SIAM J. COMPUT. © 1998 Society for Industrial and Applied Mathematics
Vol. 28, No. 2, pp. 447–462

COMPETITIVE ALGORITHMS FOR LAYERED GRAPH TRAVERSAL[*]

AMOS FIAT[†], DEAN P. FOSTER[‡], HOWARD KARLOFF[§], YUVAL RABANI[¶], YIFTACH RAVID[‖], AND SUNDAR VISHWANATHAN[**]

Abstract. A layered graph is a connected graph whose vertices are partitioned into sets $L_0 = \{s\}, L_1, L_2, ...$, and whose edges, which have nonnegative integral weights, run between consecutive layers. Its width is $\max\{|L_i|\}$. In the on-line layered graph traversal problem, a searcher starts at s in a layered graph of unknown width and tries to reach a target vertex t; however, the vertices in layer i and the edges between layers $i-1$ and i are only revealed when the searcher reaches layer $i-1$.

We give upper and lower bounds on the competitive ratio of layered graph traversal algorithms. We give a deterministic on-line algorithm which is $O(9^w)$-competitive on width-w graphs and prove that for no w can a deterministic on-line algorithm have a competitive ratio better than 2^{w-2} on width-w graphs. We prove that for all w, $w/2$ is a lower bound on the competitive ratio of any randomized on-line layered graph traversal algorithm. For traversing layered graphs consisting of w disjoint paths tied together at a common source, we give a randomized on-line algorithm with a competitive ratio of $O(\log w)$ and prove that this is optimal up to a constant factor.

Key words. competitive analysis, layered graphs, search strategies

AMS subject classifications. 68Q10, 68Q25

PII. S0097539795279943

1. Introduction.

Finding the shortest path in a graph from a source to a target is a well-studied problem. Dijkstra's algorithm [Dij] appeared in 1959. Other algorithms can be found in [Bel, Flo, FF, AMOT].

Baeza-Yates, Culberson, and Rawlins [BCR] and Papadimitriou and Yannakakis [PY] consider a large family of shortest path problems that operate with incomplete information. They describe algorithms that start at a source, search for the target, and learn about the environment as they progress. The complexity measure associated with such an algorithm is the ratio of the total distance traversed by the algorithm to the length of the shortest source–target path. Related work on exploring graphs with incomplete information is considered in [DP].

[*]Received by the editors January 13, 1995; accepted for publication (in revised form) December 23, 1996; published electronically July 28, 1998.
http://www.siam.org/journals/sicomp/28-2/27994.html

[†]Computer Science Department, School of Mathematics, Tel-Aviv University, Tel-Aviv 69978, Israel (fiat@math.tau.ac.il).

[‡]Department of Statistics, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104-6302 (foster@hellspark.wharton.upenn.edu). This work was done while the author was at the Graduate School of Business, University of Chicago.

[§]College of Computing, Georgia Tech, Atlanta, GA 30332-0280 (howard@cc.gatech.edu). This author was supported in part by NSF grant CCR-8807534. This work was done while the author was at the Department of Computer Science, University of Chicago.

[¶]Computer Science Department, The Technion, Haifa 32000, Israel (rabani@cs.technion.ac.il). This work was done while the author was a graduate student at the Computer Science Department, Tel Aviv University.

[‖]IBM Haifa Research Laboratory–Tel Aviv Annex, IBM House, 2 Weizmann St., Tel Aviv 61336, Israel (yiftach@haifa.vnet.ibm.com). This work was done while the author was a graduate student at the Computer Science Department, Tel Aviv University.

[**]Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, India 400076 (sundar@cse.iitb.ernet.in). This author was supported in part by NSF grants CCR-8710078 and CCR-8906799. This work was done while the author was a graduate student at the Department of Computer Science, University of Chicago.

This measure is closely related to the concept of *competitive analysis*, introduced by Sleator and Tarjan [ST], which gives a worst case complexity measure for on-line algorithms. An *on-line algorithm* is an algorithm which must deal with a sequence of events, responding to events in real time without knowing what the future holds. The *competitive ratio* of an on-line algorithm A is defined as the supremum, over all sequences of events σ, and all possible (on- or off-line) algorithms ADV, of the ratio between the cost associated with A to deal with σ and the cost associated with ADV to deal with σ. We say that A is c-competitive if this supremum is at most c. (In some of the on-line literature, especially that dealing with paging and the k-server problem, from the cost of A on σ a constant additive term is subtracted, before dividing by the cost of ADV on σ. Where ambiguity might arise, we shall say that A is *strictly* c-competitive, meaning that the definition without an additive term is used.)

The *layered graph traversal problem* was introduced in [PY] and generalizes work of [BCR]. A *layered graph* is a connected graph in which the vertices are partitioned into sets $L_0 = \{s\}, L_1, L_2, L_3, ...$ and all edges run between L_{i-1} and L_i for some i. Each edge has a nonnegative integral weight. Vertex s is known as the *source*. Let $w = \max\{|L_i|\}$; w is called the *width* of the graph. An on-line layered graph traversal algorithm starts at the source and, without knowing w, moves along the edges of the graph, paying a cost equal to the weight of the edge traversed. Its goal is to reach the vertex t in the last layer known as the "target"; which vertex is the target is not revealed until the searcher occupies a vertex in the last layer.

Edges can be traversed in either direction, but the on-line algorithm pays whenever it crosses the edge. The edges between L_{i-1} and L_i, and their lengths, become known only when a node in L_{i-1} is reached.

We define the competitive ratio of a layered graph traversal algorithm to be the worst case ratio between the total distance traveled by the on-line algorithm and the length of the shortest source–target path. (If the algorithm is randomized, we use the expected distance it travels.) The competitive ratio of a layered graph traversal algorithm is given as a function of the width w.

A layered graph is said to consist of w *disjoint paths* if it is formed from w paths which are vertex disjoint except that each contains the common source. [BCR] gave optimal deterministic algorithms for all w with a competitive ratio which is asymptotic to $2ew$.

For arbitrary layered graphs, [PY] gave an optimal algorithm for width 2, with a competitive ratio of 9. It follows from [BCR] that $1 + 2w(1 + \frac{1}{w-1})^{w-1} \sim 2ew$ is a lower bound on the competitive ratio. Prior to this paper no other bounds were known.

Section 2 proves that general layered graphs of width w weighted with arbitrary nonnegative integers are no more difficult to traverse than width-w layered *trees* whose weights are $0-1$. Notice that if we know a lower bound on the smallest nonzero weight of an edge, then we can express the weights as multiples of this lower bound and round to the closest integer, thereby converting the problem with arbitrary nonnegative weights to one with integer weights. The competitive ratio is affected by at most a constant factor due to this conversion. This factor can be made arbitrarily close to one by taking the lower bound arbitrarily close to zero.

In sections 3 and 4 we give upper and lower bounds, exponential in w, on the competitive ratio for deterministic layered graph traversal.

- Section 3 gives an algorithm which attains a competitive ratio of $O(9^w)$ on layered graphs of width w. This algorithm does not need to know w in advance and automatically adjusts itself to deal with the real width on hand.

