International Journal of Computational Geometry & Applications


# APPROXIMATE EUCLIDEAN SHORTEST PATHS IN 3-SPACE

JOONSOO CHOI JUERGEN SELLEN* CHEE-KENG YAP†
*Courant Institute of Mathematical Sciences*
*New York University*
*251 Mercer Street, New York, NY 10012*

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

Papadimitriou's approximation approach to the Euclidean shortest path (ESP) in 3-space is revisited. As this problem is *NP*-hard, his approach represents an important step towards practical algorithms. However, there are several gaps in the original description. Besides giving a complete treatment in the framework of bit complexity, we also improve on his subdivision method. Among the tools needed are root-separation bounds and non-trivial applications of Brent's complexity bounds on evaluation of elementary functions using floating point numbers.

*Keywords:* Approximate Euclidean Shortest Path, Exact Geometric Computation.

## 1. Introduction

The *Euclidean shortest path* (ESP) problem can be formulated as follows: given a collection of polyhedral obstacles in physical space $S$, and source and target points $s_0, t_0 \in S$, find a shortest obstacle-avoiding path between $s_0$ and $t_0$. Here $S$ is typically $\mathbb{E}^2$ or $\mathbb{E}^3$. It is evident that this is a basic problem in applications such as robotics. The case $S = \mathbb{E}^2$ has recently seen a breakthrough, with the $O(n \log n)$ algorithm of Hershberger and Suri.[1] However, our interest is the case $S = \mathbb{E}^3$. Here, Canny and Reif[2] proved that the problem is *NP*-hard. In contrast, Papadimitriou[3] described a polynomial-time approximation algorithm. The *approximate Euclidean shortest path* problem has, in addition to the above inputs, an input parameter $\epsilon > 0$ and we are required to construct an obstacle-avoiding path whose length is at most $(1 + \epsilon)$ times the length of the shortest path. As Papadimitriou suggested, his algorithm has not just a theoretical result, but offers hope for practical implementation. The approach is basically simple and may be very similar to ones that practitioners intuitively use, although without a corresponding analysis (see Ref.

*This author is supported by a postdoctoral fellowship from the DAAD, Germany.
†This author is supported by NSF grant #CCR-9002819.

[4]). An alternative approach using the idea of dividing the angular space evenly is given by Clarkson.[5] Basically, his algorithm reduces the dependency of the running time on $n$, while increasing the dependency on $\epsilon$.

Most geometric algorithms are developed within the context of one of two distinct computational frameworks, the *algebraic framework* or the *bit framework*. In the algebraic framework, we count the number of algebraic operations, assuming that each algebraic operation returns an exact value. Computational models appropriate for the algebraic framework include arithmetic circuits and real RAMS. In the bit framework, we are interested in the representation of algebraic values as binary strings. Algebraic operations must be reduced to Boolean operations. Computational models that cater to the bit framework include Boolean circuits and Turing machines.

Most basic problems in the algebraic framework have only one input size parameter $n$, which is usually proportional to the number of algebraic values in the input. For instance, the problem of multiplying two polynomials of degree $n$ is said to have input size $n$, and can be solved in $O(n \log n)$ algebraic operations. The result of Hershberger and Suri mentioned above also falls under the algebraic framework. Basic problems in the bit framework usually have two parameters, $n$ and $L$, where $n$ is as before and $L$ is the maximum bit size of any algebraic value. For instance, isolating the roots of a polynomial of degree $n$ where each coefficient has at most $L$ bits. Sometimes $n$ is bounded and may be omitted; an example is the GCD of two integers. Most numerical approximation problems (such as the approximate ESP above) naturally belong to the bit framework. In this case, we have an additional input parameter, $\epsilon$. Usually, $\epsilon$ is specified by a positive integer $s$ where $\epsilon = 2^{-s}$.

This paper is interested in bit complexity. The bit framework will become more important for computational geometry in the future. As the field now begins to address implementation issues in earnest, it must focus on lower-level operation. In lower-level operations, it is the bit size of numbers that is the main determinant of complexity. An important paradigm form future implementations of geometric algorithm is "exact computation".[6] In exact computation, complexity crucially depends on the bit size of numbers.

The main source of trouble in Papadimitriou's paper (and, for that matter, Clarkson's paper) lies in the fact that it is a result in the bit model and yet it lapses into the algebraic framework at several critical moments. This is inadmissible because we know of no sensible combination of the bit framework with the algebraic framework. After all, the *NP*-hardness result for ESP falls under the bit framework, and the approximation result of Papadimitriou is generally viewed as an approach to circumvent this hardness barrier. As an example of the lapses, note that lengths of polygonal paths are sums of square-roots of rational numbers. These values are no problem to compute in the algebraic framework, but in the bit framework, one must work out the precision to which one need to compute these values. We will see that the precision needed depends on how the numbers are to be used. One cannot take this precision issue lightly as it is precisely (no pun intended) the high precision needed for comparing sums of square-roots that prevents us from placing

the ESP problem (even the planar case) into the class *NP*.

Once one realizes these lapses, it is still not a routine matter to decide the precision to which we must compute these algebraic numbers. We will employ basic techniques of root bounds and algebraic computation (see Ref. [7]). One complication is that we will be using approximations of approximations (and so on). The second difficulty is that we need to compute elementary functions (exponential and logarithmic functions) to carry out various approximations. This is done by invoking Brent's results.[8] However, the application of these results in the context of deriving bit complexity bounds requires some arguments, as we will see.

The heart of Papadimitriou's algorithm is a subdivision scheme to break up edges into smaller segments. In our reworking of Papadimitriou's approach, we introduce a new subdivision scheme that has two nice features. If $\epsilon$ is halved, we can reuse all the points in the previous subdivision. Of course this can save recomputation in practice. Further, it turns out that our subdivision uses fewer points when $\epsilon$ is small. This is a slight asymptotic improvement. By exploiting this new subdivision scheme, Choi, Sellen and Yap[9] give an incremental approximation approach which yields an asymptotic improvements of running time.

The rest of this paper is organized as follows: in Section 2, we sketch Papadimitriou's algorithm in three parts and indicate where the difficulties come into each part. In Section 3, the new subdivision method for edges is described. In Section 4, we sketch some general lemmas on application of Brent's complexity bounds for floating point number computations. An Appendix is included for details. Sections 5 to 8 describe the fixes to the original algorithm, including the running time analysis in the framework of bit complexity. We conclude in Section 9.

## 2. Outline of an Approximation Algorithm

Assume that we are given a collection of polyhedral obstacles, source and target points $s_0, t_0$, and $0 < \epsilon \leq 1/2$. We assume that the obstacles are bounded by triangular facets. There are a total of $n$ edges in the collection of obstacles. Each vertex of obstacles and $s_0, t_0$ are specified by $L$-bit integers[a]. Throughout this paper, we use a parameter

$$\epsilon_1 := \frac{\epsilon}{C_0 n},$$

where $C_0$ is a suitably large constant. For the present discussion, $C_0 \geq 24$ suffices. The algorithm of Papadimitriou can be divided into three parts:

**(I)** We divide each edge $e$ into a set of *segments* by introducing points $p_0, p_{\pm 1}, p_{\pm 2}, \ldots$. We call the $p_i$'s the *break points* and $p_0$ the *origin* of $e$. Each segment is of the form $\sigma_i = [p_i, p_{i+1}]$ for $i \in \mathbb{Z}$. For a segment $\sigma$, let $\ell(\sigma)$ be the length of $\sigma$. For any two segments $\sigma, \sigma'$, let $D(\sigma, \sigma')$ denote the shortest distance between a point on $\sigma$ and a point on $\sigma'$. We can view $s_0$ and $t_0$ as degenerate segments. By construction each segment $\sigma$ satisfies

$$\ell(\sigma) \leq \epsilon_1 D(s_0, \sigma). \tag{2.1}$$

[a] We assume that the integers are in range $[-2^L, 2^L]$.

**(II)** The *visibility graph* $G = (V, A)$ is constructed. The vertices $V$ of the graph comprise the segments from (I) and also $s_0, t_0$. The *visibility edges* $A$ comprise pairs $(\sigma, \sigma')$ of nodes that can *see each other*, meaning that there exists $x \in \sigma, x' \in \sigma'$ such that the line segment $[x, x']$ avoids the (interior of) obstacles. We call such a line segment $[x, x']$ a *visibility segment.*

**(III)** We compute for each visibility edge $(\sigma, \sigma')$ a "nominal cost" $C(\sigma, \sigma')$, defined to be the Euclidean distance between the midpoints of $\sigma$ and $\sigma'$. We now apply Dijkstra's shortest path algorithm to the weighted graph $G' = (V, A; s_0, C)$. The shortest path $\pi$ in $G'$ from $s_0$ to $t_0$ is output.

