A GRAPH-THEORETIC GAME AND ITS APPLICATION TO THE k-SERVER PROBLEM*

NOGA ALON†, RICHARD M. KARP‡, DAVID PELEG§, AND DOUGLAS WEST¶

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. The main result is that the value of the game on any n-vertex graph is bounded above by \( \exp(O(\sqrt{\log n \log \log n})) \). It is conjectured that the value of the game is \( O(\log n) \).

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

At the heart of the analysis of the game is an algorithm that provides an approximate solution for the simple network design problem. Specifically, for any n-vertex weighted, connected multigraph, the algorithm 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. It also improves substantially known estimates concerning the existence of a sparse basis for the cycle space of a graph.

Key words. k servers, spanners, spanning trees, average stretch

AMS subject classifications. 05C05, 05C12, 05C85, 68R10, 90B80, 90D43

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 \( \text{cost}(T, e) \) is defined as follows. For an edge \( e \) that does not lie in the tree \( T \), let \( \text{cycle}(T, e) \) denote the weight of the unique cycle formed when edge \( e \) is added to the tree \( T \). Then

\[
\text{cost}(T, e) = \begin{cases} 
0 & \text{if } e \text{ lies in the tree } T, \\
\text{cycle}(T, e)/w(e) & \text{otherwise}.
\end{cases}
\]

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

\[
\min_p \max_q \sum_T \sum_e p(T) q(e) \text{cost}(T, e) = \max_q \min_p \sum_T \sum_e p(T) q(e) \text{cost}(T, e).
\]

*Received by the editors December 3, 1991; accepted for publication (in revised form) October 4, 1993.
†Tel Aviv University, Tel Aviv 69978, Israel. The research of this author was supported by a US-Israeli BSF Grant.
‡University of California at Berkeley and International Computer Science Institute, Berkeley, California 94720.
§The Weizmann Institute, Rehovot, Israel. The work of this author was supported by an Alon Fellowship, a Bantrell Fellowship, and a Haas Career Development Award.
¶University of Illinois. The research of this author was supported by Office of Naval Research Grant N00014-85K0570 and by NSA/MSP Grant MDA904-90-H-4011.
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 $\{G_n\}_{n \geq 1}$ 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 provides an approximate solution for the simple network design problem of [JLR]. Specifically, for any $n$-vertex weighted, connected multigraph, our algorithm 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. As a byproduct, our algorithm improves the main result of [SV] that deals with the choice of a sparse basis for the cycle space of a given graph. The authors of [SV] show that for every $n$-vertex graph there is a spanning tree so that the average length of a fundamental cycle is $O(\sqrt{n})$, whereas our result improves this estimate to $\exp(O(\sqrt{\log n \log \log n}))$.

A somewhat more general network design problem is the optimum communication spanning tree problem studied by Hu [Hu]. This problem extends the simple network design problem by introducing a communication requirement matrix on the vertices, and charging for each pair of vertices a cost proportional to their distance in the chosen spanning tree times their communication requirement. It can be shown that this seemingly more general problem is in fact reducible to the simple network design problem on multigraphs, so that any (approximate) solution for the latter problem can be translated into a solution for the former (with the same approximation ratio).

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 $w(e)$. The class of these metric spaces seems to be one of the most natural ones for considering the $k$-server problem, as any real configuration of roads forms such a network. Note that any nontrivial road network has infinitely many points; a continuous circle, for example, can be represented by a road network with two vertices and two parallel edges joining them.

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. Shortest-path trees are among the basic tools used in communication networks for various control tasks. Given a graph $H$, let $path(H, u, w)$ denote the weighted distance (i.e., the length of the shortest path) between $u$ and $w$ in $H$. A shortest-path tree for the graph $H$ w.r.t. some vertex $r$ is a tree $T$ with the property that for every vertex $v$ in $H$, its distance from $r$ in $T$ is the same as in $H$, i.e., $path(T, v, r) = path(H, v, r)$.

The obvious drawback of a shortest-path tree w.r.t. $r$ is that while it yields optimal routes from $r$ to any other node, the quality of the routes provided by the tree between other pairs of nodes may be poor. A natural measure for the quality of the path connecting nodes $u, w$ in the tree $T$ is its stretch factor (or dilation), defined as $path(T, u, w)/path(G, u, w)$. Classes of graphs for which there exist spanning trees with low worst-case stretch factor, termed tree
spanners, are studied in [Cai]. However, in general, looking for a tree with low worst-case stretch factor may be a hopeless task, since such a tree may not always exist. In particular, it is clear that there are graphs of diameter $D$ for which any spanning tree incurs a worst-case stretch of $\Omega(D)$ for some pairs of nodes (the unit-weight cycle of two-D vertices is an example for such a graph). It may therefore be useful to look for trees that attempt to minimize the average stretch factor over all graph edges, or even over all pairs of vertices in the graph. This problem is referred to in [JLR] as the simple network design problem, and is shown therein to be NP hard. Our result provides a method for constructing such a tree, which in some cases may provide an attractive alternative to the standard shortest-path tree.

Let us comment that another viable alternative involves insisting on good bounds for the worst-case stretch, at the cost of allowing cycles in the spanning structure. It is known that for every graph there exist relatively sparse spanning subgraphs, termed spanners, guaranteeing this property [ADDJ], [PS], [PU]. However, the use of a tree as our spanning structure may sometimes be preferred due to its practical advantages, in terms of simplicity of the routing and control processes, lower total channel costs, and so on.

2. Basic examples. In this section we consider the game on several example graphs. The natural road networks that are not trees include cycles and grids. The solution of the game for the grid is difficult; asymptotics will appear in §6. Here we content ourselves with easier examples. A strategy for the edge player has value $v$ if the minimum expected payoff for any tree against it is $v$, and a strategy for the tree player has value $v$ if the maximum expected payoff against it for any edge is $v$; in other words, the value of a strategy is the payoff it ensures. The min-max theorem guarantees optimal strategies with the same value. Note that if the game has value $v$, then any choice given nonzero probability in the optimal strategy for one player must have expected payoff exactly $v$ against the optimal strategy for the other player.

**Example 2.1. 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 a positive payoff. The minimum positive payoff equals the length of the shortest cycle, which is 3. Therefore 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 $n$-vertex stars, then any tree 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 $Val(G) = 3 - \frac{6}{n}$. 

With arbitrary weights, the complete graph becomes rather complex. Therefore, let us consider simpler graphs and introduce weights.

**Example 2.2. Cycles and multicycles.** 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 $Val(G, w) = 1$.

Now let $G$ be the multigraph whose underlying simple graph is an $n$ cycle, with $G$ having $n_i$ copies of edge $i$. Suppose that each copy of edge $i$ has weight $w_i$, and let $W = \sum w_i$. Each tree consists of one copy of each of $n-1$ edges. Let $T_i$ denote the trees having no copy of edge $i$, and suppose these are chosen with equal probability totaling $p_i$. If $p_i = \frac{w_i}{W}$, as before, then the expected payoff for a copy of edge $j$ is $1 + 2(1 - p_j)(1 - \frac{1}{n_j}) < 3$ (as in a multigraph, two parallel edges form a cycle of length 2). Hence $Val(G, w) < 3$. In fact, even for the unweighted case $Val(G)$ can be arbitrarily close to 3 when $n$ and $\{n_i\}$ are appropriately chosen. In particular, if $w_i = 1$ and $n_i = m$ for all $i$, then when the edge player chooses edges uniformly the expected payoff of any tree is $3 + 2(1 - \frac{1}{n})(1 - \frac{1}{m})$, so $Val(G) \geq 3 - \frac{2}{n} - \frac{2}{m} + \frac{2}{nm}$. 

In order to obtain higher values of the game for unweighted graphs, it is apparent that we need to have a substantial fraction of nontree edges, and that a large diameter will allow some of those edges to generate a large cost. (In any case, a small diameter leads to a small $Val(G)$. The cycle meets the second criterion but fails the first, which suggests the following example.

