A Graph-Theoretic Game and its Application to the k-Server Problem (Extended Abstract)

Noga Alon * Richard M. Karp † David Peleg ‡
Douglas West §

July 11, 1991

Abstract

This paper investigates a zero-sum game played on a weighted connected graph G between two players, the *tree player* and the *edge player*. At each play, the tree player chooses a spanning tree T and the edge player chooses an edge e. The payoff to the edge player is $cost(T, e)$, defined as follows: If e lies in the tree T then $cost(T, e) = 0$; if e does not lie in the tree then $cost(T, e) = cycle(T, e)/w(e)$, where $w(e)$ is the weight of edge e and $cycle(T, e)$ is the weight of the unique cycle formed when edge e is added to the tree T. Our main result is that the value of the game on an n-vertex graph is bounded above by $\exp(O(\sqrt{\log n \log\log n}))$.

The game arises in connection with the k-server problem on a *road network*; i.e., a metric space that can be represented as a multigraph G in which each edge e represents a road of length $w(e)$. We show that, if the value of the game on G is $Val(G, w)$, then there is a randomized strategy that achieves a competitive ratio of $k(1 + Val(G, w))$ against any oblivious adversary. Thus, on any n-vertex road network, there is a randomized algorithm for the k-server problem that is $k \cdot \exp(O(\sqrt{\log n \log\log n}))$-competitive against oblivious adversaries.

At the heart of our analysis of the game is an algorithm that, for any n-vertex weighted, connected multigraph, constructs a spanning tree T such that the average, over all edges e, of $cost(T, e)$ is less than or equal to $\exp(O(\sqrt{\log n \log\log n}))$. This result has potential application to the design of communication networks.

*Tel Aviv University and IBM Almaden Research Center.
†University of California at Berkeley and International Computer Science Institute, Berkeley, California.
‡The Weizmann Institute.
§University of Illinois.

1 Introduction

Let G be a connected multigraph, and let w be a function from the edge set of G into the positive reals; $w(e)$ is called the *weight* of edge e. Consider a two-person zero-sum game between a *tree player* and an *edge player*. At each play the tree player chooses a spanning tree T and, simultaneously, the edge player chooses an edge e. The payoff $cost(T, e)$ is defined as follows: if e lies in the tree T then $cost(T, e) = 0$; else $cost(T, e) = cycle(T, e)/w(e)$, where $cycle(T, e)$ is the weight of the unique cycle formed when edge e is added to the tree T.

Our main interest is in determining the value of this game for particular weighted multigraphs, and in determining an upper bound on the value in terms of the number of vertices.

We introduce some standard terminology. A mixed strategy for the tree player is a probability distribution p over the spanning trees of G, assigning to each spanning tree T a probability $p(T)$; similarly, a mixed strategy for the edge player is a probability distribution q over the edges, assigning to each edge a probability $q(e)$. The min-max theorem of game theory tells us that

$$\min_p \max_q \sum_T \sum_e p(T)q(e)cost(T, e) = \max_q \min_p \sum_T \sum_e p(T)q(e)cost(T, e).$$

The common value of these two expressions is called the *value* of the game, and is denoted $Val(G, w)$. In the special *unweighted case* in which $w(e) = 1$ for all e, the value is denoted $Val(G)$.

Our main results with respect to the game are as follows:

- for every weighted n-vertex multigraph G, w,
  $Val(G, w) \leq \exp(O(\sqrt{\log n \log\log n}))$
- There is an infinite sequence $\{G_n\}$ of graphs such that G_n has n vertices and $Val(G_n) = \Omega(\log n)$.

We also provide a near tight analysis of the game on several simple classes of graph topologies, including cycles, cycles with cross edges, grids and hypercubes.

At the heart of our analysis of the game on arbitrary multigraphs is an algorithm that, for any n-vertex weighted, connected multigraph, constructs a spanning tree T such that the average, over all edges e, of $cost(T, e)$ is less than or equal to $\exp(O(\sqrt{\log n \log\log n}))$. This algorithm provides us with a strategy for the tree player in the game.

The game arises in connection with the k-server problem on a *road network*; i.e., a metric space that can be represented as a multigraph G in which each edge

e represents a road of length $w(e)$. We show that, if the value of the game on G is $Val(G, w)$, then there is a randomized strategy that achieves a competitive ratio of $k(1 + Val(G, w))$ against any oblivious adversary. Thus, on any n-vertex road network, there is a randomized algorithm for the k-server problem that is $k \cdot \exp(O(\sqrt{\log n \log\log n}))$-competitive against oblivious adversaries.

In addition, the spanning tree algorithm has potential application to the design of communication networks, yielding a viable and useful alternative to either shortest-path trees or spanners [ADDJ, PS] in certain cases.