The correctness of the algorithm amounts to the claim that $C(\pi)$ is at most $(1+\epsilon)C(\pi^*)$ where $C(\pi^*)$ denotes the Euclidean length of an actual shortest path $\pi^*$. According to the original paper, everything hinges on a clever subdivision scheme for the points $p_{\pm i}$ $(i = 0, 1, \ldots)$ in part (I).

We now give some counter indications:

(I)' The origin $p_0$ of an edge $e$ is a specially chosen rational point. For $|i| > 0$, $\|p_0 - p_i\|_2$ is equal to $\epsilon_1 \|s_0 - p_0\|_2 (1+\epsilon_1)^{|i|-1}$. But this means that the coordinates of $p_i$ would in general be irrational. The original paper assumes that $p_i$ can be determined exactly. But in fact, we must determine the precision to which to compute the points. This precision depends on the subsequent use of these points — see (III)' below.

(II)' For any edge $e$, let $span(e)$ denote the line through $e$. In general, for any set $X$ of points, $span(X)$ is the affine span of $X$. Following Papadimitriou, let us fix two edges $e, f$ and let $x$ and $y$ be real parameters for points in $span(e)$ and $span(f)$, respectively. Let $e(x)$ and $f(y)$ denote the corresponding points. Now consider a triangular facet $T$ bounding an obstacle. We view $T$ as the intersection of three half-planes contained in $span(T)$. Let $H$ be one such half-plane. So its boundary $\partial H$ contains an edge of $T$. The set of potential visibility segments $[e(x), f(y)]$ such that $span([e(x), f(y)])$ intersects $\partial H$ is a hyperbola

$$(x - a)(y - b) = c$$

for some $a, b, c$, as noted by Papadimitriou and Guibas.[3] However, the set $H^*$ of points $(x, y)$ corresponding to potential visibility segments that intersect $H$ is never of the form $(x-a)(y-b) \leq c$ or $(x-a)(y-b) \geq c$. This is intuitively obvious since the hyperbola is solely determined by $\partial H$, and does not take into account the actual plane $span(H)$. Assuming that we correctly figure out the *configuration obstacle* $H^*$ corresponding to any half-plane $H$, the corresponding configuration obstacle $T^*$ of $T$ is now given by

$$T^* = H_1^* \cap H_2^* \cap H_3^*$$

where $H_1 \cap H_2 \cap H_3 = T$. Finally, the free space $FP$ in the $(x, y)$ parameter space is the complement of the union of all the $T^*$'s. The boundary of $FP$ will consist of axes-parallel line segments and hyperbolic segments. In the plane sweep algorithm proposed to determine $FP$, the priority of events are the $x$-coordinates of intersections of these straight lines and hyperbolas. The original paper assumes we can compute these priorities exactly.

(III)' The original paper assumes that we can compute the nominal cost function $C(\sigma, \sigma')$ exactly. There are two sources of inaccuracies: we can only compute the midpoints of $\sigma$ approximately, and then we only compute an approximation to the distance between two approximate midpoints. A more serious matter arises: suppose the algorithm outputs a sequence of segments

$$\sigma_1, \sigma_2, \ldots, \sigma_k$$

of the obstacles which are touched by an approximation to the shortest path. This means that there exists a *zig-zag* path

$$\pi = (s, q_1, r_1, \ldots, q_k, r_k, t)$$

(where $q_i, r_i \in \sigma_i$) that avoids all obstacles. (Note that $q_i, r_i$ can be quite far apart in absolute terms.) We must compute an approximation to this zig-zag path, since obtaining some explicit path is the *raison d'être* of most applications. The approximation must be *guaranteed* to be obstacle-avoiding (and still achieves an $\epsilon$-approximation).

## 3. A New Subdivision Scheme

In this section, we present a new subdivision scheme for edges. Our scheme is motivated by the desire that the subdivision for $\epsilon/2$ is a refinement of the subdivision for $\epsilon$.

As in Ref. [3], the origin $p_0$ of an edge $e$ is chosen to be the closest point on $e$ to the source $s_0$. Let $d = \|s_0 - p_0\|_2$ be this distance. We describe the subdivision of the half-edge $e'$ to one side of the origin (the other half-edge is treated by analogy). Choose $p_1$ on $e'$ at distance $d$ from $p_0$. The segment $[p_0, p_1]$ is uniformly subdivided into segments of length $d\epsilon_1/2$ by introducing the points $q_1, q_2, \ldots$ such that

$$\|p_0 - q_i\|_2 = (d\epsilon_1/2) \cdot i. \tag{3.1}$$

The segments (resp., break points) between $p_0$ and $p_1$ are called the *uniformly-divided segments* (resp., *break points*). The rest of the half-edge $e'$ is subdivided using the points $p_2, p_3, \ldots$ defined as follows. Let $m$ be the smallest integer satisfying

$$2^m \geq \frac{\log(\ell(e)/d)}{\log(1+\epsilon_1)}, \tag{3.2}$$

where $\ell(e)$ denotes the length of an edge $e$. Then we define $p_i$ such that

$$\|p_0 - p_i\|_2 = d\left(\frac{\ell(e)}{d}\right)^{(i-1)/2^m}. \tag{3.3}$$

The segments (resp., break points) outside of $p_1$ are called the *exponentially-divided segments* (resp., *break points*).

**Lemma 1**

*(i) Each segment $\sigma$ created by the subdivision satisfies the basic inequality of Eq. (2.1).*
*(ii) There are at most $N = O(n(L/\epsilon_1))$ segments in all the edges.*
*(iii) The subdivision for $\epsilon/2$ is a refinement of the subdivision for $\epsilon$.*

**Proof.**
(i) If $\sigma = [q_i, q_{i+1}]$ is a uniformly-divided segment, then the length of $\sigma$, $\ell(\sigma)$, trivially satisfies

$$\ell(\sigma) = \epsilon_1 \cdot (d/2) < \epsilon_1 \cdot D(s_0, \sigma).$$

For an exponentially-divided segment $\sigma = [p_i, p_{i+1}]$, Eq. (3.2) and Eq. (3.3) imply

$$\begin{aligned} \ell(\sigma) &= \left( \left( \frac{\ell(e)}{d} \right)^{1/2^m} - 1 \right) \cdot d \left( \frac{\ell(e)}{d} \right)^{(i-1)/2^m} \\ &\leq \epsilon_1 \|p_0 - p_i\|_2 \\ &< \epsilon_1 D(s_0, \sigma). \end{aligned}$$

The last inequality follows from triangular inequality of three points $s_0, p_i$ and $p_{i+1}$.
(ii) For any obstacle edge $e$, there are at most $2 \cdot (2/\epsilon_1)$ uniformly-divided segments and $2 \cdot 2^m$ exponentially-divided segments. In Lemma 5, we will prove that $2^m < (8L + 16)/\epsilon_1$. Therefore there are $O(n(L/\epsilon_1))$ segments in total.
(iii) Let $q'_i$ (resp., $p'_i$) be a uniformly-divided (resp., exponentially-divided) break point of the subdivision for $\epsilon/2$. Then clearly every $q_i$ of the subdivision for $\epsilon$ is the same as $q'_{2i}$.

For any $\epsilon$, let $m(\epsilon)$ be the integer $m$ defined in Eq. (3.2). Then $m(\epsilon/2)$ is equal to $m(\epsilon)$ or $m(\epsilon) + 1$, since $\epsilon_1/2 \leq \ln(1 + \epsilon_1) \leq \epsilon_1$ for $\epsilon_1 \in [0, 1]$. Therefore, if $m(\epsilon/2) = m(\epsilon)$, then both subdivisions introduce the same exponentially-divided break points. Otherwise, every $p_i$ of the subdivision for $\epsilon$ is the same as $p'_{2i-1}$, since

$$\begin{aligned} \|p_0 - p_i\|_2 &= d \left( \frac{\ell(e)}{d} \right)^{(i-1)/2^{m(\epsilon)}} \\ &= d \left( \frac{\ell(e)}{d} \right)^{(2i-2)/2^{m(\epsilon)+1}} \\ &= \|p_0 - p'_{2i-1}\|_2. \end{aligned}$$

□

Property (i) is similar to the original scheme. Property (ii) improves on the original by an additive factor of $O(n \log(1/\epsilon_1)/\epsilon_1)$. This improvement can be significant because $\epsilon$ might be exponentially small in $L$, especially if we want to guarantee that we approach an actual shortest path. Property (iii) is advantageous to have when we want to use successive refinements of $\epsilon$.

