On Traversing Layered Graphs On-line *

H. Ramesh

Courant Institute, New York University

Abstract

The following bounds on the competitive ratios of deterministic and randomized on-line algorithms for traversing width-w layered graphs are obtained.

1. A deterministic algorithm with a competitive ratio of $O(w^3 2^w)$. This ratio is close to the lower bound of $\Omega(2^w)$ and improves upon the previous best upper bound of $O(9^w)$.
2. The first known polynomially competitive randomized algorithm with a competitive ratio of $O(w^{13})$. This settles a conjecture due to Fiat et al. ([FFKRRV]).
3. A lower bound of $\Omega(\frac{w^2}{\log^{1+\epsilon} w})$ on the competitive ratio of any randomized algorithm for this problem, where ϵ is any positive number. The previous best lower bound was linear.

1 Introduction

A *layered graph* is a connected weighted graph whose vertices are partitioned into sets (i.e., layers) $L_0, L_1, L_2, \ldots$, and all edges connect vertices in consecutive layers. Each edge has a non-negative integral length. The set L_0 is a singleton set and the only vertex r in L_0 is called the *root* of the graph. The *width* of a layered graph is defined as $\max_i\{|L_i|\}$. The *on-line layered graph traversal problem* involves searching for a specified target vertex in a layered graph whose width and number of layers are unknown. The searcher starts at r and attempts to reach the target vertex. The edges between layer L_i and layer L_{i+1} and the vertices in layer L_{i+1} are revealed only when the searcher visits some vertex in L_i. Note that *all* vertices in L_{i+1} are revealed when the searcher visits some vertex in L_i. The searcher can traverse edges in both directions; however, he pays for the total distance he travels.

The layered graph traversal problem belongs to a larger family of shortest path problems which operate with incomplete information about the environment being searched. It was introduced by Papadimitriou and Yannakakis [PY] and generalizes the work of Baeza-Yates, Culberson and Rawlins [BCR]. They [BCR, PY] consider a number of such shortest path problems and give algorithms which start at a source and search for a target, learning about the environment as they progress. The complexity measure associated with such deterministic algorithms is the worst case ratio of the total distance traversed by the algorithm to the length of the shortest source-target path. For randomized algorithms, the expected distance traversed by the algorithm is considered instead of the total distance.

*This work was supported by NSF grants CCR-8902221, CCR-8906949, CCR-9202900 and CCR-8901484.

Papadimitriou and Yannakakis [PY] gave an optimal algorithm for layered graphs of width 2, with a competitive ratio of 9. It follows from the work of Baeza-Yates et al. [BCR] that $1+2w(1+\frac{1}{w-1})^{w-1} \sim 2ew$ is a lower bound on the competitive ratio for width-w layered graphs. Fiat et al. [FFKRRV] gave upper and lower bounds of $O(9^w)$ and $\Omega(2^w)$, respectively, on the competitive ratio of deterministic algorithms for traversing width-w layered graphs. Actually, their upper bound can be easily improved to $O((4.5)^w)$. They also gave a randomized lower bound of $\Omega(w)$. The only randomized upper bound known prior to this work was the exponential deterministic upper bound. Fiat et al. [FFKRRV] conjectured that a polynomial upper bound could be achieved using randomization.

Our contribution is threefold:

- We give an $O(w^3 2^w)$-competitive deterministic algorithm for traversing width-w layered graphs.
- We give the first polynomially competitive randomized algorithm for layered graph traversal, thus settling the conjecture due to Fiat et al. [FFKRRV]. Our algorithm has a competitive ratio of $O(w^{13})$.
- We show a lower bound of $\Omega(\frac{w^2}{\log^{1+\epsilon} w})$ on the competitive factor of any randomized algorithm, even when w is known in advance. Here, ϵ can be any positive constant.

In addition to being inherently interesting, the layered graph traversal problem is related to other on-line problems. It generalizes the Metrical Task Systems problem [BLS] and the k-Server problem [MMS] as explained by Fiat et al. [FFKRRV]. However, in the latter case, the width of the layered graph depends upon the cardinality of the metric space and therefore, layered graph traversal techniques are inadequate for producing solutions to the k-server problem. An important direct application of the layered graph traversal problem is to the Metrical Service Systems problem, suggested by Chrobak and Larmore [CL]. In this problem, a single server moving among points of a metric space is presented with requests; each request is a set of at most w points. The server must move to one of these points, the cost incurred being the distance moved. Chrobak and Larmore [CL] gave a competitive algorithm for uniform metric spaces and deterministic and randomized algorithms for all metric spaces for the case $w = 2$. Fiat et al. [FFKRRV] showed that the metrical service systems problem with requests of size w is equivalent to the width-w layered graph traversal problem, when w is known in advance. Thus, our bounds on layered graph traversal also apply to the metrical service systems problem. Some interesting variants of the layered graph traversal problem have also been studied [ABM, KRT].

Fiat et al. [FFKRRV] showed that traversing layered graphs on-line is equivalent to traversing layered trees on-line. All algorithms in the above work and in this paper actually traverse layered trees; the results applies unchanged to layered graphs.

Both our deterministic and randomized algorithms and the deterministic algorithm of Fiat et al. [FFKRRV] have the following flavour. The searcher starts from the root r of the layered tree T. We say that a searcher *spots* a vertex v when he reaches a vertex in the layer preceding the layer containing v. We pretend that the searcher is not looking for the target; rather, he wishes to determine a sequence of sets of vertices in T, with each set increasing the lower bound on the source-target distance. These sets are called *Active Sets* (the term is due to Fiat et al. [FFKRRV]). In this process, whenever the searcher spots the target, he aborts the search for the active set sequence and proceeds to the target along the shortest path from his current location to the target.

Active sets have the following properties. All vertices in a particular active set are at the same distance from r. Further, each path in T starting at r terminates at one of these vertices and no two of them have an ancestor-descendant relationship. Note that if the searcher has spotted all the vertices in the active set whose vertices are at distance d from the root and the target vertex has not yet been spotted then the distance of the target from the root is at least d.

The key issue in the algorithms is the determination of the active set sequence. The crucial step here is the search for the vertices of an active set, given all previous active sets in the sequence. All vertices in the current active set are twice the distance from the root as compared to the vertices in the previous active set in this sequence. We refer to this step as the *d-extension step*, where d is the distance between r and the vertices in the previous active set. d is a lower bound on the distance between r and the target vertex if the target has not been spotted till the beginning of the d-extension step. If the d-extension step completes without the target being spotted, the lower bound on the distance between r and the target is "extended" by d, i.e., $2d$ becomes a lower bound on this distance. The efficiency of the d-extension step is the critical factor which determines the overall competitive ratio of the algorithm. We give new schemes for performing the d-extension step which lead to the improved deterministic and randomized upper bounds mentioned above. Note that in our randomized algorithm, randomization is used only in the d-extension step.

Our randomized lower bound is achieved using a new construction. This construction is aimed at showing a lower bound on the expected number of times any randomized algorithm, while performing the d-extension step to determine a new active set, switches between the subtrees rooted at the vertices in the previous active set.

The rest of the paper is organized as follows. Section 2 gives some definitions and preliminaries. Section 3 describes the overall structure of our algorithms. Section 4 describes the deterministic algorithm and Section 5 describes the randomized algorithm. Section 6 describes the randomized lower bound.

2 Definitions and Preliminaries

Define a *layered 0–1 tree* to be a rooted acyclic layered graph in which each edge has a 0-1 weight and each non-root vertex has a neighbour in the previous layer.

Lemma 2.1 Fiat at al. [FFKRRV] *The on-line layered graph traversal problem is equivalent to the on-line layered 0–1 tree traversal problem, i.e., given a competitive on-line algorithm for traversing layered 0–1 trees, one can construct an on-line algorithm, with the same competitive ratio, for traversing arbitrary layered graphs.*

By virtue of Lemma 2.1, we need only consider layered 0–1 trees instead of arbitrary layered graphs. The algorithms that we present traverse layered 0–1 trees.

Let T be a layered 0–1 tree with root r, target t, and width w. Let $L_0, L_1, \ldots$ refer to the layers of T. $d(a,b)$ refers to the length of the path from vertex a to vertex b in T. Let v be a vertex in T. $p(v)$ denotes the parent of v. T_v refers to the subtree of T rooted at v. A vertex in layer L_j has *depth* j. Vertex x is said to be deeper than vertex y if the depth of x exceeds the depth of y. A vertex v is said to be *live* at an instant if it has a descendant in the deepest layer of T spotted till then.

We define active sets formally as follows. $AS_j(x)$ (read as the j-active set of vertex x) is the set consisting of those descendants v of x such that:

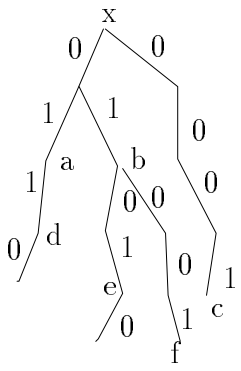

Figure 1: Active Sets

1. $v \in S_j(x) = \{\ u \mid d(x,u) = j,\ d(u,p(u)) = 1\}$, and,
2. v has a descendant in the layer containing the deepest vertex in $S_j(x)$.

In Fig. 1, $AS_1(x) = \{a, b, c\}$ and $AS_2(x) = \{e, f\}$. From Property 1 above, it follows that no two vertices in $AS_j(x)$ have an ancestor-descendant relationship. Then, from Property 2 above,

Fact 1 *$|AS_j(x)| \leq w$, for all j and all nodes x.*

The algorithms we present assume that the width of the tree is known in advance. Trees of unknown width can be traversed using these algorithms in the following manner: the searcher guesses the width w' of the tree (the first guess is 1) and traverses the tree until either the target is found or the tree is detected to have width greater than w'. In the latter case, the searcher increments w', by 1 for the deterministic case and by w' for the randomized case, and repeats the above step with the new value of w'. This process adds only an $O(1)$ multiplicative factor to the competitive ratio for both the deterministic and the randomized cases.

Instead of aiming to reach a target vertex t, our algorithms solve the harder problem of determining $AS_{d(r,t)+1}(r)$. Of course, this has to be done without prior knowledge of the value of $d(r,t)$. The searcher is sure to spot the target in the process of determining $AS_{d(r,t)+1}(r)$. In order to actually reach the target, the searcher incurs a cost of at most $2d(r,t)$ after determining $AS_{d(r,t)+1}(r)$. Thus determining $AS_{d(r,t)+1}(r)$ and then moving to the target vertex adds at most an $O(1)$ factor to the competitive ratio which is subsumed within the O notation.

A and B refer, respectively, to our deterministic and randomized algorithm for traversing tree T of known width w. Both A and B start at r and determine $AS_i(r)$, for some i. D_w and R_w denote quantities which upper bound the competitive ratios of A and B, respectively, i.e, $D_w \times i$ and $R_w \times i$ are upper bounds on the costs incurred by A and B while traversing T to determine $AS_i(r)$. Both algorithms recursively search subtrees of T which are known to have width w', $w' \leq w$. For all w', $1 \leq w' \leq w$, let $D_{w'}$ and $R_{w'}$ denote quantities which upper bound the competitive ratios of A and B, respectively, for traversing subtrees of T of known width w', i.e., the cost incurred for determining $AS_i(x)$ when the subtree rooted at x is known to have width w' is at most $D_{w'} \times i$ for the deterministic case and at most $R_{w'} \times i$ for the randomized case. We will define $D_{w'}$ and $R_{w'}$ shortly.

3 Outline of the Algorithms

Since traversing trees of width $w = 1$ is trivial, we assume that $w \geq 2$.

