote accesses, where these costs need not be related. Apart from a few results for $(n-1)$ servers in a metric space with n points [18], nothing is known about this problem. Page migration can be viewed as a special case of the 1-server problem with excursions, in which the movement cost is a constant factor times the remote access cost.

Fix a metric space M and $D \geq 1$. Let $\mathcal{A}$ be an on-line algorithm for the page migration problem in M and with page size factor D. For a given initial position s and a request sequence ϱ, we denote the location of the server managed by algorithm $\mathcal{A}$ after serving sequence ϱ by $\mathcal{A}(s, \varrho)$ and the cost incurred by $\mathcal{A}$ on a request sequence ϱ by $cost_{\mathcal{A}}(s, \varrho)$.

Algorithm $\mathcal{A}$ is called *c-competitive* if for each $s \in M$ there is a constant a_s such that

$$cost_{\mathcal{A}}(s, \varrho) \ \leq \ c \cdot cost_{\mathcal{A}'}(s, \varrho) + a_s \tag{1}$$

for all request sequences ϱ and all algorithms $\mathcal{A}'$. (However, in this paper all algorithms will have $a_s = 0$.)

Formally, a randomized algorithm for the page migration problem is a probability measure on the set of all deterministic algorithms for that problem. If $\mathcal{A}$ is such a randomized algorithm, we let $cost_{\mathcal{A}}(s, \varrho)$ denote the expected cost incurred by $\mathcal{A}$ on a request sequence ϱ, given initial position s, and we say that $\mathcal{A}$ is *c-competitive* if (1) then holds for all choices of x and ϱ and for every deterministic algorithm $\mathcal{A}'$.

In the literature of competitive analysis, an on-line problem is often viewed as a game played against an adversary that generates requests on-line. An adversary is called *adaptive* if during the game it is allowed to know the state of the on-line algorithm, and hence adjust its strategy in response to the random choices of the algorithm. An adversary is called *oblivious* if it only knows the on-line algorithm and, consequently, it knows the probability distribution of the state at every time step but not the actual state. An adversary must choose its own algorithm to satisfy the requests and is charged the cost incurred by its algorithm on the request sequence. An *off-line* adversary is allowed to choose its algorithm after the entire request sequence has been generated. An *on-line* adversary must choose its algorithm before any requests are generated. The oblivious on-line and oblivious off-line adversaries are equivalent in power. Our definition of randomized competitiveness assumes that the adversary is oblivious.

Black and Sleator [6] consider two classes of networks: uniform spaces (complete graphs with each edge having length 1) and trees with arbitrary edge lengths. They develop 3-competitive deterministic algorithms for these two cases. In addition, they show that no deterministic algorithm could be better than 3-competitive in any metric space with at least two points. Black and Sleator conjecture that a 3-competitive algorithm exists for all metric spaces. Very recently, Awerbuch *et al.* [1] have given a 6-competitive algorithm for the general case, which is the best upper bound known so far.

Westbrook [23] gives a randomized algorithm for the uniform network whose competitiveness constant approaches $(5 + \sqrt{17})/4 \approx 2.28$ as D approaches infinity, a $(3 + \sqrt{5})/2 \approx 2.618$-competitive randomized algorithm for general networks, and a randomized algorithm that is 3-competitive against an on-line adaptive adversary.

Similar memory management problems are examined in [4, 7, 12, 14, 15, 18, 20]. Practical issues and applications of page migration are discussed more fully in [5, 6, 22]. The *file assignment* problem involves the static allocation of multiple copies of data files to the nodes of a distributed system. File assignment has received a great deal of attention in the database community (see *e.g.* [11]). Adaptive versions of this problem have been studied [25], and our work has bearing on these problems.

A major difficulty in traditional competitive analysis has been to understand optimal off-line algorithms. In particular, it is almost always impossible to know when, or to where, an optimal algorithm will move its server (the page). The approach in this paper is to use *work functions*, which completely describe the actions of the adversary to the extent that an on-line algorithm can know them, and which provide a systematic approach to problems of competitive analysis. Work functions were defined in [9]. A similar concept was defined earlier in [18] in the context of the server problem. Each algorithm in this paper maintains a *current work function* ω, where, for each $x \in M$, $\omega(x)$ is the algorithm's estimate of the optimal cost of servicing the past request sequence in such a way that the current server location is x. An algorithm then uses the work function to decide a move.

Overview of the paper. We first consider randomized algorithms. In Section 4 we consider the simple case where the metric space consists of only two points. We present a $(2+\frac{1}{2D})$-competitive randomized algorithm, EDGE, and prove a matching lower bound. The lower bound proof uses a potential argument, similar to the method frequently used for proving upper bounds. EDGE will be used as a building block to construct more complex algorithms.

In Section 5 we consider uniform spaces of arbitrary size. Quite unexpectedly, this problem appears to be very hard. We give an extension of EDGE to uniform spaces, prove that this extension is optimal for $D = 1, 2$, and conjecture that it is optimal for any $D \geq 3$.

In Section 6 we present a randomized algorithm for trees called FACTOR, which can be viewed as a product of a number of instances of EDGE (one instance for each edge). FACTOR is $(2+\frac{1}{2D})$-competitive, *i.e.*, optimal. The "factoring" technique is similar to the technique used in [3, 13] in studying sequential search. FACTOR can also be applied, with the same competitive constant, to metric spaces that are Cartesian products of trees with the $\mathcal{L}^1$ metric, including hypercubes and meshes.

We then present a number of results about deterministic algorithms. We show how to use continuity in certain spaces to turn a randomized algorithm into a deterministic one with the same competitive constant. This leads to $(2+\frac{1}{2D})$-competitive algorithms for certain spaces (such as $\mathbb{R}^n$ under the Manhattan metric) which are products of continuous trees. We also derive an algorithm for $\mathbb{R}^n$ under any norm with a competitive ratio that tends to $1+\phi$ as D grows large, where ϕ is the golden ratio, ≈ 1.618. (This algorithm actually works for any normed metric space over $\mathbb{R}$.) Next we disprove the conjecture of Black and Sleator by giving a simple metric space for which there is no deterministic algorithm with competitiveness constant better than $\frac{85}{27} \approx 3.148$ for $D = 1$. For $D = 1$, we also give an optimal 3-competitive algorithm for any 3-point metric space. This algorithm uses *forgiveness*, a technique that has recently been applied to develop an 11-competitive algorithm for the 3-server problem [8]. We also have a computer-aided proof showing that this result cannot be extended to 4-point spaces.

2 Work Functions and Competitive Analysis

Following [9], we introduce the idea of a *work function*. Informally, at each step in the processing of a request sequence ϱ, we will have an associated work function ω such that $\omega(x)$ is a lower bound on the cost paid by any algorithm to serve the request sequence up to that point and end with the server located at point x. Although, due to the lack of information about future requests, an on-line algorithm usually cannot know the optimum location of the server, it can use the work function to estimate how much an optimum algorithm has paid so far given that its server is currently located at a particular point.

Fix a space M and a page size factor D. We say that $\omega : M \to \mathbb{R}^+$ is a *work function* if it satisfies the following two conditions:

(wf1) $\omega(x) - \omega(y) \leq D \cdot d_{xy}$ for all $x, y \in M$.
(wf2) For all $x \in M$, $\sup_y \{D \cdot d_{xy} - \omega(y)\} < \infty$.

Note that by (wf1), we can replace the universal quantifier in (wf2) by the existential one.

Given an initial position s and a request sequence ϱ, we say that a work function ω is a *work function for* (s, ϱ) if it satisfies the following additional condition:

(wf3) For all $x \in M$, the cost incurred by any algorithm that starts at point s, serves ϱ, and ends with the server at point x, is at least $\omega(x)$.

By $W_{M,D}$ we denote the set of work functions for a given metric space M and page size D. For simplicity, in the remainder of this section we write $W = W_{M,D}$, assuming M and D are fixed.

the *update operator* $\wedge : W \times M \to W$ is defined as

$$(\omega \wedge r)(x) \;=\; \inf_z \{\omega(z) + d_{zr} + D \cdot d_{zx}\}.$$

Lemma 2.1 *If ω is a work function then, for any $r \in M$, the updated function $\omega \wedge r$ is also a work function. Furthermore, if ω is a work function for some (s, ϱ), then $\omega \wedge r$ is a work function for $(s, \varrho r)$.*

Proof: We first verify that (wf1) holds for $\omega' = \omega \wedge r$. For any $x, y \in M$,

$$\begin{aligned} \omega'(x) &= \inf_z \{\omega(z) + d_{zr} + D \cdot d_{zx}\} \\ &\leq \inf_z \{\omega(z) + d_{zr} + D \cdot d_{zy} + D \cdot d_{xy}\} \quad = \quad \omega'(y) + D \cdot d_{xy}. \end{aligned}$$

Next, we verify that (wf2) holds for ω'. For any $x \in M$,

$$\begin{aligned}\sup_y \{D \cdot d_{xy} - \omega'(y)\} &= \sup_{y,z} \{D \cdot d_{xy} - \omega(z) - d_{zr} - D \cdot d_{zy}\} \\ &\leq \sup_z \{D \cdot d_{zx} - \omega(z)\}.\end{aligned}$$

The last quantity is finite since (wf1) holds for ω.

It is easy to see that condition (wf3) is true for ω' with respect to ϱr. If $\mathcal{A}$ is an algorithm that has its server in position y when the request r arrives, and then moves its server to x, by induction its cost on ϱr is at least

$$\omega(y) + d_{yr} + D \cdot d_{yx} \;\geq\; \inf_z \{\omega(z) + d_{zr} + D \cdot d_{zx}\} \;=\; \omega'(x).$$

▌

Inductively, we extend the update operator to sequences of requests (ε = the empty request sequence):

$$\begin{aligned}\omega \wedge \varepsilon &= \omega, \quad \text{and} \\ \omega \wedge \varrho r &= (\omega \wedge \varrho) \wedge r.\end{aligned}$$

For $x \in M$, let $\chi_x(y) = D \cdot d_{xy}$. We call χ_x a *characteristic function.* For each $s \in M$ and request sequence ϱ, define $\omega_{s,\varrho} = \chi_s \wedge \varrho$.

Lemma 2.2 *The function $\omega_{s,\varrho}$ is a work function for (s, ϱ).*

Proof: By induction on ϱ. For the basis, we observe that $\omega_{s,\varepsilon} = \chi_s$ is a work function for the empty sequence, since any algorithm must pay at least $D \cdot d_{sx}$ to move its server from s to x. The inductive step follows from Lemma 2.1. ▌

We now prove that $\omega_{s,\varrho}$ is the work function for (s, ϱ) that has maximum value at each point.

