Random Walks on Weighted Graphs, and Applications to On-line Algorithms

Don Coppersmith* Peter Doyle† Prabhakar Raghavan*
Marc Snir*

Abstract

We study the design and analysis of randomized on-line algorithms. We show that this problem is closely related to the synthesis of random walks on graphs with positive real costs on their edges. We develop a theory for the synthesis of such walks, and employ it to design competitive on-line algorithms.

*IBM T.J. Watson Research Center, Yorktown Heights, NY 10598.
†AT&T Bell Laboratories, Murray Hill, NJ 07974.

1 Overview

Much recent work has dealt with the competitive analysis of on-line algorithms [5, 16, 18]. In this paper we study the design of randomized on-line algorithms. We show here that the synthesis of random walks on graphs with positive real costs on theirs edges is related to the design of these randomized on-line algorithms. We develop methods for the synthesis of such random walks, and use them to design competitive randomized on-line algorithms.

Let G be a weighted undirected graph with n vertices $\{1, \ldots, n\}$; $c_{ij} = c_{ji} > 0$ is the *cost* of the edge connecting vertices i and j, $c_{ii} = 0$. Consider a random walk on the graph G, executed according to a transition probability matrix $P = (p_{ij})$; p_{ij} is the probability that the walk moves from vertex i to vertex j, and the walk pays a cost c_{ij} in that step. Let e_{ij} (not in general equal to e_{ji}) be the expected cost of a random walk starting at vertex i and ending at vertex j (e_{ii} is the expected cost of a round trip from i). We say that the random walk has *stretch* c if there exists a constant a such that, for any sequence $i_0, i_1, \ldots, i_\ell$ of vertices $\sum_{j=1}^{\ell} e_{i_{j-1}i_j} \leq c \cdot \sum_{j=1}^{\ell} c_{i_{j-1}i_j} + a$. We prove the following tight result:

> Any random walk on a weighted graph with n vertices has stretch at least $n - 1$, and every weighted graph with n vertices has a random walk with stretch $n - 1$.

The upper bound proof is constructive, and shows how to compute the transition probability matrix P from the cost matrix $C = (c_{ij})$. The proof uses new connections between random walks and effective resistances in networks of resistors, together with results from electric network theory. Consider a network of resistors with n vertices, and *conductance* σ_{ij} between vertices i and j (vertices i and j are connected by a resistor with *branch resistance* $1/\sigma_{ij}$). Let R_{ij} be the *effective resistance* between vertices i and j (i.e., $1/R_{ij}$ is the current that would flow from i to j if one volt were applied between i and j; it is known that $1/R_{ij} \geq \sigma_{ij}$). Let the *resistive random walk* be defined by the probabilities $p_{ij} = \sigma_{ij} / \sum_k \sigma_{ik}$. In Section 3 we show that this random walk has stretch $n - 1$ in the graph with costs $c_{ij} = R_{ij}$. Thus, a random walk with optimal stretch is obtained by computing the *resistive inverse* (σ_{ij}) of the cost matrix (c_{ij}): a network of branch conductances ($\sigma_{ij} \geq 0$), so that, for any i, j, c_{ij} is the effective (not branch)

resistance between i and j. Unfortunately, not all cost matrices have resistive inverses (with positive conductances). However, we show in Section 4 that every matrix (c_{ij}) has a *generalized resistive inverse*: a network of nonnegative branch conductances σ_{ij} with associated effective resistances R_{ij}, such that either $R_{ij} = c_{ij}$, or $R_{ij} < c_{ij}$ and $\sigma_{ij} = 0$. The resistive random walk has stretch $n - 1$ for the graph with costs R_{ij}, and consequently for the graph with costs c_{ij}, since it never traverses those edges whose costs it underestimates.

Chandra *et al.* [7] use electric networks to *analyze* a particular random walk, in which $p_{ij} = (1/c_{ij})/(\sum_k 1/c_{ik})$. Traditionally, this is how electric networks have been used in studying random walks: to *analyze* a given random walk (cf. Doyle and Snell [10]). Here we instead use electric networks to *synthesize* a (different, in general) random walk with optimal stretch.

Next, we outline the relevance of this random walk synthesis problem to the design of on-line algorithms. Consider the following game played between a *cat* and a *mouse* on the graph G. Round r starts with both cat and mouse on the same vertex i_{r-1}. The mouse then moves to a new vertex i_r not known to the cat; the cat then walks on the graph until it reaches the mouse at i_r, at which point round $r + 1$ starts with the mouse moving to a new node. Each move of the mouse may depend on all previous moves of the cat. The cat may use a randomized algorithm, and choose its next move probabilistically, as a function of its previous moves. The games stops after a fixed number of rounds. A strategy for the cat is *c-competitive* if there exists a constant a such that for any number of rounds and any strategy of the mouse the cat's expected cost is at most c times the mouse's cost $+a$. A random walk with stretch c defines a strategy for the cat that is c-competitive: in each round, the cat executes a random walk according to P until it finds the mouse. This strategy is very simple, and *memoryless*: the cat need not remember its previous moves, and the next cat move depends only on its current position.

Some special cases of the cat-and-mouse game have been studied by Baeza-Yates *et al.* [1]. We show that this cat-and-mouse game is at the core of many other on-line algorithms that have evoked tremendous interest of late [3, 4, 5, 8, 9, 11, 18, 20, 21, 22]. We consider two settings. The first is the *k-server problem*, defined in [18]. An on-line algorithm manages k mobile servers located at the vertices of a graph G whose edges have positive real lengths; it has to satisfy on-line a sequence of requests for service at vertex v_i, $i = 1, 2, \ldots$, by moving a server to v_i unless it already has a server there.

Each time it moves a server, it pays a cost equal to the distance moved by that server. We compare the cost of such an algorithm, to the cost of an adversary that, in addition to moving its servers, also generates the sequence of requests. The competitiveness of an on-line algorithm is defined with respect to these costs (Section 8) [3, 21]. It was conjectured in [18] that for every cost matrix there exists a k-competitive algorithm for this problem. Repeated attempts to prove this conjecture have met only with limited success [8, 9, 21]. We use our optimal random walk to derive optimal randomized k-competitive server algorithms in two situations: (1) when the graph G has a resistive inverse, and (2) when the graph G has $k+1$ vertices. This includes all previously known cases where the conjecture was proven true, as well as many new cases. We do so with a single unified theory — that of resistive inverses. The algorithm is very simple, randomized and memoryless.

The other setting is the *metrical task system (MTS)*, defined in [5]. A MTS consists of a weighted graph (the vertices of the graph are positions, and edge weights are the costs of moving between positions). An algorithm occupies one position at any time. A *task* is represented by a vector $(c_1, \ldots, c_n)$, where c_i is the cost of processing the task in position i. The algorithm is presented a sequence of tasks $\mathcal{T} = T_1, T_2, \ldots$ and can move to a new position before processing each task. The cost incurred by the algorithm is the sum of the costs of moving and processing tasks. A $(2n-1)$-competitive on-line algorithm for MTS is presented in [5], and shown to be optimal. The algorithm is deterministic, but somewhat complex. In Section 9 we present a simple, memoryless randomized algorithm for any MTS that is $(2n-1)$-competitive, and show that no randomized algorithm can do better against the adaptive on-line adversary.

Deterministic on-line algorithms traditionally make use of the following seemingly paradoxical idea: not knowing the future, they base their decisions on their record of the past. In a number of interesting cases, the maintenance of appropriate information from the past suffices to make these algorithms competitive. Our main theme here seems even more paradoxical: our randomized on-line algorithms ignore the past as well, maintaining no history. Instead they base their choice at each step just on the relative costs of the alternatives at hand. This yields simple memoryless randomized algorithms that are competitive for various situations.

The remainder of this paper is organized as follows. Sections 2–4 deal with random walk strategies for the cat-and-mouse game; more general strategies

are discussed in Section 5. Sections 6 and 7 derive some additional properties of resistive random walks that will prove to be of use later on. Section 8 considers the k-server problem, and Section 9 deals with metrical task systems. We conclude with directions for further work in Section 10.

2 Lower bound on Stretch

In this section we give a lower bound on the stretch of any random walk on any graph.

Theorem 2.1 *For any $n \times n$ cost matrix C and any transition probability matrix P, the stretch of the random walk defined by P on the graph with weights given by C is $\geq n-1$.*

Proof: We can assume w.l.o.g. that P is irreducible (the underlying directed graph is strongly connected). Let ϕ_i be the ith component of the left eigenvector of P for the eigenvalue 1 (when P is aperiodic, this is the stationary probability of vertex i), so that $\phi_j = \sum_i \phi_i p_{ij}$ [17]. Let $e_i = \sum_j p_{ij} c_{ij}$ denote the expected cost of the first move out of vertex i, and let $E = \sum_i \phi_i e_i = \sum_{i,j} \phi_i p_{ij} c_{ij}$ be the average cost per move. We have

$$\begin{aligned} \sum_{i,j} (\phi_i p_{ij}) e_{ji} &= \sum_i \phi_i \left(\sum_j p_{ij} e_{ji} \right) = \sum_i \phi_i (e_{ii} - e_i) \\ &= \sum_i \phi_i (E/\phi_i - e_i) = (n-1)E, \end{aligned}$$

(see [17], for instance, for a proof of the identity $e_{ii} = E/\Phi_i$), while

$$\sum_{i,j} (\phi_i p_{ij}) c_{ji} = \sum_{i,j} (\phi_i p_{ij}) c_{ij} = E.$$

Thus,

$$\sum_{i,j} (\phi_i p_{ij}) e_{ji} = (n-1) \sum_{i,j} (\phi_i p_{ij}) c_{ji}.$$

Notice that, if each directed edge (ji) (note the order!) is counted with multiplicity proportional to $\phi_i p_{ij}$, then a flow condition is satisfied: the total multiplicity of edges leading out of i is equal to that of those leading into i.

Thus, the above equation represents a convex combination of cycles so that there is some cycle $(i_1, i_2, \ldots i_\ell, i_{\ell+1} = i_1)$ with stretch at least $n-1$; thus,

$$\sum_{j=1}^{\ell} e_{i_j i_{j+1}} \geq (n-1) \sum_{j=1}^{\ell} c_{i_j i_{j+1}}.$$

□

The symmetry of the cost matrix C is necessary for the theorem. If we drop the condition of symmetry, a cost matrix might be

$$C = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix},$$

and the random walk with transition probabilities

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

has a stretch of one, rather than at least two, as would be demanded by the theorem.

3 Upper bound — resistive case

We next consider the complementary upper bound problem: given C, to synthesize a matrix P that achieves a stretch of $n-1$ on C. In this section we will describe a construction and proof for a class of matrices C known as *resistive matrices*. In Section 4 we will generalize our construction to arbitrary cost matrices.

Let (σ_{ij}) be a non-negative symmetric real matrix with zero diagonal. Build the support graph (V, E), with vertex set $V = \{1, 2, ..., n\}$ and edge set $E = \{(i\ j) \mid \sigma_{ij} > 0\}$, and let (V, E) be connected. Consider a network of resistors based on (V, E), such that the resistor between vertices i and j has *branch conductance* σ_{ij}, or *branch resistance* $1/\sigma_{ij}$.