**Example 2.3. Cycles with diagonals.** Given $n$ even, let $G$ be the graph consisting of an $n$ cycle $C$ together with the chords joining antipodal points on the cycle. We prove $2 - \frac{4}{3n} \leq Val(G) \leq 3 - \frac{6}{n}$; we do not have an exact solution for this graph. The upper bound is achieved as follows. Let $T$ be a tree consisting of a path of $\frac{n}{2}$ edges on $C$ and the $\frac{n}{2} - 1$ diagonals having one endpoint in the path. If the tree player plays the $n$ rotations of $T$ with equal probability, then the expected payoff of an edge of $C$ is $0 + 4 + 2 \frac{n}{2} + 1)$, and the expected payoff of a diagonal is $0 \cdot (1 - \frac{1}{n/2}) + (n/2 + 1) \cdot \frac{1}{n/2} = 1 + \frac{1}{n}$. This upper bound is not optimal, since in particular, the strategy of sticking to the edges of $C$ (and choosing them uniformly) does not guarantee a payoff greater than 1 for the edge player.

The uniform edge strategy establishes the lower bound. A tree containing no diagonals has expected payoff $[n + (\frac{n}{2})(\frac{n}{2} + 1)] \frac{2}{3n}$, which is quite large. (Indeed, such a tree is simply the cycle $C$ except one of its edges. The cost of this edge with respect to the tree is $n$ and the cost of each diagonal is $n/2 + 1$). For any other tree $T$, we claim there are at least two edges of $C$ not in $T$ that complete fundamental cycles of length at least $\frac{n}{2} + 1$. Therefore, the expected cost of $T$ is at least $[\frac{n}{2} + 1] 2 - \frac{2}{n}$ (since the cost of each nontree edge is clearly at least 4). To prove the claim, view $G$ as a Möbius band or twisted ladder (see Fig. 1); the cycle wraps around twice, and the diagonals become rungs of the ladder. Select a diagonal edge $e$ in $T$. Label the vertices $u_1, \ldots, u_{n/2}$ counterclockwise from one end of $e$ and $v_1, \ldots, v_{n/2}$ counterclockwise from the other end. Let $F$ be the component of $T$ containing $e$ that is obtained from $T$ by deleting either or both of $u_{i}v_{i}$ or $v_{i}u_{i}$ from $T$, if they are present. Let $i$ be the largest index such that $u_i \in F \{v_j \in F\}$. Then $cost(T, u_i u_{i+1}) \geq \frac{n}{2} + 1$, because the choice of $i$ implies that the path in $T$ between $u_i$ and $u_{i+1}$ must contain at least one of $u_k, v_k$ for each $k$ in $\{i + 1, \ldots, \frac{n}{2}, 1, \ldots, i\}$, and both of them for some $k$. This is similar for $v_j v_{j+1}$.

**Example 2.4. Weighted cycle with diagonals.** Consider the graph $G$ given in Example 2.3, but suppose the edges of $C$ have weight 1 and the diagonals have weight $w > 1$. Assume that $n$ is an even multiple of $k$. Let $T_k$ be a tree obtained by taking $\frac{n}{2k}$ equally spaced diagonals, growing a path of length $\lceil \frac{k+1}{2} \rceil$ clockwise and $\lceil \frac{k-1}{2} \rceil$ counterclockwise from each endpoint of each diagonal taken, and adding cycle edges on one semicircle to connect these components. The cost of a nontree cycle edge is $2k + 2w$, except for two edges with cost $\frac{n}{2} + w$. The total cost of a set of $k - 1$ consecutive nontree diagonals is

$$2(k - 1) + \frac{2}{w} \left[ \left\lceil \frac{(k + 1)/2}{2} \right\rceil + \left\lceil \frac{(k + 1)/2}{2} \right\rceil \right] = 2(k - 1) + \frac{1}{w} \left\lceil \frac{k^2}{2} \right\rceil.$$
If the rotations of $T_k$ are selected with equal probability, then the expected cost of an edge of
$C$ is $\frac{1}{n} \left( \left( \frac{n}{2} + w \right) 2 + (2k + 2w)(\frac{n}{2k} - 1) \right) = 2 + \frac{n}{k} - \frac{2k}{n}$. The expected cost of a diagonal (if $k$ is
even) is $2 + \frac{k}{2w} - \frac{1}{k}$. If we choose $k \approx \sqrt{2}w$, then we obtain $\text{Val}(G, w) < 2.707 \ldots$. □

All of our examples here have values bounded by constants. However, it is relatively easy
to construct examples for which $\text{Val}(G) = \Omega(\log n)$. As is well known there are 4-regular
graphs on $n$ vertices with girth $c \log n$ (cf. [Bo], pp. 107–109). For any tree, the $n + 1$ nontree
edges each have cost at least $c \log n$. Hence against the uniform edge strategy every tree
has expected cost at least $(\frac{c}{2}) \log n$. In §5, we will see that the $N$-vertex grid also has cost
$\Theta(\log N)$.

3. 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 $\pi, \rho$, where $\pi$ specifies the initial positions of the $k$ servers and $\rho$ 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 $\rho$, starting with the servers in
the initial positions $\pi$. We may also associate with the pair $\pi, \rho$ a real number $\text{OPT}(\pi, \rho)$
denoting the minimum possible cost of processing the request sequence $\rho$, starting with the
servers in the $k$-tuple $\pi$ of initial positions; $\text{OPT}(\pi, \rho)$ 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 $\pi, \rho$,

$$A(\pi, \rho) \leq C \cdot \text{OPT}(\pi, \rho) + a.$$  

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 tree-like 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 tree-like 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 [MaMcS1].

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 $\pi$ of initial positions and every request sequence $\rho$,

$$E[A(\pi, \rho)] \leq C \cdot COST(\pi, \rho) + a,$$

where $E$ denotes mathematical expectation. Several examples are known in which randomized algorithms can achieve a smaller competitive factor than deterministic ones [BBK], [BLS], [FKLMSY].

We consider the $k$-server problem on a class of metric spaces that generalize the tree-like 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 3.1.** 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 $k(1 + \text{Val}(G, w))$-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 uniformly 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 tree-like 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 tree-like road network.

Let the random variable $A(\pi, \rho)$ denote the cost that the above randomized algorithm incurs in processing the request sequence $\pi$, starting from the initial server positions $\rho$. Let $OPT(\pi, \rho)$ denote the optimal off-line cost of processing $\rho$, starting from server positions $\pi$, 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 $\rho$, starting from the server positions $\pi$, in the tree-like road network constructed by the algorithm. By the $k$-Competitiveness of the Chrobak–Larmore algorithm, $A(\pi, \rho) \leq k \cdot OPT'(\pi, \rho) + a$. 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 nontree 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 \( \rho \) starting from the initial \( k \) tuple of server positions \( \pi \) at cost \( OPT(\pi, \rho) \). Let \( \rho = \rho_1 \rho_2 \ldots \rho_i \). 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 \( \rho \) on the derived tree-like road network by having server \( x(i) \) process request \( \rho_i \) 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 tree-like 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 (1) follows. \( \Box \)

**Corollary 3.2.** There is a 2k-competitive randomized algorithm on the circle.

**Proof.** This follows immediately from Example 2.2 (or even from its special case corresponding to a cycle of two vertices). \( \Box \)

Note that as shown in [FRRS] there is a deterministic \( O(k^2) \)-competitive algorithm for the circle.

### 4. 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 connected 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 endpoints of \( e \). Define

\[ cost^*(T, e) = \frac{path(T, e)}{w(e)}. \]
Also, for every subset of edges \( E' \), denote
\[
\text{cost}^* (T, E') = \sum_{e \in E'} \text{cost}^*(T, e).
\]

Let \( S(G, w, T) = \text{cost}^*(T, E)/|E| \). Let \( S_{\text{opt}}(G, w) = \min_T S(G, w, T) \), where \( T \) ranges over all spanning trees of \( G \). In the unweighted case, where \( w(e) = 1 \) for all \( e \), we abbreviate \( S(G, w, T) \) by \( S(G, T) \) and \( S_{\text{opt}}(G, w) \) by \( S_{\text{opt}}(G) \). 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 \( \text{cost}^*(T, e) \). Let the value of this game be denoted \( \text{Val}^*(G, w) \). In the unweighted case, we abbreviate \( \text{Val}^*(G, w) \) by \( \text{Val}^*(G) \). Then, since \( |\text{cost}(T, e) - \text{cost}^*(T, e)| \leq 1 \) for every edge \( e \), \( |\text{Val}(G, w) - \text{Val}^*(G, w)| \leq 1 \). 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 endpoints 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 4.1.** \( \text{Val}^*(G, w) = \sup_{G', w'} S_{\text{opt}}(G', w') \), where \( G', w' \) ranges over all weighted multigraphs obtained from \( G, w \) by replication.