Lemma 2.3 *For each $\epsilon > 0$, $s, x \in M$, and request sequence $\varrho = r_1 r_2 \ldots r_n$, for $n \geq 0$, there is a service sequence $x_0, x_1, x_2, \ldots, x_n = x$ for ϱ such that*

$$D \cdot d_{sx_0} + \sum_{i=0}^{n-1} \left(d_{x_i r_{i+1}} + D \cdot d_{x_i x_{i+1}}\right) \;\leq\; \omega_{s,\varrho}(x) + \epsilon.$$

Proof: By induction on n. If $n = 0$, let $x_0 = x$. For $n \geq 1$, let $\varrho' = r_1 \ldots r_{n-1}$. Pick y such that

$$\omega_{s,\varrho'}(y) + d_{yr_n} + D \cdot d_{xy} \;\leq\; \omega_{s,\varrho}(x) + \epsilon/2$$

By the inductive assumption, there are $s = x_0, \ldots, x_{n-1} = y$ such that

$$D \cdot d_{sx_0} + \sum_{i=0}^{n-2} \left(d_{x_i r_{i+1}} + D \cdot d_{x_i x_{i+1}}\right) \;\leq\; \omega_{s,\varrho'}(y) + \epsilon/2,$$

and the lemma follows. ∎

Every work function described in this paper will have a well-defined minimum (*i.e.*, a point $x \in M$ such that $\omega(x) = \inf_z \omega(z)$), and hence we will replace the infimum by a minimum in the update operator. This implies we can let the ϵ in Lemma 2.3 go to zero, and speak of the "optimum" algorithm, denoted OPT. (It is possible to construct metric spaces in which there is no optimum algorithm and no minimum cost.) We abbreviate $\min_z \omega(z)$ by $\min(\omega)$. The optimum cost of servicing ϱ starting at s is $cost_{\mathrm{OPT}}(s, \varrho) = \min(\omega_{s,\varrho})$.

2.1 Offset functions

The *offset operator* $\downarrow : W_{M,D} \to W_{M,D}$ is defined as follows:

$$(\downarrow\omega)(x) \;=\; \omega(x) - \min(\omega)$$

A function σ is an *offset function* if $\sigma = \downarrow\omega$ for some work function ω. Equivalently, σ is an offset function if it satisfies conditions (wf1) and (wf2) and $\min(\sigma) = 0$.

The *optimal offset function* is the function $\sigma_{s,\varrho} = \downarrow\omega_{s,\varrho}$. The optimal offset function tells how much more than the current minimum an optimal algorithm must have paid providing that its server is currently at point x.

It is generally easier to deal with offset functions than work functions. In our problems, offset functions are easily characterized.

2.2 Competitive analysis

We can prove that a given algorithm $\mathcal{A}$ is c-competitive by showing that $cost_{\mathcal{A}}(s, \varrho) \le c \cdot cost_{\mathrm{OPT}}(s, \varrho)$ for any initial positions s and request sequence ϱ.

Let $\varrho = r_1 r_2 \ldots r_m$, and for $1 \le i \le m$ let $\varrho_i = r_1 r_2 \ldots r_i$. In our proofs of competitiveness, we will use a *potential function* $\Phi : M \times M^* \to \mathbb{R}^+$. We will show that for each $1 \le i \le m$,

$$cost_{\mathcal{A}}(s, \varrho_i) - cost_{\mathcal{A}}(s, \varrho_{i-1}) + \Phi(s, \varrho_i) - \Phi(s, \varrho_{i-1}) \le c \cdot (cost_{\mathrm{OPT}}(s, \varrho_i) - cost_{\mathrm{OPT}}(s, \varrho_{i-1}))\,. \qquad (2)$$

Summing over i, this gives $cost_{\mathcal{A}}(s, \varrho) + \Phi(s, \varrho) - \Phi(s, \varepsilon) \le c \cdot cost_{\mathrm{OPT}}(s, \varrho)$. If $\Phi(s, \varrho) - \Phi(s, \varepsilon) \ge 0$, the competitiveness of $\mathcal{A}$ follows. We shall typically abbreviate Inequality (2) by

$$\Delta cost_{\mathcal{A}} + \Delta\Phi \;\le\; c \cdot \Delta cost_{\mathrm{OPT}}.$$

As mentioned above, we shall work with offset functions rather than work functions. We use the following property: if $\sigma = \downarrow\omega$ then

$$\min(\sigma \wedge r) = \min(\omega \wedge r) - \min(\omega). \qquad (3)$$

This property is easily verified using the definitions of update and offset. The current offset suffices, therefore, to compute $\Delta cost_{\text{OPT}}$ following a request.

The potential functions we use in this paper will in fact depend only on the server location and the current offset function, and thus we will view them as functions $\Phi : M \times W \to \mathbb{R}^+$.

3 Randomized Algorithms

In the next several sections we present randomized algorithms for a single edge, a tree, and any Cartesian product of trees under the $\mathcal{L}^1$ metric. In this section we give some preliminary definitions.

A randomized algorithm is a probability distribution on the set of all deterministic algorithms for a given problem. Equivalently, one can view a randomized algorithm as an algorithm that makes random choices at each step. Given a new request, such an algorithm determines a probability distribution of the new server location.

Finite distributions. A probability distribution μ on a set S is said to be *finite* if there exists a finite set $X \subseteq M$ such that $\mu(X) = \sum_{x \in X} \mu(x) = 1$. If μ is a finite distribution on S and $\mu(X) = 1$, we will also say that X *supports* μ. The smallest set that supports μ is called the *support of* μ. If the support of μ is a singleton, $\{x\}$, by an abuse of notation, we will identify x and μ, *e.g.*, we will write $\mu = x$. Throughout this paper, unless otherwise stated, we say "distribution" for short when we mean finite probability distribution.

If $\mu_1, \ldots, \mu_k$ are distributions, and $\sum_{i=1}^{k} p_i = 1$, then the linear combination

$$\mu = \sum_{i=1}^{k} p_i \mu_i$$

where $\mu(x) = \sum_{i=1}^{k} p_i \mu_i(x)$ for all x, is also a distribution.

We use two formalisms to describe randomized algorithms for an on-line problem. A *barely random* algorithm, $\mathcal{A}$, for a problem is a fixed distribution on a finite set of deterministic algorithms, $\{\mathcal{A}_0, \mathcal{A}_1, \ldots \mathcal{A}_N\}$, for that problem. $\mathcal{A}$ operates by first choosing $\mathcal{A}_i$ with probability p_i, then following $\mathcal{A}_i$ deterministically. Hence a random choice is made only once, during initialization. We denote a barely random algorithm constructed in this way by

$$\mathcal{A} = \sum_{i=1}^{N} p_i \mathcal{A}_i .$$

Some barely random algorithms are given in [21, 23].

On the other hand, a *distribution-specified* algorithm $\mathcal{A}$ is a deterministic function, where $\mathcal{A}(s, \varrho)$ is a distribution on a set of states for each start state s and request sequence ϱ. If μ is the distribution of the current state, and if r is a request, then the algorithm must choose, deterministically, a distribution ν of the next state. The method of transition from μ to ν is determined by a "minimal transport," defined below.

Transports. Suppose that μ and ν are distributions on sets A and B, respectively, and that a "distance" d_{xy} is given for every $x \in A$ and $y \in B$. We define a *transport* from μ to ν to be distribution α on $A \times B$ such that

$$\begin{aligned}\mu(x) &= \sum_y \alpha(x,y) \text{ for all } x \in A,\\ \nu(y) &= \sum_x \alpha(x,y) \text{ for all } y \in B.\end{aligned}$$

A transport describes a way for a randomized algorithm $\mathcal{A}$ to change the distribution of its state from μ to ν, by specifying for each pair of states (x,y) the probability that it starts at x and ends at y. To move between distributions μ and ν, the algorithm examines its actual state. If that state is x, then $\mathcal{A}$ moves to state y with probability $\alpha(x,y)/\mu(x)$.

We define

$$\begin{aligned}\langle\alpha\rangle &= \sum_x \sum_y d_{xy} \cdot \alpha(x,y), \quad \text{and}\\ \delta_{\mu\nu} &= \min\{\langle\alpha\rangle : \alpha \text{ is a transport from } \mu \text{ to } \nu\}.\end{aligned}$$

We call an α for which $\langle\alpha\rangle = \delta_{\mu\nu}$ a *minimal transport* from μ to ν. The value of $\delta_{\mu\nu}$ is called the *transport distance between μ and ν*. It can be calculated in polynomial time by a linear program.

For $x, y \in M$, note that d_{xy} (the distance from x to y) and δ_{xy} (the transport distance between distributions supported by $\{x\}$ and $\{y\}$) are equal.

We assume that any distribution-specified algorithm uses a minimum transport to move between distributions. In order to agree with the formal definition of a randomized algorithm, we must show how a distribution-specified randomized algorithm is a probability measure on the set of all deterministic algorithms. Let $\mathcal{A}$ be any distribution-specified randomized algorithm. For each real number $0 < t \le 1$, we define a deterministic algorithm $\mathcal{A}_t$, in such a manner that

$$\mathcal{A} = \int \mathcal{A}_t dt$$

That is, for any $0 \le p \le 1$, $\mathcal{A}$ follows $\mathcal{A}_t$ for some $t \le p$ with probability p.

The $\mathcal{A}_t$ are defined in such a way that if the probability is π that $\mathcal{A}$ moves from x to y, then $\mathcal{A}_t$ moves deterministically from x to y for all t in some set whose measure is π.

We define $\mathcal{A}_t$ as follows. Let s be the initial position and ϱ the sequence. of requests up to the present step. Let μ be the distribution of $\mathcal{A}$ before the next request at r, and ν is the distribution of

$\mathcal{A}$ after that request, and choose a minimal transport α from μ to ν. Let $U(s, \varrho, x, t)$ be the Lebesque measure of $\{0 < u \leq t \mid \mathcal{A}_u(s, \varrho) = x\}$. Let $\{y_1, \dots y_n\}$ be the support of ν. If $\mathcal{A}_t$ is in state x, before the request, then it moves to state y_i if and only if

$$\sum_{j=1}^{i-1} \alpha(x, y_j) < U(s, \varrho, x, t) \leq \sum_{j=1}^{i} \alpha(x, y_j)$$

Finally, we note that every barely random algorithm gives rise to a distribution-specified algorithm in a natural way. Let

$$\mathcal{A} = \sum_{i=1}^{N} p_i \mathcal{A}_i$$

be a barely random algorithm. Let $\mathcal{A}'$ be the distribution-specified algorithm defined as follows. If s is a start state and ϱ is a sequence of requests, let

$$\mathcal{A}'(s, \varrho) = \sum_{i=1}^{N} p_i \mathcal{A}_i(s, \varrho)$$

We call $\mathcal{A}'$ the distribution-specified algorithm *generated* by $\mathcal{A}$.