## 4. Floating Point Number Computations

Approximate computations introduce a host of detailed calculations which we highlight in this section. Our subdivision scheme requires us to compute values of

the form $E^F$ where $E, F$ are numerical expressions. The following problem captures what is needed:

> (P1) Let $s, L_1, L_2$ be given natural numbers and $E, F$ be numerical expressions satisfying $2^{-L_1} \leq F \leq 1$ and $-L_2 \leq \lg E \leq -2^{-L_2}$. The problem is to compute $E^F$ to precision $s$, in time polynomial in $s, L_1, L_2$.

In application, we have $s = O(\log(1/\epsilon_1) + L)$, $L_1 = O(\log(1/\epsilon_1) + L)$, and $L_2 = O(L)$ (see Section 7).

Note that $\widehat{x}$ approximates $x$ to *absolute precision* (resp., *relative precision*) $s$ means $|\widehat{x} - x| < 2^{-s}$ (resp., $|(\widehat{x} - x)/x| < 2^{-s}$). In the sequel, we will use the term "precision" for relative precision.

We resort to Brent's results[8] on computation of elementary functions, and compute $E^F$ using the following obvious straight line program:

(i) Compute $V_1$ which approximates $E$ to precision $s_1$;
(ii) Compute $V_2$ which approximates $F$ to precision $s_2$;
(iii) Compute $V_3$ which approximates $\ln(V_1)$ to precision $s_3$;
(iv) Compute $V_4$ which approximates $V_2 \cdot V_3$ to precision $s_4$;
(v) Compute $V_5$ which approximates $\exp(V_4)$ to precision $s_5$.

We want to conclude that (with suitable choices of $s_i$'s) $V_5$ approximates $E^F$ to precision $s$. We would hope that the cost of these steps is polynomial in $s, L_1, L_2$. The sequence of approximations is compounded by the fact that application of Brent's result is not straightforward: his bounds for computing elementary functions assume that each function is defined over an arbitrary but fixed domain $[a, b]$. Under this assumption, one can compute $\exp(x)$ and $\ln(x)$ to precision $s$ with time complexity (bit complexity) $O(\mu(s) \log s)$, where

$$\mu(s) = O(s \log s \log\log s)$$

is the time complexity (bit complexity) of multiplying $s$-bit integers. When the domain $[a, b]$ is variable, we need to pay for the conversion to the fixed range. Towards a satisfactory solution of (P1), consider a subproblem:

> (P2) Let $s, L$ be given natural numbers and $x$ a number satisfying $2^{-L} \leq |x| \leq 2^L$. Compute $\exp(x)$ to precision $s$ in time polynomial in $s, |x|$.

Before we solve this problem, let us briefly describe the floating point number system we will be assuming.

### *4.1. Floating point numbers*

A floating point number $\alpha$ is a pair $\langle e, f \rangle$ of binary integers where $e$ is the *exponent* and $f$ the *fraction*. Let $\langle f \rangle$ denote the number obtained by placing a binary decimal point just in front of the most significant one ('1') in $f$ (if $f = 0$, then define $\langle f \rangle = 0$). Thus $1/2 \leq \langle f \rangle < 1$ for $f \neq 0$. The number *represented by* $\alpha$ is

$$2^e \cdot \langle f \rangle .$$

We also write $\alpha \sim 2^e\langle f\rangle$, to emphasize that $\alpha$ is the representation itself, not just the value it represents. The *precision* of $\alpha$ is[b] equal to $1 + \lfloor \lg f \rfloor$.

**Lemma 2** *Assuming a floating point representation of numbers, we can solve (P2) in time $O(\mu(s+|x|)\log(s+|x|))$ if $1 \leq |x| \leq 2^L$, and $O(\mu(s+\lg L)(L+\log(s+\lg L)))$ if $2^{-L} \leq |x| < 1$.*

Let us sketch a proof of Lemma 2. Let $\langle e, f\rangle \sim x$. We define

$$y := x \cdot 2^{-\lceil \lg |x| \rceil} = \begin{cases} \langle 1, f\rangle; & f = 1, \\ \langle 0, f\rangle; & f > 1, \end{cases}$$

which lies in the range $(1/2, 1]$. Note that $\exp(x) = \exp(y)^{2^{\lceil \lg |x| \rceil}}$. For $x \in (1/2, 1]$, we can directly appeal to Brent's result. For $|x| > 1$, let $s' = s + |x|$. Applying Brent's result, we can compute a value $z$ which approximates $\exp(y)$ to precision $s'$ in time $O(\mu(s')\lg s')$. Next we square $z$ repeatedly for a total of $\lceil \lg x \rceil$ times, each time taking care to truncate the result to precision $s'$. We then argue that the final result approximates $\exp(x)$ to precision $s$. For $|x| \leq 1/2$, we compute square root of $z$ repeatedly for a total of $-\lceil \lg x \rceil$ times, each time taking care to truncate the result to some suitable precision.

The upshot of a series of arguments similar to the above eventually leads us to:

**Lemma 3** *Assuming that we can compute $E, F$ to any precision $r$ in time $T_E(r)$, $T_F(r)$, respectively, then we can solve problem (P1) in time*

$$O(T_E(s+L_2)+T_F(s+L_2)+\mu(s+L_2)\log(s+L_2)+\mu(s+\lg L_0)(L_0+\log(s+\lg L_0))),$$

*where we write $L_0$ for $L_1 + L_2$. If $L_0 = O(s)$, this simplifies to $O(T_E(s) + T_F(s) + \mu(s)s)$.*

**Proof.** See Appendix. □

## 5. Configuration Obstacles

Fix any two edges $e_1, e_2$ of obstacles. Write $e_i = [p_i, q_i]$ and let us parameterize the points on $span(e_i)$ such that $xp_i + (1-x)q_i$ is denoted by $e_i(x)$, for $x \in \mathbb{R}$. The set of pairs $(x, y)$ corresponding to a potential visibility segment $[e_1(x), e_2(y)]$ is thus parameterized by the unit square $Q = [0,1] \times [0,1]$. Of course, if $x < 0$ or $x > 1$, the notation $e_i(x)$ is still meaningful and refers to a point on $span(e_i) \setminus e_i$. Let $Q$ be embedded in $Q' = \mathbb{R}^2$, which denotes the parameter space for $span(e_1) \times span(e_2)$. For any obstacle $X \subseteq \mathbb{R}^3$, let $X^*$ denote the set of points $(x, y) \in Q'$ such that $[e_1(x), e_2(y)]$ intersects the relative interior of $X$. We call $X^*$ the *configuration obstacle* corresponding to $X$.

As in Section 2, it is enough to understand the configuration obstacle $H^*$ where $H$ is a half-plane. Assume that $\partial H$ contains the edge $[u, v]$ of a triangular face. Let $\ell(x, y)$ be the line passing through the points $e_1(x), e_2(y)$. Consider the simplex

$$\triangle = \triangle(u, v, e_1(x), e_2(y)).$$

[b] We write 'log' when the actual base is irrelevant. But $\lg = \log_2$ and $\ln = \log_e$.

The requirement that $\ell(x,y)$ intersects $\partial H$ amounts to saying $\mathrm{Vol}(\triangle) = 0$, where

$$\begin{aligned}
2 \cdot \mathrm{Vol}(\triangle) &= \det \begin{pmatrix} u & v & e_1(x) & e_2(y) \\ 1 & 1 & 1 & 1 \end{pmatrix} \\
&= -\det(v-u \mid e_1(x)-u \mid e_2(y)-u) \\
&= -\det(v-u \mid (q_1-u)+x(p_1-q_1) \mid (q_2-u)+y(p_2-q_2)) \\
&= -\det(v-u \mid p_1-q_1 \mid p_2-q_2)xy - \det(v-u \mid q_1-u \mid p_2-q_2)y - \\
&\quad \det(v-u \mid p_1-q_1 \mid q_2-u)x - \det(v-u \mid q_1-u \mid q_2-u) \\
&= -Axy - By - Cx - D. \qquad (5.1)
\end{aligned}$$

The coefficients $A, B, C, D$ are defined as the corresponding determinants. If $A \neq 0$, we may rewrite the equation in a standard form for hyperbolas:

$$(Ax+B)(Ay+C) = -AD + BC.$$

### *5.1. Degenerate Position*

We say the two edges $e_1, e_2$ are in *degenerate position* with respect to $H$ if $A = 0$ or $(A \neq 0) \wedge (AD - BC = 0)$. Let us analyze two edges in degenerate position.