**Proof.** Recall that
\[
\text{Val}^*(G, w) = \max_q \min_p \sum_T \sum_e p(T)q(e)\text{cost}^*(T, e),
\]
where \( q \) ranges over probability distributions for the edge player, and \( p \) over probability distributions for the tree player. For each fixed probability distribution \( q \), \( \sum_T \sum_e p(T)q(e)\text{cost}^*(T, e) \) is minimized by a probability distribution \( p \) concentrated on one tree; this is an instance of the general observation that if one of the players in a zero-sum two-person game announces his mixed strategy, then the other player can play optimally by choosing a pure strategy. It follows that
\[
\text{Val}^*(G, w) = \max_q \min_p \sum_T \sum_e q(e)\text{cost}^*(T, e).
\]

Now consider any weighted multigraph \( G', w' \) obtained from \( G, w \) by replication. Let \( d(e) \) be the number of copies of edge \( e \), and let \( q(e) \) be \( d(e)/|\sum d(e)| \). Then \( q \) is a probability distribution, and
\[
S_{\text{opt}}(G', w') = \min_T \sum_e q(e)\text{cost}^*(T, e).
\]

It follows that \( S_{\text{opt}}(G', w') \leq \text{Val}^*(G, w) \). Conversely, let \( q \) be the probability distribution for the edge player that maximizes \( \min_T \sum_e q(e)\text{cost}^*(T, e) \). For any (large) integer \( M \), let \( G^M, w^M \) be obtained from \( G, w \) by making \( 1 + [Mq(e)] \) copies of each edge \( e \). Then clearly, as \( M \) tends to infinity, \( S_{\text{opt}}(G^M, w^M) \) converges to \( \text{Val}^*(G, w) \). The conclusion of the theorem follows. \( \Box \)

An edge-transitive graph is a graph \( G \) such that for any edges \( e, e' \) in \( G \) there is an automorphism \( \sigma \) in the automorphism group \( \Gamma(G) \) such that \( \sigma \) 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 4.2.** If \( G \) is edge-transitive, then \( \text{Val}^*(G) = S_{\text{opt}}(G) \).
Proof. If the edge player plays each edge with equal probability, the expected payoff is at least $S_{\text{opt}}(G)$ for any tree. Let $T$ be a tree with $S(G, T) = S_{\text{opt}}(G)$ and let $T'$ be the collection of images of $T$ under the elements of $\Gamma(G)$. For each $T' \in T$, let the probability of playing $\sigma T$ be $\frac{1}{|\Gamma|}$ times the number of automorphisms $\sigma$ such that $\sigma T = T'$. We claim that the expected payoff for this tree strategy is $S_{\text{opt}}(G)$ for every edge, which completes the proof. Note that $\text{cost}^*(T, e) = \text{cost}^*(\sigma(T), \sigma(e))$. Also, Lagrange’s theorem states that if a group $\Gamma(G)$ acts on a set $S (= E(G))$, then the number of group elements taking $e \in S$ to any member of its orbit (including to itself) is the same. Hence the expected payoff for edge $e$ is

$$\frac{1}{|\Gamma|} \sum_{\sigma} \text{cost}^*(\sigma(T), e) = \frac{1}{|\Gamma|} \sum_{\sigma} \text{cost}^*(T, \sigma^{-1}(e)) = \frac{1}{|\Gamma|} |E| \sum_{e'} \text{cost}^*(T, e') = S_{\text{opt}}(G). \quad \square$$

5. Upper and lower bounds for the tree game.

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

THEOREM 5.1. There exists a positive constant $c$ such that for all $n$, there exists an $n$-vertex graph $G_n$ such that $Val^*(G_n) > c n\ln n$.

Proof. The following is a known result in extremal graph theory (cf. [Bo], pp. 107–109): There exists a positive constant $a$ such that for all $n$ there exists an $n$-vertex graph $G_n$ with $2n$ edges such that every cycle in $G_n$ is of length at least $a n\ln n$. Let $T$ be any spanning tree in $G$. Then, for any nontree edge $e$, $\text{cost}^*(T, e) > a n\ln n - 1$. Since more than half the edges are nontree edges, it follows that for every $T$, $\frac{1}{|E|} \sum_{e} \text{cost}^*(T, e) \geq \frac{1}{2} (a n\ln n - 1)$. Thus $S_{\text{opt}}(G) \geq \frac{1}{2} (a n\ln n - 1)$, and it follows from Theorem 4.1 that $Val^*(G) \geq \frac{1}{2} (a n\ln n - 1)$. $\square$

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. (Note that in the context of the $k$-server problem we are really interested only in graphs. However, the construction technique employed in our proof makes it necessary to prove the result in the more general setting of multigraphs.)

LEMMA 5.2. For every $n$-vertex multigraph $G$, there exists a multigraph $G'$, $\omega'$ on the same set of vertices and at most $n(n+1)$ edges such that $S_{\text{opt}}(G, \omega) \leq 2 \cdot S_{\text{opt}}(G', \omega')$.

Proof. Let $E$ be the edge multiset of $G$, and let $E^{\text{set}}$ be the (maximal) set of distinct edges in $E$, where two edges are considered distinct if they don’t have the same pair of endpoints. (That is, $E^{\text{set}}$ contains a single representative edge $uv$ for every pair of vertices $u$ and $v$ that are adjacent in $G$.) For each edge $e \in E^{\text{set}}$, let $d(e)$ be the number of copies of $e$ in $E$, and let $\omega'(e)$ be the lowest weight of a copy of $e$ in $G, \omega$. Then the cardinality of $E$ (i.e., the total number of edges in $G$ counting repetitions) is

$$|E| = \sum_{e \in E^{\text{set}}} d(e).$$

Consider a new multigraph $G'$ with the same set of distinct edges, but with each edge $e$ occurring $r(e)$ times instead of $d(e)$ times, where

$$r(e) = 1 + \left\lfloor \frac{d(e) |E^{\text{set}}|}{|E|} \right\rfloor . \quad (2)$$

Then the cardinality of the edge multiset $E'$ of $G'$ satisfies

$$|E'| = \sum_{e \in E^{\text{set}}} r(e) \leq |E^{\text{set}}| + \frac{|E^{\text{set}}|}{|E|} \sum_{e \in E^{\text{set}}} d(e) = 2 |E^{\text{set}}| . \quad (3)$$
Since $E^{set}$ contains at most one edge per pair of endpoints, including self-loops, it follows that $G'$ has at most $n(n + 1)$ edges, as required.

It remains to bound $S_{opt}(G', w')$. Combining (2) and (3) we get

$$r(e) \geq \frac{d(e)|E^{set}|}{|E|} \geq \frac{d(e)|E'|}{2|E|}.$$  \hspace{1cm} (4)

The multigraph $G'$ has a spanning tree $T$ such that

$$S_{opt}(G', w') = \frac{1}{|E'|} \sum_{e \in E^{set}} r(e) \text{cost}^*(T, e, w'),$$  \hspace{1cm} (5)

with the notation $\text{cost}^*(T, e, u)$ denoting $\text{cost}^*$ defined with respect to a weight function $u$. Using (4) and (5) and the choice of $w'$, we find that

$$S_{opt}(G', w') \geq \frac{1}{2|E|} \sum_{e \in E^{set}} d(e) \text{cost}^*(T, e, w).$$

But since $T$ is a spanning tree of $G$ as well as $G'$, this last expression is at least $S_{opt}(G, w)/2$, so $S_{opt}(G, w) \leq 2S_{opt}(G', w')$. \hspace{1cm} \square

5.2. An upper bound for unweighted graphs: outline. Before we state and prove our upper bound for $S_{opt}(G, w)$ on general (weighted) graphs, it is instructive to sketch the construction in the simpler setting of an unweighted graph.