- Section 4 proves that for all w, 2^{w-2} is a lower bound on the competitive ratio of any deterministic on-line layered graph traversal algorithm.

Thus arbitrary layered graphs are much harder to traverse than those consisting of disjoint paths.

Randomized on-line algorithms are addressed in several papers including [BLS, RS, CDRS, FKLMSY, BBKTW, KRR]. An oblivious adversary is one who constructs the sequence of events in advance and deals with the sequence optimally. For this adversary model [BLS] and [FKLMSY] give examples where randomization can improve the competitive ratio exponentially. This adversary models a world in which the on-line algorithm's actions do not themselves influence future events. One can consider a situation where the on-line algorithm's actions have a direct influence on the future. In such cases [BBKTW] showed that randomization cannot improve the competitive ratio more than polynomially. We deal with randomized layered graph traversal algorithms (assuming an oblivious adversary), and present the following results.

- Section 5 gives a randomized on-line algorithm for the disjoint path traversal problem. The competitive ratio is $O(\log w)$. We also show that this is optimal up to a constant factor. This is an exponential improvement over the bound for deterministic algorithms. This result immediately gives a randomized min operator [FRR] for on-line k-server algorithms: given a set of w possibly conflicting on-line strategies, a new on-line strategy can be devised which is no worse than $O(\log w)$ times the best of these strategies on every input.
- Section 6 gives a lower bound of $w/2$ on the competitive ratio of any randomized traversal algorithm for general layered graphs.

The problem of traversing layered graphs generalizes numerous on-line problems. For instance, metrical task systems (see [BLS]) can be modeled as layered graphs where layers represent tasks, and in each layer there is a node for each possible state. The k-server problem (see [MMS]), viewed in the servers' configuration space, is the problem of traversing the layered graph of permitted configurations for each request. Unfortunately, the width of this graph depends on the cardinality of the metric space, and not just on the number of servers, so layered graph techniques are inadequate for producing solutions to the k-server problem directly. However, the algorithm given in [BCR] for traversing layered graphs consisting of disjoint paths was used by [FRR] in their construction of competitive k-server algorithms.

As an additional example of the power of layered graph traversal as a tool for designing on-line algorithms, consider the problem of metrical service systems, suggested by [CL]. A single server moving among points of a metric space is presented with requests. Each request is a set of at most w points. One of these points is then selected by the on-line algorithm, and the server is moved to that point; the cost is the distance moved. [CL] gave a competitive metrical service system algorithm for uniform metric spaces and deterministic and randomized algorithms for all metric spaces for the case of $w = 2$. Note that the k-server problem can be reduced to the metrical service systems problem in the configuration space. Section 7 shows that the metrical service systems problem with requests of size w (in metric spaces with integral distances) is equivalent to the width-w layered graph traversal problem, when w is known in advance, in that a c_w-competitive algorithm exists for one problem if and only if one exists for the other. Related recent work appears in [FL].

2. Trees are sufficient.

We first prove that given a competitive on-line algorithm for traversing width-w layered trees, in which each edge has a $0-1$ weight and each nonsource vertex has a neighbor in the previous layer, one can construct an on-line algorithm, with the same competitive ratio, for traversing arbitrary width-w layered graphs.

Definition 1. *Let H be any layered graph with source s, and let v be a vertex in H in, say, layer L_j. Define H_v to be a shortest $s-v$ path in H which contains no vertex of $L_{j+1} \cup L_{j+2} \cup L_{j+3} \cup \cdots$ (if such a path exists).*

Let G be a layered graph of width at most w with nonnegative integral edge weights and with source s. We start by proving that an on-line algorithm traversing G can construct, on the fly, a layered tree T with the following properties.

1. A vertex v is in T's ith layer if and only if v is in G's ith layer and G_v exists.
2. For all v, the length of T_v is at most the length of G_v (if G_v exists).
3. Each nonsource vertex in T has exactly one neighbor in the previous layer. (We call such a tree *rooted*.)

Furthermore, any on-line traversal algorithm for T can be simulated on G without increasing the cost.

The tree $T = T(G)$ is defined by induction on the layer index i, starting from a one-node graph ($i = 0$). Let $i > 0$. For every v in G's ith layer L_i for which G_v exists, one vertex and one edge are added to T as follows. Let $u_0 = s$ and let $G_v = \langle u_0, u_1, u_2, \ldots, u_\ell, v\rangle$. Let u_k be the first vertex in G_v which is in layer L_{i-1}. Add to T vertex v and edge (u_k, v) with weight equal to the weight of the portion of G_v between u_k and v.

Lemma 2. *For all v, the length of T_v is at most the length of G_v.*

Proof. The proof by induction on the index of the layer containing v.

Basis: $i = 0$. Trivial.

Inductive Step: $i > 0$. Assume correctness for $i-1$. Suppose that v is adjacent in T to u_k in T's $i-1$st layer. In path G_v, let a be the length of the prefix from s to u_k and let b be the length of the $u_k - v$ suffix. The length of T_v equals the length of T_{u_k} plus b. By the inductive hypothesis, the length of path T_{u_k} is at most the length of G_{u_k}, which is itself at most a. Therefore, the length of T_v is at most $a + b$, the length of G_v. ☐

Given an algorithm $\mathcal{A}$ to traverse T, we show how to traverse G without increasing the cost. Suppose that $\mathcal{A}$ moves in T from u in layer $i-1$ to v in layer i. The weight of the edge traversed in T is the length of a portion of G_v in G. This portion avoids layers $i+1, i+2, \ldots$, so the G-traversal algorithm can follow it. Similarly, if $\mathcal{A}$ moves from v in layer i to u in layer $i-1$, the G-traversal algorithm can traverse backward the corresponding portion of G_v.

A layered tree with arbitrary nonnegative integral weights can be converted to a layered tree with $0-1$ weights by inserting additional intermediate layers, on the fly.

3. A deterministic algorithm.

Without loss of generality, we may assume that the original problem asks for a traversal algorithm for $0-1$, rooted, layered trees of arbitrary width, each having a target. Instead, for each w we will build a traversal algorithm A_w that maintains the following property. For each $0-1$ rooted tree T of width at most w without a target, for each i, the cost incurred by A_w on T between the start and the time it visits its first layer-i vertex is at most $8 \cdot 9^w$ times the length of a shortest path between s and any vertex of L_i.

We can easily solve the original problem via algorithms $A_1, A_2, \ldots$. We need only run A_j, starting with $j = 1$, until the width exceeds j, or until we reach some vertex in the same layer as the target. If, including the newly revealed layer, the width is $k > j$, we backtrack to the source and execute procedure A_k, starting at the source, forgetting everything we know about the graph. As soon as we learn that the layer we occupy contains the target, we backtrack to the source and then travel optimally to t. The total cost incurred by this algorithm on a width-w graph whose shortest source−target path is of length d is bounded by

$$d[8 \cdot 9^1 + 8 \cdot 9^2 + \cdots + 8 \cdot 9^w + (8 \cdot 9^w + 1)].$$

This is $O(9^w)$ times the source−target distance.

In order to define algorithms A_w, we need some terminology.