If $A = 0$, then

$$A = -\det(v-u \mid p_1-q_1 \mid p_2-q_2) = +\det \begin{pmatrix} u & v & u+p_1-q_1 & u+p_2-q_2 \\ 1 & 1 & 1 & 1 \end{pmatrix} = 0.$$

Therefore the points $u, v, u+p_1-q_1, u+p_2-q_2$ are on the same plane, i.e., the two edges $e_1, e_2$ are in two parallel planes (or on the same plane) that are parallel with the line $\partial H$.

Let us consider the second case. Using the relation (see Ref. [10])

$$\det(a \mid b \mid c)\det(a \mid d \mid e) - \det(a \mid b \mid e)\det(a \mid d \mid c) = \det(a \mid b \mid d)\det(a \mid c \mid e)$$

for $a, b, c, d, e \in \mathbb{R}^3$, we get

$$AD - BC = \det \begin{pmatrix} u & v & u+p_1-q_1 & q_1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \cdot \det \begin{pmatrix} u & v & u+p_2-q_2 & q_2 \\ 1 & 1 & 1 & 1 \end{pmatrix}.$$

Thus if $AD - BC = 0$, then either $span(e_1)$ or $span(e_2)$ is coplanar with $\partial H$.

Suppose $e_1, e_2$ is not in degenerate position with respect to $H$. It can be shown that the sign of $AD-BC$ depends on the direction of the vectors $p_i - q_i$ with respect to $H$ and on whether $span(e_i)$ intersects $H$, for $i = 1, 2$.

### *5.2. Configuration Obstacles*

We are ready to describe $H^* \subseteq Q'$. Let us define two points $(x_0, y_0), (x_1, y_1) \in Q'$ as follows: $span(H)$ intersects $span(e_1)$ at $e_1(x_0)$ and intersects $span(e_2)$ at $e_2(y_0)$. Let $P_i$ be the plane containing $\partial H$ and parallel to $e_i$. Then $P_1$ intersects $span(e_2)$ at $e_2(y_1)$ and $P_2$ intersects $span(e_1)$ at $e_1(x_1)$. Assuming non-degeneracy,

these intersection points are well-defined. Furthermore, imagine $span(H)$ to be the horizontal plane $\{Z = 0\}$ in physical $(X, Y, Z)$-space, and that $p_i - q_i$ have a positive $Z$-component. The vertical and horizontal lines through $(x_0, y_0)$ divide $Q'$ into four quadrants (I), (II), (III) and (IV). Note that a pair of quadrants (I, III) or (II, IV) is always disjoint with $H^*$. By symmetry, we may assume that $e_1, e_2$ are parameterized to increase in the positive $Z$ direction. Then the first and third quadrants are necessarily be disjoint from $H^*$. Let us consider one possibility (the other cases are similarly treated), where the line $\partial H$ separates $e_1(x_0)$ and $e_2(y_0)$ on the plane $span(H)$ and $e_1$ intersects with $H$. We may assume $(x_1, y_1)$ lies in quadrant (I) (the other case is quadrant (III), which is symmetric). See Figure 1. The asymptotes of the hyperbola in this case are the vertical and horizontal lines through $(x_1, y_1)$. Furthermore, $(x_0, y_0)$ lies on the hyperbola. Suppose the hyperbola lies to the northeast and southwest of $(x_1, y_1)$. Then $H^*$ is shown as shaded in Figure 1. Note that only one of the two components of the hyperbola is used: this is invariably the case since one component of the hyperbola is located in a quadrant that is disjoint with $H^*$.

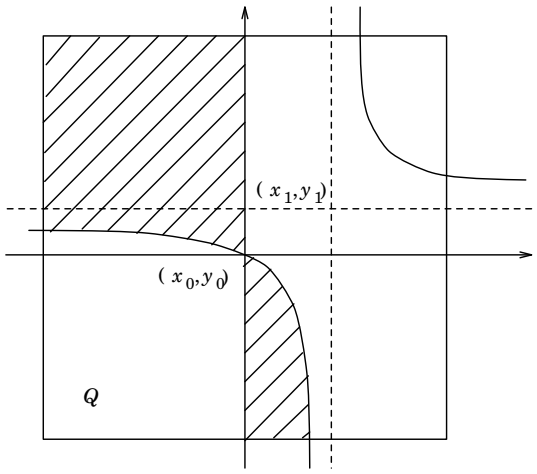


Fig. 1. Configuration obstacle for a half-space.

## 6. Construction of the Visibility Graph

In this section, we shall (like Papadimitriou) count the number of algebraic operations, to give some idea of the global complexity. To avoid confusion with the bit complexity accounting in the next section, we call each algebraic operation a "step".

Again focus on a pair of fixed edges $e_1, e_2$. The parameter space $Q$ corresponds to all potential visibility segments. The break points (these are the $p_i$'s and $q_i$'s in Section 3) on $e_1$ and $e_2$ define a grid on $Q$. For segments $\sigma_i \subseteq e_i$ $(i = 1, 2)$, there is a corresponding *grid rectangle* $R = R(\sigma_1, \sigma_2) \subseteq Q$. We say that $R$ is *covered* if and

only if

$$R(\sigma_1, \sigma_2) \setminus K^* = \emptyset,$$

where $K^*$ denotes all the configuration obstacles. Clearly $(\sigma_1, \sigma_2)$ is an edge in the visibility graph $G$ if and only if $R(\sigma_1, \sigma_2)$ is not covered.

Our goal is to decide for every grid rectangle $R$ whether or not it is covered; if it is uncovered, we will further determine a rational sample point $p$ such that a ball of some small radius $\delta$ centered at $p$ is contained in $R \setminus K^*$. Note that this determination will allow us to compute a *guaranteed* obstacle-avoiding path.

Let

$$M = \Theta(L/\epsilon_1)$$

be a bound on the maximum number of break points on any edge. Then there are at most $M^2$ grid rectangles in $Q$. We proceed slightly differently than Papadimitriou.

First we compute the arrangement of the collection of boundary arcs corresponding to all the configuration obstacles. This can be computed in $O(n\lambda_4(n))$ time where $\lambda_4(n) = O(n \cdot 2^{\alpha(n)})$ and $\alpha(n)$ is the inverse of an Ackermann-like function. This follows from the fact that any two boundary arcs can intersect at most twice, and from the incremental algorithm in Ref. [11]. We next convert this arrangement into an explicit representation of the boundary of the configuration obstacle $K^*$, say, in $O(n^2 \log n)$ steps. Finally we perform an interesting kind of line sweep over this representation of $K^*$ in $Q$. Our goal is to decide for each grid rectangle whether it is covered by $K^*$. There are three kinds of events for the vertical sweepline:

(i) Vertices on the boundary of $K^*$ (including vertical asymptotes of hyperbolic curves).

(ii) The break points $q_{\pm i}, p_{\pm i}$ on $e_1$.

(iii) Intersection points between horizontal grid lines (or horizontal asymptotes of hyperbolic curves) and boundary arcs of $K^*$. Unlike the first two types, this is dynamically inserted into the priority queue.

The state of the sweepline is represented by a balanced binary tree $T$ that contains a leaf for each of the horizontal grid lines and for each of the boundary arcs that intersect the sweepline. Whenever a grid line and a boundary arc become adjacent in $T$, we check if they will generate an event of type (iii). Whenever we come to an event of type (ii), we have just completely swept a column of at most $M$ grid rectangles and will begin to sweep a new column of at most $M$ grid rectangles. We need $O(M)$ steps to do the necessary processing, i.e., to put out the decision for each grid rectangle whether it is covered. There are $O(n^2)$ events of type (i) and $O(nM)$ events of type (iii). They can each be processed at $O(\log(M+n))$ steps. The events of type (ii) take $O(M)$ steps each, and $O(M^2)$ altogether.

## 7. Bit Complexity

We shall now determine the bit complexity of the algorithm in the preceding section. In particular, this calls for determining the precision to carry out the various tasks. We need this, for example, to make sure that we can do line sweep correctly and to be able to determine sample points in each grid rectangle $R$ that is not

covered. We shall employ a variety of classical techniques in algebraic computing.[7] It is clearly of practical interest to obtain small constants for this precision.

The upshot of the calculations below is that

$$W = O(\log(1/\epsilon_1) + L)$$

bits of precision suffices in the computation of points.

Having calculated the precision $W$, we can determine the bit complexity of our algorithm. The break points $p_{\pm i}, q_{\pm i}$ on all edges will be approximated to $O(W)$ bits, and as we shall show, these can be computed altogether in $O(nM\mu(W)W)$ time. Now let us again fix a pair of edges. To compute the arrangement of the boundary curves of $K^*$, we need to compute the intersection points between two hyperbolic arcs and/or lines to precision $W$. One readily available method is to compute the resultants (with respect to each of the two variables in turn) of the two equations, and isolate the roots to precision $W$. This can be done in $O(\mu(W))$ time.[12] The intersection points can be easily extracted from these roots. Thus we can compute the arrangement of boundary curves in time $O(n\lambda_4(n) \cdot \mu(W))$. The plane sweep to compute an explicit representation of $K^*$ takes time $O(n^2 \log n \cdot W)$.