Our construction is based on the concepts of clusters and partitions. A cluster is a subset of the vertices whose induced subgraph is connected. A partition of a given graph $G = (V, E)$ is a collection of disjoint clusters whose union is $V$. The basic procedure used in the construction is a variant of the clustering algorithm of [Aw]. The construction involves a parameter $x$ depending on $n$. This parameter is required to satisfy $x(n) > 1$ and be monotone increasing in $n$ (with $x(n) \to \infty$), but its precise definition is left to be specified later. The output of the procedure is a partition of the given graph into clusters with low radii (specifically, radii at most $y(n) = O(x(n) \ln n)$), with the additional property that “most” of the graph edges are internal to clusters, and only a fraction of $1/x(n)$ of the edges connect endpoints in different clusters. We refer to edges of these two types as “internal” and “intercluster” edges, respectively.

A spanning tree can be built on the basis of such a partition as follows. First, construct a shortest-path spanning tree $T_C$ (as defined earlier) for every cluster $C$ in the partition. Note that since the partition is composed of disjoint clusters, these spanning trees form a spanning forest in the graph. Now connect the forest into a single tree by selecting a suitable tree of intercluster edges.

This last step can be carried out recursively. Given the partition, create an auxiliary multigraph $\tilde{G}$ by collapsing each cluster $C$ into a single vertex $v_C$, and including an edge between two such vertices of $\tilde{G}$ for each original edge connecting these clusters. (Here is why it is useful to handle multigraphs in this algorithm, rather than simple graphs.) Next, apply the same procedure recursively to $\tilde{G}$, and obtain a tree $\tilde{T}$. The final tree $T$ will consist of the union of $\tilde{T}$ and the trees $T_C$ constructed for each cluster $C$.

Observe that internal edges will typically enjoy a lower cost than intercluster edges. This is because the path connecting the endpoints of an edge $uw$ internal to a cluster $C$ in the final tree $T$ is the path connecting $u$ and $w$ on the tree $T_C$, and the partitioning algorithm guarantees that $C$ has a low radius. In contrast, for an intercluster edge $u_1u_2$, where $u_1 \in C_1$ and $u_2 \in C_2$, the path connecting $u_1$ and $u_2$ in $T$ is not simply the path connecting $v_{C_1}$ and $v_{C_2}$ in the tree $\tilde{T}$. Rather, this path is expanded when retracting from $\tilde{G}$ to $G$ by replacing each vertex $v_C$ on...
it with an appropriate path segment on the internal tree \( T_C \). This heavier cost for intercluster edges is compensated for by the fact that these edges are only a \( 1/x(n) \) fraction of the entire edge set, and therefore their contribution to the average cost is controllable.

The cost analysis is carried along the following lines. Let \( f(n, m) \) be the maximum of \( S(G, T) \) over all connected multigraphs \( G \) with up to \( n \) vertices and \( m \) edges, where \( T \) is the spanning tree of \( G \) constructed by the above algorithm. Then

\[
 f(n, m) \leq 2y(n) + \frac{1}{x(n)} \cdot f(n, m/x(n)) \cdot 5y(n). 
\]

This inequality can be explained as follows. The first term in the right-hand side of the inequality denotes the contribution of the internal edges. This term is bounded above by \( 2y(n) \) since each cluster is of radius at most \( y(n) \). The second term denotes the contribution of the intercluster edges. In this term, the factor \( 1/x(n) \) is an upper bound on the fraction of intercluster edges. The number of edges in \( G \) is at most \( m/x(n) \), thus the expected cost of an edge in \( G \) is at most \( f(n, m/x(n)) \). Now consider an intercluster edge \( e = uv \), and let \( \tilde{y} \) be the path connecting its endpoints in \( G \), namely, \( \tilde{y} = (e_1, \ldots, e_q) \), where each \( e_i = u_i v_i \) is an intercluster edge connecting the vertices \( v_{i-1} \) and \( v_i \) in \( G \), for \( 1 \leq i \leq q \), and \( u \in C_0 \) and \( v \in C_q \). The path \( \tilde{y} \) can be expanded in \( G \) to a path \( y \) connecting \( u \) and \( v \), by taking the following steps:

1. inserting between \( e_i \) and \( e_{i+1} \) a subpath of the tree spanning \( C_i \), connecting \( v_i \) and \( v_{i+1} \) (for \( 1 \leq i \leq q - 1 \)),
2. inserting a subpath of the tree spanning \( C_0 \), connecting \( u \) and \( v_1 \), and
3. inserting a subpath of the tree spanning \( C_q \), connecting \( v_q \) and \( v \).

Each of these \( q + 1 \) subpaths is of length at most \( 2y(n) \), namely, twice the radius of the corresponding cluster. Therefore the total length of the resulting path \( y \), for a given path \( \tilde{y} \) of length \( q \), is at most \( q + 2(q + 1)y(n) \). Thus each connecting path in \( G \) expands by a factor of at most \( (q + 2(q + 1)y(n))/q \leq 5y(n) \) in \( G \). This accounts for the factor \( f(n, m/x(n)) \cdot 5y(n) \) in the second term.

Finally, using the above inequality and choosing \( x(n) \) optimally as \( \exp(\sqrt{\ln n \ln \ln n}) \), we find that \( f(n) \leq \exp(O(\sqrt{\ln n \ln \ln n})) \).

5.3. An upper bound for weighted graphs: outline. For understanding the weighted case, it is convenient to think of the above algorithm as an iterative, rather than recursive one. From this point of view, think of the main clustering procedure as a “machine,” to which the graph is “fed” for a number of iterations. In each iteration \( j \geq 1 \), the procedure constructs a partition for the current graph \( G_j \) (where \( G_1 \) is the original graph \( G \)), and then contracts the clusters into single vertices, thus creating the graph \( G_{j+1} \) to be processed in the next iteration. Each such iteration also reduces the number of “uncovered” edges by a factor of \( x \), until all edges are “covered.” (An edge is covered if it is internal to some cluster we have already constructed.)

Given this view of the partitioning process, one can analyze the radii of the clusters constructed at each iteration \( j \geq 1 \) (henceforth called “\( j \) clusters”). For that purpose, it is instructive to review the partitioning process in more detail. The partitioning procedure creates the clusters sequentially. Each cluster \( C \) is “grown” by starting at a single vertex, and successively merging it with layers of neighbouring vertices. This merging process is stopped once the number of outgoing edges of the current cluster \( C \) (namely, edges with one endpoint in \( C \) and one outside \( C \)) is at most a \( 1/x(n) \) fraction of the number of internal edges (namely, edges with both endpoints in \( C \)). As will be shown later on (in the formal proof for the weighted case), this merging process is bound to halt after adding no more than, say, \( y(n)/3 \) layers. Hence each \( j \) cluster has radius at most \( y(n)/3 \) in the current graph \( G_j \). However,
in the original graph $G$, such $j$ clusters have radius $r_j \leq y^{j+1}(n)$. This can be easily argued by noting that when a $j$ cluster is constructed, each merged layer increases the radius by an additive factor of up to

$$1 + 2r_{j-1},$$

where the 1 is contributed by the added edge, and the $2r_{j-1}$ by the diameter of the $(j - 1)$ cluster corresponding to the endpoint of that edge in $G_j$. The bound on $r_j$ follows inductively since at most $y(n)/3$ layers are merged.

Let us next outline the modifications needed for handling the weighted case. The main problem that needs to be overcome is that in this case, all edges cannot be treated alike, since when growing a cluster, a single layer merging step will increase the radius of the resulting cluster by the weight of the heaviest merged edge, rather than by just one, thus the radii of constructed clusters cannot be controlled.

The algorithm thus needs to be modified as follows. It is necessary to break the set of edges $E$ into classes $E_i$, $i \geq 1$, according to weights, with $E_i$ containing all edges whose weight is in the range $[y^{i-1}, y^i)$ for an appropriately chosen parameter $y = y(n, x)$ (with $x$ a parameter to be determined in the same spirit as in the unweighted case). Intuitively, we would like to handle the lighter edges first. (We may, and will, assume that the minimum weight of an edge is 1.)