1. We refer to the time just after layer t and the edges from layer $t-1$ to t have been revealed as *time* t. The algorithm must move to a vertex in layer t after time t and before time $t+1$.

2. Vertex v is *active* at time t if it has a descendant in layer t. At time t, vertices in layer t are called *active leaves*.

3. At time t, $SP(v)$ denotes the length of the shortest path from v to a descendant of v in layer t (if v is active at time t).

Now we construct the algorithms. A_1 is the obvious algorithm. A_w for $w > 1$ is constructed from $A_1, A_2, A_3, \ldots, A_{w-1}$ as follows. Its execution is divided into phases. Within each phase, a vertex r, initially the source, is designated as the root for that entire phase. We will maintain the invariant that every path from the source to an active leaf passes through the root r. The searcher occupies r at the start of the phase. Furthermore, an integer d is fixed for the entire duration of the phase.

To start a phase, we let $d = SP(r)$. If $d = 0$, the searcher moves along length-0 edges from r, visiting all descendents of r at distance 0 from r (using, say DFS), then returning back to r, all at no cost.

At this point, $d = SP(r) \geq 1$ is fixed for the phase, and the searcher occupies r. If y is a descendant of x, let $d(x, y)$ denote the length of the unique $x - y$ path. At all times, let $S = \{s | s$ is an active descendant of r, $d(r, s) = d$, s's parent u satisfies $d(r, u) = d - 1$, and $SP(s) < d\}$. (A function of time, S may change many times within a phase to reflect its definition; however, d is defined once at the beginning of a phase and remains constant.) Because some active leaf is at distance exactly d from r at the start of a phase, $S \neq \emptyset$ at that time. Because the active leaf descendants of different $s \in S$ are distinct, $|S| \leq w$ always.

Let S_t denote the set S at time t. A phase ends as soon as either (1) there is an $x \in S_t$ such that at time t, x has w active leaf descendants, or (2) $S_t = \emptyset$. If either (1) or (2) occurs, the current phase ends at time $t-1$, and a new phase, possibly with a new root, begins immediately afterward.

Each phase is divided into subphases. The start of a phase marks the beginning of its first subphase. A new subphase begins at a later time t if S_t is strictly smaller than S_{t-1}. (A phase may end in the middle of a subphase.) At the start of a subphase the searcher occupies the root r. He chooses an arbitrary $s \in S$ and at a cost of d moves from r to s. Where $z = |S|$, if $z = 1$, then the searcher executes procedure A_{w-1} with s as the root, and if $z \geq 2$, he executes procedure $A_{w-(z-1)}$ with s as the root.

When the subphase terminates, the searcher retraces all of his steps within that subphase back to r. This ensures that the searcher occupies r at the beginning of the next subphase.

If a phase terminates because of termination condition (1), i.e., there is an $x \in S_t$ such that the tree rooted at x has w active leaves, then $S_t = \{x\}$. In this case the searcher moves from r to x, a distance of d, and makes x the root for the next phase. If a phase terminates because of termination condition (2), i.e., $S_t = \emptyset$, the root remains the same vertex r. Notice that in this case, $SP(r)$ increased during the phase by at least d, so the next phase will begin with the new d at least double its value in the previous phase. This concludes the definition of A_w.

Analysis.

We state four easily proven facts.

FACT 3. *If $z = |S|$ at the beginning of a subphase which starts at s, then throughout that subphase the width of the subtree rooted at s is at most $w - (z - 1)$.*

Proof. At any time during the subphase, each vertex in S has at least one active leaf as a descendant. Since $|S - \{s\}|$ equals $z - 1$ during the subphase, s can have at most $w - (z - 1)$ active leaf descendants at any time, and therefore the width of the subtree rooted at s cannot exceed $w - (z - 1)$. □

FACT 4. *Within one phase, algorithm A_{w-1} is executed at most twice. For $i < w-1$, A_i is executed at most once within a phase. An invocation of A_i $(1 \leq i \leq w-1)$ starting at vertex s terminates with $SP(s) \leq d$.*

Proof. For a given z, only one recursive call is made while $|S| = z$. For $z \leq 2$, A_w calls A_{w-1}. A_i for $i < w - 1$ can be called by A_w only if $z = w - i + 1$. As soon as $SP(s) \geq d$, s is evicted from S and the subphase terminates (if not before). □

FACT 5. *If a phase ends because of phase termination condition* (1)*, i.e., there is an $x \in S$ such that the tree rooted at x has w active leaves, then the new root x satisfies $d(source, x) = d(source, r) + d$, and, at the phase end, every source–active leaf path passes through x.*

Proof. Since $x \in S$ implies that x is a descendant of r satisfying $d(r, x) = d$, clearly $d(source, x) = d(source, r) + d$. And if the tree rooted at x has w active leaves when a phase ends, the width bound of w implies that from that time onward every source–leaf path contains x. □

FACT 6. *If condition* (2) *triggers the end of a phase, then the length of a shortest path from the source to an active leaf is at least d greater at the end of the phase than at the end of the previous phase.*

Proof. When the phase starts, $SP(r) = d$. If $S = \emptyset$ at the phase end, then every vertex originally in S has been evicted from S. All vertices in S at the beginning of the phase evicted by reason of inactivity are inactive at the end of the phase.

If y is any active leaf at the phase end, on the $r - y$ path there must be a vertex x closest to r such that $d(r, x) = d$. The only possible reason why this active vertex is not in S at the end of the phase is that $SP(x) \geq d$ at the end. Therefore, $d(r, y) = d(r, x) + d(x, y) \geq d + d = 2d$ and $SP(r) \geq 2d$ at the end of the phase. □

THEOREM 7. *For each w, for each rooted $0 - 1$ tree T of width at most w, the cost incurred by A_w on T is at most $8 \cdot 9^w$ times the length of a shortest path from the source to a vertex in the highest-numbered layer.*

Proof. We prove the statement by induction on w. For $w = 1$ the statement is clear.

Let $w > 1$. At the start of a phase rooted at, say, r, the searcher occupies r. It incurs no cost until every path from r to an active leaf has positive cost. Moving from r to the designated s costs d. Within a subphase, let z denote $|S|$ at the beginning of the subphase. If $z \geq 2$, algorithm $A_{w-(z-1)}$ is invoked, and by Fact 3 the width of the tree on which $A_{w-(z-1)}$ is invoked does not exceed $w - (z - 1)$ during the subphase. A_{w-1}

is invoked if $z = 1$, but the width cannot exceed $w - 1$ during the subphase—for if it did, the tree rooted at s would have w active leaves and phase termination condition (1) would hold, thereby aborting the current phase (and subphase). Furthermore, within a subphase which starts at s, $SP(s)$ cannot exceed $d - 1$. If it did, s would be evicted from S.

By the inductive hypothesis, if $z > 1$ at the start of the subphase, the cost incurred during this subphase is bounded by d (the cost of moving from r to s), plus $8 \cdot 9^{w-(z-1)}d$, plus the cost of backtracking to s and then to r, a total of at most $d + 2(8 \cdot 9^{w-(z-1)}d) + d$. If instead $z = 1$, the cost is at most $2d + 16 \cdot 9^{w-1}d$. There is an additional cost of d at the end of a phase if we move the root forward.