Now consider our plane sweep algorithm in Section 6, to construct the visibility graph. Each queue comparison takes time $O(W)$, and time $O(\mu(W))$ is needed to compute the events of type (iii). Thus the operations of type (i) cost $O(n^2 \log(n+M) \cdot W)$ altogether, type (ii) cost $O(M^2 W)$ and type (iii) cost $O(nM \log(n+M) \cdot \mu(W))$.

Using the relations $n = O(M)$, $\lambda_4(n) = O(n \log n)$, and $W = O(\mu(W))$, we conclude with the main result of this section:

**Lemma 4** *The bit complexity of the algorithm in the previous section, applied to all pairs of edges, is*

$$O(n^3 M \log M \mu(W) + (nM)^2 W + nM\mu(W)W).$$

It remains to compute the precision $W$. For simplicity, we will ignore additive constants in the size of integer numbers. In the following, we talk about "separating a set of points" — this means we compute the $x$- and $y$-coordinates of these points to sufficient precision so that the relative $x$-order and $y$-order of any two points in the set are correctly determined. To guarantee that the plane sweep algorithm works correctly, we have to separate intersection points between boundary curves, intersection points between grid lines and boundary curves, and the grid points (i.e., the parameters of break points) themselves. In addition, we have to ensure that approximate grid points are computed to enough precision such that Eq. (2.1) is satisfied.

In the following calculations, we need the range of the variables used in Eq. (3.1) and Eq. (3.3).

**Lemma 5** *Let $e = [p, q]$ be an obstacle edge, with length $\ell(e)$ and distance $d > 0$ to $s_0$. Then*

*(i)* $1 \leq \ell(e) < 2^{L+2}$,
*(ii)* $2^{-L-2} < d < 2^{L+2}$,
*(iii)* $\ell(e) > d$ *implies* $\ell(e)/d > \sqrt{1+2^{-(4L+8)}}$ *and* $d/\ell(e) < 2^{-2^{-(4L+10)}}$,
*(iv)* $1 \leq 2^m < (8L+16)/\epsilon_1$.

**Proof.**
(i) Trivially $\ell(e) \geq 1$. An edge with endpoints $(-2^L, -2^L, -2^L)$ and $(2^L, 2^L, 2^L)$ has the biggest length.
(ii) The upper bound of $d$ is the same as that of $\ell(e)$. The lower bound can be proved in the following way: let $p_0$ be the closest point of $e$ from $s_0$. Without loss of generality, we assume that $s_0p_0p$ is a right-angled triangle. Then $d = ((\|p - s_0\|_2)^2 - (\|p - p_0\|_2)^2)^{1/2}$. Also $\overline{pp_0}$ is the projection of $\overline{ps_0}$ onto $span(e)$. Therefore

$$d = \sqrt{(\|p - s_0\|_2)^2 - (\frac{\overrightarrow{pq} \cdot \overrightarrow{ps_0}}{\|p - q\|_2})^2}. \tag{7.1}$$

Here $\overrightarrow{pq} \cdot \overrightarrow{ps_0}$ denotes the inner product of two vectors $\overrightarrow{pq}$ and $\overrightarrow{ps_0}$. Since $d > 0$, we have

$$\begin{aligned} d &\geq \sqrt{1/(\|p - q\|_2)^2} \\ &> 2^{-L-2}. \end{aligned}$$

(iii) We use the same argument used to prove case (ii). Note that $\ell(e)^2, d^2$ are integers. Thus

$$\begin{aligned} \ell(e)/d &= \sqrt{\ell(e)^2/d^2} \\ &= \sqrt{\ell(e)^2 \cdot \frac{(\|p - q\|_2)^2}{(\|p - s_0\|_2 \cdot \|p - q\|_2)^2 - (\overrightarrow{pq} \cdot \overrightarrow{ps_0})^2}}. \end{aligned}$$

With $\ell(e)/d > 1$, we get

$$\begin{aligned} \ell(e)/d &\geq \sqrt{1 + \frac{1}{(\|p - s_0\|_2 \cdot \|p - q\|_2)^2}} \\ &> \sqrt{1 + 2^{-(4L+8)}}. \end{aligned}$$

From $1 + x \leq \exp(x) \leq 1 + 2x$ for $0 \leq x \leq 1$, we finally get $\sqrt{1 + 2^{-(4L+8)}} > 2^{2^{-(4L+10)}}$. This proves the claim.
(iv) By definition

$$2^m < 2 \cdot \frac{\ln(\ell(e)/d)}{\ln(1 + \epsilon_1)}.$$

The claim $2^m < (8L+16)/\epsilon_1$ directly follows from (i), (ii), and $\ln(1 + \epsilon_1) \geq \epsilon_1/2$, for $0 \leq \epsilon_1 \leq 1$. □

*7.1. Precision of Break Points*

In this subsection, we shall determine the precision to which the parameters of the break points have to be computed such that they are well separated and

Eq. (2.1) is satisfied. This precision will be needed later when we calculate the precision needed to separate intersection points between boundary arcs and grid lines.

For an edge $e = [p, q]$ and each break point $p_i \in e$, let $x_i \in [0, 1]$ be the parameter of $p_i$ on $e$, i.e., $p_i = x_i p + (1 - x_i)q$.

**Lemma 6** *A precision of* $\lg(1/\epsilon_1) + 14L + 30$ *suffices to separate two parameters* $x_i$, $x_j$ *of break points.*

**Proof.** In parameter space, the gap between any two parameters $x_i$ and $x_j$ is the same as the smallest length of segments divided by the length of $e$. For each segment $\sigma \subseteq e$, let $\ell_e(\sigma) = \ell(\sigma)/\ell(e)$.

Each uniformly-divided segment $\sigma$ satisfies

$$\begin{aligned} \ell_e(\sigma) &= (1/2)d\epsilon_1/\ell(e) \\ &> (1/2)2^{-L-2}\epsilon_1 2^{-L-2} \\ &= 2^{-(\lg(1/\epsilon_1)+2L+5)}. \end{aligned}$$

The shortest exponentially-divided segment is $\sigma_1 = [p_1, p_2]$. Therefore, each exponentially-divided segment $\sigma$ satisfies

$$\begin{aligned} \ell_e(\sigma) &\geq (d(\frac{\ell(e)}{d})^{1/2^m} - d)/\ell(e) \\ &> \frac{d}{\ell(e)}((1 + 2^{-(4L+8)})^{1/2^{m+1}} - 1) \\ &\geq \frac{d}{\ell(e)}(1 + 2^{-(m+4L+10)} - 1) \\ &> \frac{1}{2^{L+2}}\frac{1}{2^{L+2}}\frac{1}{2^{m+4L+10}} \\ &\geq 2^{-(\lg(1/\epsilon_1)+14L+30)}. \end{aligned}$$

□

Similar calculations show that the same absolute precision is also sufficient to guarantee the validity of Eq. (2.1):

**Lemma 7** *If two consecutive break points* $x_i, x_{i+1}$ *for a segment* $\sigma = [p_i, p_{i+1}]$ *are computed to absolute precision* $\lg(1/\epsilon_1)+14L+30$, *then* $\ell(e)|x_{i+1} - x_i| \leq \epsilon_1 D(s_0, \sigma)$.

We conclude:

**Theorem 8** *A precision of*

$$W = O(\lg(1/\epsilon_1) + L)$$

*suffices to separate the parameters of the break points and to satisfy Eq. (2.1).*

*7.2. Computation of Break Points*

The parameter $x_i$ for $p_i$ is computed by the following equation:

$$x_i = \begin{cases} x_0 + (1/2)\epsilon_1 di/\ell(e); & p_i \text{ is uniformly-divided,} \\ x_0 + (d/\ell(e))^{1-(i-1)/2^m}; & p_i \text{ is exponentially-divided.} \end{cases} \tag{7.2}$$

The parameter $x_0$ of the origin $p_0$ of $e$ is

$$x_0 = |\frac{\overrightarrow{qs_0} \cdot \overrightarrow{qp}}{\overrightarrow{qp} \cdot \overrightarrow{qp}}|.$$

Therefore $x_0$ is a rational number that can be represented with two integers of maximum $2L$ bits.

Since we have to compute the approximation of $x_i$ to precision $W$, each term in Eq. (7.2) should be computed to precision $W+1$. Trivially, we can compute $x_0$ to this precision in $O(\mu(W))$ time.