Fix a w', $1 \leq w' \leq w$. We describe A and B as they search a subtree T' of T rooted at a vertex r' to determine $AS_i(r')$, for some i. We assume that the width of the portion of T' searched is known to be w'. The knowledge of width prior to searching T' may seem peculiar. What actually happens is that the algorithms assume that width of the portion of T' searched is w' and abort if the width is discovered to exceed w'. If abortion does occur then the cost incurred is accounted for at the next higher level of recursion; at the highest level, the width is known to be w, by assumption. Therefore, we can assume that the portion of T' searched indeed has width w'.

Definitions. Let $i_0 = w^2$ for the deterministic case and $i_0 = w^7$ for the randomized case. We define the quantities $D_{w'}, R_{w'}, X^D_{w'}, X^R_{w'}, Y^D_{w'}, Y^R_{w'}$ as follows. As will will show later, the cost incurred by the A in determining $AS_i(r')$ is bounded by $D_{w'}i$ and the cost incurred by B in determining $AS_i(r')$ is bounded by $R_{w'}i$. Clearly, this is true when $w' = 1$. Note that we could have defined $R_1 = 1$; the definition of R_1 is below is for technical reasons which shall become clear in Lemma 5.10. Consider some $w' > 1$. We will assume that the above holds for all widths less than w' and show later that this holds even for w'. Note that $X^D_{w'}, Y^D_{w'}, X^R_{w'}, Y^R_{w'}$ are defined only when $w' > 1$ and $i > i_0$. Also note that $D_{w'} = O(w^3 2^{w'})$ and $R_{w'} = O(w^{12}w')$.

$D_{w'} = 1$, if $w = 1$.
$D_{w'} = w^2 w'$, if $w > 1$ and $i \leq i_0$.
$D_{w'} = (1 + \frac{1}{w} + \frac{2}{i_0})X^D_{w'} = (1 + \frac{O(1)}{w}) \max\{2D_{w'-1}, 3D_{w'-2} + D_{w'-3} + \cdots + D_1\} + O(w^2)w'$, if $w' > 1$ and $i > i_0$.

$Y^D_{w'} = (1 + \frac{1}{w} + \frac{1}{i_0}) \max\{2D_{w'-1}, 3D_{w'-2} + D_{w'-3} + \cdots + D_1\} + 4w(w'w + 2)$, if $w' > 1$ and $i > i_0$.
$X^D_{w'} = (1 + \frac{1}{w})Y^D_{w'}$, if $w' > 1$ and $i > i_0$.

$R_{w'} = O(w^6)$, if $w = 1$.
$R_{w'} = w^7 w'$, if $w > 1$ and $i \leq i_0$.
$R_{w'} = (1 + \frac{1}{w} + \frac{2}{i_0})X^R_{w'} = (1 + \frac{1}{w} + \frac{2}{i_0})(1 + \frac{1}{w}) \max\{(1 + \frac{O(1)}{w})R_{w'-1} + O(w^{10}), O(w^{12})w'\}$, if $w' > 1$ and $i > i_0$.

$Y^R_{w'} = \max\{(1 + \frac{O(1)}{w})R_{w'-1} + O(w^{10}), O(w^{12})w'\}$, if $w' > 1$ and $i > i_0$.
$X^R_{w'} = (1 + \frac{1}{w})Y^R_{w'}$, if $w' > 1$ and $i > i_0$.

Both A and B share the following basic framework. The following sequence of active sets is determined.

$$AS_1(r'), AS_2(r'), \ldots, AS_{i_0-1}(r'), AS_{i_0}(r'), AS_{2i_0}(r'), AS_{2^2 i_0}(r'), \ldots, AS_{2^j i_0}(r'), \ldots,.$$

Note that $AS_i(r')$ need not be in this sequence. The search to determine this sequence terminates when one of two situations is reached.

1. $AS_i(r')$ has been determined.

2. A vertex v which belongs to last determined active set in the above sequence is discovered to have the following property: all vertices in some layer in T' are descendants of v, i.e., v and its descendants are the only live vertices in T'.

In situation (1), the algorithm terminates, having incurred a cost of at most $D_{w'} \times i$ in the deterministic case and at most $R_{w'} \times i$ in the randomized case. Situation (2) leads to a *redefinition* of the root from r' to v, i.e., the searcher moves to v and determines $AS_{i-d(r',v)}(v)$ recursively. $D_{w'}$ and $R_{w'}$ will be defined so that the cost incurred by the algorithm till the redefinition of the root is at most $D_{w'} \times d(r', v)$ in the deterministic case and at most $R_{w'} \times d(r', v)$ in the randomized case, and further, the cost incurred by the algorithm following the redefinition of the root is at most $D_{w'}(i - d(r', v))$ in the deterministic case and at most $R_{w'}(i - d(r', v))$ in the randomized case. We refer to situations (1) and (2) as *termination conditions*.

The motivation behind redefining the root is the following. In the search to determine each active set in the sequence, the subtrees rooted at vertices in the previous active set are recursively searched. Clearly, if a redefinition of the root does not occur, then each such recursive search will involve a subtree of width at most $w' - 1$. This condition is necessary in order to set up the recurrences which describe $D_{w'}$ and $R_{w'}$.

The determination of the active set sequence involves two different schemes; one scheme determines $AS_1(r'), \ldots, AS_{i_0}(r')$ and another determines the rest of the sequence. The first of these is quite naive.

A Naive Algorithm. $AS_1(r')$ is determined by traversing all 0 weight edges which lead down from r'. Clearly, this incurs no cost. Next, we show how to determine $AS_k(r')$, given $AS_{k-1}(r')$, $2 \leq k \leq i_0$. The algorithm proceeds in at most $|AS_{k-1}(r')| \leq w'$ iterations. The searcher is located at r' at the beginning and end of each iteration. In each iteration, a live vertex v in $AS_{k-1}(r')$ is chosen. The searcher moves from r' to v. He then searches T_v by traversing all possible 0 weight edges that lead down from v in order to determine $AS_1(v)$. Finally, he returns to r'. The total cost incurred in each iteration is at most $2(k-1)$. The total cost over all iterations is thus at most $2(k-1)w'$.

Lemma 3.1 *The cost incurred by the naive algorithm to determine $AS_i(r')$, $i \leq i_0$, is at most $i(i-1)w'$. Therefore, $D_{w'}$ and $R_{w'}$ bound the competitive ratio of A and B, respectively, for determining $AS_i(r')$, $i \leq i_0$.*

It remains to show that $D_{w'}$ and $R_{w'}$ bound the competitive ratios of A and B, respectively, when $w' > 1$ and $i > i_0$. This involves determining the rest of the active set sequence, which in turn involves extension steps.

The 2^j-Extension Step. This step determines $AS_{2^{j+1}i_0}(r')$ after $G = AS_{2^j i_0}(r')$ has been determined and is accomplished by recursively searching the subtrees rooted at each of the vertices in G. If a redefinition of the root does not occur during this step then the width of each of these subtrees is at most $w' - 1$, as mentioned earlier.

Our improved results stem from performing this crucial step efficiently. We review the performance of this step in Fiat et al. [FFKRRV] before describing our algorithms for performing this step. We define the competitive ratio of the 2^j-extension step to be the total cost incurred in this step divided by $2^j i_0$. As we will show later, this ratio will be bounded by $X^D_{w'}$ for the deterministic case and by $X^R_{w'}$ for the randomized case.

The 2^j-Extension Step of Fiat et al. [FFKRRV]. Fiat et al. show that at most two subtrees of width $w' - 1$ and at most one subtree each of width $w' - 2, w' - 3, \ldots, 1$ need to be traversed recursively in the worst case. The worst case scenario for their algorithm is illustrated in Fig. 2. The widths of the subtrees appear within the subtrees in parentheses. Let $F_{w'}$ denote the competitive

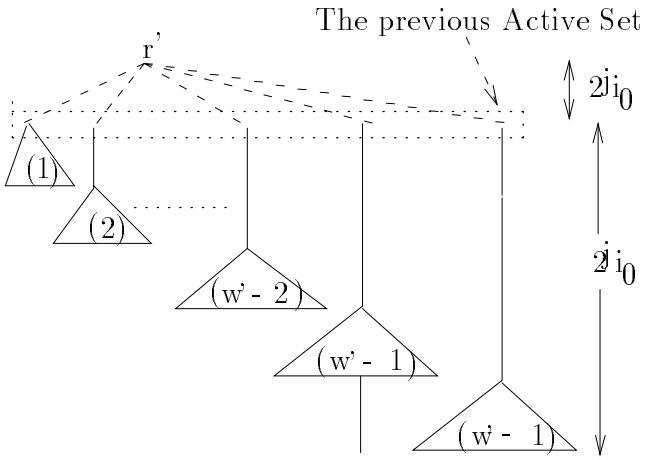

Figure 2: Typical Scenario for the Algorithm of Fiat et al.

ratio of their algorithm. The competitive ratio of an extension step for this algorithm is at most $2F_{w'-1} + F_{w'-2} + \cdots + F_1 + O(w')$; the last term reflects the cost incurred in switching between vertices in G. In order to derive $F_{w'}$ for this algorithm, suppose that a redefinition of the root from r' to v takes place during the 2^j-extension step. Then, in the worst case, $d(r', v)$ could be as small as $2^j i_0$. The total cost incurred by the algorithm prior to this extension step is at most $(i_0)^2 w' + (F_1 + F_2 + \cdots + F_{w'-2} + 2F_{w'-1} + O(w'))(2^j i_0 - i_0) \sim (F_1 + F_2 + \cdots + F_{w'-2} + 2F_{w'-1} + O(w'))(2^j - 1)i_0$. The cost incurred in this extension step could be as much as $(F_1 + F_2 + \cdots + 2F_{w'-1} + O(w'))2^j i_0$. The total cost incurred till the redefinition is the sum of these two costs. Setting $F_{w'}(2^j i_0)$ equal to this total cost yields that, roughly, $F_{w'} \sim 2(F_1 + \ldots + F_{w'-2} + 2F_{w'-1} + O(w'))$ and therefore, $F_{w'} = O((4.5)^{w'})$.

Thus, the competitive ratio of an extension step in Fiat et al.ś algorithm is at most $2F_{w'-1} + F_{w'-2} + \cdots + F_1 + O(w')$ and $F_{w'}$ is roughly twice this competitive ratio.

Our 2^j-Extension Step. We wish to show that $D_{w'}$ and $R_{w'}$ bound the competitive ratios of A and B, respectively, This is done by achieving the following goals.

1. Performing each extension step in a manner such that $X^D_{w'}, X^R_{w'}$ bound the competitive ratios of an extension step in the deterministic and the randomized case, respectively.

2. To show that the competitive ratio of A is at most $(1 + \frac{1}{w} + \frac{2}{i_0})X^D_{w'}$ and the competitive ratio of B is at most $(1 + \frac{1}{w} + \frac{2}{i_0})X^R_{w'}$.

We describe how to achieve each of these two goals.

Goal 2. The solution to the second goal is common to both the deterministic and the randomized cases. The solution is simply to perform the 2^j-extension step in phases instead of doing it in one shot. There are up to w phases. The phases proceed as follows. At the beginning and end of each phase, the searcher is at r'. In phase i', $1 \leq i' \leq w$, $AS_{2^j i_0 + i'd'}(r')$ is determined, where $d' = \lceil \frac{2^j i_0}{w} \rceil$. A phase is terminated prematurely if one of the termination conditions occurs, following which either the root is redefined or the algorithm terminates. A redefinition of the root occurs when at most one

of the vertices in the previous active set (i.e., the active set determined at the end of the previous phase) is detected to be live.

We show that the above algorithm satisfies Goal 2. We need the following definitions.

Definitions. Let the competitive ratio of a phase be defined analogous to the competitive ratio of an extension step, i.e., it is the ratio between the cost incurred in a phase and d'. As we will show in subsequent sections, a phase can be performed so that $Y^D_{w'}$ upper bounds the competitive ratio of a phase for the deterministic case and $Y^R_{w'}$ upper bounds the competitive ratio of a phase for the randomized case.