Let c_{ij} be the *effective resistance* between vertices i and j. (A unit voltage between i and j in this network of resistors results in a current of $1/c_{ij}$.) We require that the support graph be connected so that the effective resistances will be finite.

Definition 1 *A cost matrix* (c_{ij}) *is* resistive *if it is the matrix of effective resistances obtained from a connected non-negative symmetric real matrix* (σ_{ij}) *of conductances. The matrix* (σ_{ij}) *is the* resistive inverse of C. $\square$

The following facts are not difficult to prove, and follow from standard electric network theory [23]. Resistive cost matrices are symmetric, positive off the diagonal, zero on the diagonal, and satisfy the triangle inequality: $c_{ij} + c_{jk} \geq c_{ik}$. A principal submatrix of a resistive cost matrix is resistive.

Define two $(n-1) \times (n-1)$ matrices $\bar{\sigma}, \bar{C}$ by

$$\bar{\sigma}_{ii} = \sum_{j \leq n, j \neq i} \sigma_{ij}, \quad 1 \leq i \leq n-1, \tag{1}$$

$$\bar{\sigma}_{ij} = -\sigma_{ij}, \quad i \neq j, \quad 1 \leq i, j \leq n-1, \tag{2}$$

$$\bar{c}_{ij} = [c_{in} + c_{jn} - c_{ij}]/2, \quad 1 \leq i, j \leq n-1. \tag{3}$$

Then $\bar{\sigma}$ is the inverse of $\bar{C}$:

$$\sum_{j=1}^{n-1} \bar{\sigma}_{ij} \bar{c}_{jk} = \delta_{ik}. \tag{4}$$

It can happen that a given cost matrix $C = (c_{ij})$ gives rise to a putative resistive inverse with some negative conductances:

$$\exists i, j : \quad \sigma_{ij} < 0$$

and in this case there is no resistive inverse for C.

Examples of resistive cost matrices include:
(1) Any three points with the distances satisfying the triangle inequality.
(2) Points on a line: vertex i is at a real number r_i, with $c_{ij} = \mid r_i - r_j \mid$.
(3) The uniform cost matrix $c_{ij} = d$, if $i \neq j$.
(4) Tree closure: given a tree T on n vertices and positive costs for the tree edges, points are located on the edges of the tree, and the distance between any pair of points equals the distance between them on the tree. The previous three examples are particular cases of tree closure.
(5) A cost matrix C given by a graph with $m+n$ vertices $x_1, x_2, \ldots, x_m$, $y_1, y_2, \ldots, y_n$, $m, n > 1$, where $c_{x_i,x_j} = 2m$, $c_{y_i,y_j} = 2n$, and $c_{x_i,y_j} = m + n - 1$. The associated resistive inverse is a complete bipartite graph $K_{m,n}$ with

resistors of resistance mn on each edge. This example cannot be expressed as any of the previous examples: for if C were a tree closure, then the midpoint of the tree path joining x_1 and x_2 would be at distance $n-1$ from both y_1 and y_2, contradicting $c_{y_1,y_2} = 2n > 2(n-1)$.

If C is a resistive cost matrix, its resistive inverse (σ_{ij}) provides a way of synthesizing an optimal random walk P achieving a stretch of $n-1$. In fact, in determining the stretch of a random walk, it suffices to consider sequences of vertices $v_1, v_2, \ldots v_\ell, v_{\ell+1} = i_1$ that form simple cycles in G. Indeed, assume that the random walk defined by P has a stretch of c on simple cycles. Any cycle can be decomposed into the union of disjoint simple cycles, so that the claim holds for arbitrary closed paths. Let d be the maximum length of an edge in G. Then, for any path $v_1, v_2, \ldots, v_\ell$ we have

$$\begin{aligned}
\sum_{i=1}^{\ell-1} e_{v_i v_{i+1}} &= \sum_{i=1}^{\ell-1} e_{v_i v_{i+1}} + e_{v_\ell v_1} - e_{v_\ell v_1} \\
&\leq c \cdot \left(\sum_{i=1}^{\ell-1} c_{v_i v_{i+1}} + c_{v_\ell v_1} \right) \\
&\leq c \cdot \left(\sum_{i=1}^{\ell-1} c_{v_i v_{i+1}} \right) + c \cdot d.
\end{aligned}$$

Let C be a resistive cost matrix, and (σ_{ij}) be its resistive inverse, Let P be the resistive random walk on the graph with cost matrix C, i.e.

$$p_{ij} = \frac{\sigma_{ij}}{\sum_k \sigma_{ik}}.$$

Let ϕ_i be the steady state probability of vertex i under the random walk P, so that

$$\phi_i = \sum_j \phi_j p_{ij}$$

and

$$\sum_i \phi_i = 1.$$

The following properties of the resistive random walk will prove to be of much use.

One can verify, by substitution, that

$$\phi_i = \frac{\sum_k \sigma_{ik}}{\sum_{gh} \sigma_{gh}}.$$

The steady state probability of a move on edge (ij) is

$$\phi_i p_{ij} = \frac{\sigma_{ij}}{\sum_{gh} \sigma_{gh}}$$

and the expected cost of a move is

$$E = \sum_{gh} \phi_g p_{gh} c_{gh} = \frac{\sum_{gh} \sigma_{gh} c_{gh}}{\sum_{gh} \sigma_{gh}}.$$

By Foster's Theorem [13, 14, 23] on electric networks,

$$\sum_{gh} \sigma_{gh} c_{gh} = 2(n-1),$$

so that

$$E = \frac{2(n-1)}{\sum_{gh} \sigma_{gh}}.$$

The average cost of a round trip to vertex i is [17]

$$e_{ii} = E/\phi_i = \frac{2(n-1)}{\sum_k \sigma_{ik}}.$$

Theorem 3.1 *Let $C = (c_{ij})$ be a resistive cost matrix and let P be the resistive random walk on the graph with cost matrix C. Then every cycle $(v_1, v_2, ..., v_\ell, v_{\ell+1} = v_1)$ has stretch $n-1$:*

$$\sum_{i=1}^{\ell} e_{v_i v_{i+1}} = (n-1) \cdot \sum_{i=1}^{\ell} c_{v_i v_{i+1}}.$$

Proof: Following Doyle and Snell [10] we define the *escape probability* $P_{esc}(ij)$ to be the probability that a random walk, starting at vertex i, will reach vertex j before returning to vertex i. Doyle and Snell [10] show that

$$P_{esc}(ij) = \frac{1/c_{ij}}{\sum_k \sigma_{ik}}.$$

The average cost of a round trip from vertex i to vertex j and back to vertex i is

$$e_{ii}/P_{esc}(i,j) = 2(n-1)c_{ij} = (n-1)[c_{ij} + c_{ji}].$$

This cost is also, by definition, $e_{ij} + e_{ji}$, so that

$$e_{ij} + e_{ji} = (n-1) \cdot [c_{ij} + c_{ji}].$$

So the stretch of any two-cycle is $n - 1$.

We need a bound on the stretch of any cycle, not just two-cycles. The stationary probability of traversing the directed edge (ij) is $\sigma_{ij} / \sum_{gh} \sigma_{gh}$, which is symmetric because σ is symmetric. Thus our random walk is a *reversible* Markov chain [17]. For any cycle $(v_1, v_2, ..., v_\ell, v_{\ell+1} = v_1)$, the expected number of forward traversals of the cycle (not necessarily consecutive) is the same as the expected number of backward traversals of the cycle, and the expected cost per forward traversal is the same as the expected cost per backward traversal. Thus

$$\begin{aligned}
\sum_{i=1}^{\ell} e_{v_i v_{i+1}} &= \sum_{i=1}^{\ell} e_{v_{i+1} v_i} \\
&= \frac{1}{2} \left[\sum_{i=1}^{\ell} e_{v_i v_{i+1}} + \sum_{i=1}^{\ell} e_{v_{i+1} v_i} \right] \\
&= \frac{1}{2} \sum_{i=1}^{\ell} \left[e_{v_i v_{i+1}} + e_{v_{i+1} v_i} \right] \\
&= \frac{1}{2} \sum_{i=1}^{\ell} (n-1) \left[c_{v_i v_{i+1}} + c_{v_{i+1} v_i} \right] \\
&= \sum_{i=1}^{\ell} (n-1) c_{v_i v_{i+1}}.
\end{aligned}$$

So every cycle has stretch $n - 1$. $\square$

Note that this theorem only holds for cycles, not for individual directed edges. If

$$C = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix}, \quad (\sigma_{ij}) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 1 & 0 \\ 1/2 & 0 & 1/2 \\ 0 & 1 & 0 \end{bmatrix}$$

then

$$\begin{aligned}
e_{21} &= 3 > 2 = (n-1) c_{21} \\
e_{12} &= 1 < 2 = (n-1) c_{12} \\
e_{21} + e_{12} &= 4 = (n-1)(c_{21} + c_{12}).
\end{aligned}$$

However, we note that since

$$e_{ij} + e_{ji} \leq (n-1)[c_{ij} + c_{ji}] = 2(n-1)c_{ij}$$

we have

$$e_{ij} \leq 2(n-1)c_{ij};$$

the stretch on individual edges is at most $2(n-1)$. Furthermore, if the cost matrix (c_{ij}) fulfils the triangle inequality, then $e_{ji} \geq c_{ji}$, so that

$$e_{ij} \leq 2(n-1)c_{ij} - e_{ji} \leq (2n-3)c_{ij}.$$

4 Upper bound — non-resistive case

In this section we prove the existence of a generalized resistive inverse. The generalized resistive inverse turns out to be the solution to a convex variational problem, and we present a simple iterative algorithm for finding it. From the generalized resistive inverse we get an $n-1$-competitive strategy for the cat-and-mouse game with an arbitrary positive symmetric cost matrix.

Theorem 4.1 *Let C be any positive symmetric cost matrix. Then there is a unique resistive cost matrix $\hat{C}$ with associated conductance matrix σ, such that $\hat{c}_{ij} \leq c_{ij}$, $\sigma_{ij} \geq 0$ and $\hat{c}_{ij} = c_{ij}$ if $\sigma_{ij} \neq 0$.*

Thus, σ is the generalized resistive inverse of C.

Proof: For simplicity, we will limit the discussion to the case of the triangle graph ($n = 3$), with costs $c_{1,2} = R_0,\ c_{1,3} = S_0,\ c_{2,3} = T_0$, and with edge conductances $\sigma_{1,2} = a,\ \sigma_{1,3} = b,\ \sigma_{2,3} = c$ and corresponding effective resistances $R = R_{1,2},\ S = R_{1,3},\ T = R_{2,3}$. This case will exhibit all the features of the general case, and yet allow us to get by without cumbersome subscripts. Please note, however, that for a triangle graph a cost matrix is resistive if and only if it satisfies the triangle inequality, while for a general graph the triangle inequality is necessary but by no means sufficient. Needless to say, we will make no use of this condition for resistivity in our analysis of the 3-node graph.