Lemma 3.1 *If $\mathcal{A}'$ is generated by a barely random algorithm $\mathcal{A}$, then $\Delta cost_{\mathcal{A}'} \leq \Delta cost_{\mathcal{A}}$*

Proof: Suppose $\mathcal{A} = \sum_{i=1}^{N} p_i \mathcal{A}_i$. Let s be the start state, ϱ the request sequence, and r the next request. Let $\mu = \sum_{i=1}^{N} p_i x_i$ and $\nu = \sum_{i=1}^{N} p_i y_i$, where $x_i = \mathcal{A}_i(s, \varrho)$ and $y_i = \mathcal{A}_i(s, \varrho r)$. Then

$$\Delta cost_{\mathcal{A}'} = \delta_{\mu\nu} \leq \sum_{i=1}^{N} p_i d_{x_i y_i} = \Delta cost_{\mathcal{A}}$$

since $\sum_{i=1}^{N} p_i(x_i, y_i)$ is a transport from μ to ν. ∎

4 Migration on Two Points

In this section we consider the page migration problem when the metric space M has only two points, x and y. We call such a space an *edge*. We give a randomized algorithm, EDGE, that is $(2 + \frac{1}{2D})$-competitive on any edge, and show that this constant is optimal.

The edge is a special case of the uniform space of n points, $\{x_1, x_2, \dots, x_n\}$, in which $d_{xy} = \ell$ for all $x \neq y$, ℓ a fixed constant. Since we are always free to change the scale of distance, we can assume without loss of generality that $\ell = 1$. The optimal work function can be written as a vector $(b_1, b_2, b_3, \dots, b_n)$, of non-negative integers, where $b_i = \omega_{s,\varrho}(x_i)$. The offsets derived from $\omega_{s,\varrho}$ have particularly simple form.

All costs are non-negative integers, and therefore the property (wf1) implies the following lemma.

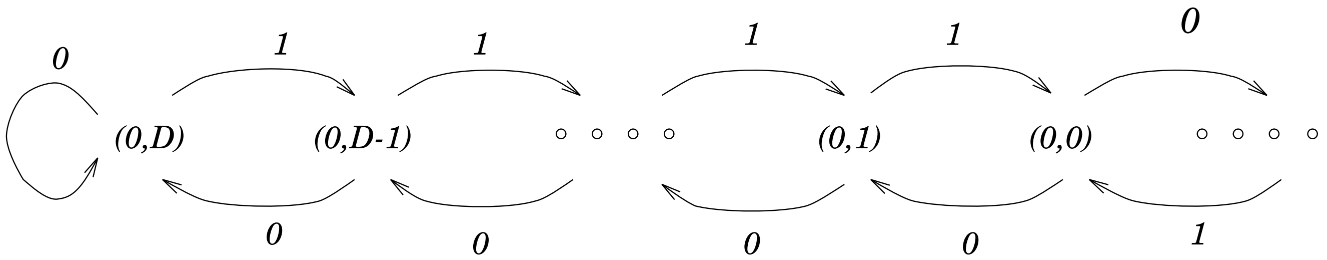

Figure 1: The offset functions for migration on two points. Each offset function is a pair of integers $(\omega(x), \omega(y))$, one of which is zero, the other between 0 and D. Only the part where $\omega(x) = 0$ is shown, the other part is symmetric. A label $z : c$, for $z = x, y$ and $c = 0, 1$, on an edge means that this edge corresponds to the change of the offset function when the request is on z and that the optimum cost of this move is c.

Lemma 4.1 *In the uniform space of size n, any optimal offset function $\sigma_{s,\varrho}$ has the following form: b_i is an integer in the range $[0, D]$, and at least one b_i is zero.*

In the case of the edge $\{x, y\}$, Lemma 4.1 implies that there are $2D - 1$ optimal offset functions, each of the form $(0, b)$ or $(b, 0)$, where b is an integer between 0 and D. Figure 1 shows how the offset function changes in response to a request.

Algorithm EDGE. Define the barely random algorithm

$$\text{EDGE} \;=\; \sum_{i=1}^{2D} \frac{1}{2D} \text{EDGE}_i$$

where the EDGE_i are deterministic algorithms defined as follows. Let s_i be the position of EDGE_i's server. If r is a request and not every EDGE_i has its server at r, pick the smallest j such that $s_j \neq r$. EDGE_j moves its server to r, while all other EDGE_i do not move their servers. By an abuse of notation, we shall also use EDGE to denote the distribution-specified randomized algorithm generated by this barely random algorithm. (The costs are the same.)

Let σ be the current offset function, and let $b = \sigma(y) - \sigma(x)$. Let $p_b = \frac{D+b}{2D}$. By simple induction, it is easy to see that algorithm EDGE maintains the following probability distribution:

$$\mu(x) \;=\; p_b \qquad\qquad \mu(y) \;=\; p_{(-b)} \;=\; 1 - p_b$$

Thus if the current offset function is $(0, b)$ then $\mu = (\frac{D+b}{2D}, \frac{D-b}{2D})$.

Let μ be the distribution prior to processing request r, and let ν be the distribution after processing request r. The service cost of the request is $\mu(y)$ if $r = x$, and $\mu(x)$ if $r = y$. The movement cost of the request will be D times the transport cost, $D \cdot \delta_{\mu\nu}$, which will be zero if $\mu = \nu$, a possibility which only occurs when $\sigma = (0, D)$ and $r = x$ or $\sigma = (D, 0)$ and $r = y$, and is $\frac{1}{2}$ in all other cases.

For example, if the request is at x, and if $\mu(x) = (D + b)/2D$ (with $b < D$), then the algorithm behaves as follows. If the server is currently at x, then it stays at x with probability 1. If the server is currently at y, then it is moved to x with probability $(1/2D)/((D - b)/2D) = 1/(D - b)$ and it stays at y with probability $1 - 1/(D - b)$.

Theorem 4.2 EDGE *is* C_D*-competitive for* $C_D = 2 + \frac{1}{2D}$.

Proof: We use the following potential function.

$$\Phi = \sum_{i=1}^{D-|b)|} \left(\frac{1}{2} + \frac{i}{2D} \right)$$

Since all definitions are symmetric with respect to x and y, we can assume, without loss of generality, that the current offset function is $(0, b)$. We analyze the change of the potential in two cases: when the request is on x, and when the request is on y. In both cases we need to show that

$$\Delta cost_{\text{EDGE}} + \Delta\Phi \leq C_D \cdot \Delta cost_{\text{OPT}}.$$

In fact, since it is useful in the lower bound proof later, we show the equality holds:

$$\Delta cost_{\text{EDGE}} + \Delta\Phi = C_D \cdot \Delta cost_{\text{OPT}}. \tag{4}$$

Equation (4) and Equation (3, Section 2.2) together imply that Inequality (2) of Section 2.2 holds. Since initially $\Phi = 0$, and $\Phi \geq 0$ for all offset functions, we can conclude that EDGE is $(2 + 1/2D)$-competitive.

Case 1: Request on x. The subcase $b = D$ is trivial, since all terms in (4) are zero. Suppose that $b \leq D - 1$. The new offset function is $(0, b + 1)$.

We have $\Delta cost_{\text{EDGE}} = \mu(y) + 1/2$, $\Delta\Phi = -(1/2 + (D - b)/2D)$, and $\Delta cost_{\text{OPT}} = 0$. Therefore

$$\Delta cost_{\text{EDGE}} + \Delta\Phi \;=\; \frac{D - b}{2D} + \frac{1}{2} - \left(\frac{1}{2} + \frac{D - b}{2D} \right) \;=\; 0 \;=\; C_D \cdot \Delta cost_{\text{OPT}}.$$

Case 2: Request on y. We can assume $b \geq 1$. (If $b = 0$ and the request is on y, just exchange the labels "x" and "y" and reduce to Case 1.) Then the new offset function is $(0, b - 1)$.

We have $\Delta cost_{\text{EDGE}} = \mu(x) + 1/2$, $\Delta\Phi = 1/2 + (D - b + 1)/2D$, and $\Delta cost_{\text{OPT}} = 1$. Therefore

$$\Delta cost_{\text{EDGE}} + \Delta\Phi \;=\; \frac{D + b}{2D} + \frac{1}{2} + \frac{1}{2} + \frac{D - b + 1}{2D} \;=\; 2 + \frac{1}{2D} \;=\; C_D \cdot \Delta cost_{\text{OPT}}.$$

This completes the proof. ∎

Theorem 4.3 *No randomized on-line algorithm for the page migration problem on two points can be c-competitive for any $c < C_D$.*

Proof: Suppose $\mathcal{A}$ is any randomized algorithm for the page migration problem on two points. We will define a potential function Ψ that assigns a value $\Psi(\mu, \omega) \in \mathbb{R}^+$ to each distribution μ and offset function ω. We will also give an adversary strategy for generating requests such that

(i) Ψ is bounded.

(ii) For request sequences generated by this adversary, the inequality

$$\Delta cost_{\mathcal{A}} + \Delta\Psi \;\geq\; C_D \cdot \Delta cost_{\mathrm{OPT}} \tag{5}$$

holds at each request.

(iii) The adversary can generate a sequence of arbitrarily high optimum cost.

We first show that the above three conditions suffice to prove the statement of the theorem.

Summing (5) over an adversary sequence gives

$$cost_{\mathcal{A}} + \Psi_f - \Psi_i \quad \geq \quad C_D \cdot cost_{\mathrm{OPT}}$$

where Ψ_i and Ψ_f are the initial and final values of Ψ, respectively. If A were c-competitive for $c < C_D$ then for some constant a, $cost_{\mathcal{A}} \leq c \cdot cost_{\mathrm{OPT}} + a$, so that

$$(C_D - c) \cdot cost_{\mathrm{OPT}} \quad \leq \quad a - \Psi_f + \Psi_i.$$

By (iii), the left side can be made arbitrarily large, contradicting the fact that the right side is bounded.