For the purpose of Lemmas 3.2 and 3.3, assume that $Y^D_{w'}$ upper bounds the competitive ratio of a phase for the deterministic case and $Y^R_{w'}$ upper bounds the competitive ratio of a phase for the randomized case. Lemma 3.2 then shows that the competitive ratio of any extension step is bounded by $X^D_{w'}$ and $X^R_{w'}$ for the deterministic and the randomized cases, respectively. Lemma 3.3 shows that Goal 2 is satisfied.

Lemma 3.2 *The total cost incurred in the 2^j-extension step is bounded by $X^D_{w'}2^j i_0$ in the deterministic case and $X^R_{w'}2^j i_0$ in the randomized case.*

Proof. Since the cost incurred in a phase is at most $Y^D_{w'}d'$ in the deterministic case and $Y^R_{w'}d'$ in the randomized case, and since there are up to w phases in the 2^j-extension step, the total cost incurred in the 2^j-extension step is bounded by $Y^D_{w'}d'w$ for the deterministic case and $Y^R_{w'}d'w$ for the randomized case. Since $\frac{d'w}{2^j i_0} = \frac{w}{2^j i_0}\lceil\frac{2^j i_0}{w}\rceil \leq 1 + \frac{w}{2^j i_0} \leq 1 + \frac{1}{w}$, $Y^D_{w'}d'w \leq X^D_{w'}2^j i_0$ and $Y^R_{w'}d'w \leq X^R_{w'}2^j i_0$. The lemma follows. □

Lemma 3.3 *The total cost incurred by algorithm A to determine $AS_i(r')$, $i > i_0$, is at most $(1 + \frac{1}{w} + \frac{2}{i_0})X^D_{w'}i = D_{w'}i$. The total cost incurred by algorithm B to determine $AS_i(r')$, $i > i_0$, is at most $(1 + \frac{1}{w} + \frac{2}{i_0})X^R_{w'}i = R_{w'}i$.*

Proof. We show the lemma only for the deterministic case. The randomized case is identical. Suppose that the root is redefined from r' to v (the second termination condition) in phase i' of the 2^j-extension step. A similar analysis holds if the first termination condition occurs instead of the second. Clearly, it suffices to show that the cost incurred by the algorithm till the redefinition plus the cost of moving to v in order to recursively determine $AS_{i-d(r',v)}(v)$ is bounded by $(1+\frac{1}{w}+\frac{2}{i_0})X^D_{w'} \times d(r',v) = D_{w'} \times d(r',v)$.

Note that $d(r',v) \geq 2^j i_0 + (i'-1)d'$. The cost incurred in the previous extension steps is at most $(i_0)^2 w' + X^D_{w'}(2^j i_0 - i_0) \leq X^D_{w'}2^j i_0$ (this is because the final values of $X^D_{w'}, X^R_{w'}$ derived are greater than $i_0 w'$). The cost incurred in the first $i'-1$ phases of this extension step is at most $Y^D_{w'}(i'-1)d' \leq X^D_{w'}(i'-1)d'$. The cost incurred in the i'th phase is at most $Y^D_{w'}d' \leq X^D_{w'}d'$. The cost incurred in moving to v is $d(r',v)$. The total cost incurred till the redefinition is thus $X^D_{w'}(2^j i_0 + (i'-1)d') + X^D_{w'}d' + d(r',v) \leq X^D_{w'}(d(r',v) + d') + d(r',v)$. Since $d' = \lceil\frac{2^j i_0}{w}\rceil$, $\frac{d'}{d(r',v)} \leq \frac{1}{w} + \frac{1}{i_0}$, and $X^D_{w'}(1 + \frac{d'}{d(r',v)})d(r',v) + d(r',v) \leq X^D_{w'}(1 + \frac{1}{w} + \frac{1}{i_0} + \frac{1}{X^D_{w'}})d(r',v) \leq X^D_{w'}(1 + \frac{1}{w} + \frac{2}{i_0})d(r',v)$, as claimed. □

Goal 1. By Lemma 3.2, for showing Goal 1, it suffices to show how to perform a phase so that $Y^D_{w'}$ and $Y^R_{w'}$ bound the competitive ratios of a phase in the deterministic and randomized cases, respectively. While describing a phase, we assume that the phase runs to completion without any of the termination conditions occurring. The algorithm for performing a phase is different in the deterministic and the randomized cases. The two cases are described in Sections 4 and 5, respectively.

4 The Deterministic Case

Recall that the competitive ratio of an extension step for the algorithm of Fiat et al. is at most $2F_{w'-1} + F_{w'-2} + \cdots + F_1 + O(w')$. If a phase is performed using this algorithm, this leads to the competitive ratio of a phase possibly being $2D_{w'-1} + D_{w'-2} + \cdots + D_1 + O(w \times w')$, which is too expensive for solving Goal 1. Instead, we divide a phase further into subphases. There are up to w subphases in a phase; each subphase determines a new active set. Recall $d' = \lceil \frac{2^j i_0}{w} \rceil$. Let $d'' = \lceil \frac{d'}{w} \rceil$. Subphase j_1 of phase j_2 of the 2^j-extension step determines the active set $AS_{2^j i_0 + (j_2-1)d' + j_1 d''}(r')$, if $j_1 d'' < d'$, and the active set $AS_{2^j i_0 + (j_2-1)d' + d' - (j_1-1)d''}(r')$, otherwise. A subphase continues as long as at least 3 vertices in the previous active set (i.e., the active set determined in the previous subphase) are live. This has the effect of ensuring that the portions of subtrees rooted at vertices in the previous active set which are searched in this subphase have width at most $w' - 2$. If in some subphase, at most 2 vertices in the previous active set are discovered to be live then this subphase and the sequence of subphases terminates. The phase is then completed by recursively searching the subtrees rooted at these two vertices, each of which has width at most $w' - 1$ as long as none of the termination conditions occurs. A subphase itself is performed similar to the algorithm of Fiat et al.

For the sake of completeness, we give the algorithm for performing a subphase. Let $W = AS_{2^j i_0 + (j_2-1)d' + d_1}(r')$ be the last active set determined (W was determined either at the end of the previous subphase, or at the end of the previous phase if there was no previous subphase, or at the end of the previous extension step if there was no previous phase either). In a subphase, there are up to $|W| = n \leq w'$ iterations. At the beginning and the end of each iteration, the searcher is located at the root r'. In the kth iteration, a vertex v in W is chosen. The searcher moves down from r' to v and attempts to search T_v recursively to determine $AS_v(d_2)$, where $d_2 = d''$ if $d_1 + d'' \leq d'$, and $d_2 = d' - d_1$ otherwise. The searcher then returns to r'. The searcher performs the recursive search in T_v only as long as the width of the portion of T_v traversed is at most $k' = \min\{k, w' - 2\}$; he abandons this search and returns to r' the moment this width is found to exceed k'. If this width does not exceed k' then v is removed from W at the end of this iteration, i.e., it is not considered in subsequent iterations; otherwise, v remains in W. In either case, all those vertices in W which are no longer live are removed from W at the end of this iteration. In addition, before beginning the recursive search of T_v, the searcher strips of all those vertices from T_v which do not have a descendant in the deepest layer of T spotted till then. The following lemma holds.

Lemma 4.1 *The above algorithm determines $AS_{d_2}(v')$ for all those vertices v' which are initially in W in at most $n \leq w'$ iterations.*

Proof. We show that after the kth iteration, if the subphase does not terminate then there are at most $n - k$ vertices in W. Further, we show that whenever a vertex v is removed from W, $AS_{d_2}(v)$ has already been determined.

The proof is by induction on k. Before the first iteration, there are at most n vertices in W. Suppose after iteration $k - 1$ and before iteration k, there are at most $n - k + 1$ vertices in W. Consider iteration number k and suppose vertex $v \in W$ is chosen in this iteration. If $AS_{d_2}(v)$ is determined in this iteration and v is removed from W then the lemma is clearly true. So suppose that T_v is found to have width more than $k' = \min(k, w - 2)$ in this iteration.

Let L_a be the deepest layer in T_v seen till immediately before T_v was searched in this iteration. Since the searcher strips off all vertices in T_v which do not have a descendant in L_a before searching

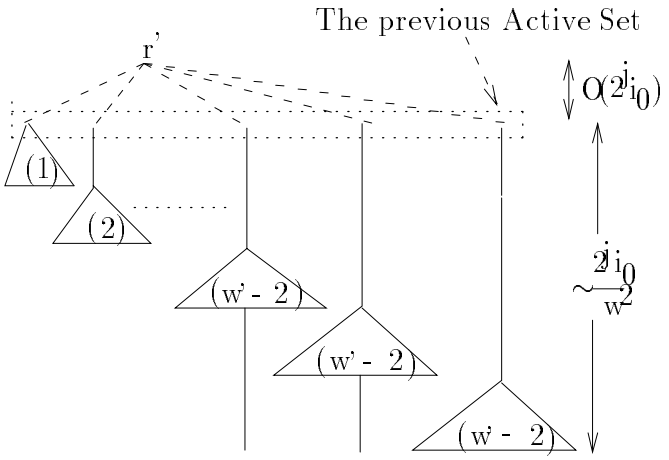

Figure 3: Worst Case Scenario in a Subphase

T_v, the width of layer L_a in T_v is at least as large as the width of any previous layer in T_v (after stripping). Therefore, either L_a or some layer deeper than L_a in T_v achieves width greater than k'. Further, all vertices initially in W are at most as deep as L_a. It follows that at most $n - k'$ of the vertices (including v) initially in W can be live after the kth iteration. If $k' = w - 2$ then at most two vertices in W remain live and the subphase terminates. If $k' < w - 2$, $k' = k$. Since vertices which are not live are removed from W and since when a vertex in detected not to be live all its descendants are known to the searcher, the lemma follows. □

Corollary 4.2 *The cost incurred in a subphase is at most* $(3D_{w'-2} + D_{w'-3} + \cdots + D_1)d_2 + 4w'wd'$ *(See Fig. 3. and contrast with Fig. 2.).*

Proof. In iteration k, $1 \leq k \leq w - 3$, the width of the subtree traversed recursively is at most k; the cost incurred in this recursive traversal is thus at most $D_k d_2$. The cost incurred in recursively traversing subtrees in the last three iterations is at most $3D_{w'-2}d_2$. The cost incurred in moving from r' to a vertex in W at the beginning of each iteration and back to r' at the end of each iteration is at most $2w' \times 2wd'$. □

Next, we derive the total cost incurred in a phase. Goal 1 follows from Lemma 4.3.

Lemma 4.3 *The total cost incurred in a phase is at most* $(1 + \frac{1}{w} + \frac{w}{i_0}) \max\{2D_{w'-1}, 3D_{w'-2} + D_{w'-3} + \cdots + D_1\}d' + 4w(w'w + 2)d' = Y^D_{w'}d'$.

Proof. First, suppose each of the subphases in a phase is performed to completion (i.e., in none of the subphases are at most two of the vertices in the previous active set found to be live). The total cost incurred is at most $(3D_{w'-2} + D_{w'-3} + \cdots + D_1)d' + 4w'w^2d'$. The lemma follows for this case.