We now feed the classes $E_i$ to the “machine” in a pipelined fashion, with overlaps. Namely, in iteration 1 only the edges of $E_1$ are considered, in the next iteration $E_2$ is added, and so on. In general, the class $E_i$ is taken into consideration for the first time in the $i$th iteration, and is processed for the next $\rho = O(\ln n/\ln x)$ iterations, each reducing the number of unsatisfied edges in it by a factor of $x$, until the entire set $E_i$ is exhausted.

A crucial point that must be explained at this point is the role of the parameter $y$ in the construction. In the weighted case, this parameter has two different functions. The first is similar to the one it had in the unweighted case, i.e., it is (more or less) the radius increase bound for constructed clusters. That is, clusters constructed for the graph $G_j$ in the $j$th iteration will have at most radius $y/3$ in $G_j$. The second function of the parameter $y$ is governing the weight range of the edge classes.

The combination of these two functions implies that in the construction of new clusters during a given iteration $j$, there is a balance between the contributions to the radius made by previously constructed clusters and by new edges. This is what guarantees that cluster radii are properly bounded in the original graph $G$ as well. Specifically, the radius of a $j$ cluster (constructed in iteration $j$) is bounded by $r_j \leq y^{j+1}$. Formally, this can again be deduced inductively, noting that when a $j$ cluster is constructed, each merged layer increases the radius by up to

$$y^j + 2r_{j-1} \leq 3y^j,$$

where (in analogy with (6)) the $y^j$ is contributed by the added edge and the $2r_{j-1}$ by the diameter of the $(j - 1)$ cluster corresponding to the endpoint of that edge in $G_j$, and $r_{j-1} \leq y^j$ follows by the inductive hypothesis. This, combined with the fact that at most $y/3$ layers are added, yields the desired bound of $r_j \leq y^{j+1}$ by induction.

5.4. The upper bound for weighted graphs: formal proof. We are now ready to formally state and prove the upper bound on $S_{opt}$.

**Theorem 5.3.** There exists a constant $c$ such that for $n$ sufficiently large, every $n$-vertex connected multigraph $G$ and every weight function $w$ satisfy $S_{opt}(G, w) \leq \exp(c\sqrt{\log n \log \log n})$. 
By Lemma 5.2, 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 $G$. Again, the construction involves a parameter $1 \leq x \leq n$ depending on $n$, to be fixed later. Define

$$\rho = \left\lfloor \frac{3 \ln n}{\ln x} \right\rfloor, \quad \mu = 9 \rho \ln n, \quad y = x \mu.$$

Break the edge multiset $E$ into subclasses $E_i$, for $i \geq 1$, according to weights, defining

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

(Recall that the edge weights are normalized to be greater than or equal to 1.)

As described above, the algorithm proceeds in iterations. In each iteration $j$ we compute some clusters in the graph $G_j$, and mark some of the edges as “covered.” We then contract the clusters into single nodes, thus preparing the multigraph $G_{j+1}$ for the next iteration.

More precisely, let $E_{i,j}$, for $i, j \geq 1$, denote the multiset of edges from $E_i$ that are still uncovered at the beginning of iteration $j$. The purpose of the construction at iteration $j$ is to partition the vertex set into disjoint clusters such that

- each cluster has a spanning tree of radius at most $y^{j+1}$ in $G$;
- in every nonempty edge class $E_i$, $1 \leq i \leq j$, the fraction of intercluster edges is at most $1/x$; i.e., $|E_{i,j+1}| \leq |E_{i,j}|/x$.

Note that this definition for the purpose of an iteration, and particularly, restricting the second requirement to classes $E_i$ for $i \leq j$, implies that iteration $j$ need only consider (and process) edges from the edge multisets $E_i$ for $i \leq j$, i.e., edges of weight less than $y^j$, and may ignore heavier edges. Avoiding heavier edges is crucial for guaranteeing the radius bound in the first requirement, as discussed earlier.

We next present the procedure used for forming the partition in iteration $j$. This procedure is a modified version of the clustering algorithm of [Aw]. The partition is constructed in a “greedy” fashion by a sequence of stages, each stage building a single cluster. After each stage, all the vertices of the graph that were included in the constructed cluster are eliminated from the graph.

Consider the beginning of a stage and let $K$ be a connected component of the subgraph induced by the remaining vertices (i.e., the ones not yet included in any of the previously built clusters). Choose arbitrarily a “root vertex” $u$ in $K$. Stratify the vertices and edges of $K$ into layers according to their (unweighted) distance from $u$ as follows. For each integer $\ell \geq 0$, let $V(\ell)$ be the set of vertices at (unweighted) distance $\ell$ from $u$ in $K$ (where the unweighted distance between two vertices is defined as the minimal number of edges in a path connecting them). Also 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 $\ell^*$ be the least $\ell$ such that

$$\forall 1 \leq i \leq j, |E^i_j(\ell + 1)| \leq \frac{1}{x} |E^i_j(1) \cup E^i_j(2) \cup \cdots \cup E^i_j(\ell)|.$$

Note that such $\ell$ always exists. The resulting cluster is the vertex set $V(1) \cup V(2) \cup \cdots \cup V(\ell^*)$. The vertices of this cluster are now eliminated from the graph, and the next stage is started. This process of cluster generation continues until the graph is exhausted (i.e., all vertices are assigned to clusters).

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 $x = x(n), \rho, \mu, y$, and the edge classes $E_i$, as defined above.

While $\bigcup_i E_i \neq \emptyset$ do the following:

1. Partition the vertex set of $G_j$ into clusters as described above.
2. Construct a shortest-path spanning tree $T_C$ in each cluster $C$ of $G_j$.
3. Add each edge $e$ of each of the constructed trees to the output tree $T$.
4. Construct the next multigraph $G_{j+1}$ by contracting each cluster $C$ into a single vertex $v_C$, discarding internal edges from $\bigcup_i E_i$, and replacing each intercluster edge $u_1u_2$, for $u_1 \in C_1$ and $u_2 \in C_2$, by a new edge connecting the corresponding contracted vertices $v_{C_1}$ and $v_{C_2}$.
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 $E_i$ considered by the algorithm (i.e., the only nonempty ones) are those satisfying $j - \rho \leq i \leq j$.

**Lemma 5.4.** $|E_i^j| \leq |E_i^j|/x^{j-i}$ for every $1 \leq i \leq j$.

**Proof.** The claim follows from the fact that $|E_i^{j+1}| \leq |E_i^j|/x$ for every $1 \leq i \leq j$.

To see this, note that by definition of the cluster construction process, each time a cluster is constructed in a connected component $K$, the number of edges of $E_i^j$ connecting that cluster to the vertices of $K$ not selected for inclusion is at most $1/x$ times the number of edges of $E_i^j$ included in the cluster. Thus, in the entire graph the number of edges of $E_i^j$ joining vertices in different clusters (constituting $E_i^{j+1}$) is at most $1/x$ times $|E_i^j|$. □

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

**Lemma 5.5.** In iteration $j$, the radius of each cluster in $G$ is bounded above by $y^{j+1}$.

**Proof.** Let $B_j$ denote the number of layers $\ell^*$ merged into any cluster in iteration $j$. We first prove that $B_j$ is bounded above by

$$B_j \leq y/3 = 3\rho x \ln n.$$  

To see this, note that by Lemma 5.4, $|E_i^{j+\rho}| = 0$ for every $i \geq 1$, which implies that the augmentation rule (*) governing the addition of layers to the cluster need only consider edges from classes $E_i^j$ for $j - \rho + 1 \leq i \leq j$. Each additional layer $\ell$ implies that some set $E_i^j(\ell)$ fails to satisfy this condition. Namely, for all $\ell$ such that $1 < \ell < \ell^*$, at least one $i$ such that $j - \rho + 1 \leq i \leq j$, satisfies

$$|E_i^j(\ell)| > \frac{1}{x}|E_i^j(1) \cup E_i^j(2) \cup \cdots \cup E_i^j(\ell-1)|.$$  

Thus

$$|E_i^j(1) \cup E_i^j(2) \cup \cdots \cup E_i^j(\ell)| > \left(1 + \frac{1}{x}\right)|E_i^j(1) \cup E_i^j(2) \cup \cdots \cup E_i^j(\ell-1)|.$$  