We now give a potential function Ψ and prove that it satisfies the three conditions. Let $b = \sigma(y) - \sigma(x)$ as before. Let q be the probability that $\mathcal{A}$'s server is at x. Let $\Lambda_{\mathcal{A}} = \mid q - p_b \mid$, and define

$$\Psi \quad = \quad \Phi + D \cdot \Lambda_{\mathcal{A}},$$

where Φ is the potential used in the upper bound analysis above. Note that Ψ is bounded, since both Φ and $\Lambda_{\mathcal{A}}$ are bounded.

We first remark that we can assume that if the offset function is a characteristic function at a point, $\mathcal{A}$ will be concentrated at that point, according to a theorem of [16]. Let the adversary make requests as follows:

- If $q < p_b$, the adversary requests x.
- If $q > p_b$, the adversary requests y.

- If $b = D$, we can assume that $q = 1$. The adversary requests y.
- If $b = -D$, we can assume that $q = 0$. The adversary requests x.
- If $\mid b \mid < D$ and $q = p_b$, the adversary requests either x or y arbitrarily.

We now verify (5). Using (4) and the definition of Ψ, (5) is equivalent to

$$\Delta cost_{\mathcal{A}} + D \cdot \Delta\Lambda_{\mathcal{A}} \;\geq\; \Delta cost_{\text{EDGE}}, \tag{6}$$

where $\Delta\Lambda_{\mathcal{A}}$ is the change in $\Lambda_{\mathcal{A}}$.

Case 1: The adversary requests x. In this case the new offset function is $(0, b+1)$. Suppose that after the request $\mathcal{A}$ has mass q' at x. Then $cost_{\mathcal{A}} = 1 - q + D\,|q - q'|$, $D \cdot \Delta\Lambda_{\mathcal{A}} = D\,|q' - p_{b+1}| - D\,|q - p_b|$, and $cost_{\text{EDGE}} = 1 - p_b + 1/2$. Since $q < p_b < p_{b+1}$,

$$\begin{aligned}
\Delta cost_{\mathcal{A}} + D \cdot \Delta\Lambda_{\mathcal{A}} &= 1 - q + D\,|q - q'| + D\,|q' - p_{b+1}| - D\,|q - p_b| \\
&\geq 1 - q + D\,|q - p_{b+1}| - D\,|q - p_b| \\
&= 1 - q + D\,(p_{b+1} - p_b) \\
&= 1 - q + 1/2 \\
&\geq 1 - p_b + 1/2 \\
&= \Delta cost_{\text{EDGE}}
\end{aligned}$$

Case 2: The adversary requests y. In this case the new offset function is $(0, b - 1)$. Suppose that after the request $\mathcal{A}$ has mass q' at x. Then $cost_{\mathcal{A}} = q + D\,|q - q'|$, $D \cdot \Delta\Lambda_{\mathcal{A}} = D\,|q' - p_{b-1}| - D\,|q - p_b|$, and $cost_{\text{EDGE}} = p_b + 1/2$. Since $q \geq p_b > p_{b-1}$,

$$\begin{aligned}
\Delta cost_{\mathcal{A}} + D \cdot \Delta\Lambda_{\mathcal{A}} &= q + D\,|q - q'| + D\,|q' - p_{b-1}| - D\,|q - p_b| \\
&\geq q + D\,|q - p_{b-1}| - D\,|q - p_b| \\
&= q + D\,(p_b - p_{b-1}) \\
&= q + 1/2 \\
&\geq p_b + 1/2 \\
&= \Delta cost_{\text{EDGE}}
\end{aligned}$$

Finally, we must show that the adversary will generate sequences of arbitrarily large optimal cost. At each step, $\Delta cost_{\text{OPT}} = 1$ if $|b|$ increases, and is 0 if it decreases. (The way we define our adversary, b must change at each step.) Since its maximum value is D, in any request sequence of length n the value of $|b|$ must increase at least $(n - D)/2$ times, and thus $cost_{\text{OPT}} \geq (n - D)/2$, which can be made arbitrarily large by choosing sufficiently large n. ▮

5 Uniform Spaces

In this section, we define a distribution-specified randomized algorithm, UNIFORM, which is a generalization of EDGE, for any uniform space, *i.e.*, a metric space M where $d_{xy} = 1$ for all distinct pairs $x, y \in M$.

Let σ be the current offset function. For each $x \in M$ we define its *weight* as

$$k_\sigma(x) \;=\; \frac{D - \sigma(x)}{D + \sigma(x)}.$$

We let UNIFORM have the distribution

$$\mu(x) \;=\; \frac{k_\sigma(x)}{\sum_{y \in M} k_\sigma(y)}.$$

Note that if M has only two points, UNIFORM is EDGE.

Theorem 5.1 *In a uniform space,* UNIFORM *is* $(2 + \frac{1}{2D})$*-competitive for* $D = 1, 2$.

Proof: First we give the proof for $D = 1$. By Lemma 4.1, the offset function σ is uniquely defined by the number of points a_σ of offset 0. UNIFORM is simply the uniform distribution on those points. We define the potential by

$$\Phi(\sigma) \;=\; \Phi_a \;=\; 2(a-1)/a,$$

where $a = a_\sigma$. We must show that in every move

$$\Delta cost_{\mathrm{UNIFORM}} + \Delta\Phi \;\le\; \frac{5}{2} \cdot \Delta cost_{\mathrm{OPT}}. \tag{7}$$

Let σ be the current offset function, let $a = a_\sigma$, and let r be the new request point. We have two cases.

Case 1: $\sigma(r) = 0$. The new offset function $\downarrow(\sigma \wedge r)$ has one point of offset 0, namely r. Then $\Delta cost_{\mathrm{OPT}} = 0$, $\Delta cost_{\mathrm{UNIFORM}} = 2(a-1)/a$, $\Delta\Phi = -2(a-1)/a$, and therefore (7) holds.

Case 2: $\sigma(r) = 1$. Then the new offset function has $a+1$ points of offset 0. Thus we have $\Delta cost_{\mathrm{OPT}} = 1$, $\Delta cost_{\mathrm{UNIFORM}} = 1 + 1/(a+1)$, and $\Delta\Phi = 2a/(a+1) - 2(a-1)/a$. Thus we have

$$\Delta cost_{\mathrm{UNIFORM}} + \Delta\Phi \;=\; 1 + \frac{1}{a+1} + \frac{2}{a(a+1)} \;\le\; 1 + \frac{1}{2} + 1 \;=\; \frac{5}{2} \cdot \Delta cost_{\mathrm{OPT}}.$$

This completes the proof for $D = 1$.

Now we sketch the proof for $D = 2$. Each offset function σ is defined by two sets: the set of points of offset 0, and the set of points of offset 1. Let a_σ and b_σ denote, respectively, the cardinalities of those sets.

Given an offset function σ, such that $a_\sigma = a$ and $b_\sigma = b$, UNIFORM's distribution, μ, is specified as follows:

$$\mu(x) = \begin{cases} \frac{3}{3a+b} & \text{if } \sigma(x) = 0 \\ \\ \frac{1}{3a+b} & \text{if } \sigma(x) = 1 \\ \\ 0 & \text{if } \sigma(x) = 2 \end{cases}$$

We define a potential function as follows:

$$\Phi_{a,b} = \begin{cases} 4 - \frac{3}{a+2} - \frac{9}{3a+b} & \text{if } b \leq 6 \\ \\ 4 - \frac{9}{a+2} + \frac{2b-3}{3a+b} & \text{if } b \geq 6. \end{cases}$$

We must show that, for every move

$$\Delta cost_{\text{UNIFORM}} + \Delta\Phi \leq \frac{9}{4} \cdot \Delta cost_{\text{OPT}}.$$

Verification of this inequality involves elementary though tedious calculations, and is left to the interested reader. ▌

Conjecture: UNIFORM is $(2 + \frac{1}{2D})$-competitive for each $D \geq 3$.

Subsequent to this work, Lund, *et al.* [17] constructed a randomized algorithm for the uniform case that is C_D-competitive for all D. Their algorithm is a different generalization of the two point algorithm.

6 Trees

We define a finite metric space M to be a *tree* if its points are the vertices of an acyclic connected graph whose edges have nonnegative weights, and its distances are the weighted path lengths in that graph. Given such an M, there is just one such graph, and we will freely use graph terminology (such as "edge") for M when we really mean the associated weighted graph. We will denote an edge of M between vertices $x, y \in M$ by $\{x, y\}$, and the set of all edges of M by E. For $x, y \in M$, we denote the set of vertices that are on the unique path in M from x to y by $[x, y]$.

In this section we present a $(2 + \frac{1}{2D})$-competitive algorithm, FACTOR, for trees. In the next section we show that this algorithm can be extended to products of trees, that is metric spaces that are Cartesian products of trees with the $\mathcal{L}^1$ metric.

Algorithm FACTOR is based on a technique called "factoring" that divides one instance of the page migration problem on a tree M to a number of instances of the problem on two-point spaces, one for each edge of M.

We refer to the page migration problem on M as PMP^M. PMP^M "induces" in a natural way an on-line problem PMP^e on each edge $e = \{x, y\}$, which we view as space of two points at distance d_{xy}. The removal of e disconnects M into two subtrees M_x and M_y, containing x and y, respectively. Let $F^e : M \to e = \{x, y\}$ take M_x to x and M_y to y. For convenience we abbreviate $F^e(z)$ by z^e. The induced request sequence ϱ^e is given by $F^e(\varrho) = r_1^e r_2^e \ldots r_m^e$. An algorithm $\mathcal{A}$ on M induces an algorithm $\mathcal{A}^e$ in PMP^e as follows: Whenever $\mathcal{A}$ moves its server from u to v, $\mathcal{A}^e$ moves a server from u^e to v^e.