Example 1.1 *The complete graph.* Let G be the unweighted n-vertex simple complete graph. If the edge player chooses uniformly among the edges, then selection of any tree will have probability $\frac{2}{n}$ of payoff 0 and probability $\frac{(n-2)}{n}$ of some nonzero payoff at least 3, so the expected payoff is at least $3 - \frac{6}{n}$. Equality holds only for the n-vertex stars, because other trees have fundamental cycles of lengths exceeding 3. If the tree player chooses uniformly among the n stars, then any edge has probability $\frac{2}{n}$ of payoff 0 and probability $\frac{(n-2)}{n}$ of payoff 3, guaranteeing payoff at most $3-\frac{6}{n}$. Hence uniform edge selection and uniform star selection are optimal strategies, and $Val(G) = 3 - \frac{6}{n}$. $\Box$

Example 1.2 *Cycles and multi-cycles.* If G is the n-cycle, let T_i be the tree omitting edge i, and let w_i be the weight on edge i, with $W = \sum_i w_i$. If the tree player assigns probability $p_i = \frac{w_i}{W}$ to T_i, then every edge has expected payoff 1. If the edge player assigns probability $\frac{w_i}{W}$ to edge i, then every tree has expected payoff 1. Hence $Val(G, w) = 1$. $\Box$

2 An Application: The k-Server Problem on an Undirected Network

Our game on graphs first suggested itself in connection with the *k-server problem*, which was introduced in [MaMcSl] and has been studied by many researchers. The problem is set in a metric space M, and involves the use of k *servers* to process a sequence of *requests*. At any stage in the processing of the sequence of requests, each server is located at a point in M; the initial locations of the servers are stipulated as part of the problem. Each request is specified by a point $r \in M$. A request at r is processed by moving one of the servers from its present location to r; the cost of processing the request is the distance the server moves. A *deterministic on-line algorithm* is a deterministic rule for deciding which server to move in response to a request. The choice must be determined by the initial positions of the servers and the sequence of requests up to the present request.

It is because the choice is not allowed to depend on knowledge of future requests that the algorithm is called on-line.

For any pair π, ρ, where π specifies the initial positions of the k servers and ρ specifies a finite sequence of requests, and for any deterministic on-line algorithm A, let $A(\pi, \rho)$ denote the cost that A incurs in processing the sequence of requests ρ, starting with the servers in the initial positions π. We may also associate with the pair π, ρ a real number $OPT(\pi, \rho)$ denoting the minimum possible cost of processing the request sequence ρ, starting with the servers in the k-tuple π of initial positions; $OPT(\pi, \rho)$ can be viewed as the cost that would be incurred by an *optimal off-line algorithm* that knew the entire request sequence, as well as the initial positions of the servers, in advance, and thus could determine the least expensive way of processing the entire request sequence.

Research on the k-server problem focuses on determining the factor in extra cost that a deterministic on-line algorithm must pay because of the handicap of making its decisions on-line, without knowing the future. This is formalized as follows. Let C be a positive constant. The deterministic on-line algorithm A is called *C-competitive* if there exists a positive constant a such that, for all pairs π, ρ, $A(\pi, \rho) \leq C \cdot OPT(\pi, \rho) + a$. In [MaMcSl] it is shown that, for every positive integer k, every metric space M with at least $k+1$ distinct points, and every C less than k, there does not exist a C-competitive deterministic on-line algorithm. Thus, except in trivial cases, it is impossible to achieve a competitive factor less than k. On the other hand, in [FiRaRa] it is shown that, for every positive integer k and every metric space M, there exists a C-competitive deterministic on-line algorithm, for some C, where C depends on k but not on the metric space. Thus, a bounded competitive factor is always achievable. Chrobak and Larmore [ChLa] consider a special type of metric space, that might be called a *treelike road network*. Such a network consists of a tree T in which each edge $\{u, v\}$ is an interval of length $w(u, v)$ between vertex u and vertex v. The metric space consists of the union of all these intervals. The distance between two points x and y is just the length of the unique simple path between x and y in the network. Chrobak and Larmore show constructively that, for any treelike road network, and any number of servers k, there is a k-competitive deterministic on-line algorithm, matching the lower bound of k established in [MaMcSl].

It is also possible to consider randomized on-line algorithms, in which the server to move at each step is chosen by a random experiment which, of course, takes into account the initial positions of the servers, the sequence of requests up to the present one, and the choices made by the algorithm at previous steps. In the case of a randomized algorithm, the cost $A(\pi, \rho)$ is a random variable. In defining the competitive factor achieved by a randomized on-line algorithm, we view the request sequence as being specified by an *adversary* who is trying to foil the algorithm. We distinguish between two types of adversaries: the *oblivious adversary*, who specifies the entire sequence of requests in advance, and the *adaptive adversary*, who, in specifying the next request, can take into account the algorithm's responses to all previous requests. In this paper we restrict attention to oblivious adversaries. The randomized algorithm A is said to be C-*competitive* (against oblivious adversaries) if there exists a positive constant a such that, for every k-tuple π of initial positions and every request sequence ρ, $E[A(\pi, \rho)] \leq C \cdot COST(\pi, \rho) + a$, where E denotes mathematical expectation. Several examples are known in which randomized algorithms can achieve a smaller competitive factor than deterministic ones [BBKTW, BLS, FKLMSY].

We consider the k-server problem on a class of metric spaces that generalize the treelike road networks of [ChLa]. Such a space is specified by a multigraph G in which each edge e has a positive real weight $w(e)$. The edge $e = \{u, v\}$ may be viewed as an interval of length $w(e)$ between vertices u and v. The metric space $M(G, w)$ consists of the union of all these intervals; the distance between two points x and y is just the length of the shortest path between x and y; we shall sometimes refer to such a metric space as a *road network*.