Since there are at most $\rho$ possible values of $i$, by the pigeonhole principle it follows that for $\ell^*$, there exists at least one such $i$ that caused the addition of a layer for at least $[\ell^*/\rho]$ times. This $i$ satisfies

$$|E_i^j(1) \cup E_i^j(2) \cup \cdots \cup E_i^j(\ell^*)| \geq \left(1 + \frac{1}{x}\right)^{[\ell^*/\rho]}.$$  

Since the number of edges in the multigraph is at most $n(n+1)$, it follows that, for $n$ sufficiently large, $\ell^*/\rho \leq 3x \ln n$, so (8) is proved.
We now prove the main claim of the lemma, i.e., the bound on the radius of a cluster created in the $j$th iteration by induction on $j$. In order to analyze all iterations together (including also iteration 1), hypothesize iteration 0 as the one that yielded the initial graph $G$, with each vertex representing a cluster of radius 0 (thus trivially satisfying the inductive claim). Now for $j \geq 1$, suppose the claim holds up to $j - 1$, and consider the $j$th iteration. At the beginning of the iteration, every vertex of the multigraph represents a cluster of radius up to $y^j$, by the inductive hypothesis. Also, every edge considered in iteration $j$ is of length up to $y^j$. Hence in constructing a cluster, each additional layer contributes up to $3y^j$ to the radius. By inequality (8) above, the final radius is at most $3y^j \cdot y/3 = y^{j+1}$, as required.

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

**Lemma 5.6.** $\text{cost}^*(T, E) \leq 4x^2 \mu^{p+1} |E|$.

**Proof.** Let $H^j_i$ denote the set of edges from $E_i$ that were covered during iteration $j$, i.e.,

$$H^j_i = E^j_i \setminus E^{j+1}_i.$$

We first claim that for every edge $e \in H^j_i$,

$$\text{cost}^*(T, e) = \frac{\text{path}(T, e)}{w(e)} \leq 2y^{j-i+2}.$$

(9)

To see this, note that since $e$ is covered during iteration $j$, by the previous lemma $\text{path}(T, e) \leq 2y^{j+1}$, while on the other hand, $e \in E_i$ implies $w(e) > y^{i-1}$.

By Lemma 5.4,

$$|H^j_i| \leq |E^j_i| \leq |E_i|/x^{j-i}.$$

(10)

It follows from inequalities (9) and (10) that for every $1 \leq i \leq j$,

$$\text{cost}^*(T, H^j_i) \leq 2x^2 \mu^{j-i+2} |E_i|.$$

(11)

Summing these costs over $i \leq j \leq i + \rho - 1$, we get that

$$\text{cost}^*(T, E_i) \leq \sum_{j=i}^{i+\rho-1} \text{cost}^*(T, H^j_i) \leq 4x^2 \mu^{p+1} |E_i|$$

for every $i \geq 1$. Summing the costs over all weight classes we get

$$\text{cost}^*(T, E) = \sum_{i=1}^{\rho+1} \text{cost}^*(T, E_i) \leq 4x^2 \mu^{p+1} |E|,$$

and the lemma follows.

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

$$S(G, w, T) \leq 4x^2 \mu^{p+1} \leq 4x^2 \left( \frac{27 \ln^2 n}{\ln x} \right)^1 + \left[ \frac{\ln n}{\ln x} \right].$$

Selecting $x = \exp(\sqrt{\log n \log \log n})$ optimizes this bound on $S(G, w, T)$ as $S(G, w, T) \leq \exp(O(\sqrt{\log n \log \log n}))$. This completes the proof of Theorem 5.3.

**Corollary 5.7.** There exists a constant $c$ such that for $n$ sufficiently large, every $n$ vertex connected multigraph $G$, $w$ satisfies $\text{Val}^*(G, w) \leq c^{\sqrt{\log n \log \log n}}$. 

$$\text{Val}^*(G, w) \leq c^{\sqrt{\log n \log \log n}}.$$
Proof. By Theorem 4.1, \(\text{Val}^*(G, w) = \max_{G', w'} S_{\text{opt}}(G', w')\), where \(G', w'\) ranges over all the weighted multigraphs obtained from \(G, w\) by replication. The claim thus follows from Theorem 5.3. \(\square\)

We observe that the tree construction algorithm as described above can be used within a procedure (based on the ellipsoid algorithm) for implementing the first step in the online \(k\)-server algorithm of Theorem 3.1, yielding a polynomial-time algorithm for the problem; i.e., the optimal strategy for the tree player can, in fact, be efficiently approximated.

6. Grids and hypercubes. In this section, we discuss the value of the game on unweighted grids and hypercubes.

6.1. Upper bounds for \(d\)-dimensional grids. We consider the \(d\)-dimensional grid \(G_{n,d}\) that is the Cartesian product of \(d\) \(n\)-vertex paths and has \(N = n^d\) vertices. Formally, the vertex set of \(G_{n,d}\) consists of all vectors of length \(d\) whose coordinates are in \(\{1, \ldots, n\}\). The grid contains an edge \(uv\) if the vectors \(u\) and \(v\) differ in exactly one coordinate where their values differ by exactly 1.

A hypercube is edge-transitive, so \(\text{Val}^*(G_{2,d}) = S_{\text{opt}}(G_{2,d})\). 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 has edge-merging. 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 \(an\), then \(S(G', T) \leq S(G, T)(1 - \frac{1}{n}) + \alpha\). On the other hand, \(\text{Val}^*(G') \geq \text{Val}^*(G)\), 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 6.1.** For the \(d\)-dimensional grid \(G_{n,d}\) with \(N = n^d\) vertices, \(S_{\text{opt}}(G_{n,d}) < 2\log N\).

**Proof.** Suppose \(n = 2^k\). For the upper bound, we construct a tree \(T_k\) in \(G_{n,d}\) of diameter

\[
d_k \leq (2d - 1)(2^k - 1) = (2d - 1)(n - 1)
\]

such that \(S(G_{n,d}, T_k) < 2d \log n = 2\log N\). Partition the vertices of \(G_{n,d}\) into \(2^d\) subsets inducing copies of \(G_{n/2,d}\). There is a unique \(d\) cube \(Q\) consisting of one vertex from each of these subsets. Form \(T_k\) by using a copy of \(T_{k-1}\) in each of the \(2^d\) subsets and a spanning tree of diameter \(2d - 1\) in \(Q\) (a breadth-first search tree from any vertex will do). Figure 2 illustrates the construction for \(G_{8,2}\). We have \(d_k \leq (2d - 1) + 2d_{k-1}\), with \(d_0 = 0\). Hence \(d_k \leq (2d - 1)(2^k - 1)\). Note that \(G_{n,d}\) has \(dn^{d-1}(n - 1) = dN(1 - \frac{1}{n})\) edges; let \(N' = \frac{n}{2} d^d\). Also, the number of non-tree edges that join the copies of \(G_{n/2,d}\) is \(dn^{d-1} - (2d - 1)\). We thus have

\[
S(G_{n,d}, T_k) \leq \frac{2^d dN'(1 - \frac{2}{d})S(G_{n/2,d}, T_{k-1}) + (dn^{d-1} - 2d + 1)(d_k + 1)}{dN(1 - \frac{1}{n})} \leq \frac{S(G_{n/2,d}, T_{k-1}) + \frac{(d + 1)}{n - 1} < S(G_{n/2,d}, T_{k-1}) + 2d}{J}.
\]

Hence \(S(G_{n,d}, T_k) < 2dk\). \(\square\)

For the interesting special case of the hypercube \(Q_d = G_{2,d}\), which is edge-transitive, we have a slightly better construction. This construction improves on the general case only by roughly a factor of four, but it is still of interest, since it is conjectured to be optimal.

**Theorem 6.2.** \(\text{Val}^*(Q_d) = S_{\text{opt}}(Q_d) \leq (d + 1)/2\).

**Proof.** Equality holds for \(d = 1, 2\), with no choice of tree. For \(d > 2\), construct a tree \(T_d\) by taking the optimal \(d - 1\)-dimensional tree \(T^*\) in copy A of \(Q_{d-1}\) and adding all edges between this and the other copy of \(Q_{d-1}\). The edges in copy A have the same cost* as before,
and those between the copies have $\text{cost}^*=1$. Each edge $e$ in copy $B$ corresponding to an edge $e'$ in copy $A$ has $\text{cost}^*(T_d, e) = \text{cost}^*(T^*, e') + 2$. We have

$$S_{\text{opt}}(Q_d) \leq S(Q_d, T_d) = \frac{1}{(d2^{d-1})} ((d-1)2^{d-2}(2 \cdot S_{\text{opt}}(Q_{d-1}) + 2) + 2^{d-1} \cdot 1).$$

Hence $S_{\text{opt}}(Q_d) < \frac{d-1}{d} S_{\text{opt}}(Q_{d-1}) + 1$, implying the claim inductively.