We begin by recalling the relevant electrical theory (cf. Weinberg [23] and Bott and Duffin [6]). The admittance matrix of our network is

$$K = \begin{pmatrix} a+b & -a & -b \\ -a & a+c & -c \\ -b & -c & b+c \end{pmatrix}.$$

(In general, the admittance matrix is the matrix $(\bar{\sigma}_{ij})$ defined by Equations (1,2), extended to n.) If one hooks the network up to the world outside so as to establish node voltages v_1, v_2, v_3, the currents I_1, I_2, I_3 flowing into the network at the three nodes are given by

$$\begin{pmatrix} I_1 \\ I_2 \\ I_3 \end{pmatrix} = K \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}.$$

The power being dissipated by the network is

$$(I_1 v_1 + I_2 v_2 + I_3 v_3) = \begin{pmatrix} v_1 & v_2 & v_3 \end{pmatrix} K \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \geq 0.$$

The matrix K is non-negative definite, with 0-eigenvector $(1,1,1)$. Label its eigenvalues

$$0 = \lambda_0 < \lambda_1 \leq \lambda_2.$$

On the orthogonal complement $P = \{v_1 + v_2 + v_3 = 0\}$ of $(1,1,1)$, K has eigenvalues λ_1, λ_2, and the determinant of $K|_P$ — that is, the product of the non-zero eigenvalues of K — is given by the next-to-lowest order coefficient of the characteristic polynomial of K, which can be expressed using Kirchhoff's tree formula:

$$\begin{aligned} \det K|_P &= \lambda_1 \lambda_2 \\ &= \lambda_0\lambda_1 + \lambda_0\lambda_2 + \lambda_1\lambda_2 \\ &= \begin{vmatrix} a+b & -a \\ -a & a+c \end{vmatrix} + \begin{vmatrix} a+b & -b \\ -b & b+c \end{vmatrix} + \begin{vmatrix} a+c & -c \\ -c & b+c \end{vmatrix} \\ &= (ab+ac+bc) + (ab+ac+bc) + (ab+ac+bc) \\ &= 3D. \end{aligned}$$

Here the *discriminant* $D = ab + ac + bc$ is obtained by summing over the spanning trees of the network the product of the conductivities of the edges making up the tree (cf. Bott and Duffin [6]), and 3 is the number of nodes in the network. In the general case, we get

$$\det K|_P = nD,$$

because each of the n principal minors contributes D.

The effective resistances are obtained by taking the gradient of $\log D$ in edge-conductance space:

$$(R, S, T) = (\frac{\partial}{\partial a} \log D, \frac{\partial}{\partial b} \log D, \frac{\partial}{\partial c} \log D) = \nabla_{(a,b,c)} \log D.$$

In this case, we get

$$(R, S, T) = \left(\frac{b+c}{ab+ac+bc}, \frac{a+c}{ab+ac+bc}, \frac{a+b}{ab+ac+bc} \right).$$

The numerators are obtained by summing over all spanning trees containing one specified edge the product of the conductances of the edges of the tree other than the specified edge. Since the degree of the denominator is one greater than the numerator, the units of the quotient are those of inverse conductance—that is, resistance—just as they ought to be. (This formula for the effective resistances in terms of spanning trees goes back to Kirchhoff.)

Let Π denote the positive orthant in edge-conductance space

$$\Pi = \{a, b, c > 0\},$$

and let $\bar{\Pi}$ denote its closure, the non-negative orthant

$$\bar{\Pi} = \{a, b, c \geq 0\}.$$

On $\bar{\Pi}$ the function

$$\log D = \log \det K|_P - \log 3$$

is concave. As Gil Strang has pointed out to us, this follows from the fact that on the set of positive definite matrices the function $\log \det$ is concave (see [19]). Indeed, up to the additive constant $-\log 3$, the function $\log D$ is obtained by mapping the space of edge-conductances linearly into the space of linear operators on P and then taking the logarithm of the determinant—and pulling back a convex function under a non-singular linear map yields a convex function.

Now let

$$F(a, b, c) = \log D - (R_0 a + S_0 b + T_0 c),$$

where $R_0, S_0, T_0 > 0$. The function F is concave and

$$\nabla_{(a,b,c)} F = (R - R_0, S - S_0, T - T_0).$$

The extremal problem

$$\max_{(a,b,c)\in\bar{\Pi}} F(a,b,c)$$

has a unique solution (a_0, b_0, c_0) in the non-negative orthant $\bar{\Pi}$. If the solution lies in the positive orthant Π, then we have

$$\nabla_{(a_0,b_0,c_0)} F = \mathbf{0},$$

so that $R = R_0$, $S = S_0$ and $T = T_0$, and a honest resistive inverse is obtained. If the solution lies on the boundary then the Kuhn-Tucker conditions identify this point as the unique point where

$$\frac{\partial F}{\partial a}|_{a=a_0} \leq 0,$$

with

$$\frac{\partial F}{\partial a}|_{a=a_0} = 0$$

if $a > 0$, etc. Thus, $R \leq R_0$, and $R = R_0$ if $a > 0$, etc. $\square$

This proof applies as well to the case where we demand that $\sigma_{ij} = 0$ for certain selected edges (ij), and place no upper bounds on the corresponding $\hat{c}_{ij}$ (i.e. set $c_{ij} = \infty$).

If $C = (c_{ij})$ is resistive, the matrix inversion of Section 3 will find the associated conductance matrix σ, with $\hat{c}_{ij} = c_{ij}$. If C is not resistive — or even if it is — there is an iterative algorithm that converges to the generalized resistive inverse whose existence is guaranteed by Theorem 4.1. In presenting this algorithm we will once again limit the discussion to the case where the graph is a triangle, and use the same notation as above.

By Foster's theorem $aR + bS + cT = 2$, (the 2 here being one less than the number of nodes in the graph), and hence $a_0R_0 + b_0S_0 + c_0T_0 = 2$. Thus

$$(a_0, b_0, c_0) = \arg \max_{(a,b,c)\in\bar{\Sigma}} D,$$

where $\bar{\Sigma}$ is the closure of the open simplex

$$\Sigma = \{a, b, c > 0;\ aR_0 + bS_0 + cT_0 = 2\}.$$

To locate the maximum we can use the *knee-jerk algorithm*, according to which we iterate the mapping

$$T(a,b,c) = \left(a\frac{R}{R_0}, b\frac{S}{S_0}, c\frac{T}{T_0}\right).$$

The rationale behind the knee-jerk algorithm is as follows: we begin by guessing values of the conductances a, b, c, compute the corresponding effective resistances R, S, T, and compare these numbers to the desired values R_0, S_0, T_0. Suppose for a moment that $R = 2R_0$. Then the edge a says to itself, "The resistance across me is twice what it's supposed to be; now if every edge would just double its conductance, then all the resistances would be cut in half, and my resistance would be just what it's supposed to be; so I think I'll just go ahead and double my own conductance, and hope for the best."

And the amazing thing is that everything does indeed work out for the best, or at least for the better. For as it turns out, the knee-jerk mapping is a particular instance of a general method known as the *Baum algorithm*, and from the theory of the Baum algorithm (see Baum and Eagon [2]) it follows that the mapping T takes Σ to itself, and strictly increases the objective function D for any (a, b, c) (other than (a_0, b_0, c_0)) in Σ. And it follows from this that for any starting guess $(a, b, c) \in \Sigma$ the sequence $T^n(a, b, c)$ of iterates converges to the generalized resistive inverse (a_0, b_0, c_0).

5 The cat-and-mouse game

We now return to the cat-and-mouse game. As an immediate consequence of Theorem 4.1, we have:

Theorem 5.1 *Let G be any weighted graph with n nodes. The cat has an $(n-1)$-competitive strategy for the cat-and-mouse game on G.* □

Note that the strategy we prescribe for the cat is simple and memoryless, and consists of executing the resistive random walk. The computation of the transition probabilities P is done once and for all at the beginning of the game; the execution of the strategy consists simply of making a random choice at each step. The lower bound on stretch in Theorem 2.1 shows that no cat strategy based on a random walk can do better. Could a more general cat strategy do better? The answer depends on whether or not the cat learns, at the end of each round, that it has caught up with the mouse. The random walk strategy described above can be thought of as a *blind cat*: oblivious to the fact that it has caught the mouse at the end of a round, it walks on. The only requirement imposed on the mouse is that it must move to a new

node whenever the cat arrives at the node it presently occupies. We prove below that no blind cat can achieve a competitiveness better than $n-1$ on any n-node graph, regardless of how clever its strategy is (the cat can use an arbitrary randomized algorithm to hunt for the mouse). The proof is inspired by a lower bound of Manasse *et al.* [18] for the k-server problem.

Theorem 5.2 *For any $n \times n$ cost matrix C and any blind cat strategy, there is a mouse strategy that forces the competitiveness of the cat to be at least $n-1$.*

Proof: The cat starts the game at some node v_0 of the graph. It then walks through the graph visiting some (possibly random) sequence of nodes $v_1, v_2, \cdots$, paying a total cost of $\sum_{i \geq 1} c_{v_{i-1},v_i}$. We first describe strategies for $n-1$ different mice the sum of whose costs during this time is $\sum_{i \geq 1} c_{v_{i-1},v_i}$. Each of these $n-1$ mice will obey the dictum that it move to a new node whenever the cat arrives at its present location. The proof will then be completed by choosing one of these $n-1$ mouse strategies uniformly at random at the start of the game, so that the expected cost of the cat is at least $n-1$ times that of the (randomly chosen) mouse.

We now describe the strategies for the $n-1$ mice. Each of the $n-1$ begins the game at a different node of the graph, with no mouse starting at v_0. Whenever the cat arrives at a node occupied by a mouse, that mouse moves to the node just vacated by the cat. Thus no mouse ever moves to a node occupied by one of the other $n-2$ mice, so that exactly one mouse moves at each step. Further, the mouse that moves at each step pays exactly the same cost as the cat on that step. It follows that the sum of the costs of the $n-1$ mice equals $\sum_{i \geq 1} c_{v_{i-1},v_i}$. $\square$

Thus we have shown that no blind cat can achieve a competitiveness better than $n-1$, and have complemented this with a simple blind cat (the resistive random walk) that achieves this bound. What if the restriction of blindness were removed (i.e., the cat is told whenever it catches up with the mouse)? Baeza-Yates *et al.* [1] have given (without using the cat-and-mouse terminology) examples of a number of graphs for which the cat achieves a constant competitiveness. For instance, when the nodes of the graph are uniformly spaced on the periphery of a circle, they show that a natural deterministic strategy achieves a competitiveness of 9.

We conclude this section with another example in which a cat that is not blind achieves a competitiveness less than $n-1$; this example has a

different flavor from any of the ones in [1]. The following simple (though not memoryless) randomized algorithm achieves a competitiveness of $n/2$ when the graph is the complete graph on n nodes with the same cost on every edge: fix a Hamiltonian cycle H through the graph. At the beginning of each round, the cat flips an unbiased coin to decide whether to traverse H clockwise or counter-clockwise during that round. Having made this decision, the cat traverses H in the appropriate direction until it catches up with the mouse. It is clear that no matter where the mouse is hiding, the expected cost of the cat in the round is $n/2$. It is easy to show that for this graph, no strategy can do better.