By Fact 4, the total cost in a phase is at most

$$d + \sum_{z=2}^{w}(2d + 16 \cdot 9^{w-(z-1)}d) + (2d + 16 \cdot 9^{w-1}d)$$

$$= (2w+1)d + 16d[(9 + 9^2 + 9^3 + \cdots + 9^{w-2} + 9^{w-1}) + 9^{w-1}]$$

$$= (2w+1)d + 16d\left[\frac{9^w - 9}{8} + 9^{w-1}\right]$$

$$< 2wd + 16d\left[\frac{17}{72} \cdot 9^w\right]$$

$$= d\left[2w + \frac{34}{9} \cdot 9^w\right]$$

$$\leq d\left[\frac{2}{9} \cdot 9^w + \frac{34}{9} \cdot 9^w\right] = 4d \cdot 9^w.$$

Suppose v is of minimum distance from the root among those vertices in the jth and final layer. For the analysis alone, add w dummy children to v via length-0 edges. At time $i+1$, v has w active leaf descendants. Thus either $d = 0$ in the current phase, or one vertex $x \in S$ has w active leaf descendants. Hence either $d = 0$, or a phase ends at time j and x becomes the new root. In either case, we can study the cost incurred during *complete* phases.

At all times, define Φ to be the distance from the source to the current root r. Define Ψ to be the length of a shortest path from the source to an active leaf; $\Psi = \Phi + SP(r)$. In a phase, either Φ increases by d, if (1) terminated the phase, or if (2) ended the phase, Ψ increases by at least d. Thus $\Phi + \Psi$ increases within a phase by at least d, and neither Φ nor Ψ ever decreases. It follows that the cost incurred by A_w to visit some vertex in L_i is at most $4 \cdot 9^w$ times the final value of $\Phi + \Psi$, which is at most twice the final value of Ψ. Therefore, A_w is $8 \cdot 9^w$-competitive. $\square$

4. A lower bound for deterministic algorithms. Fix a competitive deterministic layered graph algorithm A for arbitrary layered graphs. A traces out a path in each layered graph. We construct a layered tree that forces A to perform poorly. Figure 1 illustrates the lower bound construction. The construction is recursive. The

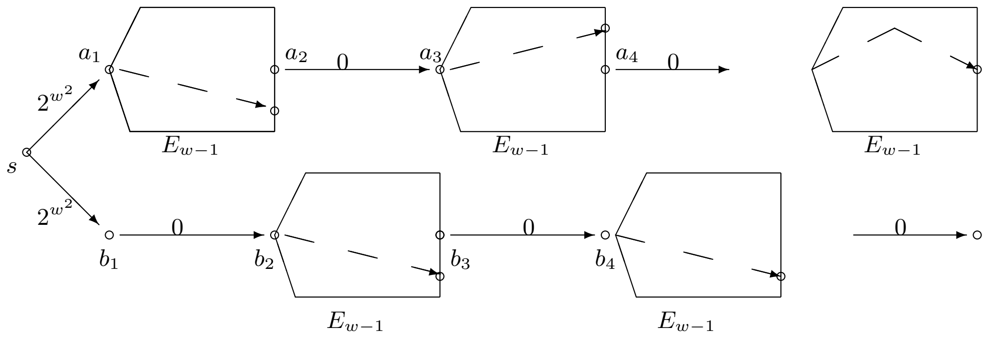

FIG. 1. *Deterministic lower bound.*

idea is that A is forced to move back and forth between the two subtrees attached to the source s, thus incurring a large cost compared with the shortest path to the target.

DEFINITION 8. *Let H be a layered tree. Suppose that $v \in L_{i-1} \neq \emptyset$ is the vertex visited by A at time i.*

1. *Define $T(v)$ to be the minimum $j > i$, if any, such that A visits a nondescendant of v at time j.*

2. *Define $L(v)$ to be the length of a shortest path from v to a descendant of v in layer $T(v)$ (if $T(v)$ and any descendants in layer $T(v)$ exist).*

3. *Define $C(v)$ to be the cost incurred by A from the time when v is first visited until the path traced out by A first exits the subtree rooted at v (if ever). This is exactly the cost incurred by A at times $i+1, i+2, ..., T(v)-1$, plus the portion of the cost incurred at time $T(v)$ attributable to edges in the subgraph rooted at v.*

LEMMA 9. *Let $w \geq 1$. Let H be a layered tree of height i, say, and arbitrary width, with at least two vertices in the ith layer, and let s be the leaf in layer i visited by A. Then there is an infinite rooted tree E_w of width at most w with these properties.*

1. *The root of E_w has $\min\{2, w\}$ children. The edge(s) out of the root are of length 2^{w^2}.*

2. *If E_w is attached to vertex s and an infinite path of length 0 is attached to all other vertices in the ith layer of H, then for this new infinite tree, $L(s)$ exists and $C(s) \geq 2^{w-1}(L(s) - 2^{w^2})$.*

Proof. The proof is by induction on w. Let $w = 1$ and let H be a tree with at least two leaves. If we attach to s an infinite path of edges of length $2^{1^2} = 2$ and attach infinite paths of length 0 to other vertices in the last layer, because A is competitive, $T(s)$ must exist. $C(s) \geq L(s) - 2$. So, clearly, $C(s) \geq 2^{1-1}(L(s) - 2^{1^2})$.

Let $w \geq 2$. Let H be a layered tree and let $s \in L_i$ be visited by A, where $L_{i+1} = \emptyset$ and $|L_i| \geq 2$. Attach to s two children a_1, b_1 via edges of length 2^{w^2}. Add to all other vertices in L_i an edge of length 0.

If A occupies neither a_1 nor b_1 at time $i+1$, then $T(s) = i+1$, $L(s) = 2^{w^2}$, and $C(s) = 0$, so clearly $C(s) \geq 2^{w-1}(L(s) - 2^{w^2})$.

So we may suppose without loss of generality that A visits a_1 at time $i+1$. By induction, there is an infinite tree E_{w-1} of width at most $w-1$ such that if a_1 is extended by E_{w-1} and all other leaves are extended by infinite paths of length 0,

$$C(a_1) \geq 2^{w-2}(L(a_1) - 2^{(w-1)^2}).$$

At time $T(a_1)$, A occupies either a descendant of b_1 or a nondescendant of s. Suppose A occupies a descendant b_2 of b_1. Choose a descendant of a_1 in layer $T(a_1)$ of minimum distance from a_1. Call it a_2. (Such a descendant exists because E_{w-1} is infinite.) "Kill" all other descendants of a_1 in layer $T(a_1)$, i.e., mark them as inactive. They will have no children. Now "truncate" the entire infinite tree to level $T(a_1)$ by removing all vertices in layers $T(a_1)+1, T(a_1)+2, T(a_1)+3, \ldots.$

By the inductive assertion we can find a new infinite tree E'_{w-1} of width at most $w-1$ so that if E'_{w-1} is attached to b_2 and all other vertices in layer $T(a_1)$ (including a_2 but no other descendants of a_1) are extended by 0-length infinite paths, $C(b_2) \geq 2^{w-2}(L(b_2) - 2^{(w-1)^2})$. Now truncate the tree to level $T(b_2)$ by eliminating all vertices in layers $T(b_2)+1, T(b_2)+2, T(b_2)+3, \ldots.$ At time $T(b_2)$, A occupies either a descendant a_3 of a_2 or a nondescendant of s. If A occupies a descendant a_3 of a_2 we attach a new infinite tree E''_{w-1} to a_3 and "kill" all descendants of b_2 in layer $T(b_2)$ except for one descendant b_3 of minimum distance from b_2.