We believe that the general construction described above is essentially optimal for large $n$, with $S_{\text{opt}}(G_{n,d}) = 2\lg N + o(\lg N)$. In the following sections we prove only that $S_{\text{opt}}(G_{n,d}) \geq c \log N$ for some constant $c$.

### 6.2. Lower bounds for two-dimensional grids

For clarity, it is useful to first present the lower bound for the case of $d = 2$, showing $S_{\text{opt}}(G_{n,2}) \geq c \lg n + o(\lg n)$. For this we need several preliminary results.

**Lemma 6.3.** If $A$ is a set of at least $\alpha^2$ vertices in $G_{n,2}$, where $|A| \leq \frac{n^2}{2}$, then there are at least $\alpha$ rows that $A$ intersects but does not fill or at least $\alpha$ columns that $A$ intersects but does not fill.

**Proof.** Suppose $A$ intersects $r$ rows and $s$ columns, with $r \geq s$. Then $rs \geq \alpha^2$ implies $r \geq \alpha$, and the result holds for $A$ unless $A$ fills at least one row. If $A$ does fill a row then $n = s \leq r$. Thus $A$ intersects every row and every column. Now our goal is to prove that there are at least $\alpha$ rows that $A$ does not fill or at least $\alpha$ columns that $A$ does not fill. For this not to happen means that $A$ fills more than $n - \alpha$ rows and more than $n - \alpha$ columns. This requires that $A$ has more than $n^2 - \alpha^2$ vertices, which is impossible if $\alpha^2 \leq |A| \leq \frac{n^2}{2}$. \(\Box\)

**Lemma 6.4.** If $A$ is a set of at least $\alpha^2$ vertices in $G_{n,2}$, where $|A| \leq \frac{n^2}{2}$, and $B$ is a set of at most four vertices in $A$, then there are at least $\frac{n}{2}$ vertices in $A$ that have neighbors outside $A$ and have distance at least $\frac{6}{16}$ from each vertex of $B$.

**Proof.** The preceding lemma guarantees a set $C$ of at least $\alpha$ vertices in $A$ that are in distinct rows (or distinct columns) and have neighbors outside $A$. Since these are in distinct
rows, a vertex of $B$ can be within distance $\frac{m-1}{4}$ of at most $\frac{n}{8}$ vertices in $C$. When we delete from $C$ the vertices that are too close to $B$, at least $\frac{n}{2}$ vertices remain. $\square$

**Lemma 6.5.** If $T$ is a spanning tree of $G_{n,2}$ and $1 \leq \alpha \leq \frac{n}{16}$, then there are at least $\frac{n^2}{2d \alpha}$ edges $e$ such that $\text{path}(T, e) > \frac{n}{16}$. 

**Proof.** Since the grid has maximum degree 4, every subtree on $m > 1$ vertices in it (not necessarily spanning) has an edge whose deletion separates it into two smaller subtrees, each having at least $\frac{m-1}{4} \geq m/8$ 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 $\lceil \frac{n^2}{8d \alpha} \rceil - 1$ edges have been deleted and the number of pieces is $\lceil \frac{n^2}{8d \alpha} \rceil$. On the average, the pieces have about $8\alpha^2$ vertices, with the smallest piece having at least $\alpha^2$ vertices. To see this, note that at every stage, the largest piece has at most 8 times as many vertices as the smallest piece, because whenever there is a new smallest size it is at least $\frac{1}{8}$ of the old largest size. Minimizing $x_1$ subject to $x_1 \leq \cdots \leq x_m \leq 8x_1$, and $\sum x_i = M$, yields $x_1 = \frac{M}{8m-3}$, achieved when all the other variables are 8 times the smallest. Note also that the largest piece must have fewer than half the vertices as soon as more than 16 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. (Since otherwise the number of deleted edges would be too large.) 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 $\frac{n}{8}$ 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 $\text{path}(T, e) > \frac{n}{16}$, 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 one of the vertices in $B$. Harvesting at least $\frac{\alpha}{2}$ endpoints of such edges from each of at least $\frac{n^2}{2d \alpha}$ pieces of $T$ (i.e., those pieces incident with at most 4 deleted edges), yields at least $\frac{n^2}{8d \alpha}$ such edges, since we might have counted each edge twice. (Of course, $\text{path}(T, e) > \frac{n}{8}$ for any edge we have counted twice, but we are not trying to optimize the constants here.) $\square$

We are now ready to prove our lower bound.

**Theorem 6.6.** $S_{opt}(G_{n,2}) \geq \frac{1}{2048} \ln n - O(1)$.

**Proof.** Given an arbitrary spanning tree $T$ of $G_{n,2}$, the preceding lemma guarantees at least $\frac{n^2}{32 \alpha}$ edges with $\text{cost}^*$ at least $\frac{n}{16}$ for any $\alpha \leq \frac{n}{16}$. We now choose an edge $e$ at random, defining a random variable $X$ by $X = \text{cost}^*(T, e)$. We seek a lower bound on $E[X]$, the average $\text{cost}^*$ of an edge, since $S_{opt}(G_{n,2}) = E[X]$. We use the fact that, for a discrete random variable $X$ attaining nonnegative integer values, we have $E[X] = \sum_{k \geq 1} \text{Prob}(X \geq k)$. For $k \leq \frac{n}{256}$, let $\alpha = 16k$. Then

$$\text{Prob}(X \geq k) \geq \frac{n^2}{1024k \cdot 2n(n-1)} > \frac{1}{2048}.$$ 

We thus obtain

$$E[X] > \frac{1}{2048} \sum_{k=1}^{\lfloor n/256 \rfloor} \frac{1}{k} \geq \frac{1}{2048} \ln n - O(1). \quad \square$$

**6.3. Lower bounds for d-dimensional grids.** More generally, we prove that Theorem 6.1 is optimal, up to a constant factor, for all $n, d \geq 2$.

**Theorem 6.7.** There exists an absolute constant $c > 0$ such that for every $n, d \geq 2$, $S_{opt}(G_{n,d}) \geq cd \log n = c \log N$. 

Downloaded 07/23/18 to 130.126.108.125. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php
The proof of Theorem 6.7 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 at least one of them does exceed $10^5$, i.e.,

\[
\max\{n, d\} \geq 10^5.
\]

Suppose $n, d \geq 2$ and let $G = G_{n,d} = (V, E)$. Denote $M = \{1, 2, \ldots, n\}$. Recall that the vertices of $G$ are naturally represented by all vectors of length $d$ 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. In the course of the proof we need to consider vectors whose coordinates except one are all determined. It is convenient to denote a nondetermined coordinate by the symbol $\ast$. Thus, for a subset $A \subset V$ and for a vector $v = (v_1, \ldots, v_d)$ with $v_i = \ast$ and $v_j \in M$ for all $j \neq i$, 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 $\{1, \ldots, d\} \setminus \{i\}$ 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 $uw$ of $G$ in direction $i$ which joins a vertex $u$ of $A$ with a vertex $w$ of $V \setminus A$, with the property that 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 $n$. See also [A1].)

**Lemma 6.8 [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 $F$ be a family of vectors of length $d$ whose coordinates, indexed by the elements of $D$, lie in $M \cap D$. For each $i, 1 \leq i \leq m$, let $F_i$ denote the set of all projections of the vectors in $F$ on $B_i$. Then

\[
|F|^k \leq \prod_{i=1}^{m} |F_i|.
\]

Returning to the graph $G = (V, E)$ we now prove the following lemma.