The following theorem establishes a connection between our two-person game and the k-server problem on a road network.

Theorem 2.1 *Let G be a multigraph and w a function assigning to each edge of G a positive real weight $w(e)$. Then, for every k, there is a polynomial $k(1 + Val(G, w))$-competitive randomized on-line algorithm for the k-server problem on the road network $M(G, w)$.*

Proof: The algorithm is as follows.

- Using an optimal mixed strategy for the tree player in the game defined by G and w, select a spanning tree T in G.
- Along each edge e not in T, choose a random point $x(e)$, and place a "roadblock" at $x(e)$; more precisely, replace $e = \{u, v\}$ by two intervals $[u, x_1(e)$ and $[x_2(e), v]$, where $x_1(e)$ and $x_2(v)$ are new points, distinct from all points in the original metric space. The edge $[u, x_1(e)]$ is of the same length as the interval $[u, x(e)]$, and the interval $[x_2(e), v]$ is of the same length as the interval $[x(e), v]$. This transformation converts the original road network to a treelike road network with the same set of points but a different distance function.
- Process the request sequence by executing the (deterministic) Chrobak-Larmore algorithm on this derived treelike road network.

Let the random variable $A(\pi, \rho)$ denote the cost that the above randomized algorithm incurs in processing the request sequence π, starting from the initial server positions ρ. Let $OPT(\pi, \rho)$ denote the optimal off-line cost of processing ρ, starting from server positions π, on the road network defined by multigraph G and weight function w. Let the random variable $OPT'(\pi, \rho)$ denote the optimal off-line cost of processing ρ, starting from the server positions π, in the treelike road network constructed by the algorithm. By the k-competitiveness of the Chrobak-Larmore algorithm, $A(\pi, \rho) \leq k \cdot OPT'(\pi, \rho)$. To complete the proof, we shall show that $E[OPT'(\pi, \rho)] \leq (1 + Val(G, w))OPT(\pi, \rho)$.

Suppose that the tree player has selected a spanning tree T of G and then for each non-tree edge e, has placed a roadblock at a random point $x(e)$. Consider a particular sequence of moves on the original road network that processes the sequence of requests ρ starting from the initial k-tuple of server positions π at cost $OPT(\pi, \rho)$. Let $\rho = \rho_1\rho_2 \ldots \rho_t$. For $i = 1, 2, \ldots, t$, let $x(i)$ be the index of the server that processes request i. Then it is possible to satisfy ρ on the derived treelike road network by having server $x(i)$ process request ρ_i, for each i. The server may not be able to follow the same path it would have followed on the original network, because of the presence of roadblocks. However, the server can simulate the path it would have followed on the original road network, except that, whenever it encounters a roadblock $x(e)$, it traverses the unique path in the treelike road network between $x_1(e)$ and $x_2(e)$ in order to get to the other side of the roadblock. The cost of each such detour is $cycle(T, e)$, the weight of the unique cycle formed by the edge e with the tree T.

We now compute the expected cost of all the detours, given that the tree player uses an optimal mixed strategy. Let $p(T)$ be the probability that the tree player chooses spanning tree T. Let $d(e)$ be the distance that servers travel along edge e in a particular sequence of moves achieving cost $OPT(\pi, \rho)$ in the original network; then $\sum_e d(e) = OPT(\pi, \rho)$. If $x(e)$ is a randomly chosen point along edge e, then the expected number of times that servers cross the point $x(e)$ is $\frac{d(e)}{w(e)}$. Each crossing of the point $x(e)$ requires a detour of cost 0 if e lies in T and $cycle(T, e)$ if e does not lie in T. Thus, the expected cost of detours associated with the edge e is exactly $d(e)cost(T, e)$, and the expected total cost of detours is $\sum_T \sum_e p(T)d(e)cost(T, e) = OPT(\pi, \rho) \sum_T \sum_e p(T)q(e)cost(T, e)$, where $q(e) = \frac{d(e)}{OPT(\pi,\rho)}$. But since the sequence of numbers $\{q(e)\}$ constitutes a probability distribution, it follows that the expected sum of detours is bounded above by $OPT(\pi, \rho)Val(G, w)$, and hence it follows that $E[OPT'(\pi, \rho)] \leq OPT(\pi, \rho)(1 + Val(G, w))$

Finally, we observe that the algorithm as described above is polynomial, i.e., the optimal strategy for the tree player can, in fact, be efficiently approximated. □