This process continues until at some point A visits a nondescendant of s. This must happen eventually, because there is at least one infinite 0-path. Since each stage adds at least 2^{w^2} to A's cost, every competitive algorithm must eventually switch at some time $T(s)$ to a nondescendant of s.

Suppose that the algorithm has constructed $a_1, b_1, a_2, b_2, \ldots, a_k, b_k$ but neither a_{k+1} nor b_{k+1}. Thus A visits either a_k or b_k but exits the subtree rooted at s at time $T(a_k)$ or $T(b_k)$, whichever is defined.

Claim. $C(s)$ increases by at least

$$2^{w^2} + 2^{w-2}(L(a_i) - 2^{(w-1)^2}) \geq 2^{w-2}L(a_i)$$

between the time when A occupies a_i and time $T(a_i)$. Similarly, between the time when A occupies b_i and time $T(b_i)$, $C(s)$ increases by at least

$$2^{w^2} + 2^{w-2}(L(b_i) - 2^{(w-1)^2}) \geq 2^{w-2}L(b_i).$$

Proof of Claim. In moving from a_i to a nondescendant of a_i, A incurs a cost of at least 2^{w^2} on the edges out of s. On the edges in the subtree rooted at a_i, A incurs a cost of

$$C(a_i) \geq 2^{w-2}(L(a_i) - 2^{(w-1)^2})$$

by the inductive case of the theorem. The proof of the second statement is similar.

But if

$$\alpha = L(a_1) + L(a_3) + L(a_5) + \cdots$$

and

$$\beta = L(b_2) + L(b_4) + L(b_6) + \cdots,$$

then

$$L(s) = 2^{w^2} + \min\{\alpha, \beta\}.$$

Thus $C(s) \geq 2^{w-2}(\alpha + \beta) \geq 2^{w-1}\min\{\alpha, \beta\} = 2^{w-1}(L(s) - 2^{w^2})$. Now make the tree infinite, as required, by attaching infinite length-0 paths to each leaf in the final layer. □

Now we prove a lower bound of 2^{w-2} on the competitive ratio.

THEOREM 10. *If A is a layered graph traversal algorithm, then its competitive ratio on width-w graphs is at least 2^{w-2}.*

Proof. We may assume $w \geq 2$. Let s be a source with two children a_1 and b_1 via edges of length 2^{w^2}. Suppose A moves from s to a_1. As in Lemma 9, we can attach to a_1 an infinite tree E_{w-1} of width at most $w-1$ such that $C(a_1) \geq 2^{w-2}(L(a_1) - 2^{(w-1)^2})$ if b_1 is extended by an infinite path of length 0. At time $T(a_1)$, A occupies a descendant b_2 of b_1. Truncate the tree to height $T(a_1)$. Let a_2 be a descendant of a_1 in layer $T(a_1)$, of minimum distance from a_1. All descendants of a_1 in layer $T(a_1)$, other than a_2, will have no children. Now attach to b_2 an infinite tree E'_{w-1}, as in Lemma 9, and to a_2 attach an infinite length-0 path

$$C(b_2) \geq 2^{w-2}(L(b_2) - 2^{(w-1)^2}).$$

At time $T(b_2)$, A occupies a descendant a_3 of a_2. Truncate the tree to height $T(b_2)$. Let b_3 be a descendant of b_2 in layer $T(b_2)$, of minimum distance from b_2. All descendants of b_2 in layer $T(b_2)$, other than b_3, will get no children.

Repeat this process *ad infinitum*. Each pair of additions increases the length of the shortest root−active-leaf path by at least $2^{(w-1)^2}$. Eventually we reach a situation in which we have constructed $a_1, b_1, a_2, b_2, ..., a_k, b_k$ so that if

$$\alpha = L(a_1) + L(a_3) + L(a_5) + \cdots$$

and

$$\beta = L(b_2) + L(b_4) + L(b_6) + \cdots,$$

then $\min\{\alpha, \beta\} \geq 2^{w^2}$. By the claim embedded in the proof of Lemma 9, by that time A's cost is at least

$$2^{w-2}(\alpha + \beta) \geq 2^{w-1} \min\{\alpha, \beta\}.$$

The adversary's cost is

$$2^{w^2} + \min\{\alpha, \beta\} \leq 2 \min\{\alpha, \beta\}.$$

Therefore the competitive ratio is at least

$$\frac{2^{w-1} \min\{\alpha, \beta\}}{2 \min\{\alpha, \beta\}} = 2^{w-2}. \qquad \square$$

5. Disjoint paths.

Let L be a layered graph which consists of a set of disjoint paths except that they share the common source. Each edge has a $0-1$ length.

We define the algorithm in phases. At the beginning, while some path has length 0, the algorithm simply chooses such a path and follows it until, if ever, its length increases. It then switches to another path of length 0, and follows that one until its length increases. This continues until all paths have positive length. Then the first phase begins.

In the kth phase ($k = 1, 2, ...$), the length of the shortest path from the source to the current layer lies in the interval $I_k = [2^{k-1}, 2^k)$. At the start of phase k the algorithm chooses a path randomly and uniformly from among those paths of length in I_k running from the source to the current layer. It then backtracks through the source to the current layer on the chosen path, incurring a cost of at most $2 \cdot 2^k$ in the process.

Whatever path the algorithm is following in phase k, it blindly continues to follow that path until its length reaches 2^k. Whenever the length of the current path reaches 2^k, the algorithm replaces it by a path chosen randomly from those paths of length less than 2^k—if any exist—backtracking through the source and incurring a cost of at most $2 \cdot 2^k$ in the process. A new phase begins and k is incremented as soon as every path has length at least 2^k.

Analysis.
Our initial backtracking cost at the start of a phase is at most $2 \cdot 2^k$. If E_w is an upper bound on the expected number of times the algorithm switches paths within any phase, then the expected cost within phase k is at most $2^{k+1} + E_w 2^{k+1} = 2^{k+1}(1+E_w)$. Let ℓ denote the number of phases. Our total expected cost is bounded above by $(1+E_w)\sum_{k=1}^{\ell} 2^{k+1} < (1+E_w)2^{\ell+2}$. The adversary's cost is at least $2^{\ell-1}$, giving us a competitive ratio bounded by $8 + 8E_w$. We show that we can take $E_w = H_w = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{w} \sim \ln w$.