**Lemma 6.9.** 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 $v$, where $v_i = \ast$ and $v_j \in M$ for all $j \neq i$. Let $A'_i$ denote the set of all vectors $v$ as above such that $A$ properly intersects $v$. Then

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

and

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

**Proof.** Apply Lemma 6.8 to the grid $G_{n,d} = (V, E)$, where $D$ is the set of $d$ coordinates in the vectors of $V$. Specifically, set $F = A$, $m = d$, $k = d - 1$ and $B_i = D \setminus \{i\}$. Clearly in this case $|F_i| = |A_i|$ and hence, by the lemma

\[
x^{d(d-1)} = |A|^{d-1} \leq \prod_{i=1}^{d} |A_i| \leq \left(\frac{1}{d} \sum_{i=1}^{d} |A_i|\right)^d.
\]

This implies the first part of Lemma 6.9.
To prove the second part, observe that since $|A_i| = x^d$, $A$ cannot intersect without properly intersecting more than $x^d/n$ vectors $v$ whose $i$th coordinate is $\ast$. Thus $|A_i'\setminus A_i| \leq x^d/n$ for all $i$, and summation over $1 \leq i \leq d$ completes the proof of the lemma.

Let $v$ be a vertex of $G$, let $y$ be a positive real, and suppose $i \in D = \{1, \ldots, d\}$. 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 $D \setminus \{i\}$. 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 6.10.** For all $y > 0$

$$L(y, d) \leq 2^{\min\{d-1, y(d-1)\}} \left(\frac{d - 1 + (d - 1)y}{d - 1}\right).$$

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 $e = 2.71828\ldots$ is the basis of the natural logarithm.

**Proof.** Suppose $v = (v_1, \ldots, v_d)$ and let $F$ be a set of vertices of $G$ whose distance from $v$ is at most $y(d-1)$. Suppose, further, that no two members of $F$ have the same projection on $D \setminus \{i\}$. Clearly, it suffices to show that the cardinality of $F$ is at most the right-hand side of (13). If $u = (u_1, \ldots, u_d)$ is a member of $F$ then, for each $j \neq i$, $u_j = v_j + \epsilon_j$, where $\sum_{j \neq i} |\epsilon_j| < y(d-1)$. The number of ways to choose the signs of all the $\epsilon_j$-s which are not $0$ is at most $2^{\min\{d-1, y(d-1)\}}$, since there are at most $\min\{d-1, y(d-1)\}$ such $\epsilon_j$. The number of ways to choose the absolute values of the $\epsilon_j$-s is less than the number of ways to write $y(d-1)$ as an ordered sum of $d$ nonnegative integers (the first $d - 1$ of which will serve as our numbers $\epsilon_j$), and this is precisely the binomial coefficient $\binom{d-1+(d-1)y}{d-1}$. This completes the proof of (13). The other two estimates follow from this one by using the fact that $\binom{a}{b} \leq (ea/b)^b$ for all integers $a$ and $b$ and the fact that $(1 + \frac{1}{2})^y \leq e$. 

We can now return to the proof of Theorem 6.7. 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 $\frac{x^d}{2}$ 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 $4d$.) It follows that the number $t$ of connected components, and hence the number of deleted edges of $T$, does not exceed $\frac{4dn^d}{x^d}$. Let $U$ denote the set of all end vertices of these deleted edges. Then $|U| \leq \frac{8dn^d}{x^d}$.

Let $x_j$ denote the positive $d$th root of the number of vertices in the $j$th connected component, ($1 \leq j \leq t$). Since $x_j \leq x \leq 0.75n$ for each $j$, Lemma 6.9 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 an endpoint of such an edge is at distance at least $y(d-1)$ from all the vertices in $U$ that lie in the same connected component as that endpoint then the length of the elementary cycle corresponding to this edge is greater than $y(d-1)$. This is because the elementary cycle has to contain a vertex in $U$. 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 6.9, we conclude that there are at least \( |A'_i| \) vertices in \( C_j \) whose projections on \( D \setminus \{i\} \) 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 \( t \) connected components and over all \( d \) directions we conclude that the number of edges that join distinct components whose elementary cycles have length that exceeds \( y(d-1) \) is at least

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

Since \( |U| \leq 8dn^d/x^d \) and since the minimum possible value of \( \sum_{j=1}^{t} x_{j}^{d-1} \) subject to the constraint that \( x \geq x_j \geq 0 \) and \( \sum_{j=1}^{t} x_{j}^{d} = n^{d} \) is \( n^{d}/x^d = n^{d}/x \) we obtain the following lemma.

**Lemma 6.11.** For each spanning tree \( T \) in \( G = G_{n,d} \) and for each real \( x, 1.5 \leq x \leq 0.75n \) and positive real \( y \), the number of edges of \( G \) whose elementary cycles have length that exceeds \( y(d-1) \) is at least

\[
\frac{dn^d}{8x} - L(y, d)8d^2n^d/x^d.
\]

We can now prove Theorem 6.7. For technical reasons it is convenient to consider three possible cases, depending on the values of the parameters \( n \) and \( d \).

**Case I.** \( n \leq 80 \).

In this case \( d \geq 10^5 \), by (12). Put \( x = 1.5, y = 1/20 \). By Lemma 6.10, \( L(y, d) \leq (2e^2)^0.05(d-1) < 1.2^{d-1} \). Hence (since \( x = 1.5 \))

\[
L(y, d)8d^2n^d/x^d < \left( \frac{12}{15} \right)^{d-1} \frac{8d}{dn} \frac{dn^d}{x} < \frac{dn^d}{16x}.
\]

Therefore, by Lemma 6.11, there are at least \( \frac{dn^d}{16x} = \frac{1}{24}dn^d \) edges whose cycles have length that exceeds \( \frac{1}{20}(d-1) \), and since the total number of edges is \( 2dn^d - n - 1 < dn^d \) and the tree \( T \) was arbitrary we conclude that in this case \( S_{opt}(G_{n,d}) \geq \frac{1}{480}(d-1) > cd \log n \), as needed.

**Case II.** \( d \leq 20 \).

In this case, which requires a little more work, \( n \geq 10^5 \), by (12). For each real \( x \) satisfying

\[
1600 \leq x \leq \frac{3}{4}n,
\]

define \( y = x/1600 \). By Lemma 6.10, \( L(y, d) \leq (2e(y + 1))^{d-1} < \frac{x^{d-1}}{160^d} \) and hence

\[
L(y, d)8d^2n^d/x^d \leq \frac{8d}{160^{d-1}} \frac{dn^d}{x} \leq \frac{dn^d}{10x^d},
\]

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

It follows from Lemma 6.11 that at least a fraction \( 1/(40x) \) of the edges have cycles of length more than \( x(d-1)/1600 \), for each \( x \) satisfying (14). 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 (14), the probability \( \text{Prob}(X \geq x/1600) \) is at least \( 1/(40x) \). Therefore, by defining \( z = x/1600 \) we conclude that for each
integer $z$ satisfying $1 \leq z \leq \frac{3}{6400} n$, $\text{Prob}(X \geq z) \geq 1/(64000z)$. Thus the expectation of $X$ is at least

$$\sum_{z=1}^{[3n/6400]} \text{Prob}(X \geq z) \geq \sum_{z=1}^{[3n/6400]} \frac{1}{64000z} \geq 10^{-6} \log n.$$  

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 $S_{\text{opt}}(G_{n,d}) \geq 10^{-6}(d - 1) \log n > cd \log n$, 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 6.10, $L(y, d) \leq \frac{x^{d-1}}{2d^2}$. 

and hence

$$L(y, d) \leq \frac{8d^2}{x^d} \leq \frac{8d}{2^{d-1}} \frac{dn^d}{x^d} < \frac{dn^d}{16x}.$$ 

Therefore, by Lemma 6.11, for every $x$, $30 \leq x \leq 0.75n$, with probability at least $1/(16x)$, the random variable $X$ defined in the previous case exceeds $(x - 4e)/(4e)$. As before (by letting $z = \lceil(x - 4e)/(4e) \rceil$ take integer values) it follows that the expectation of this random variable is at least

$$\sum_{z=\lceil30/(4e)\rceil}^{[3n/(16e)-1]} \frac{1}{64e(z + 1)} > 10^{-6} \log n,$$

and this implies that in this case too $S_{\text{opt}}(G_{n,d}) \geq cd \log n$ and completes the proof of Theorem 6.7. 

**REFERENCES**  