Next, suppose in the kth subphase, at most two of the vertices in the previous active set are found to be live. Then the cost incurred in the k subphases is $(3D_{w'-2} + D_{w'-3} + \cdots + D_1)d''k + 4kw'wd'$. The cost incurred after the subphases terminate is $2D_{w'-1}(d' - d''(k-1)) + 8wd'$. The

total cost incurred is $\max\{2D_{w'-1}, 3D_{w'-2} + D_{w'-3} + \cdots + D_1\}(d' + d'') + 4wd'(kw' + 2)$. Since $d' + d'' \leq d'(1 + \frac{d''}{d'}) \leq d'(1 + \frac{1}{w} + \frac{1}{d'}) \leq d'(1 + \frac{1}{w} + \frac{w}{i_0})$ and $k \leq w$, the total cost incurred is $(1 + \frac{1}{w} + \frac{w}{i_0})\max\{2D_{w'-1}, 3D_{w'-2} + D_{w'-3} + \cdots + D_1\}d' + 4w(w'w + 2)d'$. □

Theorem 4.4 *$D_{w'} = O(w^3 2^{w'})$. There exists a deterministic algorithm for traversing a layered graph of unknown width having competitive ratio $O(w^3 2^w)$, where w is the width of the layered graph.*

Proof. The fact that $D_{w'} = O(w^3 2^{w'})$ follows from the definition of $D_{w'}$. The second part of the lemma follows from Lemmas 3.1 and Lemma 3.3. □

5 The Randomized Case

Consider phase i' of the 2^j-extension step. In this phase, $AS_{2^j i_0 + i'd'}(r')$ is determined, where $d' = \lceil \frac{2^j i_0}{w} \rceil$. Let $e = 2^j i_0 + (i' - 1)d'$ and $W = AS_e(r')$.

This case offers more difficulties that the deterministic case. To see that a naive strategy does not work, consider Fig. 4. Suppose the searcher initially chooses one of v_1 and v_2 at random and completely searches the subtree rooted at that vertex. If he chooses v_1, he must traverse both T_1 and T_2. If he chooses v_2, he need only traverse T_2. Both T_1 and T_2 have width $w' - 1$, therefore $Y^R_{w'}$ can be as much as $\frac{1}{2}2R_{w'-1} + \frac{1}{2}R_{w'-1} = \frac{3}{2}R_{w'-1}$, which gives an exponential bound. To improve upon this, the natural thing to do is to not traverse the subtrees T_1 or T_2 completely but to decrease the probability of being in a subtree as more of it is traversed. To accomplish this, each subtree is divided into "chunks", at boundaries of which the change in probability take place.

Definition. The notion of a "chunk" is made more precise as follows.

Define $Q_h(u)$, $u \in W$, to be the set $\{u\}$ if $h = 0$ and the set of vertices $AS_{h\lceil \frac{d'}{w^5} \rceil}(u)$, if $1 \leq h \leq w^5$. Intuitively, a chunk corresponds to the collection of subtrees of T' that lie between vertices in $Q_{h-1}(u)$ and $Q_h(u)$, i.e., subtrees rooted at vertices in $Q_{h-1}(u)$ and having the vertices in $Q_h(u)$ as leaves. For purely technical reasons, the definition of $Q_h(u)$ is refined further slightly as follows. Note that $Q_h(u)$ is not always defined. This is the case if each leaf u' of T_u is such that $d(u, u') < h\lceil \frac{d'}{w^5} \rceil$. If $Q_{h-1}(u)$ exists but $Q_h(u)$ does not and u has a descendant which is deeper than the deepest vertex in $Q_{h-1}(u)$, we redefine $Q_h(u)$ to be the set of deepest leaves of T_u. We refer to $Q_h(u)$, for any h, as a Q-set *of* vertex u or a Q-set *belonging to* u.

Recall our assumption that at all times at least two vertices in W are live; otherwise the current phase and extension step terminate with a redefinition of the root. It follows that the combined width of the subtrees in T_u, $u \in W$, forming a chunk is at most $w' - 1$.

A phase is accomplished by determining all Q-sets belonging to each vertex in W. This search need not proceed independently for each $u \in W$. To see this, consider some $Q_{h_1}(u_1)$ and $Q_{h_2}(u_2)$ such that the deepest vertex in $Q_{h_1}(u_1)$ is at least as deep as the deepest vertex in $Q_{h_2}(u_2)$. Then, in the process of searching T to determine $Q_{h_1}(u_1)$, all vertices in $Q_{h_2}(u_2)$ would also be spotted by the searcher. Therefore, the searcher can *implicitly determine* $Q_{h_2}(u_2)$ while *explicitly determining* $Q_{h_1}(u_1)$. More precisely, we say that the searcher explicitly determines a Q-set $Q_h(u)$, $u \in W$, if the searcher is in the subtree rooted at u when the deepest vertex in $Q_h(u)$ is spotted. Similarly, we say that the searcher implicitly determines a Q-set $Q_h(u)$, $u \in W$, if the searcher is in the subtree rooted at some vertex in $W - \{u\}$ when the deepest vertex in $Q_h(u)$ is spotted.

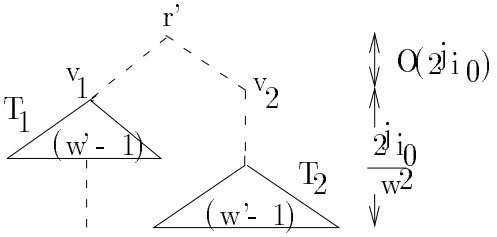

Figure 4:

This motivates a total ordering of all Q-sets belonging to vertices in W in increasing order of the depth of their respective deepest vertices, with ties broken arbitrarily. Note that any on-line algorithm determines the Q-sets in this order (modulo the ties). We say that $Q_{h_1}(u_1)$ is *larger* than $Q_{h_2}(u_2)$ or $Q_{h_1}(u_1) > Q_{h_2}(u_2)$ if the former appears after the latter in the above ordering. For any vertex $u \in W$, the largest Q-set belonging to u is called the *maximal* Q-set of u.

Note that there may be up to $w'w^5$ Q-sets. The expected number of these determined explicitly is critical. In particular, note that this expected number must be least w^5 and less than $(1+\epsilon)w^5$, for any constant ϵ independent of w, otherwise the competitive ratio becomes exponential in w. We first show, in Section 5.1, how the search for determining the Q-sets can be organized in a manner such that at most $w^5(1+\frac{O(1)}{w})$ Q-sets are determined explicitly. We then show, in Section 5.2, how each explicit search is performed, i.e., given $Q_{h-1}(u)$, how T_u is searched explicitly in order to determine $Q_h(u)$.

5.1 Minimizing Explicit Searches.

This is the heart of the randomized algorithm. We need the following definitions before proceeding with this description.

Definitions. We use the term "determination" to include both implicit and explicit determination, unless explicitly specified. Note that the various Q-sets will be determined in the order defined above, which is independent of the random choices which the algorithm will make. Further, each Q-set is determined in the above order, either implicitly or explicitly. Since all Q-sets are not determined explicitly, we associate with each $Q_h(u)$ a probability $p_h(u)$ (over all random choices made by the algorithm) that $Q_h(u)$, when determined, is determined explicitly and not implicitly. $Q_h(u)$ is called *heavy* if $p_h(u) \geq \frac{1}{w^2}$ and called *light*, otherwise. In what follows, all probabilities are unconditional, unless specified explicitly.

5.1.1 The Algorithm

The algorithm has the following framework. It proceeds in at most w' stages; each stage ends when the maximal Q-set of some vertex in W is determined. At the beginning and end of each stage, the searcher is located at r'. Each stage proceeds in a number of substages; at the beginning and end of each substage the searcher is at some vertex in W. In each substage, some Q-set is determined explicitly.

Consider a particular stage. Let $W' \subset W$ be the set consisting of those vertices whose maximal Q-set has not been determined yet. This stage begins with the searcher selecting a vertex uniformly at random from the vertices in W' and moving to that vertex from r'. This is followed by a sequence of substages which continues until the maximal Q-set of some vertex in W' is determined. Following the last substage in the current stage, the searcher moves back to r' from his current location. In any particular substage, the searcher starts at some vertex $v \in W'$ and returns to v after determining $Q_h(v)$ explicitly, where $h \geq 1$ is the least number such that $Q_h(v)$ has not yet been determined. It remains to describe how the vertex v is picked from substage to substage.

In the first substage, the vertex v is picked uniformly at random from the vertices in W'. Next, consider a particular substage and suppose the searcher determines $Q_h(u)$ explicitly in this substage. If either $Q_h(u)$ is the maximal Q-set of u or if the maximal Q-set of some other vertex is determined implicitly in the process of determining $Q_h(u)$ explicitly, then the current stage comes to an end and the searcher moves back to r'. Otherwise, the decision of which vertex in W to move to for the next substage remains the only crucial issue. This decision requires knowledge of $p_h(u)$, which can be computed on the fly using a very simple formula, as we show later in Lemma 5.2. Since this substage is not the last substage in the current stage, every vertex in W' has a Q-set greater than $Q_h(u)$. Let $|W'| = n$. If $p_h(u) < \frac{1}{w^2}$ (i.e., $p_h(u)$ is light) then vertex u is chosen for the next substage too. Otherwise, if $p_h(u) \geq \frac{1}{w^2}$, then each of the $n-1$ vertices in $W' - \{u\}$ is chosen to be the vertex for the next substage with probability $\frac{1}{w^3 p_h(u)}$, and with probability $1 - \frac{n-1}{w^3 p_h(u)}$, the vertex for the next stage remains u.

5.1.2 Analysis

We show that the expected number of Q-sets determined explicitly is at most $w^5(1 + \frac{O(1)}{w})$. We then use this fact to determine the total cost incurred by the searcher in performing this algorithm. The searcher incurs two kinds of costs in this algorithm; the *traversal cost*, which is the total cost incurred in the various explicit searches performed, and the *switching cost*, which includes the cost incurred in switching among the vertices of W between substages along with the cost of moving from r' to a vertex in W at the beginning of a stage and the cost of moving back to r' at the end of a stage. We show that the total traversal cost is at most $w^5(1 + \frac{O(1)}{w})C$, where C is maximum over all Q-sets $Q_h(u)$ of the cost of determining $Q_h(u)$ explicitly given $Q_{h-1}(u)$, and that the total switching cost is at most $O(1)w^2\frac{1}{w^3}w^5 e = O(1)w^4 e$ (recall that $e = 2^j i_0 + (i'-1)d'$).

By the linearity of expectation, the expected number of Q-sets determined explicitly is given by the sum $\sum p_h(u)$ over all Q-sets $Q_h(u)$. Further, note that the sum $\sum p_h(u)$ over all light Q-sets is at most $\frac{1}{w^2}$ times the number of light Q-sets; this is at most $\frac{1}{w^2}w^5 w' \leq w^4$. Bounding the sum $\sum p_h(u)$ over all heavy Q-sets is a more involved process. We derive an upper bound on this sum next.

We analyze each stage separately. Consider a particular stage. Suppose this stage begins after the maximal Q-set of vertex v_1 has been determined and ends when the maximal Q-set of vertex v_2 is determined. Consider the set J consisting of Q-sets which lie between the maximal Q-sets of v_1 and v_2. Recall that the various Q-sets are determined in the order described earlier, independent of the random choices made by the algorithm. It follows that for each Q-set $Q_h(u)$ determined in this stage, $p_h(u)$ is independent of the probabilities with which random choices are made by the algorithm in all the previous stages and depends only upon the probabilities with which the algorithm makes its various random choices in the current stage.

Let W' be the set of those vertices in W whose maximal Q-sets are larger than the maximal Q-set of v_1. Let $|W'| = n$. Let $\alpha \geq 0$ be the least number such that each vertex in W' contains α heavy Q-sets in J. We show that the expected number of heavy Q-sets determined explicitly in this stage is at most $n + \alpha + O(1)\frac{|J|+w^5}{w^2}$. Over all w' stages, the expected number of Q-sets determined explicitly can be shown to be at most $w^5(1 + \frac{O(1)}{w})$ as follows.

Note that $nw' \leq w^2 \leq w^4$. The α term sums to at most w^5 over all stages. To see this, suppose that $Q_h(u)$ is the largest of all Q-sets. Let α_k, J_k denote the value of α, J, respectively, in the kth stage, $1 \leq k \leq w'$. Then there are at least α_k heavy Q-sets of u in J_k, $1 \leq k \leq w'$. Note that the J_k's are all disjoint. Since u has at most w^5 Q-sets, $\Sigma_{k=1}^{w'}\alpha_k \leq w^5$, as claimed. The $\frac{|J|}{w^2}$ term sums to at most $\frac{w'w^5}{w^2} \leq w^4$ over all stages. It follows that over all w' stages, the expected number of heavy Q-sets determined explicitly is at most $w^5(1 + \frac{O(1)}{w})$.