6 The Loop Ratio

In this section and in Section 7 we study some additional properties of resistive random walks that will prove to be of use in analyzing algorithms for metrical task systems in Section 9. Let C be a cost matrix, (σ_{ij}) be its generalized resistive inverse, and $\hat{C}$ be the resitive matrix such that (σ_{ij}) is the resistive inverse of $\hat{C}$: $\hat{c}_{ij} \leq c_{ij}$ and $\hat{c}_{ij} < c_{ij}$ only if $\sigma_{ij} = 0$. The resistive random walk does not use edges where $\hat{C}$ and C differ. Thus, the estimates given in Section 3 for E, the expected cost of a move and e_{ii} the expected cost of a round trip from vertex i are valid for nonresistive graphs as well:

$$E = \frac{2(n-1)}{\sum_{gh} \sigma_{gh}},$$

and

$$e_{ii} = \frac{2(n-1)}{\sum_{k} \sigma_{ik}}.$$

Let e_i be the expected cost of the first move out of node i;

$$e_i = \sum_j p_{ij} c_{ij} = \frac{\sum_j \sigma_{ij} c_{ij}}{\sum_j \sigma_{ij}}.$$

Define the *loop ratio* at i to be $L_i = e_{ii}/e_i$, and let the loop ratio of the walk be defined to be $L = \max_i L_i$.

Theorem 6.1 *Let P be the transition probability matrix designed by our procedure. Then $L \leq 2(n-1)$ for any cost matrix C on any n-node graph.*

Proof: We can assume without loss of generality that C is resistive (otherwise, we replace C by $\hat{C}$, with no change in the loop ratio). We have

$$L_i = \frac{2(n-1)}{\sum_j \sigma_{ij} c_{ij}} \quad \forall i.$$

It suffices to show that the denominator is ≥ 1. To this end, we use the fact that the matrices $\bar{\sigma}$ and $\bar{C}$ are inverses of each other. Consider the diagonal element $(\bar{\sigma}\bar{C})_{ii}$ of their product; thus

$$1 = c_{in} \sum_{j=1}^{n} \sigma_{ij} - \sum_{j=1}^{n-1} \sigma_{ij}(c_{in} + c_{jn} - c_{ij})/2.$$

Rearranging, we have

$$1 = \sum_{j=1}^{n} c_{ij}\sigma_{ij} + \sum_{j=1}^{n-1} \sigma_{ij}(c_{in} - c_{jn} - c_{ij})/2. \tag{5}$$

By the triangle inequality, every term in the second summation is ≤ 0, yielding the result. $\square$

Notice that $L_i = 2(n-1)$ if and only if every term in the second summation of (5) is equal to zero. The following simple example demonstrates that the equality $L_i = 2(n-1)$ can occur. Let $c_{ij} = \mid i - j \mid$. This forms a resistive cost matrix, and yields transition probabilities

$$p_{1,2} = p_{n,n-1} = \quad 1 \tag{6}$$

$$p_{i,i-1} = p_{i,i+1} = \quad 1/2, \quad 2 \leq i \leq n-1. \tag{7}$$

Thus the walk is the standard random walk on the line with reflecting barriers at vertices 1 and n. Clearly, $\phi_1 = \phi_n = 1/(2n-2)$, so that $e_{1,1} = e_{n,n} = 2n-2$. Further, $e_1 = e_n = 1$.

7 Graphs with Self Loops

The results of Sections 3–6 can be extended to graphs with self-loops. We assume now that each node is connected to itself by an edge ii with cost $c_{ii} > 0$. A random walk on this graph is defined by a transition probability

matrix (p_{ij}), where p_{ii} is the probability of using edge ii. The other definitions extend naturally.

Let C be the cost matrix for a graph G with self-loops, with vertex set $V = \{1, \ldots, n\}$. Construct a $2n$ vertex graph $\hat{G}$ without self loops, by adding an extra vertex on each self loop. The vertex set of $\hat{G}$ is $\hat{V} = \{1, \ldots, n, \hat{1}, \ldots \hat{n}\}$ and its edge set is $\hat{E} = \{ij : c_{ij} < \infty\} \cup \{i\hat{i} : c_{ii} < \infty\}$. The cost matrix $\hat{C}$ for the new graph is defined by

$$\begin{aligned} \hat{c}_{ij} = c_{ij}, \text{ if } i \neq j, \\ \hat{c}_{i\hat{i}} = \hat{c}_{\hat{i}i} = c_{ii}/2, \\ \hat{c}_{i\hat{j}} = \hat{c}_{\hat{i}j} = \hat{c}_{\hat{i}\hat{j}} = \infty, \text{if } i \neq j. \end{aligned}$$

Let $\hat{p}_{rs}$ be the transition probabilities for a random walk on the graph with cost matrix $\hat{C}$. Then

$$\hat{p}_{i\hat{j}} = \hat{p}_{\hat{i}j} = 0, \text{ if } i \neq j,$$

and $\hat{p}_{\hat{i}i} = 1$. Consider now the random walk for the original graph, with transition probabilities

$$\begin{aligned} p_{ij} = \hat{p}_{ij}, \text{ if } i \neq j \\ p_{ii} = \hat{p}_{i\hat{i}}. \end{aligned}$$

Let $\hat{\pi} = u_1, u_2, \ldots, u_k$ be a finite cost path in $\hat{G}$. If $u_j = \hat{i}$ then $u_{j-1} = u_{j+1} = i$. Let π be the path in G obtained from $\hat{\pi}$ by deleting all hatted nodes. Each loop of the form $i\hat{i}i$ is replaced by a self-loop of the form ii. Then π has the same cost as π', and the probability that path π occurs in the random walk defined by (p_{ij}) equals the probability that path $\hat{\pi}$ occurs i the random walk defined by $(\hat{p}_{ij})$. In particular, the random walks associated with (p_{ij}) and $(\hat{p}_{ij})$ have the same stretch. Conversely, given a random walk process on G we can build a random walk process on $\hat{G}$ so that corresponding paths (of the same cost) occur with the same probability.

We define the *resistive random walk* for an n-vertex graph G with self-loops to be the random walk derived from the resistive random walk on the $2n$-vertex graph $\hat{G}$. We have

Theorem 7.1 *Any random walk on a graph with self-loops has stretch* $\geq 2n - 1$. *The resistive random walk achieves a stretch of at most* $2n - 1$*;*

Let C' be the cost matrix C, with self-loops omitted (diagonal elements replaced by zeros). The generalized resistive inverse $(\hat{\sigma}_{ij})$ of the cost matrix $\hat{C}$ can be easily derived from the generalized resistive inverse (σ'_{ij}) of the cost matrix C'. Indeed, let D' be the discriminant for the matrix C' and let $\hat{D}$ be the discriminant for the matrix $\hat{C}$. Bearing in mind that the discriminant is the sum, over all spanning trees of the graph, of the product of the conductances of the edges in the spanning tree, we obtain that

$$\hat{D} = D' \cdot \prod_i \hat{c}_{i\hat{i}} = 2^{-n} \cdot D' \cdot \prod_i c_{ii}.$$

Thus, the generalized resistive inverse for $\hat{C}$ is the solution to the extremal problem

$$\begin{aligned} &\max_{\hat{\sigma}_{gh} \geq 0} && \log \hat{D} - \textstyle\sum_{gh} \hat{\sigma}_{gh} \hat{c}_{gh} \\ &= \max_{\hat{\sigma}_{gh} \geq 0} && \log D' - \textstyle\sum_{ij} c_{ij} \hat{\sigma}_{ij} + \sum_i (\log \hat{\sigma}_{i\hat{i}} - c_{ii} \hat{\sigma}_{i\hat{i}}/2). \end{aligned}$$

The solution to this problem is given by

$$\hat{\sigma}_{ij} = \sigma'_{ij}, \quad \hat{\sigma}_{i\hat{i}} = \frac{2}{c_{ii}}.$$

Let p'_{ij} be the transition probabilities of the resistive random walk on the loopless graph with cost matrix C', and p_{ij} be the transition probabilities for the resistive random walk on the graph with loops with cost matrix C. Then,

$$p_{ij} = \frac{\hat{\sigma}_{ij}}{\sum_k \hat{\sigma}_{ik}} = \frac{\sigma'_{ij}}{\sum_k \sigma'_{ik} + 2/c_{ii}},$$

and

$$p_{ii} = \frac{2/c_{ii}}{\sum_k \sigma'_{ik} + 2/c_{ii}}.$$

It follows that the conditional probability that the random walk uses edge ij, given that it does not use the self-loop ii, is

$$p_{ij}/(1 - p_{ii}) = \frac{\sigma'_{ij}}{\sum_k \sigma'_{ik}} = p'_{ij}.$$

Thus, the resistive random walk on a graph with self-loop is obtained from a probabilistic process whereby one first chooses whether to stay at the same

node; then, if the decision is to move to a new node, a move is made in the resistive random walk on the graph without loops.

Also,

$$\lim_{c_{ii}\to 0} \frac{1-p_{ii}}{c_{ii}} = \frac{1}{2}\sum_k {\sigma'}_{ik}.$$

Note that $c_{ii}/(1-p_{ii})$ is the expected cost of a sequence of moves starting from node i and ending at the first move out of node i (i.e., a maximal sequence of consecutive moves on the edge ii). We recall that ${e'}_{ii}$, the expected cost of a round trip from vertex i and back in the graph with cost matrix C', is given by

$${e'}_{ii} = \frac{2(n-1)}{\sum_k {\sigma'}_{ik}}.$$

Thus,

$$\lim_{c_{ii}\to 0} \frac{c_{ii}}{1-p_{ii}} = \frac{{e'}_{ii}}{n-1}.$$

In the limit, when the cost of a self-loop goes to zero, the expected cost of consecutive moves up to the first move out of node i is $1/(n-1)$ times the expected cost of a round trip from node i (ignoring self-loops).

8 The k-Server Problem

We consider now the *k-server problem* of Manasse *et al.* [18] defined in Section 1. We compare the performance of an on-line k-server algorithm to the performance of an adversary with k servers. The adversary chooses the next request at each step, knowing the current position of the on-line algorithm, and moves one of its servers to satisfy the request (if necessary). The on-line algorithm then moves one of its servers if necessary, without knowing the position of the adversary. The moves of the on-line algorithm may be probabilistic. The game stops after a fixed number of steps. The algorithm is *c-competitive* if there exists a constant a such that, for any number of steps and any adversary, E[cost on-line algorithm] $\leq c \cdot$ [cost adversary] $+ a$. Such an adversary is said to be an *adaptive on-line* [3, 21] adversary. One can weaken the adversary by requiring it to choose the sequence of requests in advance, so that it does not know of the actual random choices made by the on-line algorithm in servicing the request sequence; this is an *oblivious*

adversary. Alternatively, one can strengthen the adversary by allowing it to generate the requests adapting to the on-line algorithm's moves, but to postpone its decisions on its server moves until the entire sequence of requests has been generated; this is an *adaptive off-line* adversary. These three types of adversaries for randomized algorithms are provably different [3, 11, 21]. However, they all coincide when the on-line algorithm is deterministic. Furthermore, if there is a randomized algorithm that is c-competitive against adaptive on-line adversaries, then there is a c^2-competitive deterministic algorithm [3].