Now let us consider the exponentially-divided break points. By Eq. (7.1), $d/\ell(e)$ is the square-root of a rational number of bit size $O(L)$, and can be approximated to precision $W$ in time $O(\mu(W)W)$. By Lemma 5, we have $2^{-2L-4} < d/\ell(e) < 2^{-2^{-4L-10}}$. Therefore by Lemma 3, we can compute the second term in $O(\mu(W)W)$ time.

This shows that we can compute the parameter $x_i$ of each exponentially-divided break point in time $O(\mu(W)W)$. Using the same arguments, it is easy to see that the same holds for the uniformly-divided break-points. Concluding, we get:

**Theorem 9** *The parameters of all break points on all edges can be computed to precision $W$ in time $O(Mn\mu(W)W)$.*

### *7.3. Separation of Intersection Points*

The approximate parameters of the break points define the grid lines in our plane sweep algorithm. In this section we calculate the precision to which we have to compute the intersection points between boundary curves and/or grid lines, such that events are scheduled properly, and such that the order of curves intersecting the sweep line is determined correctly.

Let $\alpha$ be the $x$- or $y$-coordinate of an intersection point. The precision to which we have to compute $\alpha$ is the sum of (i) the bit size of the integral part of the maximum value of $\alpha$ and (ii) the absolute precision needed to separate any two intersection points or the absolute precision needed to separate any intersection point from 0. (Note that, although we are only interested in the configuration obstacle inside the parameter space $Q' = [0,1] \times [0,1]$, the plane sweep algorithm requires us to keep track of boundary arcs (resp., intersection points) outside $Q'$.)

Let us consider the intersection between two hyperbolic functions $P_i(x,y) = A_i xy + B_i y + C_i x + D_i$, $i = 1,2$, that bound configuration obstacles. Our derivation (Eq. (5.1)) shows that $A_i, B_i, C_i, D_i$ are integers of size $3L$. By symmetry, it suffices to consider either the $x$- or the $y$-coordinates of points. The $x$-coordinates of the intersection points between $P_1(x,y)$ and $P_2(x,y)$ are roots of the resultant

$\mathrm{res}_y(P_1, P_2)$, where

$$\mathrm{res}_y(P_1(x,y), P_2(x,y)) = \det \begin{vmatrix} A_1x + B_1 & C_1x + D_1 \\ A_2x + B_2 & C_2x + D_2 \end{vmatrix}. \tag{7.3}$$

The resultant is an integer polynomial with coefficients of length $6L$. By Cauchy's upper and lower bounds on roots of polynomials,[7] $6L$ bits suffice to represent the integral part of any root, and $6L$ bits suffice to separate the roots from 0.

Next let us calculate a lower bound on the gap between the $x$-coordinates of two intersection points, assuming they are different. Let $\alpha$ be the root of the resultant $R_{(1,2)} = \mathrm{res}_y(P_1(x,y), P_2(x,y))$ and $\beta$ the root of the resultant $R_{(3,4)} = \mathrm{res}_y(P_3(x,y), P_4(x,y))$, where $P_1, P_2, P_3, P_4$ are hyperbolic functions as above. The polynomials $R_{(1,2)}$ and $R_{(3,4)}$ are integer polynomials of degree $d \leq 2$, with coefficients of size $6L$. Applying the generalized Hadamard's bound[c] (see Ref. [7])

$$|\alpha - \beta| \;\geq\; \frac{1}{2(2^{d+1}||R_{(1,2)}||_2 \cdot ||R_{(3,4)}||_2)^d}$$

to the two resultants, we get that $24L$ bits are needed to separate $\alpha$ and $\beta$. Summing up this separation gap and the maximal bit size of the integral part of $\alpha$ (resp., $\beta$) yields:

**Lemma 10** *A precision of $30L$ suffices to separate the intersection points between the hyperbolic curves.*

Next let us determine the precision needed to separate the above intersection points and the grid points. Let $R(x) = Ax^2 + Bx + C$ be the resultant defined by Eq. (7.3). Each coefficient of a grid point (i.e., the approximate parameter of a break point) can be represented by a rational number $D/E$ where $E = 2^W$ and $D$ is an integer such that $|D| \leq E$. Therefore it is the root of $Q(x) = Ex - D$. The generalized Hadamard's bound again shows that $2W + 12L$ bits suffice to separate the roots of $P(x)$ and $Q(x)$. Adding again the size of the integral part of intersection point coordinates, we conclude that a precision of $2W + 18L = O(W)$ suffices to separate intersection points of hyperbolic curves from grid points.

Proceeding in the same way for the remaining intersections, we finally get:

**Theorem 11** *If intersection points are computed with precision $O(W)$, then they are separated from each other and from the grid points.*

## 8. Coda

Having constructed the visibility graph $G = (V, A)$, we wish to compute the nominal cost of each visibility edge in $A$. Unlike the original algorithm, we do not pick the distance between midpoints of segments to serve as nominal cost. This is because these midpoints need not be rational points. However, we can pick *any* rational point in each segment instead of the midpoint. If we run Dijkstra's algorithm on $G$, assuming that we know the *exact* nominal cost of each visibility edge, then

[c] For a polynomial $P$, $||P||_2$ denotes $(\Sigma_{i=0}^{d} |p_i|^2)^{1/2}$, where $p_0, \ldots, p_d$ are the coefficients of $P$.

the original calculation shows that the length of the "approximate shortest path" $\pi$ is at most $1+\epsilon$ times the length of the actual shortest path $\pi^*$.

But, of course, our calculation only uses approximations to the nominal cost. The approximations arise because we must take square-roots. Because we choose rational points, it turns out that we only need to compute square-roots of rational numbers $x_i$ to some absolute precision $s$.

If we take $s = \lg(4n/\epsilon)$ and $\epsilon_1 = \epsilon/(24n)$, then we guarantee that the approximate length of $\pi$ is less than $(1+\epsilon)C(\pi^*)$. This can be shown as follows: let $\widehat{C(\pi)}$ be the approximate length of a path $\pi$ consisting of $k$ edges, and $\widehat{\sqrt{x_i}}$ the approximate length of the $i$-th edge of $\pi$. Then

$$\begin{aligned}
\widehat{C(\pi)} &= \Sigma_{i=1}^{k} \widehat{\sqrt{x_i}} \\
&\leq \Sigma_{i=1}^{k} \sqrt{x_i} + 2n \cdot 2^{-s} \\
&\leq (1+\epsilon/2)C(\pi^*) + \epsilon/2 \\
&\leq (1+\epsilon)C(\pi^*).
\end{aligned}$$

The last inequality follows from $C(\pi^*) \geq 1$. The inequality $\Sigma_{i=1}^{k} \sqrt{x_i} \leq (1+\epsilon/2)C(\pi^*)$ can be derived analogous to Ref. [3].

Since the maximum length of each edge of $\pi$ is $2^{L+2}$, we have to compute the approximate length of each edge to precision $s + L + 2 = \lg(4n/\epsilon) + L + 2 = O(W)$. It thus takes $O(\mu(W))$ time to compute the approximate length of each edge in the visibility graph (see Appendix).

Finally, we may invoke Dijkstra's algorithm to compute the approximate shortest path $\pi$. Note that intermediate numbers are sums of at most $2n$ approximate distances, and hence they have sizes at most $O(W + \log n) = O(W)$. Each operation in Dijkstra's algorithm involving these numbers is either an addition or a comparison, and thus takes time $O(W)$. Therefore the time complexity of Dijkstra's algorithm is

$$O(N^2 \cdot W + N^2 \mu(W)),$$

where $N := nM$ is an upper bound on the number of segments in all the edges.

One final point: suppose Dijkstra's algorithm yields a sequence of segments $(s_0, \sigma_1, \sigma_2, \ldots, \sigma_k, t_0)$. For each consecutive pair $(\sigma_i, \sigma_{i+1})$ of segments, we can compute a rational *sample point* $(x_i, y_i) \in \sigma_i \times \sigma_{i+1}$ such that the segment $[x_i, y_i]$ is a visibility segment (Section 2 (III)). Using the sample points, we finally obtain an explicit zig-zag path (Section 2 (III)') that is obstacle-avoiding.

Thus we get:

**Theorem 12** *There is an approximation algorithm for ESP that computes an explicit obstacle-avoiding path with relative error $\epsilon > 0$ in*

$$O((n^3 M \log M + (nM)^2) \cdot \mu(W))$$

*time, where $W = O(\log(n/\epsilon) + L)$, and $M = O(nL/\epsilon)$.*

## 9. Conclusion

Papadimitriou's paper on approximate Euclidean shortest paths has become recognized as a key result. In this paper, we rectified several gaps in his paper, and presented for the first time, a true bit complexity bound for an $\epsilon$-approximation algorithm for ESP.