Note that J does not contain any maximal Q-sets. Also note that in the current stage, at most 1 Q-set apart from those in J can be determined explicitly. This Q-set is one of the following: for each vertex v in W', the least Q-set of v which is greater than or equal to the maximal Q-set of v_2. Thus the expected number of heavy Q-sets determined explicitly in the current stage is at most 1 more than the expected number of heavy Q-sets in J which are determined explicitly. We now bound the latter by bounding the sum $\sum p_h(u)$ over all heavy Q-sets $Q_h(u)$ in J.

Definitions. For vertices $u, u' \in W'$ and Q-set $Q_h(u)$ in J, $q_h(u, u')$ is defined to be the conditional probability of the searcher switching to vertex u' for the next substage, given that Q-set $Q_h(u)$ is determined explicitly in the current substage. The product $p_h(u)q_h(u, u')$ is denoted by $r_h(u, u')$ and is called the *exit-to-u' probability* of $Q_h(u)$. The total *exit probability* of $Q_h(u)$ is defined to be $\sum_{u' \in W'-\{u\}} r_h(u, u')$. Let $Par_h(u)$ denote the largest Q-set belonging to u in J, if any, which is less than $Q_h(u)$. Note that if $Par_h(u)$ is defined then it equals $Q_{h-1}(u)$. Let $C_h(u)$ denote the set of Q-sets in J defined as follows. If $Par_h(u)$ is defined then $C_h(u)$ is the set of those Q-sets in J which belong to a vertex other than u, precede $Q_h(u)$, and follow $Par_h(u)$. If $Par_h(u)$ is undefined then $C_h(u)$ is the set of those Q-sets in J which belong to a vertex other than u and precede $Q_h(u)$.

The following fact is easily seen from the algorithm description in Section 5.1.1.

Fact 2 *The exit probability of light Q-sets is 0. For vertices $u, u' \in W'$, $u \neq u'$, and each heavy Q-set $Q_h(u) \in J$, $r_h(u, u') = \frac{1}{w^3}$.*

Lemma 5.1 *Let $Q_h(u) \in J$ and let $Q_{h_1}(u)$ equal $Par_h(u)$, if the latter is defined. If $Par_h(u)$ is defined then $p_h(u) = p_{h_1}(u)(1 - \sum_{u' \in W'-\{u\}} q_{h_1}(u, u')) + \sum_{Q_{h'}(u') \in C_h(u)} r_{h'}(u', u)$. If $Par_h(u)$ is not defined then $p_h(u) = \frac{1}{n} + \sum_{Q_{h'}(u') \in C_h(u)} r_{h'}(u', u)$.*

Proof. First, consider the case when $Par_h(u)$ is defined. $Q_h(u)$ is explicitly determined if and only if one of the following events occurs: either $Q_{h_1}(u)$ is determined explicitly following which the searcher remains at u for the next substage or some $Q_{h'}(u') \in C_h(u)$ is determined explicitly following which the searcher switches to u for the next substage. Note that these events (one event corresponding to each $Q_{h'}(u') \in C_h(u)$ and one corresponding to $Q_{h_1}(u)$) are mutually exclusive. Summing up the probabilities for each of these events gives the required expression for $p_h(u)$.

The lemma for the case when $Par_h(u)$ is not defined follows from similar considerations with the event that $Q_{h_1}(u)$ is determined explicitly replaced with the event that vertex u is selected by the searcher for the first substage. □.

Definitions. For each Q-set $Q_h(u)$ in J, we define sets $Danc_h(u)$ (direct ancestors of $Q_h(u)$) and $Ianc_h(u)$ (indirect ancestors of $Q_h(u)$) as follows. $Danc_h(u)$ is the set of all heavy Q-sets of u in J which precede $Q_h(u)$. $Ianc_h(u)$ is the set of all heavy Q-sets in J which precede $Q_h(u)$ but do not belong to u. In addition, we define $Ianc_h(u,u')$ to be the set consisting of those Q-sets in $Ianc_h(u)$ which belong to vertex u'.

The following important lemma gives a way of computing, on the fly, the explicit determination probabilities of Q-sets in J.

Lemma 5.2 *For $Q_h(u) \in J$, $p_h(u) = \frac{1}{n} + (\sum_{u' \in W' - \{u\}}(|Ianc_h(u,u')| - |Danc_h(u)|))\frac{1}{w^3}$.*

Proof. Using Fact 2 and solving the recurrence relation described by Lemma 5.1, we obtain the following: $p_h(u)$ equals $\frac{1}{n}$ minus the sum of exit probabilities of the Q-sets in $Danc_h(u)$ plus the sum of the exit-to-u probabilities of the Q-sets in $Ianc_h(u)$. The lemma now follows immediately from Fact 2. □

Corollary 5.3 *For any vertex $u \in W'$, the number of heavy Q-sets in J is at most $\alpha + w^3 + 1$.*

Proof. Let u be the vertex with the maximum number of heavy Q-sets in J, with ties for the maximum broken in favour of the vertex whose largest Q-set in J is the smallest. Let $Q_{h'}(u)$ be the largest Q-set of u in J. Let u' be a vertex in W' such that u' has exactly α heavy Q-sets in J. In Lemma 5.2, each of the subterms of the second term in the expression for $p_{h'}(u)$ is non-positive. Therefore, for $p_{h'}(u)$ to be non-negative, $\frac{1}{n} + (|Ianc_{h'}(u,u')| - |Danc_{h'}(u)|)\frac{1}{w^3}$ must be non-negative. Note that $|Ianc_{h'}(u,u')| \leq \alpha$. It follows that $|Danc_{h'}(u)| \leq \alpha + \frac{w^3}{n} \leq \alpha + w^3$. □

Next, we use Lemma 5.2 to obtain an upper bound on the sum $\sum p_h(u)$, over all heavy Q-sets in J.

Lemma 5.4 *$\sum p_h(u)$ over all heavy Q-sets in J is at most $\alpha + w^3 + 1 + w'\frac{1}{w^3}|J| \leq \alpha + O(1)\frac{w^5+|J|}{w^2}$.*

Proof. We use a charging scheme to prove the above. Recall that in Lemma 5.2, $p_h(u)$ was expressed as the sum of two terms, A and B, say. B, in turn, is a sum of a number of subterms. Clearly, it is sufficient to account for A and the positive subterms in B.

First, we account for the positive subterms in B over all heavy Q-sets in J. Some negative subterms in B will be used to offset these positive terms. Suppose the subterm corresponding to vertex $u' \in W'$, $u' \neq u$, is positive, i.e., $|Ianc_h(u,u')| - |Danc_h(u)| > 0$.

Consider the Q-set $Q_{h'}(u') < Q_h(u)$ obtained by the following constructive procedure: consider each of the heavy Q-sets in J which belong to either u or u' in increasing order; for each such Q-set, push it on a stack if either the stack is empty or contains Q-sets belonging to the same vertex as itself, and pop the stack otherwise; the Q-set popped by $Q_h(u)$ is the required Q-set. That $Q_h(u)$ does indeed pop the stack is shown as follows. Note that at each step in the above constructive procedure, the size of the stack at the beginning of the step is the difference between the number of heavy Q-sets of u in J and and the number of heavy Q-sets of u' in J which are smaller than the Q-set considered in that step; further the stack contains only Q-sets of u if the former number is more than the latter, only Q-sets of u' if the latter number is more than the former, and no Q-sets if both numbers are equal. Then, since $|Ianc_h(u,u')| - |Danc_h(u)| > 0$, the stack contains exactly $|Ianc_h(u,u')| - |Danc_h(u)| > 0$ Q-sets of u' when $Q_h(u)$ is considered. Therefore, $Q_h(u)$ pops the stack.

Note that $|Danc_{h'}(u')| - |Ianc_{h'}(u', u)|$ is non-negative and equals the number of Q-sets on the stack immediately before $Q_{h'}(u')$ was pushed on it. Further, $|Ianc_h(u, u')| - |Danc_h(u)|$ is also non-negative and equals the number of Q-sets on the stack immediately before $Q_{h'}(u')$ is popped from the stack. Clearly, $|Danc_{h'}(u')| - |Ianc_{h'}(u', u)| = |Ianc_h(u, u')| - |Danc_h(u)| - 1$. Consequently, the positive subterm $(|Ianc_h(u, u')| - |Danc_h(u)|)\frac{1}{w^3}$ in the expression for $p_h(u)$ is largely offset by the subterm $(|Ianc_{h'}(u', u)| - |Danc_{h'}(u')|)\frac{1}{w^3}$ in the expression for $p_{h'}(u')$. The difference of $\frac{1}{w^3}$ between the two is charged to $Q_{h'}(u')$. It can easily be seen that, by this charging scheme, every heavy Q-set in J is charged at most n times overall, giving a total charge of $n\frac{1}{w^3}|J| \leq w'\frac{1}{w^3}|J|$.

It remains to account for the A terms. There are n vertices in W', each of which has at most $\alpha + w^3 + 1$ heavy Q-sets in J. Clearly, the sum of the A terms over all these Q-sets is at most $\alpha + w^3 + 1$.

The lemma follows. □

Corollary 5.5 *The expected number of heavy Q-sets determined explicitly in the current stage is at most $\alpha + O(1)\frac{w^5+|J|}{w^2} + 1 \leq \alpha + O(1)\frac{w^5+|J|}{w^2}$. The expected total number of Q-sets determined explicitly over all w' stages is at most $w^5(1 + \frac{O(1)}{w})$.*

Proof. As described earlier, at most 1 Q-set outside J can be determined explicitly in the current stage. Since $1 \leq w^3$, $\alpha + O(1)\frac{w^5+|J|}{w^2} + 1 \leq \alpha + O(1)\frac{w^5+|J|}{w^2}$. As explained earlier, the α term sums to at most w^5 over all $n \leq w' \leq w$ stages, the $\frac{w^5}{w^2}$ term sums to at most $\frac{w^5}{w}$ and the $\frac{|J|}{w^2}$ term sums to at most $\frac{w'w^5}{w^2} \leq \frac{w^5}{w}$. The corollary follows. □

Theorem 5.6 *The total expected traversal cost over all stages is at most $w^5(1 + \frac{O(1)}{w})C$, where C is maximum over all Q-sets $Q_h(u)$ of the cost of determining $Q_h(u)$ explicitly, given $Q_{h-1}(u)$. The total expected switching cost is at most $O(1)w^2\frac{1}{w^3}w^5e = O(1)w^4e$ (recall that $e = 2^j i_0 + (i' - 1)d'$). The total expected cost is thus $w^5(1 + \frac{O(1)}{w})C + O(w^4)e$.*

Proof. The traversal cost follows from Corollary 5.5.

The switching cost can be shown as follows. Each switch between stages and each move from r' to a vertex in W and back to r' costs $O(1)e$. The expected number of switches between stages is clearly bounded by the sum of the exit probabilities over all heavy Q-sets, which is at most $w'\frac{1}{w^3}w'w^5$. The number of moves from r' to a vertex in W and then back to r' is at most w', one per stage. The theorem follows. □

5.2 Algorithm for Explicit Q-set Determination

For each vertex $u \in W$, $Q_1(u)$ is determined explicitly by using B recursively to search T_u (recall from Section 2 that B is our randomized algorithm for traversing layered trees of known width w). The expected cost incurred in this process is $R_{w'-1}\lceil\frac{d'}{w^5}\rceil$.