The *cache problem* where we manage a fully associative cache with k locations is a special case of the k-server problem [18]: we have a vertex for each possible memory item, and a uniform cost matrix with unit costs $c_{ij} = 1$. The *weighted cache* problem, where the cost of loading various items in cache may differ, is also an instance of the k-server problem [18, 21]: we have one vertex for each memory item, and a cost matrix $c_{ij} = (w_i + w_j)/2$, where w_i is the cost of loading item i in cache. (We are charging for each cache miss half the cost of the item loaded and half the cost of the item evicted; this yields the same results as if we were charging the full cost of the loaded item only.) Such a cost matrix corresponds to the distances between leaves in a star tree, where vertex i is connected to the star root by an edge of length $w_i/2$.

Theorem 8.1 *Let C be a resistive cost matrix on n nodes. Then we have a randomized k-competitive strategy for the k-server problem against an adaptive on-line adversary. More generally, if every $(k+1)$-node subgraph of C is resistive, we have a k-competitive strategy for the k-server problem on C.*

Proof: We exhibit a k-competitive randomized on-line algorithm for the more general case; we call this algorithm RWALK. If a request arrives at one of the k vertices that RWALK's servers cover (let us denote these vertices by $a_1, a_2, ..., a_k$), it does nothing. Suppose a request arrives at a vertex a_{k+1} it fails to cover. Consider the $(k+1)$-vertex subgraph C' determined by $a_1, a_2, ..., a_k, a_{k+1}$. By hypothesis, C' is resistive. Let σ' denote its resistive inverse. With probability

$$p'_i = \frac{\sigma'_{i,k+1}}{\sum_{j=1}^{k} \sigma'_{j,k+1}}$$

it selects the server at vertex a_i to move to vertex a_{k+1}. Since C' is finite, σ' is connected, and the denominator $\sum_{j=1}^{k} \sigma'_{j,k+1}$ is nonzero, the probabilities are well defined and sum to 1.

We need to prove that the RWALK is k-competitive. To this end, we define a *potential* Φ. (This is not to be confused with an electrical potential.) Say the RWALK's servers are presently at vertices $\mathbf{a} = \{a_1, a_2, ..., a_k\}$, and the adversary's servers are presently at vertices $\mathbf{b} = \{b_1, b_2, ..., b_k\}$, where $\mathbf{a}$ and $\mathbf{b}$ may overlap. We define $\Phi(\mathbf{a}, \mathbf{b})$ as the sum of the costs of all the edges between vertices currently occupied by RWALK's servers, plus k times the cost of a minimum-weight matching between vertices occupied by RWALK's servers and the adversary's servers. That is,

$$\Phi(\mathbf{a}, \mathbf{b}) = \sum_{1 \le i < j \le k} c_{a_i, a_j} + \min_{\pi} k \cdot \sum_{i=1}^{k} c_{a_i, b_{\pi(i)}},$$

where π ranges over the permutations on $\{1, 2, ..., k\}$. We also define a quantity Δ depending on the present position and the past history:

$$\Delta(\mathbf{a}, \mathbf{b}, \text{ History}) = \Phi(\mathbf{a}, \mathbf{b}) + (\text{RWALK's Cost}) \ - k \cdot (\text{Adversary's Cost}),$$

where both "Cost"s are cumulative. We will show that the expected value of Δ is a non-increasing function of time, and then show how this will imply the theorem.

Let us consider the changes in Δ due to (i) a move by the adversary (which could increase Φ), and (ii) a move by RWALK, which (hopefully) tends to decrease Φ. By showing that in both cases, the expected change in Δ is ≤ 0, we will argue that over any sequence of requests the expected cost of RWALK is at most k times the adversary's cost plus an additive term independent of the number of requests.

If the adversary moves one of its servers from b_j to b'_j, its cumulative cost is increased by c_{b_j, b'_j}. The potential Φ can increase by at most k times that quantity, since the minimum-weight matching can increase in weight by at most c_{b_j, b'_j}. (Obtain a new matching π' from the old one by matching $a_{\pi^{-1}(j)}$ to b'_j instead of b_j, and note that the weight of this new matching is no more than c_{b_j, b'_j} plus the weight of the old one; the new minimum-weight matching will be no heavier than this constructed matching.) So in this case Δ does not increase.

Next, we consider a move made by RWALK, and compare its cost to the expected change in Φ. First, we suppose that $\mathbf{a}$ and $\mathbf{b}$ overlap in $k-1$ places (later we remove this assumption):

$$a_i = b_i, \;\; i = 2, 3, ..., k; \;\; a_1 \neq b_1.$$

Define $b_{k+1} = a_1$. For convenience, set $m = k+1$, and let c_{ij}, σ_{ij}, for $i, j = 1, 2, ..., m$ be defined by $c_{ij} = c_{b_i, b_j}$. Recall equations (1-4) relating σ and C, specialized to the entries of interest:

$$\begin{aligned} \bar{\sigma}_{11} &= \sum_{j=2}^{m} \sigma_{1j} \\ \bar{\sigma}_{1j} &= -\sigma_{1j}, \; 2 \leq j < m \\ \bar{c}_{ji} &= [c_{jm} + c_{im} - c_{ji}]/2 \\ \sum_{j=1}^{m-1} \bar{\sigma}_{1j} \bar{c}_{ji} &= \delta_{1i}, i < m \end{aligned}$$

Multiply this last equation by 2 and sum over $i = 2, 3, ..., m-1$, noticing that in this range $\delta_{1i} = 0$. We obtain:

$$\begin{aligned} 0 &= 2 \sum_{i=2}^{m-1} \sum_{j=1}^{m-1} \bar{\sigma}_{1j} \bar{c}_{ji} \\ &= 2 \sum_{i=2}^{m-1} \left(\bar{\sigma}_{11} \bar{c}_{1i} + \sum_{j=2}^{m-1} \bar{\sigma}_{1j} \bar{c}_{ji} \right) \\ &= \sum_{i=2}^{m-1} \left\{ \sum_{j=2}^{m} \sigma_{1j} \left[c_{1m} + c_{im} - c_{i1} \right] \; - \; \sum_{j=2}^{m-1} \sigma_{1j} \left[c_{jm} + c_{im} - c_{ji} \right] \right\} \end{aligned}$$

For $j = m$ the latter bracketed expression $[c_{jm} + c_{im} - c_{ji}]$ is zero, so we can include it in the sum, extending the limits of summation to m:

$$\begin{aligned} 0 &= \sum_{i=2}^{m-1} \left\{ \sum_{j=2}^{m} \sigma_{1j} [c_{1m} + c_{im} - c_{i1}] \; - \; \sum_{j=2}^{m} \sigma_{1j} [c_{jm} + c_{im} - c_{ji}] \right\} \\ &= \sum_{j=2}^{m} \sigma_{1j} \left[(m-2) c_{1m} + \sum_{i=2}^{m-1} c_{im} - \sum_{i=2}^{m-1} c_{i1} - (m-2) c_{jm} - \sum_{i=2}^{m-1} c_{im} + \sum_{i=2}^{m-1} c_{ji} \right] \end{aligned}$$

$$= \sum_{j=2}^{m} \sigma_1 j \left[(m-1)c_{1m} - \sum_{i=2}^{m} c_{i1} - (m-1)c_{jm} + \sum_{i=2}^{m} c_{ji} \right]$$
$$= \sum_{j=2}^{m} \sigma_1 j \left[(m-1)c_{1m} - \sum_{i=2}^{m} c_{i1} - (m-1)c_{jm} + \sum_{1 \le i \le m,\, i \ne j} c_{ji} - c_{j1} \right]$$

Defining

$$\tau_\ell = (m-1)c_{\ell m} + \sum_{1 \le i < j \le m,\, i,j \ne \ell} c_{ij} = (m-1)c_{\ell m} + \sum_{1 \le i < j \le m} c_{ij} - \sum_{i=1}^{m} c_{i\ell}$$

we discover

$$\sum_{j=2}^{m} \sigma_{1j}[\tau_1 - \tau_j - c_{j1}] = 0.$$

The value of Φ before RWALK makes its move is τ_1. If the server at a_j is moved then the value of Φ after ther move is τ_j, and the cost of the move is c_{j1}. Thus, the expected change in Δ, as RWALK makes its random move with probability $(\sigma_{1j})/(\sum_{i=2}^{m} \sigma_{1i})$, is

$$\frac{1}{\sum_{i=2}^{m} \sigma_{1i}} \times \sum_{j=2}^{m} \sigma_{1j}[\tau_j - \tau_1 + c_{j1}] = 0.$$

The expected change in Δ is zero on RWALK's move.

Finally, we verify the case in which $\mathbf{a}$ and $\mathbf{b}$ overlap in fewer than $k-1$ vertices, and RWALK makes a move. Suppose the request is at vertex b_1. Suppose the current minimum-weight matching pairs a_i with b_i, $i = 1, 2, \ldots, k$. Let $b'_1 = b_1$, and $b'_i = a_i$, $i = 2, \ldots, k$. Let Φ' be the potential computed using $\mathbf{b}'$, rather than $\mathbf{b}$. We obtain, from the previous analysis, that the expected decrease in Φ' is equal to the expected cost of RWALK's move. The true potential Φ differs from Φ' only in the weight of the minimum-weight matching. Suppose that RWALK moves the server at a_j to b_1. Then, Φ' decreases by

$$c_{a_1,a_j} - c_{a_1,b_1}.$$

Consider a new matching, not necessarily of minimum weight, after our current move from a_j to b_1, obtained from the old matching by matching a_1 to b_j, a_j to b_1, and a_i to b_i for $i \ne 1, j$. This new matching differs from the old one by

$$c_{a_1,b_j} - c_{a_1,b_1} - c_{a_j,b_j} \le c_{a_1,a_j} - c_{a_1,b_1}$$

by the triangle inequality. Thus, Φ decreases by at least $c_{a_1,a_j} - c_{a_1,b_1}$. It follows that the expected decrease of Φ is no smaller than the expected decrease of Φ' and, hence, no smaller than the expected cost of RWALK's move.

So the expected value of

$$\Delta(\mathbf{a}, \mathbf{b}, \text{History}) = \Phi(\mathbf{a}, \mathbf{b}) + (\text{RWALK's Cost}) - k \cdot (\text{Adversary's Cost})$$

is nonincreasing at every step. Since Φ is positive, we find that

$$(\text{RWALK's Cost}) - k \cdot (\text{Adversary's Cost})$$

remains bounded, in expectation, by the initial value of Δ. So the competitiveness is k. $\square$

The last result is valid even if the graph is infinite; one only requires that the cost of a simple path be bounded and every $k+1$-node subgraph be resistive. The potential Φ we developed to prove the last result seems to be very natural and useful for the server problem. It has been subsequently used by several authors [8, 9] for analyzing algorithms for the k-server problem. While the second term in our potential function (involving the minimum weight perfect matching) is a natural measure of the distance between an on-line algorithm's servers and those of an off-line algorithm, the first term $(\sum_{1 \le i < j \le k} c_{a_i,a_j})$ was originally motivated by the hitting potentials of the random walk P on a graph with cost matrix C.