We now describe a probabilistic game which models the path selection process in a phase. Let S be a set of size n. There are two players A and B. Initially B randomly and uniformly picks one element, hiding his choice from A. At each step A chooses one element of S and removes it from S. Whenever A discards the element selected by B, B pays A \$1 and B uniformly at random picks a new item (if S is still nonempty).

We prove that the expected cost F_n incurred by B is exactly H_n. Clearly, $F_1 = 1$, and for $n \geq 2$, F_n satisfies

$$F_n = \frac{1}{n}(1 + F_{n-1}) + \frac{1}{n}(1 + F_{n-2}) + \frac{1}{n}(1 + F_{n-3}) + \cdots + \frac{1}{n}(1 + F_1) + \frac{1}{n}(1 + 0).$$

This recurrence and the fact that $F_1 = 1$ imply that $F_n = H_n$ for all n since

$$F_n = 1 + \frac{1}{n}(F_1 + F_2 + F_3 + \cdots + F_{n-1}).$$

Thus

$$nF_n = n + (F_1 + F_2 + \cdots + F_{n-1})$$

and

$$(n-1)F_{n-1} = (n-1) + (F_1 + F_2 + \cdots + F_{n-2}),$$

if $n \geq 3$, so

$$nF_n - (n-1)F_{n-1} = 1 + F_{n-1}.$$

Therefore, for $n \geq 3$, $n(F_n - F_{n-1}) = 1$ and $F_n = F_{n-1} + 1/n$. Since $F_2 = 3/2$, it follows that $F_n = H_n$ for all n.

The connection between the experiment and layered graph traversal.
A corresponds to the adversary and B corresponds to the algorithm. Each element in the set is associated with a path in the layered graph of length less than 2^k at the beginning of the kth phase. A discards an element from the set when the length of the corresponding path reaches 2^k. He pays \$1 every time this happens. The expected

number of times B backtracks is at most the expected cost to B of the game above. Thus we may take $E_w = H_w$. We have proven the following theorem.

THEOREM 11. *The competitive ratio of the randomized algorithm above for traversing disjoint paths is at most* $8 + 8H_w$.

A lower bound.

THEOREM 12. *Let* w *and* M *be any positive integers. For any randomized on-line algorithm* A *for traversing disjoint paths of width at most* w*, there is a width-*w *layered graph for which the length of the shortest source−target path is* M*, but on which* A*'s expected cost is at least* $M(2H_w - 1)$.

Proof. Each path in the width-w layered graph begins with M unit-cost edges. For a layered graph that begins this way, at time M there is at least one layer-M vertex which is occupied by the searcher with probability at least $1/w$. We give that vertex no children, but to every other layer-M vertex we give a child via a length-0 edge. At time $M + 1$, at least one of the $w - 1$ layer-$(M + 1)$ vertices is occupied by the searcher with probability at least $1/(w - 1)$. We add a length-0 edge to layer $M + 2$ from every layer-$(M + 1)$ vertex but that one. That one dies. We repeat this process for layers $M + 2, M + 3, ..., M + (w - 1)$; in layer $M + i$ there are exactly $w - i$ vertices, $i = 0, 1, 2, ..., w - 1$. The unique vertex in layer $M + w - 1$ is the target. The expected cost incurred by A is bounded below by M plus $2M$ times the sum, over each leaf in the graph other than the target, of the probability that A visits that leaf. This sum of probabilities is $\frac{1}{w} + \frac{1}{w-1} + \frac{1}{w-2} + \cdots + \frac{1}{2} = H_w - 1$. The total expected cost is hence at least $M(1 + 2(H_w - 1)) = (2H_w - 1)M$. □

6. A randomized lower bound.

Now we return to general layered graphs. Fix an integer $m \geq 2$. Let $r_w = w(1 - 1/m)$ for all w.

By induction on w, we construct for each w a probability distribution $\mathcal{G}(w)$ on a finite family of layered graphs of width w. Every graph drawn from $\mathcal{G}(w)$ has a designated vertex as the root and another as the target; the target is the unique vertex in the final layer. From the inductive construction it will be easy to verify that the following quantities depend only on w and m:

- the length L_w of the shortest root−target path in the graph,
- the sum S_w of the edge lengths,
- the number F_w of layers, excluding L_0 (the layer containing the source).

Let $E_w = 2S_wF_w$. It is clear that this is an upper bound on the distance traversed by any algorithm when it traverses any layered graph drawn from $\mathcal{G}(w)$.

Now we construct the probability distributions. See Figure 2.

Basis: $w = 1$. With probability 1 we draw a single edge (s, t) of length 1 with s the root and t the target.

Inductive Step: $w > 1$. We start with a vertex designated as the root, say s. To s we attach two edges $(s, u_1), (s, l_1)$ of length $(1/2)E_{w-1}$ each. We now construct the graph in stages. For stage 1 we draw a copy H_1 from $\mathcal{G}(w - 1)$ and attach it to u_1 (i.e., make u_1 the root of this copy). The target of H_1 we call u_2. H_1 has F_{w-1} layers of nonsource vertices in it. For these F_{w-1} layers we extend l_1 by a path of F_{w-1} length-0 edges ending at l_2. For stage 2, we extend l_2 by independently drawing a graph H_2 from $\mathcal{G}(w - 1)$, and we extend u_2 by a path of F_{w-1} length-0 edges. We continue this pattern for $N = N_w = mr_{w-1}E_{w-1}$ stages (N is an even integer). In the ith stage, for i odd, we independently select a graph H_i as in stage 1, and for i even, we choose H_i independently as in stage 2. In the last layer we have vertices

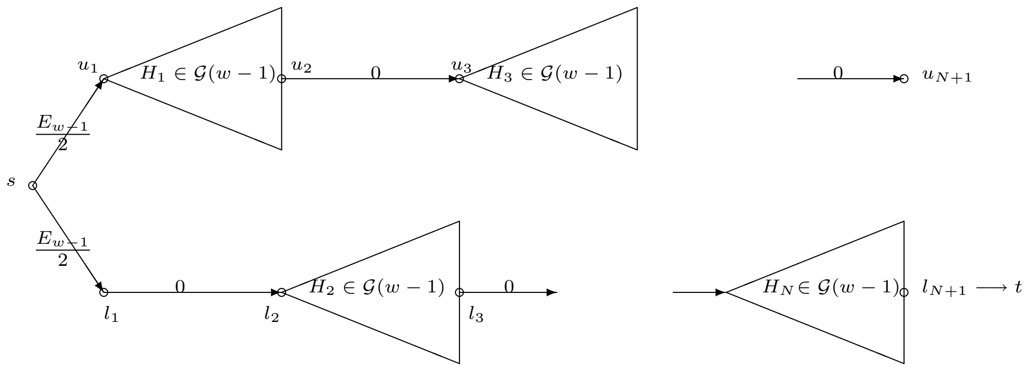
u_1
$H_1 \in \mathcal{G}(w-1)$
u_2
0
u_3
$H_3 \in \mathcal{G}(w-1)$
0
u_{N+1}
$\frac{E_{w-1}}{2}$
s
$\frac{E_{w-1}}{2}$
0
$H_2 \in \mathcal{G}(w-1)$
0
$H_N \in \mathcal{G}(w-1)$
$l_{N+1} \longrightarrow t$
l_1
l_2
l_3

FIG. 2. *Randomized lower bound.*

u_{N+1} and l_{N+1}. We toss a coin and equiprobably choose one. It gets a child, the target, via a length-0 edge; the other gets none. This completes the construction.