Next, we show how to explicitly determine Q-set $Q_h(u)$ given Q-set $Q_{h-1}(u)$, $h > 1$. Note that the portion of T between $Q_h(u)$ and $Q_{h-1}(u)$ is not a single tree of width $w' - 1$ and therefore, cannot be assumed to be traversable with $R_{w'-1}\lceil\frac{d'}{w^5}\rceil$ expected cost. Rather, it is a collection of subtrees of combined width $w' - 1$, with the additional property that the least common ancestor of vertices in

$Q_{h-1}(u)$ is at most distance d' from each vertex in $Q_{h-1}(u)$. This situation is akin to the situation handled by an extension step and is dealt with in a similar manner as described below.

We define a procedure $S_1(v, x_1, x_2, w_1)$ which determines the set $AS_{x_1+x_2}(v)$ given $AS_{x_1}(v)$, $x_1 \geq x_2$, starting and terminating at v, and knowing that the portion of T_v searched in the process has width at most w_1. Clearly, explicit determination of $Q_h(u)$ given $Q_{h-1}(u)$ can be performed by $S_1(u, (h-1)\lceil\frac{d'}{w^5}\rceil, \lceil\frac{d'}{w^5}\rceil, w'-1)$. If $w_1 = 1$ then $S_1(v, x_1, x_2, w_1)$ is performed using the obvious algorithm. If $x_2 < w^2$, we use a naive algorithm similar to the one used used for determining the active set $AS_{i_0}(r')$ in Section 3. This algorithm determines the active sets $AS_{x_1+1}(v), AS_{x_1+2}(v), \ldots, AS_{x_1+x_2}(v)$ in sequence. Otherwise, if $w_1 > 1$ and $x_2 \geq w^2$, the algorithm for $S_1(v, x_1, x_2, w_1)$ is similar to that for an extension step. This procedure is performed in a sequence of up to w phases. Each phase begins and ends with the searcher at v; in the kth phase, $AS_{x_1+k\lceil\frac{x_2}{w}\rceil}(v)$ is determined. The sequence of phases continues till either the procedure completes or at most one vertex, v'' say, in the last active set determined is detected to be live. In the latter case, the search is completed by using B recursively to search $T_{v''}$. It remains to describe each phase.

We define another search procedure $S_2(v', y_1, y_2, w_1')$ which is used to perform a phase. In this procedure, the searcher determines $AS_{y_1+y_2}(v')$, given $AS_{y_1}(v')$, $y_1 \geq y_2$, starting and terminating at v', and knowing that the portion of $T_{v'}$ searched has width at most w_1'. Note that the kth phase of $S_1(v, x_1, x_2, w_1)$ is performed by $S_2(v, x_1 + (k-1)\lceil\frac{x_2}{w}\rceil, \lceil\frac{x_2}{w}\rceil, w_1)$.

If $w_1' = 1$ then $S_2(v', y_1, y_2, w_1')$ is performed using the obvious algorithm. If $y_2 < w^6$, we use a naive algorithm similar to the one used used for determining the active set $AS_{i_0}(r')$ in Section 3. This algorithm determines the active sets $AS_{y_1+1}(v), AS_{y_1+2}(v), \ldots, AS_{y_1+y_2}(v)$ in sequence. Otherwise, if $w_1' > 1$ and $y_2 \geq w^6$, in order to perform $S_2(v', y_1, y_2, w_1')$, we divide the portion of $T_{v'}$ between the sets $AS_{y_1}(v')$ and $AS_{y_1+y_2}(v')$ into chunks (with Q-sets defined in a similar fashion) and use the algorithm in Section 5.1 to determine all Q-sets (replace w', W, r', d' and e in Section 5.1 by w_1', $AS_{y_1}(v')$, v', y_2 and y_1, respectively). In this process, the procedure S_1 is used recursively whenever explicit determination of a Q-set is required, i.e., to determine $Q_h(u)$ explicitly given $Q_{h-1}(u)$, $S_1(u, (h-1)\lceil\frac{y_2}{w^5}\rceil, \lceil\frac{y_2}{w^5}\rceil, w_1'-1)$ is used.

Note that $S_2(r', e, d', w')$ is equivalent to the procedure which performs phase i' of the 2^j-extension step (recall $d' = \lceil\frac{2^j i_0}{w}\rceil$ and $e = 2^j i_0 + (i'-1)d'$). Further, note that the two search procedures S_1 and S_2 recurse mutually; however, when S_2 recursively uses S_1 the width parameter decreases. This causes the depth of recursion to be bounded by the width parameter at the outermost recursion level.

5.3 Analysis of the Overall Algorithm

We analyze the search procedures S_1 and S_2 and show that $Y_{w'}^R$ upper bounds the competitive ratio of a phase in an extension step. In order to show this, it suffices to bound the competitive ratio of procedure $S_2(r', e, d', w')$.

Let $s_1(x_1, x_2, w_1) \times x_2$ be the expected cost incurred by the searcher while performing $S_1(v, x_1, x_2, w_1)$. Let $s_2(y_1, y_2, w_1') \times y_2$ be the expected cost incurred by the searcher while performing $S_2(v', y_1, y_2, w_1')$. The following lemmas define the recurrences which describe s_1 and s_2.

Lemma 5.7 $s_1(x_1, x_2, w_1) \leq (1 + \frac{O(1)}{w}) \max\{s_2(x_1 + x_2, \lceil\frac{x_2}{w}\rceil, w_1), R_{w_1}\} + O(1)\frac{x_1+x_2}{x_2}$, *for* $x_2 \geq w^2$, $w_1 \geq 2$.
$s_1(x_1, x_2, w_1) \leq O(1)(x_1 + w^2)w_1$, *for* $x_2 < w^2$, $w_1 \geq 2$.
$s_1(x_1, x_2, w_1) = 2\frac{x_1+x_2}{x_2}$, *for* $w_1 = 1$.

Proof. When $w_1 = 1$, the cost incurred by the algorithm is at most $2(x_1 + x_2)$. Assume that $w_1 > 1$. For $x_2 < w^2$, the naive algorithm which determines the active sets $AS_{x_1+1}(v), AS_{x_1+2}(v), \ldots, AS_{x_1+x_2}(v)$ in sequence incurs a cost of at most $2(x_1 + x_2)w_1$ for each active set. The total cost is thus at most $O(1)w_1(x_1 + x_2)x_2$. The lemma follows for this case.

Next, consider the case when $w_1 \geq 2$ and $x_2 \geq w^2$. Recall S_1 is performed in a sequence of up to w phases; if at most one vertex in some intermediate active set determined is determined to be live then this sequence is terminated and the procedure is completed by recursively searching the subtree rooted at that vertex using procedure B. To determine to the cost of the entire procedure, we consider two cases.

First, suppose that in each phase, at least two vertices in the active set determined in the previous phase remain live. Then the total cost is clearly at most $ws_2(x_1 + x_2, \lceil \frac{x_2}{w} \rceil, w_1)\lceil \frac{x_2}{w} \rceil$. The lemma follows for this case since $x_2 \geq w^2$.

Next, suppose during the hth phase, $1 \leq h \leq w$, at most one vertex v'' in the active set determined after the $h-1$th phase is determined to be live. Then the total cost incurred is at most $s_2(x_1 + x_2, \lceil \frac{x_2}{w} \rceil, w_1)h\lceil \frac{x_2}{w} \rceil + R_{w_1}(w - (h-1))\lceil \frac{x_2}{w} \rceil + 2(x_1 + x_2)$. The last term here is the cost of moving down from v to v'' and moving back to v. The lemma follows immediately from the fact that $x_2 \geq w^2$. □

Lemma 5.8 $s_2(y_1, y_2, w_1') \leq (1 + \frac{O(1)}{w}) \max\{s_1(y_2, \lceil \frac{y_2}{w^5} \rceil, w_1' - 1), R_{w_1'-1}\} + O(1)w^4 \frac{y_1}{y_2}$, *for* $y_2 \geq w^6$, $w_1' \geq 2$.
$s_2(y_1, y_2, w_1') \leq O(1)(y_1 + w^6)w_1'$, *for* $y_2 < w^6$, $w_1' \geq 2$.
$s_2(y_1, y_2, w_1') = 2\frac{y_1+y_2}{y_2}$, *for* $w_1' = 1$.

Proof. The latter two cases follow as in Lemma 5.7.

Consider the first case next. Recall that S_2 works by determining a sequence of Q-sets, some of which are determined explicitly. Further, the Q-set $Q_1(u)$ for any vertex $u \in AS_{y_1}(v)$ is determined explicitly using the algorithm B recursively; this incurs a cost of at most $R_{w_1'-1}\lceil \frac{y_2}{w^5} \rceil$. In addition, any Q-set $Q_j(u)$ for any vertex u and $j > 1$ is determined explicitly using S_1; this explicit search incurs cost at most $s_1(y_2, \lceil \frac{y_2}{w^5} \rceil, w_1' - 1)\lceil \frac{y_2}{w^5} \rceil$. The lemma now follows from Theorem 5.6 and the fact that $y_2 \geq w^6$. □

Lemma 5.9 *In all invocations of* S_1 *and* S_2, $x_1 \leq w^5 x_2$, *where* x_1 *and* x_2 *are the second and third arguments to procedure* S_1, *and* $y_1 \leq 2w^6 y_2$, *where* y_1 *and* y_2 *are the second and third arguments to procedure* S_2.

Proof. Recall that procedure $S_2(r', e, d', w')$ actually performs phase i' of the 2^j-extension step (recall $d' = \lceil \frac{2^j i_0}{w} \rceil$ and $e = 2^j i_0 + (i' - 1)d'$). Clearly, $e \leq 2^{j+1} i_0 \leq 2wd'$. The lemma now follows immediately from an examination of the recurrences defined in Lemma 5.7 and 5.8. □

The above lemma now enables us to define $s_1'(w_1)$ and $s_2'(w_1')$ as below. Clearly, $s_1'(w_1)$ upper bounds $s_1(w^5 x_2, x_2, w_1)$ which in turn upper bounds $s_1(x_1, x_2, w_1)$. Similarly, $s_2'(w_1')$ upper bounds $s_2(2w^6 y_2, y_2, w_1')$ which in turn upper bounds $s_2(y_1, y_2, w_1')$.

$s_1'(w_1) = \max\{(1 + \frac{O(1)}{w}) \max\{s_2'(w_1), R_{w_1}\} + O(w^5), O(w^7)w_1\}$, for $w_1 \geq 2$.
$s_1'(w_1) = O(1)w^5$, for $w_1 = 1$.

$s_2'(w_1') = \max\{(1 + \frac{O(1)}{w}) \max\{s_1'(w_1' - 1), R_{w_1'-1}\} + O(w^{10}), O(w^{12})w_1'\}$, for $w_1' \geq 2$.
$s_2'(w_1') = O(1)w^6$, for $w_1' = 1$.

Goal 1 follows for the randomized case from Lemma 5.10.

Lemma 5.10 *The competitive ratio of a phase is bounded by $Y^R_{w'}$, $w' > 1$.*

Proof. Recall that $S_2(r', e, d', w')$ actually performs phase i' of the 2^j-extension step (recall $d' = \lceil \frac{2^j i_0}{w} \rceil$ and $e = 2^j i_0 + (i'-1)d'$). Therefore, the competitive ratio of a phase in an extension step is bounded by $s'_2(w')$. It follows from the above that $s'_2(w') = \max\{(1 + \frac{O(1)}{w}) \max\{s'_2(w'-1), R_{w'-1}\} + O(w^{10}), O(w^{12})w'\}$, for $w' \geq 3$, and that $s'_2(2) = O(w^{12})$ and $s'_2(1) = O(w^6)$. It now suffices to show that $R_{w'-1} \geq s'_2(w'-1)$. This is shown by induction. Let h be the induction parameter, $1 \leq h \leq w'-1$. When $h = 1$, by an appropriate choice of the constant in the definition of R_1, $R_1 \geq s'_2(1)$. Next, assume that $R_{w'-2} \geq s'_2(w'-2)$ and $w' - 1 > 1$. Then $s'_2(w'-1) = Y^R_{w'-1}$ for an appropriate choice of constants in the definition of $Y^R_{w'-1}$. Since $R_{w'-1} \geq Y^R_{w'-1}$, the claim follows. □

Theorem 5.11 *$R_{w'} = O(w^{12}w')$. There exists a randomized algorithm for traversing a layered graph of unknown width having competitive ratio $O(w^{13})$, where w is the width of the layered graph.*

Proof. The fact that $R_{w'} = O(w^{12}w')$ follows from the definition of $R_{w'}$. The second part of the lemma follows from Lemmas 3.1 and Lemma 3.3. □

6 A Randomized Lower Bound

We show a lower bound of $\Omega(\frac{w^2}{\log^{1+\epsilon} w})$ on the competitive ratio of any randomized algorithm for traversing layered trees of width w, even when the w is known in advance. Here ϵ is any positive number.