As corollaries of Theorem 8.1, we have optimal k-competitive randomized algorithms against an adaptive on-line adversary for the two server problem ($k = 2$) in any metric space [18], for servers on a line [8], for the uniform cost (cache) problem [22], for the weighted cache problem [8, 21], and for servers on a tree [9]. These algorithms are extremely simple, and memoryless. Berman *et al.* [4] prove that the HARMONIC algorithm [21] for 3 servers achieves a finite competitiveness in any metric space, and Grove [15] shows that this is true for all k. Recently, Fiat, Rabani and Ravid [12] have announced a deterministic k-server algorithm that achieves a competitiveness bounded by some function of k. At present, all known cases where we know of k-competitive on-line algorithms are in (special cases of) resistive metric spaces. Our theory based on resistive random walks unifies our current picture of the k-server conjecture, and implies k^2-competitive deterministic algorithms in resistive spaces [3].

Our algorithm does not yield k-competitive algorithms in every graph. The smallest counterexample we know of consists of a 3-server problem on a five-node graph. The five nodes can be thought of as being on the periphery of a circle of circumference 8 centered at the origin. One node lies on the x-axis, and the others are at counter-clockwise distances 1,3,5 and 6 from it along the circumference. In this case it is possible to give an infinite request sequence on which the competitiveness of our algorithm exceeds 3 (we do not know that it can be arbitrarily large, however). Moreover, as we show below, a simple modification of our algorithm achieves a competitiveness of at most $2k$ for points on the periphery of a circle.

Theorem 8.1 can be used to derive randomized competitive k-server algorithms for non-resistive spaces as well, when these can be approximated by resistive spaces. For real $\lambda > 1$, a cost matrix C' is a λ-*approximation* for the matrix C if, for all i, j, $c'_{ij}/\lambda \leq c_{ij} \leq c'_{ij}$. If a server algorithm is g-competitive for the matrix C', then it is λg-competitive for the matrix C. Using this observation, we can derive a $2k$-competitive algorithm for k servers on a circle, with distances being measured along the circumference. Consider points on a circle, with the cost c_{ij} between two points i, j given as the distance along the smaller arc joining them. We can construct a 2-approximation C' to this cost C. Each arc of the circle becomes a resistor with resistance equal to the arc-length. If the smaller and larger arc distances joining two points are a, b respectively, then the effective resistance c' is $ab/(a+b)$ while $c = a \leq b$. Then easily $c' \leq c \leq 2c'$. In conjunction with results in [3], this implies that there is a $4k^2$-competitive deterministic algorithm for k servers on the circle.

In the preceding paragraph, we made use of the fact that the distance matrix induced by a set of points on a circle has a resistive 2-approximation. For some metric spaces this is not possible.

Theorem 8.2 *For any $\lambda > 1$ there is a finite set of points in the Euclidean plane for which the Euclidean distance matrix cannot be λ-approximated by any resistive cost matrix.*

Proof: In what follows, let $L(x, y)$ denote the Euclidean distance between two points, and $R(x, y)$ the distance between them in the putative λ-approximation, so that $L(x, y)/\lambda \leq R(x, y) \leq L(x, y)$.

Given two points w, y, we define the *rhombus construction on* (w, y) as follows. Construct a rhombus whose major diagonal, the line segment (w, y),

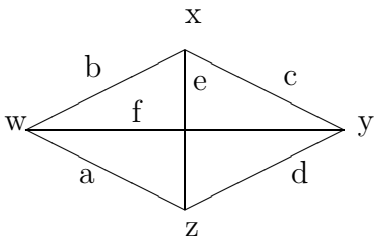

Figure 1: The Euclidean plane has no resistive approximation.

has length $2\ell \equiv L(w,y)$, and whose minor diagonal (x,z) has length $L(x,z) = \ell/\lambda$. Of course $L(w,x) = L(x,y) = L(y,z) = L(z,w) = \ell\sqrt{1+1/(2\lambda)^2}$.

Let the effective resistances be as given in Figure 1; for example, $f = R(w,y)$. We have

$$e = R(x,z) \geq L(x,z)/\lambda = \ell/\lambda^2,$$

and

$$f = R(w,y) \leq L(w,y) = 2\ell,$$

so that

$$e \geq f/(2\lambda^2).$$

Assume without loss of generality that $c = \max(a,b,c,d)$. Suppose the four-point network (w,x,y,z) has a resistive inverse. Consider the distance matrix C for these four points and the associated matrix $\bar{C}$ (recall the definitions in Section 3):

$$C = \begin{bmatrix} 0 & a & f & b \\ a & 0 & d & e \\ f & d & 0 & c \\ b & e & c & 0 \end{bmatrix}, \; \bar{C} = \frac{1}{2}\begin{bmatrix} 2b & b+e-a & b+c-f \\ b+e-a & 2e & c+e-d \\ b+c-f & c+e-d & 2c \end{bmatrix}.$$

Inverting $\bar{C}$ to find $\bar{\sigma}$, and demanding that the link (w,y) have nonnegative link conductance, we find

$$\begin{aligned} 0 &\geq (b+e-a)(c+e-d) - 2e(b+c-f) \\ &= e[e+2f-(a+b+2c)] + (e+b-a)(c-d) \end{aligned}$$

By the triangle inequality, $e + b - a \geq 0$, and by assumption $c - d \geq 0$, so that $0 \geq e + 2f - (a + b + 2c)$; that is,

$$4c \geq a + b + 2c \geq 2f + e \geq f\left(2 + \frac{1}{2\lambda^2}\right),$$

so that

$$c \geq f\left(\frac{1}{2} + \frac{1}{8\lambda^2}\right).$$

It follows that

$$\begin{aligned} \frac{R(x,y)}{L(x,y)} &\geq \frac{R(w,y)\,(1/2 + 1/8\lambda^2)}{\frac{1}{2}L(w,y)\sqrt{1 + 1/4\lambda^2}} \\ &= \left(\frac{R(w,y)}{L(w,y)}\right)\sqrt{1 + \frac{1}{4\lambda^2}}. \end{aligned}$$

For at least one of the four sides (x, y) of the rhombus, the ratio $R(x, y)/L(x, y)$ is greater than the corresponding ratio for the major diagonal $R(w, y)/L(w, y)$ by a factor of at least $\sqrt{1 + 1/4\lambda^2}$.

Now we build the promised counterexample. Set

$$K = 1 + \lfloor \log \lambda / \log \sqrt{1 + 1/(4\lambda^2)} \rfloor.$$

Begin with a pair of points w, y in the plane. Perform the rhombus construction on (w, y). For each of K steps, perform the rhombus construction on each of the sides $(w, x), (x, y), (y, z), (z, w)$ constructed during the previous step. Suppose we have a λ-approximation to the Euclidean metric on this finite graph. Set $w_0 = w, y_0 = y$. Iteratively for $k = 1, 2, ..., K$, let (w_k, y_k) be that side of the rhombus constructed on (w_{k-1}, y_{k-1}) with the largest effective resistance R. We get

$$\left(\frac{R(w_K, y_K)}{L(w_K, y_K)}\right)\left(\frac{R(w_0, y_0)}{L(w_0, y_0)}\right)^{-1} \geq \left(\sqrt{1 + 1/(4\lambda^2)}\right)^K > \lambda,$$

a contradiction. So no such λ-approximation can exist. We conclude that there is no resistive network whose effective resistance approximates distances in the Euclidean plane within a constant factor. $\square$

Thus, the approximation technique does not solve the server problem in the plane.

We now turn to the case $k = n - 1$.

Theorem 8.3 *Let C be any cost matrix on n nodes. Then RWALK is an $(n-1)$-competitive strategy for the $k = n-1$ server problem on C.*

Note that Theorem 8.3 holds even when the c_{ij} do not satisfy the triangle inequality.

Proof: Both the on-line algorithm and the adversary have one unoccupied node which we consider, respectively, to be "cat" and "mouse". Whenever a server moves from i to j the cat (resp. the mouse) moves from j to i, at cost $c_{ij} = c_{ji}$. We can assume that the adversary always requests the unique node (cat's position) which is not occupied by the on-line algorithm (see [21]). It has to move one of its own servers to satisfy this request only when the positions of the cat and of the mouse coincide. This situation corresponds exactly to the cat-and-mouse game, and the result follows from Theorem 5.1. □

9 Metrical Task Systems

We now consider Metrical Task Systems, introduced by Borodin *et al.* [5]. A metrical task system has the set S of positions $\{1, 2, \cdots n\}$ with an associated cost matrix C (positive off-diagonal, zero on diagonal, symmetric, triangle inequality) where c_{ij} is the cost of moving between position i and position j. An algorithm can reside in exactly one of the positions at any time. A task is a vector $T = (T(1), T(2), \cdots T(n))$ where $T(j)$ is the cost of processing t while in position j. Given a sequence of tasks $\mathcal{T} = T^1, T^2 \cdots$, an algorithm must decide for each task T^i the position $\sigma(i)$ it is processed in. The cost of this schedule is the sum of all position transition costs and task processing costs:

$$c(\mathcal{T}; \sigma) = \sum_i c_{\sigma(i-1),\sigma(i)} + \sum_i T^i(\sigma(i)).$$

An on-line algorithm must decide on $\sigma(i)$ before knowledge of any task $\sigma(j), j > i$. The next position may be chosen probabilistically.

Here too, we consider adaptive on-line adversaries that generate the sequence of tasks and satisfy them on-line: the adversary selects the next task at each step, knowing the current state of the on-line algorithm, and possibly changes position to satisfy the request. An on-line algorithm is c-competitive if there exists a constant a such that, for any number of steps and any on-line adversary, $E[\text{cost of on-line algorithm}] \leq c \cdot E[\text{cost adversary}] + a$. One can

weaken the adversary to be oblivious, and require it to choose the sequence of tasks in advance. Or, one can strengthen it to be adaptive off-line, by allowing it to postpone its decision on its moves until the entire sequence of tasks has been generated.

In this section, we give a lower bound of $2n-1$ on the competitiveness of any randomized on-line algorithm against such an adaptive on-line adversary, and complement this with a simple, memoryless randomized algorithm that achieves this bound.

Borodin *et al.* [5] define an on-line algorithm for metrical task systems to be a *traversal algorithm* if:
(1) The positions are visited in a fixed sequence $s_1, s_2, \cdots$ independent of the input task sequence $\mathcal{T}$.
(2) There is a sequence of positive *threshold costs* $c_1, c_2, \cdots$ such that the transition from s_j to s_{j+1} occurs when the total task processing cost incurred since entering s_j reaches c_j. In fact, they set $c_j = c_{s_j, s_{j+1}}$. Thus, the total cost incurred by the on-line algorithm is exactly twice the moving cost.

There is a technical difficulty to take care of: the accumulated cost incurred at a node can jump substantially above the threshold c_j. Borodin *et al.* overcome this difficulty by considering *continuous* schedules, where position changes can occur at arbitrary real points in time, and a task may be processed in several positions for fractional periods of time (the ith task is continuously serviced during the ith time interval). Thus, a transition always occurs when the accumulated cost exactly reaches the threshold. Such a continuous schedule can always be replaced by a discrete schedule which is no worse: if the continuous schedule visits several positions during one time interval, then the corresponding discrete schedule will stay in that position where the current task is cheapest to process. The state of an on-line algorithm consists of its "virtual position", the position it would arrive at using a continuous schedule, and of the accumulated processing cost at that virtual position. The real position of the algorithm may be distinct from its virtual position. We shall analyze the costs of the algorithm assuming it uses a continuous schedule and is at its virtual position, bearing in mind that real costs are no larger. The reader is referred to [5] for details.