Lemma 6.1 *For any request sequence, ϱ, and initial position s, $cost_{\mathcal{A}}(s, \varrho) = \sum_e cost_{\mathcal{A}^e}(s^e, \varrho^e)$.*

Proof: For a move from u to v, $\mathcal{A}$ pays $D \cdot d_{uv}$ while $\mathcal{A}^e$, for $e = \{x, y\}$, pays $D \cdot d_{xy}$ if $[x, y] \subseteq [u, v]$ and zero otherwise. Thus for any movement of the server, the cost incurred by $\mathcal{A}$ is the sum of all the movement costs incurred by the $\mathcal{A}^e$. Similarly, any remote access cost incurred by $\mathcal{A}$ is the sum of all the remote access costs incurred by the $\mathcal{A}^e$. ▌

Corollary 6.2 *If each $\mathcal{A}^e$ is c-competitive then $\mathcal{A}$ is c-competitive.*

Proof: By Lemma 6.1 the cost of an optimal algorithm OPT for PMP^M is the sum of the costs of the induced algorithms OPT^e. The cost of each OPT^e, however, can be no less than the optimum cost in PMP^e, and $\mathcal{A}^e$ is c-competitive compared to that optimum cost. ▌

Comment: Note that in the general case, when we allow positive additive constants in the definition of competitiveness (see Inequality (1)), the additive constant of $\mathcal{A}$ will be the sum of the additive constants of all the $\mathcal{A}^e$, and thus it can depend on the size of the tree. However, in our application all $\mathcal{A}^e$ will have additive constant zero, and thus so will $\mathcal{A}$.

As suggested by Lemma 6.1 and Corollary 6.2, we will construct FACTOR such that each FACTOR^e is a copy of EDGE. This will show that FACTOR is C_D-competitive.

Define the barely random algorithm

$$\text{FACTOR} = \sum_{i=1}^{2D} \frac{1}{2D} \text{FACTOR}_i$$

where the FACTOR$_i$ are deterministic algorithms defined as follows. Let s_i be the position of FACTOR$_i$'s server. Let r be a request, and let T denote a continuous subtree of M. Pick $\hat{s}_i$ according to the following program:

```
$\hat{s}_1 \leftarrow r$
$T \leftarrow [s_1, \hat{s}_1]$
for $i$ from 2 to $2D$ do
    $\hat{s}_i \leftarrow$ the nearest point to $s_i$ in T
    $T \leftarrow T \bigcup [s_i, \hat{s}_i]$
```

Each FACTOR$_i$ then moves its server to $\hat{s}_i$.

Theorem 6.3 *If M is a tree then* FACTOR *is C_D-competitive on M, for $C_D = 2 + \frac{1}{2D}$.*

Proof: FACTOR^e_i = EDGE$_i$ for every edge e and all $i = 1, \ldots 2D$. Thus FACTOR^e = EDGE. By Corollary 6.2, we are done. ∎

FACTOR can be extended to infinite trees and to dynamic trees. By a *dynamic tree* we mean a tree that can grow branches during the request sequence. More precisely, at every step a request r is given by (a) choosing an edge $\{x, y\}$ of T and putting a branch point w on $[x, y]$ (so that $d_{xy} = d_{xw} + d_{wy}$), (b) attaching r to T via w, and (c) specifying the length rw of the new edge $\{r, w\}$. Let T' be the tree obtained from T after executing (a)-(c). FACTOR then satisfies the request at r according to its normal algorithm.

Let T be the initial tree, and consider the first time that it is modified. Let ϱ be the request sequence up to this time. Assume we replace $\{x, y\}$ by edges $\{x, w\}$, $\{w, y\}$, and $\{w, r\}$ as above.

The actions performed by FACTOR$_i$ on ϱ would have been the same had the new edges always been present. Hence the cost incurred by FACTOR can be no more than $2 + 1/2D$ times the optimal cost to serve ϱ starting with T' rather than T. On the other hand, the optimal cost to satisfy ϱ starting with T' can be no more than the optimal cost starting with T, since any move that can take place in T can also take place T', and the distances in T' are no more than the distances in T. Extending this argument inductively on the sequence of dynamic modifications, we conclude that FACTOR is $(2 + 1/2D)$-competitive on dynamic trees. Since FACTOR does not need to know the entire tree to run correctly, an infinite tree can be treated as a dynamic tree.

7 Cartesian Products

In this section we show that FACTOR can be used to design an optimally competitive randomized algorithm for any product of trees, by using one instance of FACTOR in each dimension [6]. The most common examples of such product topologies include hypercubes and meshes.

For $i = 1, 2$, let M_i be a metric space with distance function d^i. The Cartesian product set $M_1 \times M_2$ has a natural metric d defined by

$$d_{(x,y)(u,v)} \quad = \quad d^1_{xu} + d^2_{yv}$$

We refer to d as the $\mathcal{L}^1$ *metric* on $M_1 \times M_2$.

If $\mathcal{A}_i$ is an algorithm for the page migration problem on M_i, we define an algorithm $\mathcal{A}_1 \times \mathcal{A}_2$ for the page migration problem on $M_1 \times M_2$ as follows. Let (s_1, s_2) be the start position of the server in the product space. Let $\varrho = (r^1_1, r^1_2) \ldots (r^n_1, r^n_2)$ be any request sequence in $M_1 \times M_2$. For $i = 1, 2$, let $x^1_i \ldots x^n_i$ be the service of the request sequence $r^1_i \ldots r^n_i$ in M_i using algorithm $\mathcal{A}_i$, starting at s_i. Then

$\mathcal{A}_1 \times \mathcal{A}_2$ serves ϱ with the service sequence $(x_1^1, x_2^1) \ldots (x_1^n, x_2^n)$. The Cartesian product construction works for both deterministic and randomized algorithms, *i.e.*, if each $\mathcal{A}_i$ is deterministic, $\mathcal{A}_1 \times \mathcal{A}_2$ is deterministic, while if each $\mathcal{A}_i$ is randomized, $\mathcal{A}_1 \times \mathcal{A}_2$ is randomized. We leave it to the reader to verify that if $\mathcal{A}_i$ is c-competitive for each i, then $\mathcal{A}_1 \times \mathcal{A}_2$ is c-competitive.

The product construction for two metric spaces and on-line algorithms for the page migration problem extends naturally to products of finitely many spaces. Thus we obtain the following result.

Theorem 7.1 *If M is a Cartesian product of finitely many trees, then there is a C_D-competitive randomized algorithm for the page migration problem on M.*

In particular, Theorem 7.1 holds for hypercubes and meshes, since a hypercube is the N-fold product of two-point spaces, and a mesh is the Cartesian product of linear graphs (finite subsets of the real line).

8 Deterministic Algorithms

In this section we consider deterministic algorithms. First we will give an optimal $(2+\frac{1}{2D})$-competitive algorithm for the vector space $\mathbb{R}^n$, which we call BARY, the *barycentric* algorithm. We then generalize BARY to an arbitrary continuous tree.

Next, we will prove that there is no general deterministic algorithm for the page migration problem with competitive constant better than $\frac{85}{27}$, for $D = 1$. Finally, we will give an optimal 3-competitive algorithm for three points and $D = 1$.

8.1 Page Migration in $\mathbb{R}^n$

A point $\mathbf{x} \in \mathbb{R}^n$ is an n-dimensional real-valued vector. Fix an arbitrary norm $\|\cdot\|$ in $\mathbb{R}^n$. Then $d_{\mathbf{xy}} = \|\mathbf{x} - \mathbf{y}\|$ is a distance function.

To construct deterministic algorithms for $\mathbb{R}^n$ we use a technique called "barycentric simulation." If μ is any distribution on $\mathbb{R}^n$, define the *barycenter of* μ as

$$\bar{\mu} \;=\; \sum_{\mathbf{x}} \mathbf{x}\mu(\mathbf{x})$$

Lemma 8.1 *Let μ and ν be two distributions on $\mathbb{R}^n$. Then $d_{\bar{\mu}\bar{\nu}} \le \delta_{\mu\nu}$.*

Proof: Let α be any transport from μ to ν. We have

$$\begin{aligned}
\|\alpha\| &= \sum_{\mathbf{x}} \sum_{\mathbf{y}} \|\mathbf{x} - \mathbf{y}\| \alpha(\mathbf{x}, \mathbf{y}) \\
&= \sum_{\mathbf{x}} \sum_{\mathbf{y}} \|(\mathbf{x} - \mathbf{y}) \alpha(\mathbf{x}, \mathbf{y})\|
\end{aligned}$$

$$\begin{aligned}
&\geq \left\| \sum_{\mathbf{x}} \sum_{\mathbf{y}} (\mathbf{x} - \mathbf{y}) \alpha(\mathbf{x}, \mathbf{y}) \right\| \\
&= \left\| \sum_{\mathbf{x}} \mathbf{x} \sum_{\mathbf{y}} \alpha(\mathbf{x}, \mathbf{y}) - \sum_{\mathbf{y}} \mathbf{y} \sum_{\mathbf{x}} \alpha(\mathbf{x}, \mathbf{y}) \right\| \\
&= \left\| \sum_{\mathbf{x}} \mathbf{x} \mu(\mathbf{x}) - \sum_{\mathbf{y}} \mathbf{y} \nu(\mathbf{y}) \right\| \\
&\geq \|\bar{\mu} - \bar{\nu}\|.
\end{aligned}$$

The third step follows from the triangle inequality on the norm. ▮

Theorem 8.2 *Let $\mathcal{A}$ be a randomized algorithm for $\mathbb{R}^n$ that maintains a finite distribution on the location of its server and is c-competitive against an oblivious adversary. Then there exists a c-competitive deterministic algorithm,* BARY($\mathcal{A}$)*, for $\mathbb{R}^n$.*

Proof: BARY($\mathcal{A}$) simulates $\mathcal{A}$ on the request sequence and computes the distribution μ of the server of $\mathcal{A}$ on $\mathbb{R}^n$. BARY($\mathcal{A}$) keeps its server on $\bar{\mu}$, the barycenter of μ.

By Lemma 8.1, the movement cost of BARY($\mathcal{A}$) cannot exceed the expected movement cost of $\mathcal{A}$ at any given step. If r is the request point, we can consider r to be a distribution supported by $\{r\}$, and the barycenter of that distribution is trivially r. Thus, by Lemma 8.1, the remote service cost of BARY($\mathcal{A}$) cannot exceed the expected remote service cost of $\mathcal{A}$, at any given step. ▮

Theorem 8.2 indicates that randomization cannot help improve competitiveness on $\mathbb{R}^n$. On the other hand, it allows us to construct several deterministic algorithms using known randomized algorithms.

Theorem 8.3 *There is a $(2+\frac{1}{2D})$-competitive deterministic barycentric algorithm for page migration in $\mathbb{R}^n$ under the $\mathcal{L}^1$ metric.*

Proof: For $n = 1$, this theorem follows from a barycentric simulation of FACTOR in the real line. We first note that if requests are made in $\mathbb{R}^1$, then the dynamic tree constructed in the manner described in Section 6 can be uniquely embedded in $\mathbb{R}^1$. Thus, it makes sense to refer to the algorithm FACTOR on $\mathbb{R}^1$.