The lower bound we show is for randomized algorithms which are required only to spot the target vertex (i.e., reach the layer immediately preceding the layer containing the target vertex) and not to reach it. Since spotting the target is a weaker requirement than reaching it, this lower bound also holds for randomized algorithms which are required to actually reach the target.

Fix some value of ϵ, $0 < \epsilon < 1$. Let w_0 be a constant such that for all $j \geq w_0$, $1 > \frac{\log^{1+\epsilon} j}{j} \geq \frac{\log^{1+\epsilon}(j+1)}{j+1}$. Fix a width $w > w_0$ and an algorithm C for spotting the target vertex in any layered graph of known width at most w.

To show the lower bound, the adversary constructs a infinite family F_w of trees of width w. The distance of the source from the target is different in each tree in F_w; the minimum such distance over all trees in F_w is denoted by d_w. Consider any tree $T \in F_w$. All leaves in T are at the same level and at least distance d_w from the root. The target is always the leaf which is least distant from the root.

The construction of F_w is inductive. As the base case, we let F_{w_0} consist of a family of trees of width w_0 in which all leaves are at the same level and the distance of the source from the target is i in the ith tree in the family, $i \geq 1$. As the induction hypothesis, suppose the adversary has already constructed an infinite family F_{w-1} of trees of width $w-1$ with the requisite properties. Let $T_{w-1}(h)$ denote the tree in the family F_{w-1} in which the target is distance h, $h \geq d_{w-1}$, from the root. We show how to construct a tree $T \in F_w$ using a number of copies of trees in F_{w-1}.

Let R_i be the lower bound on the competitive ratio of C for spotting the target in trees in F_i, $w_0 \leq i \leq w$, i.e., R_i is the infimum, over all h such that $T_i(h) \in F_i$, of the expected cost incurred by C to spot the target in $T_i(h)$ divided by h.

6.1 Construction of T

T is constructed as follows. The root r of T has two children, a and b, each of which is distance $\frac{R_{w-1}d_{w-1}}{2}$ from the root. We refer to the subtrees T_a and T_b as the left and right *limbs* of T, respectively.

The construction of the rest of T consists of a number of *augmentation steps*. All leaves of the portion of T constructed so far will be at the same level before and after each augmentation step. In each augmentation step, the tree constructed till before that step is augmented as follows. Consider a particular step. Let c and e be the leaves in the left and right limbs, respectively, which are least distant from the root. In the augmentation step, one of c, e is made the root of a new instance of a tree $T' \in F_{w-1}$ while the other is made the root of a chain of 0-weight edges having the same number of layers as T'. We say that the augmentation step augments to the left if c is the root of the new instance of T', and to the right otherwise. This augmentation step is said to use tree T'.

The series of augmentation steps is divided into two stages; these stages are designed with the following motivation. In searching T, the searcher incurs two kinds of costs: the cost of traversing subtrees in each limb and the cost of switching between limbs. The purpose of Stage 1 is to make the switching cost substantial. Stage 2 is designed with the following observation in mind. Suppose the searcher occupies a particular limb with probability greater than half. Then, if that limb is repeatedly augmented, the probability of the searcher being in that limb had better decrease. If this probability decreases too slowly, then the expected cost incurred by the searcher in searching subtrees in this limb is high and all this work is wasted if the target is chosen to be in the other limb. Otherwise, if this probability does not decrease too slowly, then the expected cost of switching between the limbs is high.

Definitions. Let y be a number such that $y \geq 8R_{w-1}d_{w-1}$ and $T_{w-1}(y) \in F_{w-1}$. Define $d_0 = \max\{y \max\{R_{w-1}, w\}, \max\{(R_{w-1})^2, w\}d_{w-1}\}$.

Stage 1. Stage 1 is as follows. At the beginning of this stage, T consists of just r, a and b. Fix some d such that $d > d_0$, where d_0 will be defined in Stage 2, and d is a multiple of d_{w-1}. Different values of d yield different trees in F_w. T is alternately augmented to the right and left using tree $T_{w-1}(d_{w-1})$ till each limb has $\frac{d}{d_{w-1}}$ instances of $T_{w-1}(d_{W-1})$.

We require the following definitions in order to describe Stage 2. Define p_e, for any vertex e in T, to be the probability that C is in the limb containing e when e is spotted for the first time, i.e., the layer preceding the layer containing e is reached. Clearly, this probability is independent of the portion of T contained in the layers deeper than the layer containing e. Therefore, the adversary can vary the portion of T deeper than e and still keep p_e unchanged. Let $z = \lceil \frac{d}{y} \rceil$. Note that $z \geq \max\{R_{w-1}, w\}$. Let $x = \frac{w}{64z \log^{1+\epsilon} w}$. Clearly, $0 < x \leq \frac{1}{64}$.

Stage 2. In Stage 2, a stack is used to aid the construction. This stack will contain some of the trees of width $w-1$ used in the augmentation steps in this stage. At any instant, all trees in the stack will be in the same limb of T. The number of trees in the stack will exactly equal the difference between the current number of trees in the left and right limbs. Initially, the stack is empty. Each augmentation step either pops the stack or pushes into the stack. The following step is repeatedly used to augment T until either the stack is popped z times or the size of the stack reaches $\frac{1}{2x}$. Let c and f be the leaves nearest to the root at the beginning of this step in the left and right limbs, respectively. Since c and f are in the same layer, $p_c = 1 - p_f$. Let n be the number of elements currently in the stack. One of the following cases occurs.

1. $n = 0$. T is augmented to the left using the tree $T_{w-1}(y)$ if $p_c > p_f$ and to the right using the same tree, otherwise. The instance of $T_{w-1}(y)$ used in this augmentation is pushed on the stack.

2. $0 < n < \frac{1}{2x}$ and the topmost tree on the stack is in the same limb as the limb which was augmented last. Without loss of generality, let this limb be the left limb. In this case, the previous augmentation step pushed a tree on the stack. Let e be the root of the bottommost tree in the stack. If $p_e - p_c < nx$ then T is augmented to the left using the tree $T_{w-1}(y)$ and this tree is pushed on the stack. Otherwise, if $p_e - p_c \geq nx$ then T is augmented to the right using the tree $T_{w-1}(y)$ and the stack is popped.

3. $0 < n < \frac{1}{2x}$ and the topmost tree on the stack does not belong to the limb which was augmented last. In this case, the previous augmentation step popped the stack. Without loss of generality, let the limb augmented last be the right limb. Let e be the root of the topmost tree in the stack. This tree is in the left limb. If $p_c > p_e$ then T is augmented to the left using the tree $T_{w-1}(y)$ and this tree is pushed on the stack. Otherwise, if $p_c \leq p_e$ then T is augmented to the right using the tree $T_{w-1}(y)$ and the stack is popped.

The target is chosen to be the leaf in T which has the least distance from the root.

This completes the construction of T.

6.2 Analysis

We now analyze the above construction. Consider Stage 1 first.

Lemma 6.1 *The cost incurred by the searcher in Stage 1 is at least $R_{w-1}d$.*

Proof. Each switch from one limb to another costs at least $R_{w-1}d_{w-1}$ as does the explicit traversal of each tree used for augmentation in Stage 1. Clearly, the number of switches plus the number of trees (i.e., those used in augmentation) traversed explicitly is at least $\frac{d}{d_{w-1}}$ in this stage. □

Consider Stage 2 next. The cost of each switch from a vertex in one limb to a vertex in the other limb in Stage 2 is at least $2d + R_{w-1}d_{w-1}$. The cost of traversing an instance of the tree $T_{w-1}(y)$ used in the augmentation in Stage 2 is at least $R_{w-1}y$. Consider one instance T' of this tree and let e be the root and f be any leaf of T'. Let q be the conditional probability that the algorithm C does not reach a vertex at least as deep as e in the limb which does not contain T' until it first spots f, given that it is in the limb containing e when it first spots e. The expected cost incurred by the searcher in between the layers containing e and f is thus at least $p_e q R_{w-1}y + p_e(1-q)(2d + R_{w-1}d_{w-1})$. Since $y \leq \frac{d}{R_{w-1}}$, this cost is at least $p_e R_{w-1}y + p_e(1-q)(d + R_{w-1}d_{w-1})$. Since $p_e q \leq p_f$ and therefore, $p_e(1-q) \geq p_e - p_f$, this cost is at least $p_e R_{w-1}y + (p_e - p_f)(d + R_{w-1}d_{w-1})$. We refer to the two components of this sum as the expected *traversal cost* and the expected *switching cost*, respectively, of T'.

Lemma 6.2 *At the end of each augmentation step in Stage 2, $p_b - p_t \leq n_t x$, where b is the root of the bottommost tree T_b in the stack, t is the root of any other tree T_t in the stack and n_t is the number of trees in the stack below T_t.*

Proof. By induction on the order of the augmentation steps. This is clearly true for the first augmentation step as $b = t$ and $n_t = 0$. Next, assume that the statement is true after the $i-1$th augmentation step. We show that it is true after the ith augmentation step too. If the ith augmentation step pops the stack then the lemma is clearly true. So assume that the ith augmentation step does not pop the stack; it pushes tree T_f on the stack. There are two cases depending upon whether or not the $i-1$th augmentation step popped the stack.

First, suppose the $i-1$th augmentation step did not pop the stack. Then it pushed a tree, T'' say, on the stack. The target of T'' is the root of T_f. The lemma is obvious from the construction of T. Second, suppose the $i-1$th augmentation step pops the stack. If the stack is empty following the $i-1$th augmentation step then the lemma follows in the same manner as the base case. Otherwise, let f be the root of T_f and let e be the root of the tree T_e which is on top of the stack when T_f was pushed. Then $p_f > p_e$ by construction and $p_b - p_e \leq n_e x$ by the induction hypothesis. Therefore $p_b - p_f \leq p_b - p_e \leq n_e x \leq n_f x$. □

Lemma 6.3 *Consider any sequence of m augmentation steps in which the stack is popped with the property that the augmentation step immediately preceding the first step of this sequence did not pop the stack. Let T_1 and T_2 be the trees popped in the first and last steps, respectively, in this sequence. Let g be a leaf of T_1 and f be the root of T_2. Then $p_f - p_g \geq mx$. Further, the sum of the expected switching cost of the trees popped in this sequence of augmentation steps is at least $mx(d + R_{w-1}d_{w-1})$.*

Proof. Let e be the root of T_1. Let b be the root of the bottommost tree on the stack during the sequence of augmentation steps in consideration. Then, by construction, $p_b - p_g \geq n''x$, where n'' is the number of elements in the stack at the beginning of the first augmentation step in the sequence of augmentation steps in consideration. But, by Lemma 6.2, $p_b - p_f \leq n'x$, where n' is the number of elements in the stack after the last augmentation step in the sequence. Since $p_b - p_g = p_b - p_f + p_f - p_g$, it follows that $p_f - p_g \geq (n'' - n')x$. Clearly, $n'' - n' = m$ and the first part of the lemma follows.