We begin with a negative result:

Theorem 9.1 *For any metrical task system with n positions, no randomized algorithm achieves a competitiveness lower than* $2n-1$ *against an adaptive*

on-line adversary.

An immediate corollary of Theorem 9.1 is a lower bound of $2n-1$ for deterministic algorithms for metrical task systems; although this result appears in [5], our proof here seems to be considerably simpler. The proof strategy uses ideas from the proof of Theorem 5.2.

Proof of Theorem 9.1: Let N be the length of the request sequence. Let us denote the on-line algorithm by R. The adversary will be a *cruel taskmaster* (in the terminology of Borodin *et al.* [5]): at each step, it presents R with a task with processing cost ϵ at that position that is currently occupied by R, and zero in all other positions. The value of ϵ will be specified later. Given the initial position of R, let R_M denote the expected cost that R pays in moving between positions in response to a sequence of length N generated by a cruel taskmaster; let R_P denote the expected cost that R pays in processing tasks, and let $R_T = R_M + R_P$ denote the expected total cost incurred by R.

We distinguish between two cases.

Case 1: $R_M/R_T < (n-1)/(2n-1)$.

We give an on-line algorithm for the adversary whose expected cost is at most $R_M/(n-1)$; since $R_T > (2n-1)R_M/(n-1)$, the lower bound on competitiveness follows. We derive this algorithm by first giving $n-1$ on-line algorithms that together pay an expected total cost of R_M; the adversary selects one of these uniformly at random to achieve the expected cost $R_M/(n-1)$.

We now describe the $n-1$ possible on-line algorithms for the adversary. Each starts at at different one of the $n-1$ positions not occupied initially by R. Whenever one of these algorithms faces a task having positive cost in its current position, it moves to that position just vacated by R. It is easy to see that no two of these algorithms ever enter the same position, so that at most one moves in response to any task. None of these $n-1$ on-line algorithms ever pays a cost for task processing, and their total moving cost equals the total moving cost of R on the sequence. Thus the expected total cost of these $n-1$ on-line algorithms is R_M as desired.

Case 2: $R_M/R_T \geq (n-1)/(2n-1)$.

In this case (by simple manipulation) $R_T \geq (2n-1)R_P/n$. Let $d = \min_{i,j} c_{ij}$; this is the minimum moving cost an algorithm must pay anytime it moves. We will choose ϵ to be small compared to d. On the request sequence of N tasks, let N_1 be the number of tasks to which R responds by moving to

a new position (incurring a cost of at least d for each of these moves), and $N_2 = N - N_1$ the number of tasks on which R remains in its position. Thus $R_P = \epsilon N_2$.

We will exhibit an on-line algorithm for the adversary paying an expected total cost of $\epsilon N/n$ on the sequence. If $N_1 > (2\epsilon/d)N$, we are done because the moving cost paid by R is at least $2\epsilon N$, which is bigger than $2n-1$ times the cost of the adversary's algorithm. Therefore assume $N_1 < (2\epsilon/d)N$, so that $N_2 \geq N(1 - 2\epsilon/d)$. Thus

$$R_P \geq \epsilon N(1 - \frac{2\epsilon}{d}),$$

and therefore

$$R_T \geq \frac{2n-1}{n}\epsilon N(1 - \frac{2\epsilon}{d}).$$

It remains to exhibit an on-line algorithm for the adversary whose expected cost on this sequence is at most $\epsilon N/n$. We do so by giving n possible on-line algorithms for the adversary that together pay a cost of ϵN; choosing randomly from among them will yield the result as in Case 1. Each of the n algorithms stays in a different one of the n positions throughout the game, never moving at all. Thus these n on-line algorithms never pay any moving costs, and their net task processing cost on any sequence equals $N\epsilon$. Thus their expected total cost is ϵN, as claimed.

The proof is completed by letting ϵ go to zero, much as in the proof of Borodin *et al.* [5]. □

Ben-David *et al.* [3] have studied the relative powers of the adaptive on-line and adaptive off-line adversaries. They show, in a very general game-theoretic setting (see also [21]), that randomization affords no benefit against an adaptive off-line adversary. More precisely, they showed that if the competitiveness of any deterministic algorithm is at least c, then no randomized algorithm can achieve a competitiveness lower than c against an adaptive off-line adversary. They left open the possibility that in some situations, an on-line algorithm could do better against an adaptive on-line adversary. There is a lower bound of k on the competitiveness of any algorithm [18, 21] for the k-server problem against an adaptive on-line adversary; for many special cases of the problem k is also an upper bound. Theorem 9.1 provides further evidence that randomization affords no help against an adaptive on-line adversary, proving as it does an analogous result for metrical task systems.

We now give two simple randomized algorithms for the metrical task systems of Borodin *et al.* [5]. The first is a variant of the traversal algorithm of Borodin *et al.* [5], and is $4(n-1)$-competitive — the traversal algorithm of Borodin *et al.* [5] is $8(n-1)$-competitive. We then make a slight modification to our algorithm to obtain a simple, traversal algorithm that is $(2n-1)$-competitive. A further modification of this algorithm yields a memoryless algorithm that is $(2n-1)$-competitive. This is optimal against an adaptive on-line adversary, as we have shown in Theorem 9.1. Borodin *et al.* [5] present a deterministic algorithm that is $(2n-1)$-competitive, but their algorithm differs from ours in that it is not a traversal algorithm, and uses memory.

Our first algorithm is similar to a traversal algorithm, with two changes:
(i) We do not employ a single fixed sequence of positions for the traversal, but rather execute a random walk through the positions following the transition probabilities derived from a resistive inverse (or its modification as in section 4) of the distance matrix d of the task system. The set of positions we visit is nevertheless independent of $\mathcal{T}$.
(ii) Let e_i be the expected cost of a move out of position i, given that we are currently at position i. We make a transition out of the current position s when the total task processing cost incurred since entering this position equals e_i (the machinery of continuous schedules is used to achieve equality).

This algorithm still has the property that the expected total cost incurred by the on-line algorithm is twice the expected cost of the moves done by the algorithm, for any adversary strategy. The design of the random walk is done once at the beginning, and assigns to each position the next-position transition probabilities. This determines the move threshold costs e_i for all positions i. Thus, the algorithm is on-line. It is not memoryless, since it uses a counter for the accumulated task processing cost in the current position.

Theorem 9.2 *The above algorithm based on a random walk with stretch c and loop ratio ℓ is $2\max(c,\ell)$-competitive.*

Proof: We can assume without loss of generality that the adversary is a "cruel taskmaster" that generates a task which has positive cost only at the position currently occupied by the on-line algorithm. Also, one can assume without loss of generality that the adversary changes position only at the time the on-line adversary reaches its current position.

We consider the computation as consisting of a sequence of phases; a new phase starts when the on-line algorithm reaches a position where the

adversary currently is. Suppose this is position i. There are two possibilities:
(i) The adversary moves to position j at the start of the current phase, and then stays put at j during the entire phase. The adversary has then a cost of c_{ij} for the current phase, whereas the on-line algorithm has an expected moving cost e_{ij}.
(ii) The adversary stays put in position i during the entire phase. Then the adversary has a cost of e_i, whereas the on-line algorithm has an expected moving cost of e_{ii}, for the current phase.

We distinguish *moving phases*, where the adversary changes position, and *staying phases*, where the adversary stays in the same position. Then E[on-line moving costs in moving phases] $\leq c \cdot$ (adversary cost of moving phases) $+ a$, and E[on-line moving costs in staying phases] $\leq \ell \cdot$ (adversary cost in staying phases). The theorem follows from the fact that the total expected costs of the on-line algorithm are twice its moving costs. $\square$

The resistive random walk has a stretch of $n-1$ and loop ratio $2(n-1)$, thus yielding a $4(n-1)$-competitive algorithm.

We apparently gain the factor of 2 over the Borodin *et al.* [5] traversal algorithm because they round up all edge costs to the nearest power of 2, whereas we avoid this rounding and directly use the edge costs for our resistive inverse. We now describe a modification that brings the competitiveness of our algorithm down to $2n-1$. In the traversal algorithm described above (and in that of Borodin *et al.* [5]), we move out of position j when the cumulative task processing cost incurred in that position reaches the expected cost of the next move; instead, we will now make the move when the task processing cost in j reaches β_j, where β_j is a threshold associated with vertex j. This allows us to introduce two improvements:
1. The loop ratio L_j is not the same for all j, so that some positions are better for the off-line adversary to "stay put" in than others. The choice of different thresholds will compensate for this imbalance.
2. In our traversal algorithm, we fared better against an adversary who moved than we did against an adversary who stayed at one place; we will correct this imbalance as well in the improved scheme.

Let p_{ij} be the transition probabilities for the resistive random walk on matrix C, let e_{ii} be the expected cost of a round trip from i, and let e_i be the expected cost of a move out of node i, given that the current position is

i. Our choice of β_i will be

$$\beta_i = \frac{2}{\sum_j \sigma_{ij}} = \frac{e_{ii}}{(n-1)}.$$

We show below that this corrects both the imbalances described above.

Theorem 9.3 *The modified random traversal algorithm is* $(2n-1)$*-competitive.*

Proof: The proof is similar to the proof for the previous theorem. We can assume without loss of generality that the adversary is a "cruel taskmaster" and changes position only at the time the on-line adversary reaches its current position. As we saw in Section 3, the average cost of a move of the on-line algorithm is $E = 2(n-1)/\sum_{gh} \sigma_{gh}$, and the steady state probability of vertex i is $\phi_i = \sum_j \sigma_{ij} / \sum_{gh} \sigma_{gh}$. Thus, the expected task processing cost per move is $\sum_i \phi_i \beta_i = 2n/\sum_{gh} \sigma_{gh}$. It follows that in our random walk, the expected ratio of total cost to move cost is $(2(n-1) + 2n)/2(n-1) = (2n-1)/(n-1)$. Thus, it suffices to show that the expected moving costs of the on-line algorithm are at most $n-1$ times the costs of the adversary.

We proceed as in the proof of the previous theorem, assuming a continuous schedule, and distinguishing between staying phases, where the adversary does not move, and moving phases, where the adversary changes positions.

The cost for the adversary of a staying phase starting (and ending) at node i is β_i; the expected moving cost of that phase for the on-line algorithm is $e_{ii} = (n-1)\beta_i$. The sequence of moving phases can be analyzed as a cat and mouse game, thus also yielding a ratio of $n-1$. □

The last algorithm is not memoryless: it needs to store the present virtual node, and a counter for the accumulated task processing cost at that node. We replace this counter by a "probabilistic counter", thus obtaining a memoryless algorithm. Consider the following continuous traversal algorithm: if the on-line algorithm is at position i and the cost of the current task at position i is w, then the length of stay of the algorithm at position i is exponentially distributed, with parameter w/β_i. One can think of the algorithm as executing a continuous, memoryless process that decides when to move. The probability of a move in any interval depends only on the interval length, and the expected length of stay is β_i/w. Such a process is the limiting case of a sequence of Bernoulli trials executed at successively

shorter intervals, i.e. the limit case of a traversal algorithm of the previous form.

Before we analyze this algorithm, we introduce two modifications. It turns out that the ith task can be processed at any of the positions visited during the ith unit interval. We shall assume it is processed in the last such position. Also, one need not visit the same position more than once during one time interval. The modified algorithm MEMORYLESS is formally described below.