Corollary 2.2 *There is a $2k$-competitive randomized algorithm on the circle.*

3 An Optimization Problem Related to the Tree Game

In this section we introduce a natural optimization problem and show that it is closely related to our graph-theoretic game. Let G be a multigraph with edge multiset E and let w be a function that assigns to each edge e in G a positive weight $w(e)$. Assume that the edge weights are normalized so that the lightest edge has weight 1. For any spanning tree T of G, and any edge e of G, let $path(T, e)$ denote the weight of the path in T between the end points of e. Define $cost^*(T, e) = path(T, e)/w(e)$. Also, for every subset of edges E', denote

$$cost^*(T, E') = \sum_{e \in E'} cost^*(T, e).$$

Let $S(G, w, T) = cost^*(T, E)/|E|$. Let $S_{opt}(G, w) = \min_T S(G, w, T)$, where T ranges over all spanning trees of G. In the unweighted case, where $w(e) = 1$ for all e, we abbreviate $S_{opt}(G, w)$ by $S_{opt}(G)$. We shall show that the problem of finding a spanning tree that minimizes $S(G, w, T)$ is closely related to the problem of finding optimal strategies for the tree player and edge player in a variant of our tree game. In this new game, the payoff to the edge player when the edge player chooses edge e and the tree player chooses spanning tree T is $cost^*(T, e) = path(T, e)/w(e)$. Let the value of this game be denoted $Val^*(G, w)$. In the unweighted case, we abbreviate $Val^*(G, w)$ by $Val^*(G)$. Then, since $|cost(T, e) - cost^*(T, e)| \leq 1$ for every edge e, $|Val(G, w) - Val^*(G, w)| \leq 1$. Thus, our new game is very closely related to the original one.

Let G, w be a weighted multigraph. The operation of *replication* of an edge e replaces the edge e by one or more parallel edges of weight $w(e)$ between the end points of e. Any weighted multigraph created from G, w by a sequence of such operations is said to be *obtained from G, w by replication.*

Theorem 3.1 *$Val^*(G, w) = \max_{G', w'} S_{opt}(G', w')$, where G', w' ranges over all weighted multigraphs obtained from G, w by replication.*

An *edge-transitive* graph is a graph G such that for any edges e, e' in G there is an automorphism σ in the automorphism group $\Gamma(G)$ such that σ takes the endpoints of e to the endpoints of e'. In the case of an unweighted edge-transitive graph, we prove that the optimization problem solves the game.

Theorem 3.2 *If G is edge-transitive, then $Val^*(G) = S_{opt}(G)$.*

4 Upper and Lower Bounds for the Tree Game on n-Vertex Multigraphs

We begin with a lower bound on $Val^*(G)$.

Theorem 4.1 *There exists a positive constant c such that, for n, sufficiently large, there exists a n-vertex graph G_n such that $Val^*(G_n) \geq c \ln n$.*

We next give a preliminary result showing that for bounding $Val^*(G)$ from above, it is sufficient to consider multigraphs with at most $n(n+1)$ edges (counting multiplicities).

Lemma 4.2 *For every n-vertex multigraph G, w, there exists a multigraph G', w' on the same set of vertices and at most $n(n+1)$ edges, such that $S_{opt}(G, w) \leq 2 \cdot S_{opt}(G', w')$.*

We now state and prove our upper bound for $S_{opt}(G, w)$.

Theorem 4.3 *There exists a constant c such that, for every n-vertex multigraph G, w, $S_{opt}(G, w) \leq \exp(c\sqrt{\log n \log\log n})$.*

By Lemma 4.2, it suffices to prove the theorem for every n-vertex multigraph having at most $n(n+1)$ edges in its edge multiset.

We shall use an iterative construction to obtain a suitable spanning tree of G. The construction involves a parameter x depending on n, to be fixed later. Define $\rho = \frac{2\ln n}{\ln x}$, $\mu = 6\rho \ln n$ and $y = x\mu$. Break the set of edges E into classes E_i according to weights, defining

$$E_i \;=\; \{e \mid w(e) \in (y^{i-1}, y^i]\}.$$

The algorithm proceeds in iterations. In each iteration we compute some clusters in the graph, and mark some of the edges as "covered." (An edge is *covered* if both its endpoints are contained in some cluster we have already constructed.) We then contract the clusters into single nodes, thus preparing the multigraph for the next iteration.

More precisely, let E_i^j denote the set of edges from E_i that are still uncovered at the beginning of iteration j. The idea of the construction at iteration j is to partition the vertex set into disjoint parts (or clusters) such that:

- each part has a spanning tree of radius at most y^{j+1};
- in every nonempty edge class E_i, $1 \leq i \leq j$, the fraction of edges that join vertices in two different parts is at most $1/x$; i.e., $|E_i^{j+1}| \leq |E_i^j|/x$.

We next present the algorithm used for forming the partition in iteration j. This algorithm is a slightly modified version of the clustering algorithm of [Aw]. The partition will be constructed in a "greedy" fashion, by choosing one part at a time. At a general step, let K be a connected component of the subgraph induced by the vertices not yet selected. Choose arbitrarily a "root vertex" u. Stratify the vertices and edges of K according to their distance from u as follows. For each integer $\ell \geq 0$, let $V(\ell)$ be the set of vertices at distance ℓ from u in K, and let $E_i^j(\ell)$ be the set of edges of E_i^j that join a vertex in $V(\ell)$ with a vertex in $V(\ell) \cup V(\ell-1)$. Let ℓ^* be the least ℓ such that

$$\forall 1 \leq i \leq j, \quad |E_i^j(\ell+1)| \leq \frac{1}{x}|E_i^j(1) \cup E_i^j(2) \cup \ldots \cup E_i^j(\ell)|. \tag{$*$}$$

If no such ℓ exists then the next part is K; otherwise the next part is the subgraph of K induced by the vertex set $V(1) \cup V(2) \cup \ldots \cup V(\ell^*)$.