Second, we show that EDGE is $(2 + 1/2D)$-competitive on the real line segment $[0, 1]$, under the restriction that all requests occur at endpoints. Algorithm EDGE is well-defined: it always keeps its server at either endpoint according to the desired probability. We must ensure that the optimum cost on such a sequence cannot be decreased by placing the server at positions inside the interval. By induction on the request sequence, one may show that the optimum work function has the form

$$\omega(x) = \min \left\{ \begin{array}{c} \omega(0) + D \cdot d_{x0} \\ \omega(1) + D \cdot d_{x1} \end{array} \right\}$$

Hence the minimum value of the work function always occurs at the endpoint, and the update costs are the same as in the two point case. Therefore algorithm EDGE is $(2 + 1/2D)$-competitive on the line segment, which implies FACTOR is $(2 + 1/2D)$-competitive on $\mathbb{R}^1$. The competitiveness of BARY(FACTOR) follows by Theorem 8.2.

To extend this to $\mathbb{R}^n$ under the $\mathcal{L}^1$ metric, simply observe that $\mathbb{R}^n$ is a n-fold Cartesian product of $\mathbb{R}^1$ and apply Theorem 7.1. ▌

Theorem 8.4 *There is a b_D-competitive deterministic barycentric algorithm for page migration in $\mathbb{R}^n$, where b_D is a function defined in* [23] *that tends to $1 + \phi$ as D grows large.*

Proof: Westbrook's randomized algorithm [23] is b_D-competitive competitive in any metric space. A barycentric simulation yields a deterministic algorithm with equivalent competitive ratio. Some of the values are $b_1 = 3$, $b_2 = 2.75$, $b_3 \approx 2.667$, $b_4 \approx 2.667$,and $b_D \to 1 + \phi \approx 2.618$ as D grows large. ▌

8.2 An Optimal Deterministic Algorithm for Continuous Trees

In this section we generalize BARY to an arbitrary "continuous tree." Let T be a tree in which each edge e has some length $0 \leq l(e) \leq \infty$. Consider each edge e of T to be an interval of length $l(e)$ and consider the union of these edges, attached according to T. We call this space the *continuous tree based on T*. Any space which is obtained from some edge-weighted tree in this fashion is called a continuous tree. Let $[x, y]$ denote the set of points $w \in M$ that lie on the path between x and y, *i.e.* $[x, y] = \{w \mid d_{xw} + d_{wy} = d_{xy}\}$.

We first note that if requests are made in a continuous tree M, then the dynamic tree constructed in the manner described in Section 6 can be uniquely embedded in M. Thus, it makes sense to refer to the algorithm FACTOR on M. In the rest of this section, we define the barycentric algorithm BARY for any continuous tree M, and prove that it is $(2 + \frac{1}{2D})$-competitive.

The goal of barycentric simulation is to find a point $\bar{\mu}$ such that for all $z \in M$, $d_{\bar{\mu}z} \leq \delta_{\mu z}$. It is relatively easy to find such a point in $\mathbb{R}^n$, but continuous trees are somewhat more complicated. We begin by considering a simple case.

Let weights $w_1, w_2 \in \mathbb{R}^+$ be placed at points $u, v \in M$, respectively. We define the *barycenter of u and v with respect to weights w_1 and w_2* to be the unique point x such that $d_{ux} = d_{uv} \cdot w_2/(w_1 + w_2)$ and $d_{xv} = d_{uv} \cdot w_1/(w_1 + w_2)$.

Lemma 8.5 *Suppose $w_1, w_2 \in \mathbb{R}^+$, and u, v, u', v' are points in M. Let x be the barycenter of u and v with respect to w_1 and w_2, and x' be the barycenter of u' and v' with respect to w_1 and w_2. Then $(w_1 + w_2) \cdot d_{xx'} \leq w_1 \cdot d_{uu'} + w_2 \cdot d_{vv'}$.*

Proof: Without loss of generality we can assume that $w_1 + w_2 = 1$, since otherwise we can normalize the weights. It is also sufficient to prove that if x'' is the barycenter of u', v with weights w_1, w_2 then $d_{xx''} \leq w_1 d_{uu'}$, because then the symmetric inequality $d_{x''x'} \leq w_2 d_{vv'}$ and the triangle inequality imply the lemma.

Let y be the point in $[u, u']$ nearest v, *i.e.*, y is the branch point of the unique Y-shaped subset of M whose three end points are u, u', and v. Then either $x \in [u, y]$ or $x \in [y, v]$, while either $x'' \in [u', y]$ or $x'' \in [y, v]$. If $x \in [u, y]$ and $x'' \in [u', y]$, then

$$\begin{aligned} d_{xx''} &= d_{uu'} - d_{ux} - d_{u'x'} \\ &= d_{uu'} - w_2 d_{uv} - w_2 d_{u'v} \\ &\leq d_{uu'} - w_2 d_{uu'} \\ &\leq w_1 d_{uu'} \end{aligned}$$

On the other hand, if $x'' \in [y, v]$, then $x, x'' \in [u, v]$, hence

$$\begin{aligned} d_{xx''} &= |d_{xv} - d_{x''v}| \\ &= w_1 |d_{uv} - d_{u'v}| \\ &\leq w_1 d_{uu'} \end{aligned}$$

The remaining case, $x' \in [y, v]$, is similar. ▮

For $n > 2$, let weights $w_1, w_2, \ldots, w_n \in \mathbb{R}^+$ be placed at points $u_1, u_2, \ldots, u_n \in M$, respectively, and let u_2' be the barycenter of u_1 and u_2 with respect to weights w_1 and w_2. We define the *barycenter of* $u_1, u_2, \ldots, u_n$ *with respect to weights* $w_1, w_2, \ldots, w_n$ as the barycenter of $u_2', u_3, \ldots, u_n$ with respect to weights $w_1 + w_2, w_3, \ldots, w_n$, recursively. If $w_1 = w_2 = \ldots = w_n$, we say simply "the barycenter of $u_1, u_2, \ldots, u_n$" for short. Notice that the barycenter depends on the order of the points $u_1, u_2, \ldots, u_n$. As an example, the reader may wish to consider a star with three edges of lengths 1, 2, and 3, respectively, and weight 1 on each leaf.

Lemma 8.6 *Let* $w_1, w_2, \ldots, w_n$ *be weights in* $\mathbb{R}^+$*, and let* $u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n$ *be points in* M*. If* $\bar{u}$ *is the barycenter of* $u_1, u_2, \ldots, u_n$ *with respect to* $w_1, w_2, \ldots, w_n$*, and* $\bar{v}$ *is the barycenter of* $v_1, v_2, \ldots, v_n$ *with respect to* $w_1, w_2, \ldots, w_n$*, then* $(\sum_{i=1}^n w_i) \cdot d_{\bar{u}\bar{v}} \leq \sum_{i=1}^n w_i \cdot d_{u_i v_i}$.

Proof: We use induction on n. The $n = 1$ case is trivial, and the $n = 2$ case is Lemma 8.5.

Suppose $n > 2$. Let u_2' be the barycenter of u_1 and u_2 with respect to weights w_1 and w_2, and let v_2' be the barycenter of v_1 and v_2 with respect to weights w_1 and w_2. Since $\bar{u}$ is the barycenter of $u_2', u_3, \ldots, u_n$ with respect to $w_1 + w_2, w_3, \ldots, w_n$, and $\bar{v}$ is the barycenter of $v_2', v_3, \ldots, v_n$ with respect to $w_1 + w_2, w_3, \ldots, w_n$ then by induction,

$$\left(\sum_{i=1}^n w_i\right) \cdot d_{\bar{u}\bar{v}} \leq (w_1 + w_2) d_{u_2' v_2'} + \sum_{i=3}^n w_i \cdot d_{u_i v_i}$$

$$\begin{aligned} &\leq w_1 \cdot d_{u_1 v_1} + w_2 \cdot d_{u_2 v_2} + \sum_{i=3}^{n} w_i \cdot d_{u_i v_i} \\ &= \sum_{i=1}^{n} w_i \cdot d_{u_i v_i}. \end{aligned}$$

The second inequality follows from Lemma 8.5. ∎

Lemma 8.7 *Let $w_1, w_2, \ldots, w_n$ be weights in $\mathbb{R}^+$, and let $u_1, u_2, \ldots, u_n$ be points in M. If $\bar{u}$ is the barycenter of $u_1, u_2, \ldots, u_n$ with respect to $w_1, w_2, \ldots, w_n$, then for all $z \in M$, $(\sum_{i=1}^{n} w_i) \cdot d_{\bar{u}z} \leq \sum_{i=1}^{n} w_i \cdot d_{u_i z}$.*

Proof: This follows from Lemma 8.6 by setting each $v_i = z$. ∎

We now construct a barycentric simulation of FACTOR. Recall that FACTOR simulates the actions of $2D$ different deterministic algorithms, FACTOR$_i$ for $i = 1, \ldots 2D$, and chooses one at random to actually execute. Let s_i be the position of FACTOR$_i$'s server. Then BARY(FACTOR)'s server will be at the barycenter of $s_1, \ldots s_{2D}$.

Theorem 8.8 BARY(FACTOR) *is $(2 + \frac{1}{2D})$-competitive on any continuous tree.*

Proof: By Lemma 8.6 the movement cost of BARY cannot exceed the expected movement cost of FACTOR. By Lemma 8.7, the remote access cost of BARY cannot exceed the expected remote access cost of FACTOR. ∎

Finally, we show BARY(FACTOR) achieves the best competitive ratio possible on a continuous tree.

Theorem 8.9 *Let $\mathcal{A}$ be an algorithm, randomized or deterministic, for the page migration problem in the interval $[0, 1]$. Then $\mathcal{A}$ is no better than $(2 + 1/2D)$-competitive.*

Proof: By Theorem 8.2, we may assume $\mathcal{A}$ is deterministic. Consider only sequences consisting of requests on points 0 and 1. Let s denote the position of $\mathcal{A}$'s server. Define a randomized algorithm, $\mathcal{B}$, that keeps its server only on points 0 or 1 according to the following distribution μ: $\mu(0) = d_{s1}$ and $\mu(1) = d_{s0}$. Observe that $\mathcal{A}$ is exactly BARY($\mathcal{B}$). It is easy to verify (just reversing the proof of Lemma 8.5) that for request of point 0 or 1, the cost to $\mathcal{A}$ equals the expected cost to $\mathcal{B}$. But by Theorem 4.3, $\mathcal{B}$ is no better than $2 + \frac{1}{2D}$-competitive. ∎

8.3 A Lower Bound

Black and Sleator [6] conjectured the existence of a 3-competitive deterministic page migration algorithm for all metric spaces and for all values of D. In this section we disprove this conjecture by demonstrating that for $D = 1$ no deterministic page migration algorithm for general metric spaces is c-competitive for any $c < \frac{85}{27}$.