> Let (p_{ij}) be the transition probabilities for the resistive random walk on the system graph. Assume the on-line algorithm is presented with task $T(1), \ldots, T(n)$. Let $p'_{ii} = e^{-T(i)/\beta_i}$, and $p'_{ij} = (1 - e^{-T(i)/\beta_i})p_{ij}$, if $i \neq j$. The random algorithm executes a random walk according to the probability distribution p'_{ij}, until it returns to a position already visited. It then selects this position as its new position.

(In reality, one need not execute an unbounded sequence of random moves. Given the cost vector $T(1), \ldots, T(n)$, and the probabilities p_{ij} one can compute directly the probability that position j will be selected by the experiment, for each j. One can then select the next position by one random choice.)

Algorithm MEMORYLESS is memoryless: the next position depends only on the current position and the current task.

Theorem 9.4 *Algorithm MEMORYLESS is* $(2n-1)$*-competitive.*

Proof: We begin be observing that we can assume without loss of generality that the adversary generates only "cruel" tasks that have nonzero cost only at the position occupied by the on-line algorithm. Intuitively, the submission of a task with several nonzero costs amounts to the submission of several unit tasks in one batch; the adversary gives up some power to adapt by doing so. Formally, suppose that the on-line algorithm is in position i, and the adversary generates a task $\mathcal{T} = (T(1), \ldots, T(n))$, and moves to a new position s. Assume, instead, that the adversary generates a sequence of cruel requests, according to the following strategy:

Generate a task with cost $T(i)$ in position i, 0 elsewhere; if the on-line algorithm moves to a new position j, then generate the task with cost $T(j)$

in position j, zero elsewhere; continue this way, until the on-line algorithm returns to an already visited position. The adversary moves to position s at the first step in this sequence.

One can check the following facts:
(1) The probability that the on-line algorithm is at position j at the end of this sequence of requests, is equal to the probability that the on-line algorithm moves to position j when submitted task $\mathcal{T}$.
(2) The expected cost for the adversary of the sequence of tasks thus generated is $\leq T(s)$, the cost of $\mathcal{T}$ for the adversary. Indeed, the sequence may contain at most one task with non-zero cost $T(s)$ at position s.
(3) The expected cost for the on-line algorithm of the sequence of tasks thus generated is $\geq$ the cost of $\mathcal{T}$ for the on-line algorithm. Indeed, if j is the next position of the on-line algorithm then, when submitted $\mathcal{T}$, the on-line algorithm pays $T(j)$, which is the cost of the last task in the sequence.

Observe that when the adversary generates only cruel tasks, the process of selecting the next position is simplified: if the on-line algorithm is in position i, and the current task has cost $w = T(i)$ in position i, zero elsewhere, then the next position is i with probability e^{-w/β_i}, $j \neq i$ with probability $p_{ij}(1 - e^{-w/\beta_i})$.

Let ϵ be a fixed positive real number. Assume first that the adversary generates only cruel tasks with cost ϵ in the position currently occupied by the adversary, zero elsewhere. Let $C(\epsilon)$ be the cost matrix obtained from C by adding a self-loop of cost ϵ at each node. Let $p_{ij}(\epsilon)$ be the transition probabilities for the resistive walk in this augmented graph ($p_{ij}(\epsilon) = p_{ij}(1 - p_{ii}(\epsilon)$, if $i \neq j$). This walk has stretch $2n - 1$. Consider an on-line algorithm that performs a random walk with transition probabilities $p_{ij}(\epsilon)$, where a transition from i to i represents a choice of staying at position i. We can assume without loss of generality that the adversary changes position only if it occupies the same position as the on-line algorithm. The on-line algorithm pays a cost of c_{ij} whenever it moves from position i to position j; it pays a (task processing) cost of $\epsilon = c_{ii}$ whenever it stays put in its current position. The same holds true for the adversary. Thus, the situation reduces to a cat and mouse game, and the algorithm is $2(n-1)$-competitive.

Assume next that the adversary generates cruel tasks of cost $k\epsilon$, k an arbitrary integer. Consider the on-line algorithm derived from the previous one, by considering a unit task of cost $k\epsilon$ to consist of k tasks of cost ϵ: The algorithm performs up to k successive trials: at each trial it moves to position

j with probability $p_{ij}(\epsilon)$; if $j \neq i$ then it halts. The new algorithm does no worse than the previous one, and is $(2n-1)$-competitive.

Let $w = k\epsilon$ the cost of the current task. The algorithm stays in the same position i with probability

$$(p_{ii}(\epsilon))^k = (p_{ii}(\epsilon))^{w/\epsilon};$$

it moves to a new position $j \neq i$ with probability

$$p_{ij} \cdot (1 - (p_{ii}(\epsilon))^k).$$

We have, by the results of Section 7

$$\lim_{\epsilon \to 0} \frac{1 - p_{ii}(\epsilon)}{\epsilon} = \frac{1}{\beta_i}.$$

Thus

$$\begin{aligned} \lim_{\epsilon \to 0}(p_{ii}(\epsilon))^{w/\epsilon} &= \lim_{\epsilon \to 0}(1 - \frac{\epsilon}{\beta_i})^{w/\epsilon} \\ &= e^{-w/\beta_i}. \end{aligned}$$

The transition probabilities of the previous algorithm converge to the transition probabilities of algorithm MEMORYLESS, when $\epsilon \to 0$. It follows, by a continuity argument, that MEMORYLESS is $(2n-1)$-competitive. □

10 Discussion and Further Work

Fiat *et al.*[11] give a randomized k-server algorithm than achieves a competitiveness of $O(\log k)$ in a graph with the same cost on all edges, against an oblivious adversary. Can a similar result be obtained for the k-server problem on other graphs? One consequence of Theorem 2.1 is that no memoryless k-server algorithm can achieve a competitiveness lower than k in any graph if it moves at most one server in response to a request. This follows from restricting the game to a $k+1$-node subgraph, so that we then have a cat-and-mouse game; since the cat is memoryless, it executes a random walk and the lower bound of Theorem 2.1 applies.

We now list several open problems raised by our work.

We do not know what stretch can be achieved by random walks when the cost matrix C is not symmetric.

It would be interesting to study the cat-and-mouse game under a wider class of strategies, for the case when the cat is not blind; this would extend the interesting work of Baeza-Yates *et al.* [1].

We have no results for the k server problem in general metric spaces. We would like to prove that the resistive random walk yields a server algorithm that achieves a competitiveness that is a function of k alone, in any metric space (against an adaptive on-line adversary). This would yield [3] a deterministic algorithm having finite competitiveness in an arbitrary metric space. Fiat, Rabani and Ravid [12] have recently given a deterministic algorithm whose competitiveness depends only on k.

We can prove that RWALK is $2k-1$-competitive in any metric space against a *lazy* adaptive on-line adversary that moves only when it must: whenever there is a node occupied by an adversary server that is not occupied by an on-line algorithm's server, the adversary requests such node. The *lazy adversary conjecture* is that the resistive on-line algorithm achieves its worst performance against a lazy adversary. A proof of this conjecture would show that the resistive algorithm is $2k-1$-competitive in any metric space. The reason is as follows: provided the (adaptive on-line) adversary plays by the lazy adversary conjecture, the operation of the algorithm on any sequence can be broken up into phases. At the beginning of each phase the k points occupied by RWALK's servers are exactly those occupied by the adversary's. The adversary moves a server at the first request of each phase, and makes no other move during the round. At every instant of the phase, there are $k-1$ points at which both RWALK and the adversary have servers, and two additional points one of which is occupied by an adversary server and one occupied by a server of RWALK. Call these two points "RWALK's hole" and "the adversary's hole" respectively (note the order). Since the adversary does not move any servers during a phase, its hole does not move during the phase. RWALK's hole executes a resisitive random walk on the $k+1$-node subgraph active during the phase. The phase ends when RWALK's hole catches up with the (static) adversary's hole.

The resistive random walk on a graph with n vertices has a stretch of at most $2(n-1)$ on any edge (see the analysis at end of section 3). The total cost incurred by RWALK during the phase is exactly the cost incurred by its hole during the phase. Suppose that the adversary moved a server a

distance d to begin the phase. Then the distance between the two holes at the beginning of of the phase is d, and the expected cost incurred by the RWALK hole during the phase is at most $(2k-1)d$.

If the cost matrix fulfils the triangle inequality, then the resistive random walk has a stretch of at most $2n-3$ on any edge, so that the lazy adversary conjecture implies that RWALK is $(2k-1)$-competitive on such graphs.

Acknowledgements

We thank Allan Borodin and Gil Strang for their helpful comments and suggestions.

References

[1] R.A. Baeza-Yates, J.C. Culberson, and G.J.E. Rawlins. Searching in the plane. To appear in *Information and Computation*, 1990.

[2] L.E. Baum and J.A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bull. Amer. Math. Soc.*, 73:363–363, 1967.

[3] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

[4] P. Berman, H.J. Karloff, and G. Tardos. A competitive 3-server algorithm. In *Proceedings 1st ACM-SIAM Symposium on Discrete Algorithms*, pages 280–290, 1990.

[5] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. *Journal of the ACM*, 39:745–763, 1992.

[6] R. Bott and R. J. Duffin. On the algebra of networks. *Trans. Amer. Math. Soc.*, 74:99–109, 1953.

[7] A. K. Chandra, P. Raghavan, W.L. Ruzzo, R. Smolensky, and P. Tiwari. The electrical resistance of a graph captures its commute and cover

times. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 574–586, Seattle, May 1989.

[8] M. Chrobak, H.J. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms*, pages 291–300, 1990.

[9] M. Chrobak and L.L. Larmore. An optimal online algorithm for k servers on trees. *SIAM Journal on Computing*, 20:144–148, 1991.

[10] P.G. Doyle and J.L. Snell. *Random Walks and Electric Networks*. The Mathematical Association of America, 1984.

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

[12] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 454–463, 1990.

[13] R. M. Foster. The average impedance of an electrical network. In *Contributions to Applied Mechanics (Reissner Anniversary Volume)*, pages 333–340. Edwards Bros., Ann Arbor, Mich., 1949.

[14] R. M. Foster. An extension of a network theorem. *IRE Trans. Circuit Theory*, 8:75–76, 1961.

[15] E. Grove. The harmonic online k-server algorithm is competitive. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 260–266, 1991.

[16] A. R. Karlin, M. S. Manasse, L. Rudolph, and D.D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):70–119, 1988.

[17] J.G. Kemeny, J. L. Snell, and A.W. Knapp. *Denumerable Markov Chains*. The University Series in Higher Mathematics. Van Nostrand, Princeton, NJ, 1966.

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

[19] A. W. Marshall and I. Olkin. *Inequalities: Theory of Majorization and Its Applications*. Academic Press, New York, 1979.

[20] C.H. Papadimitriou and M. Yanakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.

[21] P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. In *16th International Colloquium on Automata, Languages, and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 687–703. Springer-Verlag, July 1989. Revised version available as IBM Research Report RC15840, June 1990.

[22] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, February 1985.

[23] L. Weinberg. *Network Analysis and Synthesis*. McGraw-Hill, New York, 1962.