Let $x(n)$ be a function of n that will be specified later. The spanning tree T in an n-vertex multigraph G is constructed as follows:

- Set $j = 1$ and $G_j = G$.
- Set $x = x(n)$, ρ, μ, y, and the edge classes E_i, as defined above.
- While $\bigcup_i E_i \neq \emptyset$ do:
  1. Partition the vertex set of G_j into parts as described above.
  2. Construct a shortest-path spanning tree in each part of G_j.
  3. For each edge e of each of the constructed trees, add the corresponding edge of the original graph G_1 to the output tree T.
  4. Construct the next multigraph G_{j+1} by contracting each part into a single vertex, discarding covered edges (connecting endpoints in the same part) from $\bigcup_i E_i$, and replacing each uncovered edge (connecting endpoints in different parts in the partition) by a new edge connecting the corresponding contracted vertices.
  5. Set $j = j + 1$.

In order to analyze the output of our algorithm, we first bound the number of layers added to a cluster during the clustering process in some iteration j. This growth is bounded by showing that in iteration j, the only sets E_i^j considered by the algorithm (i.e., the only nonempty ones) are those satisfying $j - \rho \leq i \leq j$.

Lemma 4.4 $|E_i^j| \leq |E_i|/x^{j-i}$ *for every* $1 \leq i \leq j$*.*

The above lemma enables us to bound the radius of clusters generated during the execution.

Lemma 4.5 *In iteration* j*, the radius of each cluster is bounded above by* y^{j+1}*.*

Using the above result we can bound the costs incurred by the edges, obtaining the following.

Lemma 4.6 $cost^*(T, E) \leq 2x^2 \mu^{\rho+1} |E|$*.*

It follows that the average spanning factor of the constructed tree T is bounded above by

$$S(G, w, T) \;\leq\; 2x^2 \mu^{\rho+1} \;=\; 2x^2 \left(\frac{12 \ln^2 n}{\ln x} \right)^{1+\frac{2 \ln n}{\ln x}}.$$

Selecting $x = \exp(c\sqrt{\log n \log \log n})$ for an appropriate constant c optimizes this bound as $S(G, w, T) \;=\; \exp(O(\sqrt{\log n \log \log n}))$. This completes the proof of Theorem 4.3. □

Corollary 4.7 *There exists a constant* c *such that, for every* n*-vertex multigraph* G*,* w*,* $Val^*(G, w) \leq \exp(c\sqrt{\log n \log \log n})$*.*

5 Grids and hypercubes

In this section, we discuss the value of the game on grids and hypercubes. More generally, we consider the d-dimensional grid $G_{n,d}$ that is the product of d n-vertex paths and has $N = n^d$ vertices.

A hypercube is edge-transitive, so $Val^*(G_{2,d}) = S_{opt}(G_{2,d})$. For $G_{n,2}$, the value of the game can be approximated by the optimal tree. To see this, note that the n by n discrete torus is edge-transitive. Any tree in the grid G is also a tree in the torus G'. If we choose a tree T in G with diameter less than 1, then $S(G', T) \leq S(G, T)(1 - \frac{1}{n}) + \alpha$. On the other hand, $Val^*(G') \geq Val^*(G)$, because the edge player on the torus has the option of playing only edges in the grid. The upper bound we obtain is independent of d.

Theorem 5.1 *For the* d*-dimensional grid* $G_{n,d}$ *with* $N = n^d$ *vertices,* $S_{opt}(G_{n,d}) < 2 \log N$*.*

For the interesting special case of the hypercube $Q_d = G_{2,d}$, which is edge-transitive, we have a slightly better construction, which we conjecture to be optimal.

Theorem 5.2 *$Val(Q_d) = S_{opt}(Q_d) \leq d - 1$.*

We believe that the general construction described above is essentially optimal for large n, with $S_{opt}(G_{n,d}) = 2\lg\ N + o(\lg\ N)$. We prove (in the appendix) only that $S_{opt}(G_{n,2}) \geq c\lg\ n + o(\lg\ n)$.

Theorem 5.3 *$S_{opt}(G_{n,2}) \geq 1024^{-1}\ln n + o(\log n)$.*

More generally, we prove (in the appendix) that Theorem 5.1 is optimal, up to a constant factor, for all $n, d \geq 2$.

Theorem 5.4 *There exists an absolute constant $c > 0$ such that for every $n, d \geq 2$, $S_{opt}(G_{n,d}) \geq cd \log n = c \log N$.*

References

[ADDJ] I. Althöfer and G. Das and D. Dobkin and D. Joseph, "Generating sparse spanners for weighted graphs," *Proc. 2nd Scandianvian Workshop on Algorithm Theory*, July 1990.

[Aw] B. Awerbuch, "Complexity of Network Synchronization," J. ACM 32 (1985), 804-823.

[AwPe] B. Awerbuch and D. Peleg, "Sparse Partitions," *Proc. 31st Annual IEEE Symp. on Foundations of Computer Science*, pp. 503–513, 1990.

[BBKTW] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos and A. Wigderson, "On the Power of Randomization in Online Algorithms", *Proc. 22nd Annual ACM Symp. on Theory of Computing*, pp. 379–386, 1990.