It should be clear from our development that there are many tedious and non-obvious details when we do such approximations. The result is an interesting mix of combinatorial, numerical and algebraic complexity. Many of these tools are, in fact, those that are necessary in order to make "exact computation" a practical reality.[6]

In our solution, we tried to extract some general lemmas about approximate computing with floating point numbers, to facilitate future applications of such an approach. Note that one could avoid Brent's result by approximating Taylor's expansion directly, but this is expected to yield an inferior bit complexity.

We may generally call this the method of *generalized grids*. Of course, grids are a familiar practical technique in all of computational sciences. From a complexity theoretic viewpoint, such methods have been shunned in the past as trivial or uninteresting. This need not be so, as Papadimitriou's work has demonstrated. In fact, grid methods may be the most practical recourse for solving some intractable problems. It would be interesting to derive some general theorems about these approaches.

## References


1. J. Hershberger and S. Suri, "Efficient computation of Euclidean shortest paths in the plane," *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci.*, 1993, pp. 508–517. An improved $O(n \log n)$ bound was recently announced.
2. J. Canny and J. H. Reif, "New lower bound techniques for robot motion planning problems," *Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci.*, 1987, pp. 49–60.
3. C. H. Papadimitriou, "An algorithm for shortest-path motion in three dimensions," *Inform. Process. Lett.* **20** (1985) 259–263.
4. T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, **22** (1979) 560–570.
5. K. L. Clarkson, "Approximation algorithms for shortest path motion planning," *Proc. 19th Annu. ACM Sympos. Theory Comput.*, 1987, pp. 56–65.
6. C. -K. Yap, "Towards exact geometric computation," *Proc. Fifth Canadian conference on computational geometry*, 1993, pp. 405–419. Invited Talk.
7. C. -K. Yap, *Fundamental Problems in Algorithmic Algebra* (Princeton Univ. Press, to appear.). Available on request from author (and via anonymous ftp).
8. R. Brent, "Fast multiple-precision evaluation of elementary functions," *Journal of the ACM* **23** (1976) 242–251.
9. J. Choi and J. Sellen and C. -K. Yap, "Precision-sensitive Euclidean shortest path in 3-space," *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995.
10. H. Lass, *Vector and tensor analysis* (McGraw-Hill, 1950).
11. H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel and M. Sharir, "Arrangements of curves in the plane: topology, combinatorics, and algorithms," *Theoret. Comput. Sci.* **92** (1992) 319–336.

12. V. Pan, "Sequential and parallel complexity of approximate evaluation of polynomial zeros," *Comput. Math. Appl.* **14** (1987) 591–622.

13. A. Aho, J. Hopcroft and J. Ullman, *The design and analysis of computer algorithms* (Addition-Wesley Publishing Company, 1976).

## Appendix A. Approximate Computation of Expressions

Brent[8] showed that elementary functions ($\exp, \ln$, etc.) can be evaluated to precision $s$ in $O(\mu(s)\log s)$ time for any floating point number $x$ in a suitable finite interval. In this paper we have to evaluate various kinds of numerical expressions to precision $s$ for floating point numbers $x$ in a variable range. To decide the time complexity of computing these expressions, we use Brent's result on computing $\exp(x)$ and $\ln(x)$. The basic idea can be sketched as follows:

(i) Transform $x$ into another floating point number $f(x)$ in a fixed range by some scaling function $f$. Using our floating point representation, $f(x)$ may be defined as the fraction of $x$.

(ii) Apply Brent's result to evaluate $\exp(f(x)), \ln(f(x))$.

(iii) Rescale $\exp(f(x)), \ln(f(x))$ by a function $f'$ with $f'(\exp(f(x))) = \exp(x)$, $f'(\ln(f(x))) = \ln(x)$, respectively. The function $f'$ basically is a sequence of either multiplications or square-root computations.

Notice that the multiplications or square-roots in step (iii) have to be evaluated for numbers in a variable range. Here, the floating point representation turns out to be particularly useful.

### *A.1. Preliminaries*

In this subsection, we review some results on computing elementary functions, and extend the basic operations to the context of arbitrary precision floating point numbers.

In our calculations, it is sometimes useful to switch between the use of (relative) precision and absolute precision. First, let us identify the relationship between both notations:

**Proposition A.1** *For a numerical expression $E$, let $\hat{E}$ be a floating point number that approximates $E$ to precision $s$. Then $\hat{E}$ approximates $E$ to absolute precision $s-(\lfloor \lg|E| \rfloor + 1)$. Conversely, computing $E$ to absolute precision $s$ means computing $E$ to relative precision $s + (\lfloor \lg|E| \rfloor + 1)$.*

To compute basic operations, we shall utilize the following well known results on computing reciprocals and square-roots:

**Proposition A.2** [8,13] *Let $[a,b]$ be a fixed range. Then there exists an algorithm that, for any floating point number $x \in [a,b]$,*

*(i) evaluates $1/x$ to precision $s$ in $O(\mu(s))$ time.*

*(ii) evaluates $\sqrt{x}$ to precision $s$ in $O(\mu(s))$ time.*

**Lemma A.3** *Let $x, y$ be floating point numbers with $l$-bit exponent and $m$-bit fraction. Then there exist algorithms that*
*(i) evaluate $x + y$ to precision $s$ in $O(s + l)$ time.*
*(ii) evaluate $x \cdot y, 1/x, \sqrt{x}$ to precision $s$ in $O(\mu(s) + l)$ time.*

Note that $m$ is irrelevant in these bounds.

**Proof.** Let $x = \langle e_x, f_x \rangle$ and $y = \langle e_y, f_y \rangle$. Without loss of generality, we assume $e_x \leq e_y$.
(i) For $x + y$, we first have to normalize $x$ so that the exponent of $x$ is equal to the exponent of $y$. The following two cases occur: (1) If $e_y - e_x \leq s$, then we attach $e_y - e_x$ leading 0's to $f_x$ and add up the two numbers to precision $s$. This operation takes $O(s + l)$ time. (2) If $e_y - e_x > s$, then the result of the addition equals $y$ to precision $s$. Trivially this operation takes $O(s + l)$ time.
(ii) We have $x \cdot y = \langle e_x + e_y, f_x \cdot f_y \rangle$. Therefore multiplication takes $O(\mu(s) + l)$ time, where $f_x \cdot f_y$ to precision $s$ takes $O(\mu(s))$ time and $e_x + e_y$ takes $O(l)$ time.
(iii) If $f_x \neq 0$, then $1/x = 1/\langle 0, f_x \rangle \cdot 2^{-e_x}$. Therefore computing $1/x$ to precision $s$ takes $O(\mu(s) + l)$ time, where computing $1/\langle 0, f_x \rangle$ to precision $s$ takes $O(\mu(s))$ time by Proposition A.2, and the addition of exponents takes $O(l)$ time.
(iv) We have

$$\sqrt{x} = \begin{cases} \sqrt{\langle 0, f_x \rangle} \cdot 2^{e_x/2} & \text{if } e_x \text{ is even,} \\ \sqrt{\langle 1, f_x \rangle} \cdot 2^{(e_x - 1)/2} & \text{if } e_x \text{ is odd.} \end{cases}$$

Therefore by Proposition A.2, computing $\sqrt{x}$ to precision $s$ takes $O(\mu(s) + l)$ time. □

For the exponential and the logarithmic function, Brent showed:

**Proposition A.4** [8] *Let $[a, b]$ be a fixed range, with, for the case (ii) below, $0 \notin [a, b]$. Then there exists an algorithm that, for any floating point number $x \in [a, b]$,*
*(i) evaluates $\exp(x)$ to precision $s$ in $O(\mu(s) \log s)$ time.*
*(ii) evaluates $\ln(x)$ to precision $s$ in $O(\mu(s) \log s)$ time.*

In the following subsections, we show how to compute $\exp(x)$ and $\ln(x)$ for arbitrary floating point numbers $x$. Finally, we use these results to compute $E^F$ for numerical expressions $E$ and $F$, as needed in Section 7.