Next, we show that the sum of the switching costs of the trees popped in the current sequence of augmentation steps is at least $mx(d + R_{w-1}d_{w-1})$. The proof is by induction on the prefixes of this sequence. We show that for any prefix of this sequence, if α is the root of the last tree popped in this prefix subsequence then the expected switching cost of the trees popped in this prefix subsequence is at least $(p_\alpha - p_g)(d + R_{w-1}d_{w-1})$. As the base case, consider the prefix of length 1, i.e., the popping of tree T_1. Recall that e is the root of T_1. $p_e - p_g \geq x$ by the first part of this lemma, and the lemma then follows for the base case from the definition of switching cost. Next, suppose T_3 is the tree popped in the penultimate augmentation step in the sequence. Let h be the root of T_3. As the induction hypothesis, assume that the expected switching cost of the first $m-1$ trees popped in this sequence is at least $(p_h - p_g)(d + R_{w-1}d_{w-1})$. Consider two cases next. First, suppose $p_h > p_f$. Then $p_h - p_g \geq p_f - p_g \geq mx$ and the lemma follows from the induction hypothesis. Second, suppose $p_h \leq p_f$. We show in the next paragraph that h must be a leaf of T_2. Then the expected switching cost for T_2 is at least $(p_f - p_h)(d + R_{w-1}d_{w-1})$. The lemma follows from the induction hypothesis and the facts that $p_f - p_g = p_f - p_h + p_h - p_g$ and $p_f - p_g \geq mx$.

It remains to show that h is a leaf of T_2 if $p_h \leq p_f$. Suppose h is not a leaf of T_2. Then the augmentation step prior to the one which uses T_3 would have popped the stack. Also T_2 is on top of the stack when the augmentation using T_3 is performed. From Case 3 in the construction description, it follows that $p_h > p_f$, a contradiction. □

Corollary 6.4 *The total expected switching cost over all trees used in the augmentation steps in Stage 2 is at least $x(d + R_{w-1}d_{w-1})$ times the number of times the stack is popped.*

Lemma 6.5 *Consider an augmentation step which pops the stack. Let this augmentation step use tree T_1 and let T_2 be the tree popped. The expected traversal costs of T_1 and T_2 sum to at least $R_{w-1}y$.*

Proof. Let e and f be the roots of T_1 and T_2, respectively. We show that $p_e + p_f \geq 1$. The lemma then follows from the definition of traversal costs.

Let tree T_3 with root b be the bottom of the stack when T_2 is popped. Let n' be the number of trees in the stack below T_2. Consider two cases. First, suppose the previous augmentation step (the one prior to the step in which T_2 is popped) also popped the stack. Then from Case 3 of the construction description, $1 - p_e \leq p_f$. Second, suppose the previous augmentation step did not pop the stack. Then it must have pushed T_2 on the stack. Let e' be a leaf of T_2. Since e and e' are at the same level but in different limbs, $p_e = 1 - p_{e'}$. From Lemma 6.2, $p_b - p_f \leq n'x$. From Case 2 of the construction description, $p_b - p_{e'} \geq (n'+1)x$. Therefore, $p_f - p_{e'} \geq x$ and hence, $p_f + p_e \geq 1 + x \geq 1$. □

Lemma 6.6 *Consider the state of the stack at any instant in the above construction. The sum of the expected traversal costs of the trees in the stack is at least $\frac{1-(m-1)x}{2} m R_{w-1}y$, where m is the size of the stack.*

Proof. If T_i is the ith bottommost tree in the stack and e_i is its root then, by Lemma 6.2, $p_{e_i} \geq p_{e_1} - (i-1)x$. Therefore, the sum of the expected traversal costs of all trees in the stack is at least $\frac{2p_{e_1}-(m-1)x}{2} m R_{w-1}y$. By Case 1 of the construction, $p_{e_1} \geq \frac{1}{2}$ and the lemma follows. □

Let Case 1 refer to the case in which Stage 2 terminates as a result of there being z popping steps. Let Case 2 refer to the case in which Stage 2 terminates as a result of the stack growing to size $\frac{1}{2x}$. In Case 1, let $d' = zy$ and in Case 2, let $d' = ny$, where n is the number of popping steps in the Stage 2 construction.

Lemma 6.7 *In Case 1, the cost incurred by C in Stage 2 is at least $R_{w-1}d + xz(d + R_{w-1}d_{w-1})$. The first term is the cost incurred in traversing the subtrees in each limb and the second term is the cost incurred in switching between limbs. In Case 2, the cost incurred by C in Stage 2 is at least $R_{w-1}d' + R_{w-1}\frac{y}{8x}$.*

Proof. First, consider Case 1. The total expected switching cost is clearly $xz(d + R_{w-1}d_{w-1})$ from Corollary 6.4. The total expected traversal cost is at least $zR_{w-1}y$ from Lemma 6.5. Since $zy \geq d$, the lemma follows for this case. Next, suppose Stage 2 terminates as a result of the stack growing to size $\frac{1}{2x}$. The total expected traversal cost is at least $nR_{w-1}y$, from Lemma 6.5, plus $(1 - (\frac{1}{2x} - 1)x)\frac{1}{4x}R_{w-1}y \geq \frac{1}{8x}R_{w-1}y$, from Lemma 6.6. The lemma follows for this case too. □

Corollary 6.8 *The total expected cost over both stages is at least $R_{w-1}(2d) + xz(d + R_{w-1}d_{w-1})$ in Case 1 and at least $R_{w-1}(d' + d + \frac{y}{8x})$ in Case 2.*

Proof. Follows from Lemmas 6.1 and 6.7. □

We can now obtain the recurrence for R_w. Recall that the distance from the root to the target is $\frac{R_{w-1}d_{w-1}}{2} + d + zy$ in Case 1 and $\frac{R_{w-1}d_{w-1}}{2} + d + ny$ in Case 2.

Case 1.

$R_w(d + zy + \frac{R_{w-1}d_{w-1}}{2}) \geq R_{w-1}(2d) + xz(d + R_{w-1}d_{w-1})$.

Since $d + zy \leq 2d + y$,

$R_w 2d \geq R_{w-1}\frac{2d}{1+\frac{y}{2d}+\frac{R_{w-1}d_{w-1}}{4d}} + xz\frac{d}{1+\frac{y}{2d}+\frac{R_{w-1}d_{w-1}}{4d}}$

Since $y \leq \frac{d}{R_{w-1}}$ and $d \geq \frac{1}{2}(R_{w-1})^2 d_{w-1}$,

$R_w 2d \geq R_{w-1}\frac{2d}{1+\frac{1}{R_{w-1}}} + xz\frac{d}{1+\frac{1}{R_{w-1}}}$

Since $R_{w-1} \geq 1$ and since $\frac{1}{1+a} \geq 1 - a$, for all $a \geq 0$,

$R_w 2d \geq R_{w-1}2d - 2d + xz\frac{d}{2}$

$R_w \geq R_{w-1} + \frac{xz}{4} - 1$

Case 2.

$R_w(d + d' + \frac{R_{w-1}d_{w-1}}{2}) \geq R_{w-1}(d' + d + \frac{y}{8x})$.

$R_w(d + d' + \frac{R_{w-1}d_{w-1}}{2}) \geq R_{w-1}(d + d' + \frac{R_{w-1}d_{w-1}}{2})(1 + \frac{\frac{y}{8x} - \frac{R_{w-1}d_{w-1}}{2}}{d+d'+\frac{R_{w-1}d_{w-1}}{2}})$

Recall that $y \geq 8R_{w-1}d_{w-1}$ and $x \leq \frac{1}{64}$ and therefore, $x \leq \frac{y}{8R_{w-1}d_{w-1}}$. Therefore, $\frac{y}{8x} \geq R_{w-1}d_{w-1}$ and $\frac{y}{8x} - \frac{R_{w-1}d_{w-1}}{2} \geq \frac{y}{16x}$. Then, using the facts $d \geq \frac{R_{w-1}d_{w-1}}{2}$ and $d' \leq d + y \leq 2d$, we get:

$R_w \geq R_{w-1}(1 + \frac{y}{64xd})$

Since $z = \lceil \frac{d}{y} \rceil$,

$R_w \geq R_{w-1}(1 + \frac{1}{64xz})$

Finally, from the above, we derive that

$R_w \geq \min\{R_{w-1}(1 + \frac{1}{64xz}), R_{w-1} + \frac{xz}{4} - 1\}$

Recall that $x = \frac{w}{64z\log^{1+\epsilon} w}$. Therefore,

$R_w \geq \min\{R_{w-1}(1 + \frac{\log^{1+\epsilon} w}{w}), R_{w-1} + \frac{w}{256\log^{1+\epsilon} w} - 1\}$

Recall that from the definition of w_0, $\frac{\log^{1+\epsilon} w}{w}$ is at most 1 and non-increasing for $w \geq w_0$. Therefore,

$R_w \geq \min\{\frac{R_{w_0}}{2^{w_0}}w^{\log^{\epsilon} w}, R_{w_0} + \frac{w^2 - w_0^2}{512\log^{1+\epsilon} w} - w\}$

For sufficiently large w, the first term is larger than the second. Therefore, $R_w = \Omega(\frac{w^2}{\log^{1+\epsilon} w})$.

Theorem 6.9 *Any randomized algorithm for traversing a layered tree of width w has a competitive ratio of at least $\Omega(\frac{w^2}{\log^{1+\epsilon} w})$, for any $\epsilon > 0$.*

7 Conclusion

Tighter bounds have been obtained on the layered graph traversal problem for both the deterministic and the randomized cases. The following problem, which we call the *Local Layered Tree Traversal Problem*, is an interesting and somewhat more natural variant of the layered tree traversal problem. Suppose not all the vertices in a layer are revealed when a vertex v in the previous layer is visited; rather, only those vertices which are adjacent to the vertex v are revealed. Note that this problem is a generalization of the Cow-Path problem studied by Kao, Reif and Tate [KRT] in which the layered tree to be traversed consists of a set of disjoint paths leading away from the root. It can easily be seen that the deterministic and the randomized lower bounds for general layered tree traversal also translate to the local layered tree traversal problem. An interesting open problem is to obtain deterministic and randomized upper bounds for this problem. It is not clear that randomization can lead to a polynomially competitive algorithm for this problem.

Acknowledgements

The author thanks Ravi Boppana and Richard Cole for discussions and help with this paper. The author also thanks the anonymous referees for numerous suggestions and an observation which improved the performance of the randomized algorithm.

References

[ABM] Y. Azar, A. Broder, M. Manasse. On-line Choice of On-line Algorithms. Proceedings of *4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 432-441, 1993.

[BCR] R. Baeza-Yates, J.C. Culberson, G.J.E. Rawlins. Searching in a Plane. To appear in *Information and Computation.*

[BLS] A. Borodin, N. Linial, M. Saks. An Optimal On-line algorithm for Metrical Task Systems. In *Journal of ACM*, 39, pp. 745-763, 1992.

[CL] M. Chrobak, L. Larmore. Server problems and On-line Games. DIMACS Workshop on On-line Algorithms, Feb. 1991.

[FFKRRV] A. Fiat, D.P. Foster, H. Karloff, Y. Rabani, Y. Ravid, S. Vishwanathan. Competitive Algorithms for Layered Graph Traversal. In *Proc. 32nd Annual Symposium on Foundations of Computer Science*, Sept. 1991.

[FL] J. Friedman, N. Linial. On Convex Body Chasing. Manuscript, May 1991.

[KRT] M. Kao, J. Reif, S. Tate. Searching in an Unknown Environment: An Optimal Randomized Algorithm for the Cow-Path Problem. Proceedings of *4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 441-450, 1993.

[MMS] M.S.Manasse, L.A. McGeoch, D.D. Sleator. Competitive Algorithms For Server Problems. In *Journal of Algorithms*, 11, pp. 208-230, 1990.

[PY] C.H. Papadimitriou and M. Yannakakis. Shortest paths Without a Map. In *Proc. 16th ICALP*, pp. 610-620, July 1989.