Lemma 8.10 *There exists a 4-point metric space, K, and an initial server position, $x \in K$, such that for any deterministic competitive on-line algorithm $\mathcal{A}$ there is a request sequence ϱ with the following properties:*

1. *$cost_{\mathcal{A}}(x, \varrho) \geq \frac{85}{27} cost_{\mathrm{OPT}}(x, \varrho)$ and $cost_{\mathrm{OPT}}(x, \varrho) \geq 1$;*
2. *$\mathcal{A}(x, \varrho) = \mathrm{OPT}(x, \varrho) = u$. That is, after serving ϱ, both $\mathcal{A}$ and the optimal off-line algorithm end with the server at the same point, u.*

Proof: The finite space K has four points x, y, z, t with distances: $d_{xy} = 7$, $d_{xz} = 20$, $d_{xt} = 10$, $d_{yz} = 14$, $d_{yt} = 17$ and $d_{zt} = 10$ (See Fig. 2.)

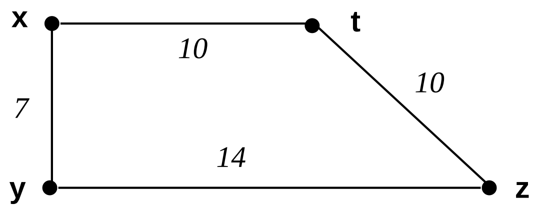

Figure 2: Metric space used in the lower bound proof. Diagonal lengths are "shortest path" distances.

We now give an adversarial strategy for picking a request sequence ϱ given that the server is initially at x. The adversarial strategy is illustrated by the tree in Figure 3, where notation $v : r$ means that in response to $\mathcal{A}$ moving its server to v, the adversary makes the next request on r. We give a description of the strategy below. The notation rr^+ means that the adversary generates two or more requests to point r until $\mathcal{A}$ moves the server to r. By a theorem of [16], we may assume that $\mathcal{A}$ eventually moves the server to r.

1. Request y. If the on-line server stays at x, go to next step. If the on-line server moves to y, z or t, request xx^+ and halt. The optimum cost is 7, and $\mathcal{A}$'s cost is at least 28.
2. Request z. If the server moves to y, go to next step. If the server moves to z or t, request yy^+ and halt; the optimum cost is 21, and $\mathcal{A}$'s cost is at least 71. If the server stays at x, request zz^+ and halt; the optimum cost is 21, and $\mathcal{A}$'s cost is at least 67.
3. Request t. If the server moves to t, go to next step. If the server moves to x, request zz^+ and halt; the optimum cost is 31, and $\mathcal{A}$'s cost is at least 98. If the server stays at y or moves to z, request tt^+ and halt; the optimum cost is 27, and $\mathcal{A}$'s cost is at least 85.
4. Request z. If the server moves to z, go to next step. Otherwise, request zz^+ and halt; the optimum cost is 31, and $\mathcal{A}$'s cost is at least 98.
5. Request t. If the server moves to t, go to next step. Otherwise, request tt^+ and halt; the optimum cost is 37, and $\mathcal{A}$'s cost is at least 118.

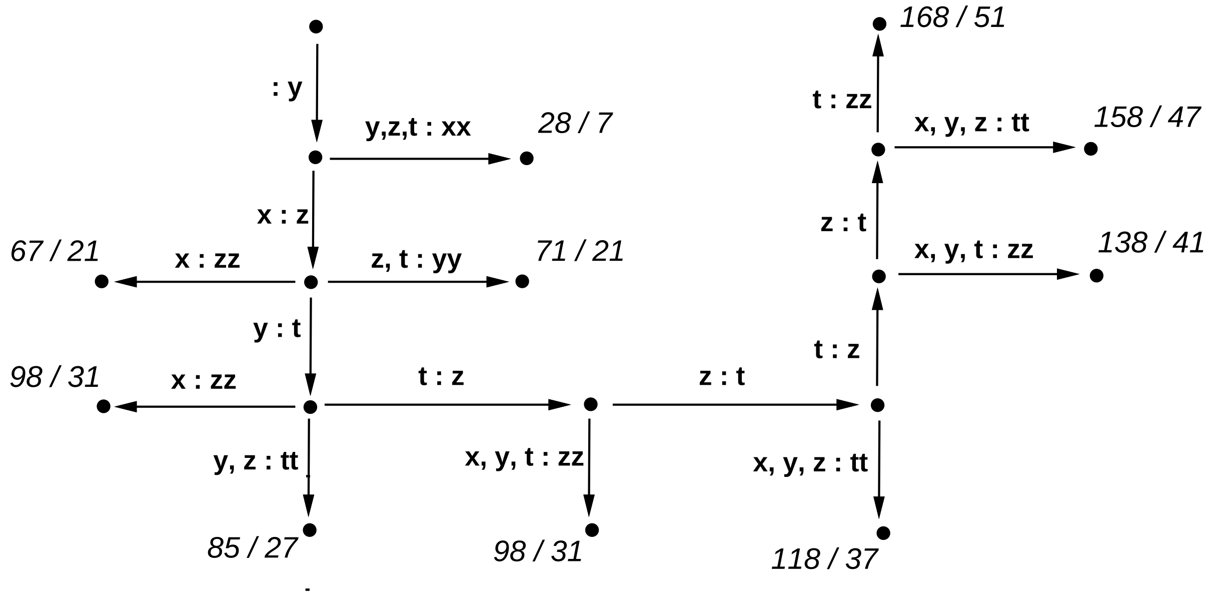

Figure 3: Adversary strategy.

6. Request z. If the server moves to z, go to next step. Otherwise, request zz^+ and halt; the optimum cost is 41, and $\mathcal{A}$'s cost is at least 138.

7. Request t. If the server moves to t, request zz^+ and halt; the optimum cost is 51, and $\mathcal{A}$'s cost is 168. Otherwise, request tt^+ and halt; the optimum cost is 47, and $\mathcal{A}$'s cost is at least 158.

Note that $\frac{85}{27}$ is the most favorable cost ratio the algorithm can obtain. ▮

Theorem 8.11 *If $c < \frac{85}{27}$, then there is no deterministic c-competitive algorithm for the page migration problem.*

Proof: For any integer p, let K_p be an isometric copy of K, and $x_p \in K_p$ the point which corresponds to x. The adversary constructs a tree-like metric space in which each tree node is some copy K_p. It begins with copy K_0 at the root. Then three copies K_1, K_2, K_3 are attached by equating x_1, x_2, x_3 with y_0, z_0, t_0, respectively. The construction procedure then recurses on the three children. The recursion stops when the branch has sufficient depth. In this metric space each point which is sufficiently close to x_0 corresponds to x_p for some copy K_p. The server is placed initially at point x_0.

Let $\mathcal{A}$ be any on-line algorithm. The adversary generates a bad request sequence for $\mathcal{A}$ in phases. At the start of phase i both $\mathcal{A}$ and the adversary have the server at the same point x_p of copy K_p. The adversary then generates the request sequence ϱ_i according to the strategy of Lemma 8.10, playing only within K_p. The adversary moves its server optimally within K_p. If $\mathcal{A}$ ever leaves K_p during ϱ_i then the adversary may continue as if the server of $\mathcal{A}$ were at the point from which it departed K_p and the cost incurred by $\mathcal{A}$ can only increase. At the end of ϱ_i both $\mathcal{A}$ and the adversary have their server at the same point u, which is x_q for some copy K_q. The adversary then begins phase $(i+1)$.

By Lemma 8.10 in each phase i we have

$$B_i = cost_{\mathcal{A}}(x, \varrho_i) - c \cdot cost_{\mathrm{OPT}}(x, \varrho_i) = \left(\frac{85}{27} - c\right) cost_{\mathrm{OPT}}(x, \varrho_i) \geq \epsilon,$$

for some $\epsilon > 0$. By using the space with arbitrarily large cardinality the value of $\sum_i B_i$ can be made arbitrarily large. Hence for $\varrho = \varrho_1\varrho_2\varrho_3\cdots$ there is no constant a_x for which $cost_{\mathcal{A}}(x, \varrho) \leq c \cdot cost_{\mathrm{OPT}}(x, \varrho) + a_x$, and so $\mathcal{A}$ cannot be c-competitive. ▌

8.4 An Algorithm for Three Points

The results of Black and Sleator [6] imply that there is a 3-competitive algorithm for each D and each metric space with two points. In this section, we show a 3-competitive algorithm for any three-point metric space and $D = 1$. We do not know whether there is a 3-competitive algorithm for 3 point spaces with $D > 1$.

Our three-point algorithm is based on a technique called *forgiveness* that has been recently used in an 11-competitive algorithm for 3 servers [8]. This technique exploits the flexibility in the definition of work functions. In the previous sections, our algorithms took advantage of the fact that the optimal offset functions for the edge, uniform space, and tree can be compactly characterized and are finite in number. With arbitrary three point spaces, however, the set of optimal offset functions can be very complicated. (The reader may wish to try computing the optimal offset functions for the metric space given by the 3-4-5 triangle as an example.) In fact, for some three point spaces there are infinitely many optimal offset functions.

In general, forgiveness decreases some values of the current work function in order to simplify the form of the corresponding offset function. The term "forgiveness" comes from viewing the on-line problem as a game played against an adversary. At each turn, the adversary chooses a request that the algorithm must serve, but the adversary must also pay something to serve the request. The work function provides a lower bound on the cost incurred by the adversary. By decreasing the work function we forgive the adversary some of that debt.

Theorem 8.12 *Let M be a metric space with only three points. Then there is a 3-competitive algorithm in M for $D = 1$.*

Proof: Let ω_i denote the work function at the time of request r_{i+1}, and $\sigma_i = \downarrow(\omega_i)$. Let $\omega_0 = \chi_s$, where s is the initial server location. To compute ω_{i+1}, we first compute $\omega' = \omega_i \wedge r_i$. We then choose ω_{i+1} so that

1. $\omega_{i+1}(x) \leq \omega'(x)$ for all x,

2. $\min(\omega_{i+1}) = \min(\omega')$,

3. the offset function $\sigma_{i+1} = \downarrow \omega_{i+1}$ always has one of just four forms given below. (These forms have properties (wf1) and (wf2) from Section 2.)