[BLS] A. Borodin, N. Linial and M. Saks, "An Optimal On-line Algorithm for Metrical Task Systems", J. of the ACM (1990).

[CFGS] F.R.K. Chung, P. Frankl, R.L. Graham and J.B. Shearer, "Some Intersection Theorems for Ordered Sets and Graphs", J. Combinatorial Theory, Ser. A 43 (1986), 23-37.

[ChLa] M. Chrobak and L. Larmore, "An Optimal On-Line Algorithm for k Servers on Trees," manuscript, July, 1989, to appear in SIAM J. on Computing.

[FiRaRa] A. Fiat, Y. Rabani and Y. Ravid, "Competitive k-server Algorithms," *Proc. 31st Annual IEEE Symp. on Foundations of Computer Science*, pp. 454–463, 1990.

[FKLMSY] A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.D. Sleator and N.E. Young, "Competitive Paging Algorithms," Technical report, Carnegie Mellon University, 1988, J. of Algorithms, to appear.

[MaMcSl] M.S. Manasse, L.A. McGeoch and D.D. Sleator, "Competitive Algorithms for On-Line Problems," *Proc. 20th Annual ACM Symp. on Theory of Computing*, pp. 322–333, 1988.

[PS] D. Peleg and Alejandro A. Schäffer, "Graph Spanners," J. of Graph Theory 13 (1989), 99-116.

A Proof of Theorem 5.3

In order to prove Theorem 5.3 we need several preliminary results.

Lemma A.1 *If A is a set of α^2 vertices in $G_{n,2}$, where $\alpha^2 \leq \frac{n^2}{2}$, then there are at least α rows that A intersects but does not fill or at least α columns that A intersects but does not fill.*

Lemma A.2 *If A is a set of α^2 vertices in $G_{n,2}$, where $\alpha^2 \leq \frac{n^2}{2}$, and B is a set of at most four vertices in A, then there are at least $\frac{\alpha}{2}$ vertices in A that have neighbors outside A and have distance at least $\frac{\alpha}{16}$ from each vertex of B.*

Lemma A.3 *If T is a spanning tree of $G_{n,2}$ and $\alpha \leq \frac{n}{4}$, then there are at least $\frac{n^2}{32\alpha}$ edges e such that $cycle(e) > \frac{\alpha}{16}$.*

Proof: Since the grid has maximum degree 4, every subtree (not necessarily spanning) has an edge whose deletion separates it into two smaller subtrees, each having at least $\frac{1}{4}$ of its vertices. Begin with T and successively delete edges from the biggest remaining component, always splitting that component as evenly as possible. Do this until $\lceil \frac{n^2}{4\alpha^2} \rceil - 1$ edges have been deleted and the number of pieces is $\lceil \frac{n^2}{4\alpha^2} \rceil$. On the average, the pieces have about $4\alpha^2$ vertices, with the smallest piece having at least α^2 vertices. To see this, note that at every stage, the largest piece has at most 4 times as many vertices as the smallest piece, because whenever there is a new smallest size it is at least $\frac{1}{4}$ of the old largest size. Minimizing x_1 subject to $x_1 \leq \ldots \leq x_m \leq 4x_1$ and $\sum x_i = M$ yields $x_1 = \frac{M}{4m-3}$, achieved when all the other variables are 4 times the smallest. Note also that the largest piece must have fewer than half the vertices as soon as 4 pieces are requested.

Since each deleted edge of T is incident to two pieces, the average number of deleted edges incident with a piece is less than 2. Hence at least half the pieces are incident with at most four deleted edges. If A is the set of vertices of such a piece, let B be the vertices in A incident to these deleted edges. By the preceding lemma, A has at least $\frac{\alpha}{2}$ vertices having neighbors outside A whose distance from each vertex of B is at least $\frac{\alpha}{16}$. The edges leaving A from these vertices do not belong to T, since the only edges of T leaving A are incident to vertices of B. For any such edge e, we have $cycle(e) > \frac{\alpha}{16}$, since the path in T between the ends of e must contain a path in the piece of T induced by A from e to B. Harvesting at least $\frac{\alpha}{2}$ endpoints of such edges from each of at least $\frac{n^2}{8\alpha^2}$ pieces of T yields at least $\frac{n^2}{32\alpha}$ such edges, since we might have counted each edge twice. (Of course, $cycle(e) > \frac{\alpha}{8}$ for any edge we have counted twice, but let us be pessimistic.) $\Box$

We can now complete the proof of the theorem. Given an arbitrary spanning tree T of $G_{n,2}$, the preceding lemma guarantees at least $\frac{n^2}{32\alpha}$ edges with cost at least $\frac{\alpha}{16}$ for any $\alpha \le \frac{n}{4}$. We now choose an edge e at random, defining a random variable X by $X = cost(T, e)$. We seek a lower bound on $E[X]$, the average cost of an edge, since $S_{opt}(G_{n,2} = E[X]$. We use the fact that, for a discrete random variable X, we have $E[X] = \sum_{k\ge1} Prob(X \ge k)$. For $k \le \frac{n}{64}$, let $\alpha = 16k$. Then $Prob(X \ge k) \ge \frac{n^2}{512k\cdot 2n(n-1)} > \frac{1}{1024k}$. We thus obtain $E[X] > 1024^{-1}\sum_{k=1}^{n/64} k^{-1} = 1024^{-1}\ln n + o(\log n)$. □