LEMMA 13. *For all positive integers w, for all deterministic algorithms A_w designed to traverse graphs drawn from $\mathcal{G}(w)$, the expected cost of A_w to traverse a graph drawn randomly from $\mathcal{G}(w)$ is at least $r_w L_w$.*

Proof. The proof is by induction on w. The $w = 1$ case is trivial.

Let $w \geq 2$. Choose a deterministic algorithm A_w for graphs drawn from $\mathcal{G}(w)$.

Within this proof, we imagine that the random graph H is generated "on the fly"; i.e., only when the searcher reaches either u_i or l_i, for i odd, are the two graphs for stages i and $i+1$ drawn from $\mathcal{G}(w-1)$, and only then are stages i and $i+1$ of H built. This makes no difference, since A_w is on-line and its behavior cannot depend on the future.

Pick an odd $i < N$. At the end of stage $i-1$, the searcher occupies either u_i or l_i. Let J be a graph having $i-1$ stages that induces the searcher to occupy u_i at the end of stage $i-1$ (if possible). Now define an algorithm A_{w-1} (dependent on J) for traversing graphs drawn from $\mathcal{G}(w-1)$, as follows. A_{w-1} mimics A_w in the graph drawn from $\mathcal{G}(w-1)$ in the F_{w-1} layers succeeding u_i, until, if ever, A_w backtracks through s to a nondescendant of u_i. At this point, A_{w-1} blindly marches ahead in a naive way, until u_{i+1} is reached.

The cost of backtracking through s is so large that the cost incurred by A_w in the $2F_{w-1}$ layers succeeding u_i, given that the first $i-1$ stages equal J, is at least the cost of A_{w-1} on those same layers. The inductive hypothesis now implies that the expected cost of A_w in the $2F_{w-1}$ layers succeeding u_i, given J, is at least $r_{w-1}L_{w-1}$.

Now choose an $i-1$-stage graph J', if possible, so that A_w occupies l_i at the end of stage $i-1$. A similar argument implies that the conditional expected cost incurred by A_w in the $2F_{w-1}$ layers succeeding l_i, given that the first $i-1$ stages of H equal J', is at least $r_{w-1}L_{w-1}$. It follows that the (unconditional) expected cost incurred by A_w in progressing from either u_i or l_i to either u_{i+2} or l_{i+2} is at least $r_{w-1}L_{w-1}$.

At the end of stage N, we flip a coin to decide which vertex, u_{N+1} or l_{N+1}, becomes the parent of the target. With probability 1/2, the searcher must backtrack through s to the target. Thus it incurs an additional expected cost of at least $(1/2)(NL_{w-1} + E_{w-1})$. The total expected cost divided by L_w is at least

$$\begin{aligned}
&\frac{(N/2)r_{w-1}L_{w-1} + (1/2)(NL_{w-1} + E_{w-1})}{(1/2)E_{w-1} + (N/2)L_{w-1}} \\
&= \frac{(N/2)r_{w-1}L_{w-1} + (1/2)r_{w-1}E_{w-1} + (1/2)(NL_{w-1} + E_{w-1}) - (1/2)r_{w-1}E_{w-1}}{(1/2)E_{w-1} + (N/2)L_{w-1}} \\
&= (r_{w-1} + 1) - \frac{(1/2)r_{w-1}E_{w-1}}{(1/2)E_{w-1} + (N/2)L_{w-1}} \\
&\geq (r_{w-1} + 1) - \frac{(1/2)r_{w-1}E_{w-1}}{(N/2)L_{w-1}} \\
&\geq r_{w-1} + 1 - \frac{r_{w-1}E_{w-1}}{N} \\
&= r_{w-1} + (1 - 1/m). \qquad \square
\end{aligned}$$

Now we prove the following theorem.

THEOREM 14. *For every positive integer w, for every randomized algorithm $\mathcal{B}$ for traversing graphs drawn from $\mathcal{G}(w)$, there exists a layered graph K of width at most w such that the ratio of the expected distance traversed by $\mathcal{B}$ to the length of the shortest root–target path in K is at least r_w.*

Proof. The proof follows Yao's observation regarding the minimax principle [Yao]. Choose a randomized algorithm $\mathcal{B}$ and a width w. Lemma 13 implies that the expected cost incurred by every deterministic algorithm $\mathcal{A}$ on a graph drawn randomly from $\mathcal{G}(w)$ is at least $r_w L_w$. However, $\mathcal{B}$ is nothing more than a probability distribution on deterministic algorithms. It follows that the expected cost of $\mathcal{B}$ on a graph drawn randomly from $\mathcal{G}(w)$ is at least $r_w L_w$. It follows that on some graph K assigned positive probability under $\mathcal{G}(w)$, $\mathcal{B}$'s expected cost is at least $r_w L_w$. But the source–target distance in K is L_w. $\square$

7. Metrical service systems.

7. Metrical service systems. In the following section, w-MSS abbreviates "metrical service systems with requests of size at most w," w-LGT abbreviates "traversal of layered graphs of width at most w," and w-LTT abbreviates "traversal of $0-1$ rooted layered trees of width at most w." (Notice that w-LGT and w-LTT algorithms traverse only graphs of width at most w.)

LEMMA 15. *If A is a c_w-competitive algorithm for w-LGT, then there exist strictly c_w-competitive on-line algorithms for w-MSS in all metric spaces with integral distances.*

Proof. Fix a metric space where the distances are integral. Given a sequence of w-MSS requests, we construct, in an on-line manner, a layered graph. Layer 0 contains a single vertex, which is the starting point of the server. The vertices of layer $i > 0$ are the points of the ith request. For every $i \geq 0$, every vertex of layer i is connected to every vertex of layer $i+1$ by an edge of weight equal to the distance between the two points. Apply the w-LGT algorithm A to this graph. When A first encounters layer i, it chooses a vertex in that layer to move to. The w-MSS algorithm serves the ith request by moving to that point. $\square$

DEFINITION 16. *Let I be an infinite rooted layered tree in which each vertex has $2w$ children. Let r denote the root of I. Let $\mathcal{M}$ be an infinite metric space whose underlying set is $V(I)$ and in which the distance between u and v is the length of the $u-v$ path in I.*

LEMMA 17. *Let B be a strictly c_w-competitive w-MSS algorithm for the infinite metric space $\mathcal{M}$. Then there exists a c_w-competitive on-line w-LTT algorithm A (and therefore one for w-LGT).*

Proof. Let T be an instance of the w-LTT problem. Let s be the source vertex of T, initially occupied by the searcher. We use B to define algorithm A which traverses T as follows. From the, say, $l_i \le w$ vertices $v_1^i, v_2^i, ..., v_{l_i}^i$ in the ith layer of T, we construct, on-the-fly, a sequence $p_1^i, p_2^i, ..., p_{l_i}^i$ of l_i vertices of the metric space $\mathcal{M}$ (p_j^i "representing" v_j^i), and then present the set $\{p_1^i, p_2^i, ..., p_{l_i}^i\}$ as a request of $l_i \le w$ points to B. B will choose one of the points, say, p_j^i, to move to. We stipulate, then, that A moves to v_j^i.