If ω_i is a work function for $\varrho_i = r_1 r_2 \cdots r_i$ then clearly ω_{i+1} is a work function for $\varrho_i r_{i+1}$.

Notation. We will label the three points of our space relative to the location of the on-line server. The point containing the on-line server is always labeled A, and the other two points are labeled B and C arbitrarily. Let a be the length of the side opposite the point currently labeled A, *i.e.* d_{BC}. We define the quantities b and c similarly. After the on-line server moves, we permute the labels as necessary. For the sake of succinctness, we will describe each offset function σ as a vector $\sigma = (\sigma(A), \sigma(B), \sigma(C))$.

We consider four *forms*, where a form is described by a current offset function σ, and some condition involving A, B, C and σ. In each form, $\sigma(A) = 0$. Thus, our server is always located at a minimum point of the offset function.

Form 1: $\sigma = \chi_A$, the characteristic function of A, that is, $\sigma = (0, c, b)$. The potential is $\Phi = 0$.

Form 2: $c \le \max\{b, a\}$ and $\sigma = (0, 0, \min\{b, a\})$. The potential is $\Phi = 2 \cdot c$.

Form 3: $\sigma = (0, 0, 0)$. The potential is $\Phi = 2 \cdot \max\{c, b\}$.

Form 4: $\max\{c, b\} \le a$ and $\sigma = (0, 0, a - c)$. The potential is $\Phi = 3 \cdot c + b - a$.

Let σ be the current offset function. By $\Delta cost_{adv}$ we denote the cost charged to the adversary for request r. As mentioned before, the algorithm will forgive *after* updating the current offset function, and therefore the charge to the adversary is simply $\Delta cost_{adv} = \min(\sigma \wedge r)$.

We now give the algorithm $\mathcal{A}$, and show that on any request

$$\Delta cost_{\mathcal{A}} + \Delta\Phi \le 3 \cdot \Delta cost_{adv}. \tag{8}$$

For request r_{i+1}, we have

$$\begin{aligned} \min(\sigma_i \wedge r_{i+1}) &= \min((\downarrow \omega_i) \wedge r_{i+1}) = \min(\omega_i \wedge r_{i+1}) - \min(\omega_i) \\ &= \min(\omega_{i+1}) - \min(\omega_i) \end{aligned} \tag{9}$$

Using Equation (9) in Equation (8), and summing over the request sequence $\varrho = r_1 \ldots r_m$, we can conclude that $cost_{\mathcal{A}}(s, \varrho) \le 3 \cdot \min(\omega_m)$. Although $\min(\omega_m)$ is not necessarily the *optimal* lower bound on the cost to serve the sequence, it is *a* lower bound, and competitiveness follows.

In each of the above forms, if the request is on A then the server stays on A and the new offset function is χ_A. Both costs are zero. The new form is of type 1, with potential zero, and thus (8) holds. Therefore we can restrict our attention to requests on B and C.

Form 1: Without loss of generality, the request is on B. We do not move the server, and our cost is c. The adversary's cost is c, and the new offset function is

$$(0, 0, \min\{b, a, b + a - c\}).$$

If $c \leq \max\{b, a\}$, the offset function is $(0, 0, \min\{b, a\})$, and we are at Form 2. If $c \geq \max\{b, a\}$, we forgive the offset function to $(0, 0, 0)$, obtaining Form 3. In either situation, the potential Φ increases from 0 to $2 \cdot c$, and we obtain

$$\Delta cost_{\mathcal{A}} + \Delta\Phi \;=\; 3 \cdot c \;=\; 3 \cdot \Delta cost_{adv}.$$

Form 2: If the request is on B, we move the server to B. Our cost is $2 \cdot c$, the adversary cost is zero, and the offset function becomes χ_B, a permutation of Form 1. The potential decreases by $2 \cdot c$, so (8) is an equality.

If the request is on C and $b \leq a$, we leave the server at A. The offset function becomes $(0, a - b, 0)$, a permutation of Form 4 with potential $3 \cdot b + c - a$. Our cost is b, and the adversary's cost is b. Then

$$\Delta cost_{\mathcal{A}} + \Delta\Phi \;=\; 3 \cdot b - (c + a - b) \;\leq\; 3 \cdot b \;=\; 3 \cdot \Delta cost_{adv}.$$

If the request is on C and $b \geq a$, we move the server to B (yes, to B not to C). The offset function becomes $(b - a, 0, 0)$, a permutation of Form 4 with potential $3 \cdot a + c - b$. Our cost is $b + c$, and the adversary's cost is a. Then

$$\Delta cost_{\mathcal{A}} + \Delta\Phi \;=\; 3 \cdot a \;=\; 3 \cdot \Delta cost_{adv}.$$

Form 3: Without loss of generality, the request is on B. We move the server to B. The offset function becomes χ_B, a permutation of Form 1. Our cost is $2 \cdot c$, and the adversary's cost is zero. The potential decreases by at least $2 \cdot c$, so (8) holds.

Form 4: If the request is on B, we move the server to B. The offset function becomes χ_B, a permutation of Form 1. Our cost is $2 \cdot c$, and the adversary's cost is zero. The potential decreases by $3 \cdot c + b - a$. Then

$$\Delta cost_{\mathcal{A}} + \Delta\Phi \;=\; 2 \cdot c - (3 \cdot c + b - a) \;\leq\; 0,$$

so (8) holds.

If the request is on C, we do not move the server. The offset function becomes $(c + b - a, c, 0)$, our cost is b, and the adversary's cost is $a - c$. We forgive the offset function to $(0, c, 0)$, a permutation of Form 2, with potential $2 \cdot b$. Then, since $b \leq a$, we have

$$\Delta cost_{\mathcal{A}} + \Delta\Phi \;=\; 2 \cdot b + a - 3 \cdot c \;\leq\; 3(a - c) \;=\; 3 \cdot \Delta cost_{adv}.$$

This completes the proof. ∎

Four point spaces. One might wonder whether Black and Sleator's conjecture holds for spaces of fixed size. According to a computer program that we have written, there exists a 4-point metric space M such that no on-line algorithm for the page migration problem for $D = 1$ on M is 3-competitive. In fact, it is the same space used for the proof of Theorem 8.10, shown in Figure 2. There exists no written proof of this statement, and any written proof based on the results of our program would be, at this time, far too long and tedious for a journal paper. We are willing to share the source code with anyone who wishes to pursue this matter.

Acknowledgements The authors thank Lásló Lovász for suggesting the approach to continuous trees taken in Section 8.2, which provides a simpler proof than in the conference version of this paper.

References

[1] B. Awerbuch, Y. Bartal and A. Fiat, Competitive distributed file allocation. *Proc. of 25th Symposium on Theory of Computation*, pages 164–173, 1993.

[2] S. Ben-David, A. Borodin, R. Karp, G. Tardos and A. Widgerson, On the power of randomization in on-line algorithms. *Proc. 22nd Symposium on Theory of Algorithms*, pages 379-386, 1990.

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

[4] P. Berman, H. J. Karloff, and G. Tardos. A competitive three-server algorithm. In *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, pages 280–290, 1990.

[5] D. Black, A. Gupta, and W. Weber. Competitive management of distributed shared memory. In *Proceedings, Spring Compcon 1989*, pages 184–190, New York, NY, 1989. IEEE Computer Society Press.

[6] D. L. Black and D. D. Sleator. Competitive algorithms for replication and migration problems. Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University, 1989.

[7] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4 (1991) 172-181. Also in *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, pages 291–300, 1990.

[8] M. Chrobak and L. L. Larmore. Generosity helps, or an 11-competitive algorithm for three servers. *Journal of Algorithms* 16 (1994) 234–263. Also in *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, pages 196–202, Philadelphia, 1992. Society for Industrial and Applied Mathematics.

[9] M. Chrobak and L. L. Larmore. The server problem and on-line games. In *Proc. of the DIMACS Workshop on On-Line Algorithms*, volume 7 of *DIMACS Series*, pages 11–64. American Mathematical Society, 1992.

[10] W. Crowther, J. Goodhue, E. Starr, R. Thomas, W. Milliken, and T. Blackadar. Performance measurements on a 128-node butterfly parallel processor. In *Proc. International Conf. on Parallel Processing*, pages 531–540, New York, NY, 1985. IEEE Computer Society Press.

[11] L. W. Dowdy and D. V. Foster. Comparative models of the file assignment problem. *ACM Computing Surveys*, 14(2):287–313, 1982.

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

[13] S. Irani. Two results on the list update problem. *Inf. Process. Lett.*, 38:301–306, 1991.

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

[15] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki. Competitive randomized algorithms for non-uniform problems. In *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, pages 301–309, 1990. Also, to appear in *Algorithmica.*

[16] C. Lund and N. Reingold. Linear programs for randomized on-line algorithms. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 382–391. Society for Industrial and Applied Mathematics, Philadelphia, 1994.

[17] C. Lund, N. Reingold, J. Westbrook, and D. Yan. On-Line Distributed Data Management. To appear in *European Symposium on Algorithms*, 1994

[18] M. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for on-line problems. *Journal of Algorithms* 11 (1990) 208-230. Also in *Proc. 20th ACM Symp. on Theory of Computing*, pages 322–333, 1988.

[19] G. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss. The IBM research parallel processor prototype: Introduction and architecture. In *Proc. International Conf. on Parallel Processing*, pages 764–771, New York, NY, 1985. IEEE Computer Society Press.

[20] P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. Research Report RC 15622 (No. 69444), IBM T. J. Watson Research Center, 1990. Also, in *16th International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science vol. 372, Springer-Verlag, 1989, 687-703.

[21] N. Reingold, J. Westbrook, and D. D. Sleator. Randomized algorithms for the list update problem. *Algorithmica*, 11:15–32, 1994.

[22] C. Scheurich and M. Dubois. Dynamic page migration in multiprocessors with distributed global memory. *IEEE Transactions on Computers*, 38(8):1154–1163, August 1989.

[23] J. Westbrook. Randomized algorithms for multiprocessor page migration. *SIAM J. Comput.*, 23(5). To appear.

[24] A. Wilson. Hierarchical cache/bus architecture for shared memory multiprocessors. In *Proc. 14th International Symp. on Computer Architecture*, pages 244–252, New York, NY, 1987. IEEE Computer Society Press.

[25] O. Wolfson. A distributed algorithm for adaptive replication data. Technical Report CUCS-057-90, Department of Computer Science, Columbia University, 1990.