B Proof of Theorem 5.4

The proof of Theorem 5.4 requires several preparations. We prove it for, say, $c = 10^{-10}$. (We note that we make no attempt to optimize the constants here and in the rest of the proof.) Clearly, with this value of c, if both n and d do not exceed 10^5 there is nothing to prove and hence we may assume that

$$Max\{n, d\} \ge 10^5. \tag{1}$$

Suppose $n, d \ge 2$ and let $G = G_{n,d} = (V, E)$. Denote $M = \{1, 2, \ldots, n\}$. Note that the vertices of G are naturally represented by all vectors of length n whose coordinates are in M. If uv is an edge of G, where $u, v \in V$, we say that the *direction* of the edge uv is i if i is the unique coordinate in which u and v differ. For a subset $A \subset V$ and for a vector $v = (v_1, \ldots, v_n)$ with $v_i = *$ and $v_j \in M$ for all $j \ne i$ we say that A *intersects* the vector v if there is a $u \in A$ such that u coincides with v on all the coordinates but the i−th (i.e., the projection of u on $M \setminus \{i\}$ is equal to that of v). We say that A *properly intersects* v if A intersects v and also its complement $V \setminus A$ intersects v. Note that in this case there is at least one edge of G in direction i which joins a vertex u of A with a vertex w of $V \setminus A$, where both u and w coincide with v on all coordinates but the i-th.

We need the following lemma, proved in [CFGS]. (The proof in [CFGS] is given only for the case $n = 2$ but the same proof works for all integers n).

Lemma B.1 *[CFGS] Let $D = \{1, \ldots d\}$ be a finite set and let $B_1, \ldots, B_m$ be subsets of D such that each element of D belongs to at least k of the sets B_i. Let $\mathcal{F}$ be a family of vectors of length d whose coordinates, indexed by the elements of D, lie in $M = \{1, \ldots, n\}$. For each i, $1 \le i \le m$, let $\mathcal{F}_i$ denote the set of all projections of the vectors in $\mathcal{F}$ on B_i. Then*

$$|\mathcal{F}|^k \le \prod_{i=1}^{m} |\mathcal{F}_i|.$$

□

Returning to the graph $G = (V, E)$ we now prove:

Lemma B.2 *Suppose $A \subset V$, $|A| = x^d$, where $x \geq 1$ is not necessarily an integer. Let A_i denote the set of all vectors v such that A intersects v, and let A'_i denote the set of all vectors v such that A properly intersects v. Then*

$$\sum_{i=1}^{d} |A_i| \geq dx^{d-1}$$

and

$$\sum_{i=1}^{d} |A'_i| \geq dx^{d-1}(1 - x/n).$$

Let v be a vertex of G, let y be a positive real, and suppose $i \in M$. Let S denote the set of all vertices of G whose distance from v is less than $y(d-1)$ and let S_i denote the set of projections of the members of S on $M \setminus \{i\}$. Denote by $L(y, d)$ the maximum possible cardinality of S_i where the maximum is taken over all possible choices of v and i.

Lemma B.3 *For all $y > 0$:*

$$L(y, d) \leq 2^{Min\{d-1, y(d-1)\}} \binom{d - 1 + (d-1)y}{d-1}. \tag{2}$$

Therefore, for all $y \geq 1$,

$$L(y, d) \leq (2e(y+1))^{d-1}$$

and for all $y \leq 1$

$$L(y, d) \leq (2e^2)^{(d-1)y},$$

where $e = 2.71828...$ is the basis of the natural logarithm.

We can now return to the proof of Theorem 5.4. Let T be a spanning tree of $G = G_{n,d}$. Let x be a real number, $1.5 \leq x \leq \frac{3}{4}n$. By deleting edges of T we can break it into connected components each containing at most x^d and at least $\frac{x^d}{2d}$ vertices. (This is possible since the maximum degree of a vertex of G (and hence of a vertex of T) is at most $2d$, and thus if we repeatedly split T into connected components by always splitting the biggest remaining component as evenly as possible we will never have two components the ratio between the sizes of which exceeds $2d$). It follows that the number t of connected components, and hence the number of deleted edges of T, does not exceed $\frac{2dn^d}{x^d}$. Let U denote the set of all end-vertices of these deleted edges. Then $|U| \leq \frac{4dn^d}{x^d}$.