Let us start by defining $v_1^0 := s$, the source vertex of T. Representing v_1^0 is $p_1^0 := r$, the root of I. A starts on the node $p = p_1^0$.

At a generic time, A will occupy some node in, say, layer i of the layered graph. When layer $i+1$ is revealed, we must choose request $i+1$ in $\mathcal{M}$, the response to which tells to which node of layer $i+1$ A should move. This is done as follows. Let the $l_{i+1} \le w$ nodes of the $i+1$st layer of T be $v_1^{i+1}, v_2^{i+1}, ..., v_{l_{i+1}}^{i+1}$. Look at the edge between a node v_j^{i+1} in the $i+1$st layer and its parent called, say, v_k^i. If the edge between v_j^{i+1} and its parent v_k^i is of weight 0, then we represent v_j^{i+1} by the same node p_k^i that represented its parent: $p_j^{i+1} := p_k^i$. If, on the other hand, the edge from v_j^{i+1} to its parent v_k^i is of weight 1, then we choose a child of p_k^i to represent v_j^{i+1}: we choose, among the $2w$ children of p_k^i, a child which is the root of a subtree in I containing no representative of a vertex in the ith (previous) layer and also containing no representative (so far) of a vertex in layer $i+1$. (Since there are at most $2w$ nodes in layers i and $i+1$, the $2w$ children of p_k^i suffice.) This child is then p_j^{i+1}.

It remains to show that for any two consecutive layers, the distance in T between any pair of vertices contained in those two layers is equal to the distance in I between their representatives. Consider any two consecutive layers numbered i and $i+1$. The proof is by induction on i. The case of $i = 0$ is easy and the proof is omitted. Now consider $i > 0$. Notice that by the inductive hypothesis the claim is true if both vertices in the pair are taken from the ith layer. As we generate the representatives for the vertices in the $i+1$st layer, we check the distances between the representatives and the representatives of vertices in the ith layer, and the distances between their representatives and the representatives already created for vertices in layer $i+1$. Consider a particular vertex v_j^{i+1} of the $i+1$st layer. If the distance to its parent v_k^i in T is 0, then, as described above, we have $p_j^{i+1} = p_k^i$, which is the representative of its parent. Since p_k^i has already been considered in the current step of the induction, the claim trivially holds. If the distance between v_j^{i+1} and v_k^i is 1, the choice of p_j^{i+1} guarantees that its distance to any representative q of a vertex in layer $i+1$ which was already considered in the current step of the induction, or of a vertex in layer i, is exactly the distance between p_k^i and q, plus 1. Thus, the claim holds in this case as well.

Therefore we conclude that at each step, the distance traversed by the w-MSS server is equal to the distance traversed by the w-LTT searcher. We also conclude that the optimal costs for both instances are the same (since an optimal path for one induces a path for the other with the same cost). This completes the proof of the lemma. □

Lemmas 15 and 17 give the following result.

THEOREM 18. *For each w, strictly c_w-competitive, deterministic or randomized algorithms exist for w-MSS for all metric spaces with integral distances if and only if a c_w-competitive, deterministic or randomized algorithm, respectively, exists for w-LGT.*

8. Concluding remarks. An obvious open problem is to close the gap between the upper bound and the lower bound for deterministic and randomized layered graph traversal. Of special interest is the question of designing an efficient randomized traversal algorithm. In an earlier version of this paper, we conjectured that a polynomial upper bound is achievable by the use of randomization. Since then, this conjecture has been proven by Ramesh [Ram], who gives an $O(w^{13})$-competitive randomized algorithm. Ramesh has also reported improvements in the deterministic upper bounds (to $O(w^3 2^w)$) and in the randomized lower bounds (to a nearly quadratic bound). Burley [Bur] recently further improved the deterministic upper bound to $O(w2^w)$ via an algorithm for metrical service systems.

REFERENCES

[AMOT] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan, *Faster algorithms for the shortest path problem*, J. Assoc. Comput. Mach., 37 (1990), pp. 213–223.

[BCR] R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins, *Searching in the plane*, Inform. and Comput., 106 (1993), pp. 234–252.

[Bel] R. Bellman, *On the routing problem*, Quart. Appl. Math., 16 (1958), pp. 87–90.

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

[BLS] A. Borodin, N. Linial, and M. Saks, *An optimal on-line algorithm for metrical task systems*, J. Assoc. Comput. Mach., 39 (1992), pp. 745–763.

[Bur] W. R. Burley, *Traversing layered graphs using the work function algorithm*, J. Algorithms, 20 (1996), pp. 479–511.

[CDRS] D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir, *Random walks on weighted graphs and applications to on-line algorithms*, J. Assoc. Comput. Mach., 40 (1993), pp. 421–453.

[CL] M. Chrobak and L. Larmore, *Server Problems and On-Line Games*, DIMACS Workshop on On-Line Algorithms, February 1991.

[Dij] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[DP] X. Deng and C. H. Papadimitriou, *Exploring an unknown graph*, in Proc. 31st IEEE Annual Symposium on Foundations of Computer Science, 1990, pp. 355–361.

[FKLMSY] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, *Competitive paging algorithms*, J. Algorithms, 12 (1991), pp. 685–699.

[FF] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.

[FL] J. Friedman and N. Linial, *On convex body chasing*, Discrete Comput. Geom., 9 (1993).

[Flo] R. W. Floyd, *Algorithm* 97 *(shortest path)*, Comm. ACM, 5 (1962), p. 345.

[FRR] A. Fiat, Y. Rabani, and Y. Ravid, *Competitive k-server algorithms*, J. Comput. System Sci., 48 (1994), pp. 410–428.

[KRR] H. J. Karloff, Y. Rabani, and Y. Ravid, *Lower bounds for randomized k-server and motion-planning algorithms*, SIAM J. Comput., 23 (1994), pp. 293–312.

[MMS] M. S. Manasse, L. A. McGeoch, and D. D. Sleator, *Competitive algorithms for on-line problems*, J. Algorithms, 11 (1990), pp. 208–230.

[PY] C. H. Papadimitriou and M. Yannakakis, *Shortest paths without a map*, Theoret. Comput. Sci., 84 (1991), pp. 127–150.

[RS] P. Raghavan and M. Snir, *Memory versus randomization in on-line algorithms*, in Proc. 16th ICALP, 1989. Springer-Verlag, New York, pp. 687–703.

[Ram] H. Ramesh, *On traversing layered graphs on-line*, J. Algorithms, 18 (1995), pp. 480–512.

[ST] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.

[Yao] A. C. C. Yao, *Probabilistic computations: Towards a unified measure of complexity*, in Proc. 18th IEEE Annual Symposium on Foundations of Computer Science, 1977, pp. 222–227.