Let x_j^d denote the number of vertices in the j-th connected component, $(1 \le j \le t)$. Since $x_j \le x \le 0.75n$ for each j, Lemma B.2 implies that the total number of edges of G emanating from the j-th connected component to vertices in other components is at least $\frac{1}{4}dx_j^{d-1}$. Observe that if y is a real positive number and if both ends of such an edge are at distance at least $y(d-1)$ from all the vertices in U that lie in the same connected component as that end then the length of the elementary cycle corresponding to this edge is at least $2y(d-1)$. We next obtain a lower bound for the number of such edges. Let us fix a direction i, and consider only edges in direction i. If we let A denote the set of vertices of the j-th component C_j and use the notation of Lemma B.2, we conclude that there are at least $|A'_i|$ vertices in C_j whose projections on $M \setminus \{i\}$ are pairwise different, such that from each such vertex there is an edge in direction i emanating to another connected component. By the definition of $L(y,d)$, each vertex of U can be at distance less than $y(d-1)$ from no more than $L(y,d)$ such vertices. Summing over all connected components and over all directions we conclude that the number of edges that join distinct components whose elementary cycles have length at least $2y(d-1)$ is at least

$$\frac{1}{2}\sum_{j=1}^{t}\frac{1}{4}dx_j^{d-1} - |U|dL(y,d).$$

Since $|U| \le 4dn^d/x^d$ and since the minimum possible value of $\sum_{j=1}^{t} x_j^{d-1}$ subject to the constraint that $x_j \ge 0$ and $\sum_{j=1}^{t} x_j^d = n^d$ is $\frac{n^d}{x^d}x^{d-1} = n^d/x$ we obtain the following lemma.

Lemma B.4 *For each spanning tree T in $G = G_{n,d}$ and for each real x, $1.5 \le x \le 0.75n$ and positive real y, the number of edges of G whose elementary cycles have length at least $2y(d-1)$ is at least*

$$\frac{1}{8}d\frac{n^d}{x} - L(y,d)4d^2\frac{n^d}{x^d}.$$

□

We can now prove Theorem 5.4. For technical reasons it is convenient to consider three possible cases, depending on the values of the parameters n and d.

Case I: $n \le 80$.

In this case $d \ge 10^5$, by (1). Put $x = 1.5$, $y = 1/20$. By Lemma B.3 $L(y,d) \le (2e^2)^{0.05(d-1)} < 1.2^{d-1}$. Hence (since $x = 1.5$),

$$L(y,d)4d^2\frac{n^d}{x^d} < (\frac{12}{15})^{d-1}4d\frac{dn^d}{x} << \frac{1}{16}\frac{dn^d}{x}.$$

Therefore, by Lemma B.4, there are at least $\frac{1}{16}d\frac{n^d}{x} = \frac{1}{24}dn^d$ edges whose cycles have length at least $\frac{2}{20}(d-1)$, and since the total number of edges is $dn^{d-1}(n-1) <$

dn^d and the tree T was arbitrary we conclude that in this case $S_{opt}(G_{n,d}) \geq \frac{1}{240}(d-1) > cd\log n$, as needed.

Case II: $d \leq 20$.

In this case, which requires a little more work, $n \geq 10^5$, by (1). For each real x satisfying

$$1600 \leq x \leq \frac{3}{4}n \tag{3}$$

define $y = x/1600$. By Lemma B.3 $L(y,d) \leq (2e(y+1))^{d-1} < \frac{x^{d-1}}{160^{d-1}}$ and hence

$$L(y,d)4d^2\frac{n^d}{x^d} \leq \frac{4d}{160^{d-1}}d\frac{n^d}{x} \leq \frac{1}{20}d\frac{n^d}{x},$$

where here we applied the fact that $d \geq 2$.

It follows that at least a fraction $1/(20x)$ of the edges have cycles of length at least $2x(d-1)/1600$, for each x satisfying (3). Therefore, if X is the random variable defined on the edges f of our graph G by letting $X(f)$ be the length of the elementary cycle of f divided by $d-1$ it follows that for each x satisfying (3), the probability $Prob(X \geq x/800)$ is at least $1/(20x)$. Therefore, by defining $z = x/800$ we conclude that for each integer z satisfying $2 \leq z \leq \frac{3}{3200}n$, $Prob(X \geq z) \geq 1/(16000z)$. Thus the expectation of X is at least

$$\sum_{z=2}^{3n/3200} Prob(X \geq z) \geq \sum_{z=2}^{\lfloor 3n/3200 \rfloor} \frac{1}{16000z} \geq 10^{-5}\log n.$$

The average cost of an edge with respect to the tree T is $d-1$ times the expectation of X and hence we conclude that in this case $S_{opt}(G_{n,d}) \geq 10^{-5}(d-1)\log n >> cd\log n$, as required.

Case III: $n > 80$ and $d > 20$.

For each real x, $30 \leq x \leq 0.75n$ define y by $y+1 = \frac{x}{4e}$. By Lemma B.3 $L(y,d) \leq \frac{x^{d-1}}{2^{d-1}}$ and hence

$$L(y,d)4d^2\frac{n^d}{x^d} \leq \frac{4d}{2^{d-1}}d\frac{n^d}{x} << \frac{1}{16}d\frac{n^d}{x}.$$

Therefore, by Lemma B.4, for every x, $30 \leq x \leq 0.75n$, with probability at least $1/(16x)$ the random variable X defined in the previous case is at least $2(x-4e)/(4e)$. As before (by letting $z = (x-4e)/2e$ take integer values) it follows that the expectation of this random variable is at least

$$\sum_{z>30/(2e)-2}^{3n/(8e)-2} \frac{1}{32e(z+2)} > 10^{-6}\log n,$$

and this implies that in this case too $S_{opt}(G_{n,d}) \geq cd\log n$ and completes the proof of Theorem 5.4. $\square$