### *A.2. The Logarithmic Function*

For a given floating point number $x$, we want to compute $\ln x$ to precision $s$ (i.e., to absolute precision $s_3 = s - (\lfloor \lg |\ln x| \rfloor + 1)$). If $x$ is in a fixed range $[a, b]$, then we will use Proposition A.4. Otherwise we will compute $\ln x$ by scaling as follows: let $y = x/2^{\lceil \lg x \rceil}$, then $y \in (1/2, 1]$. Therefore

$$\begin{aligned} \ln x &= \ln(y \cdot 2^{\lceil \lg x \rceil}) \\ &= \ln y + \lceil \lg x \rceil \ln 2. \end{aligned} \tag{A.1}$$

Let $\widehat{\ln y}$ and $\widehat{\ln 2}$ be approximations of $\ln y$ and $\ln 2$ to absolute precision $s_1$ and $s_2$, respectively:

$$\begin{aligned}\widehat{\ln y} &= \ln y + \epsilon_1, \quad |\epsilon_1| < 2^{-s_1},\\ \widehat{\ln 2} &= \ln 2 + \epsilon_2, \quad |\epsilon_2| < 2^{-s_2}.\end{aligned}$$

Let $\widehat{\ln x} = \widehat{\ln y} + \lceil \lg x \rceil \widehat{\ln 2}$. Then

$$\begin{aligned}|\widehat{\ln x} - \ln x| &= |\ln y + \epsilon_1 + \lceil \lg x \rceil (\ln 2 + \epsilon_2) - \ln x|\\ &= |\epsilon_1 + \epsilon_2 \lceil \lg x \rceil|\end{aligned}$$

We want to bound $\epsilon_1$ and $\epsilon_2$ such that

$$|\widehat{\ln x} - \ln x| < 2^{-s_3}.$$

To achieve this, we may choose

$$|\epsilon_1| < 2^{-s_3-1}, |\epsilon_2| < \frac{2^{-s_3-1}}{|\lceil \lg x \rceil|} = 2^{-s_3-\lg(|\lceil \lg x \rceil|)-1}.$$

Now let us consider the time complexity to compute $\ln(x)$ to precision $s$:
(i) Computing $\lceil \lg x \rceil$ and $y$ takes $O(|\lg x|)$ time.
(ii) Computing $\ln y$ to absolute precision $s_3 + 1$ (i.e., to precision $s - \lfloor \lg |\ln x| \rfloor$) takes $O(\mu(s - \lg |\ln x|) \log(s - \lg |\ln x|))$ time by Proposition A.4.
(iii) Computing $\ln 2$ to absolute precision $s_3 + \lg(|\lceil \lg x \rceil|) + 1$ (i.e., to precision $s - \lfloor \lg |\ln x| \rfloor + \lg(|\lceil \lg x \rceil|) + 1 = O(s)$) takes $O(\mu(s) \log s)$ time by Proposition A.4.

As the time for evaluating Eq. (A.1) is dominated by the above complexities, we arrive at running time $O(l + \mu(s') \log s')$, where $s' = \max\{s, s - \lg |\ln x|\}$. In slightly simplified form:

**Lemma A.5** *Let $x$ be a floating point number with $l$-bit exponent and $m$-bit fraction. Then we can compute $\ln(x)$ to precision $s$ in time*

$$O(l + \mu(s + \log l) \log(s + \log l)).$$

*A.3. The Exponential Function*

Our scheme of scaling and rescaling will require us to compute successively squares or square-roots. During this process, we have to keep track of the error. As a prerequisite to the following calculations, we consider two recursive functions $Y, Z$ defined as follows:

$$\begin{aligned}Y(x, \epsilon, k) &= \begin{cases} \exp(x) + \epsilon & \text{if } k = 0,\\ Y(x, \epsilon, k-1)^2 + \epsilon & \text{if } k > 0.\end{cases}\\ Z(x, \epsilon, k) &= \begin{cases} \exp(x) + \epsilon & \text{if } k = 0,\\ Z(x, \epsilon, k-1)^{1/2} + \epsilon & \text{if } k > 0.\end{cases}\end{aligned}$$

**Lemma A.6** *For $x \in (1/2, 1]$, $0 \leq \epsilon < 1$, and a non-negative integer $k$, the functions $Y$ and $Z$ satisfy*
*(i)* $Y(\pm x, \epsilon, k) \leq \exp(\pm x)^{2^k} + 2^{2^{k+2}-4}\epsilon$,
*(ii)* $Z(\pm x, \epsilon, k) \leq \exp(\pm x)^{1/2^k} + (k+1)\epsilon$.

**Proof.** We prove the inequalities by induction on $k$. For $k = 0$, the claim can be checked directly. Assuming the claim holds for $k \leq i$, consider $k = i + 1$:

$$\begin{aligned} Y(\pm x, \epsilon, i+1) &= Y(\pm x, \epsilon, i)^2 + \epsilon \\ &\leq (\exp(\pm x)^{2^i} + 2^{2^{i+2}-4}\epsilon)^2 + \epsilon \\ &< \exp(\pm x)^{2^{i+1}} + 2^{2^{i+3}-4}\epsilon. \end{aligned}$$

$$\begin{aligned} Z(\pm x, \epsilon, i+1) &= Z(\pm x, \epsilon, i)^{1/2} + \epsilon \\ &\leq (\exp(\pm x)^{\frac{1}{2^i}} + (i+1)\epsilon)^{1/2} + \epsilon \\ &< \exp(\pm x)^{\frac{1}{2^{i+1}}} + (i+2)\epsilon. \end{aligned}$$

□

To compute $\exp(x)$, we proceed as in the previous subsection. Recall that computing $\exp(x)$ to precision $s$ means computing $\exp(x)$ to absolute precision $s' = s - (\lfloor \lg \exp(x) \rfloor + 1)$. We scale as follows: let $y = x/2^{\lceil \lg x \rceil}$, then $y \in (1/2, 1]$, and

$$\exp(x) = \exp(\frac{x}{2^{\lceil \lg x \rceil}})^{2^{\lceil \lg x \rceil}} = \exp(y)^{2^{\lceil \lg x \rceil}}. \tag{A.2}$$

**Case 1:** Let $x > 1$. We define a recursive function $Y(k)$, $k \geq 0$, as follows:

$$\begin{aligned} Y(0) &= \exp(y) + \epsilon_0, \\ Y(k+1) &= Y(k)^2 + \epsilon_{k+1}. \end{aligned}$$

For $k = \lceil \lg x \rceil$, we want to bound $\epsilon_i$, $0 \leq i \leq k$, such that

$$|Y(k) - \exp(y)^{2^k}| < 2^{-s'}. \tag{A.3}$$

Let us define $\epsilon = \max\{|\epsilon_0|, |\epsilon_1|, \ldots, |\epsilon_k|\}$. Then by Lemma A.6, we have

$$\begin{aligned} |Y(k) - \exp(y)^{2^k}| &= |(\cdots((\exp(y) + \epsilon_0)^2 + \epsilon_1)^2 + \cdots)^2 + \epsilon_k - \exp(y)^{2^k}| \\ &\leq |Y(y, \epsilon, k) - \exp(y)^{2^k}| \\ &\leq 2^{2^{k+2}-4}\epsilon. \end{aligned}$$

Thus, in order to satisfy Eq. (A.3), it suffices to bound each $\epsilon_i$, $1 \leq i \leq \lceil \lg x \rceil$, by $2^{-s'-2^{\lceil \lg x \rceil + 2}+4}$.

Let us consider the time complexity to compute $\exp(x)$, $x > 1$:
(i) Computing $\lceil \lg x \rceil$ and $y$ takes $O(\lg x)$ time.
(ii) Computing $\exp(y)$ to absolute precision $s' + 2^{\lceil \lg x \rceil + 2} - 4$ (i.e., to precision

$O(s+x))$ takes $O(\mu(s+x)\log(s+x))$ time by Proposition A.4.
(iii) By Eq. (A.2), we have to square the result from (ii) $\lceil \lg x \rceil$ times. By the above analysis, each square function should be computed to precision $O(s+x)$. As the size of exponents does not exceed $O(x)$, the whole process takes time $O((\mu(s+x)+x)\log x)$ (see Lemma A.3).

Therefore, in total, computing $\exp(x), x > 1$, takes time $O(\mu(s+x)[\log x + \log(s+x)])$.

**Case 2:** Let $x \in (0, 1/2]$. Analogous to the case $x > 1$, we define a recursive function $Z(k)$ by

$$\begin{aligned} Z(0) &= \exp(y) + \epsilon_0, \\ Z(k+1) &= Z(k)^{1/2} + \epsilon_{k+1}. \end{aligned}$$

Note that $\lg x < 0$. Thus for $k = -\lceil \lg x \rceil$, we want to bound $\epsilon_i$, $0 \le i \le k$, such that

$$|Z(k) - \exp(y)^{\frac{1}{2^k}}| < 2^{-s'}. \tag{A.4}$$

Again let us define $\epsilon = \max\{|\epsilon_0|, |\epsilon_1|, \ldots, |\epsilon_k|\}$. Then by Lemma A.6, we have

$$\begin{aligned} |Z(k) - \exp(y)^{\frac{1}{2^k}}| &= |(\cdots((\exp(y)+\epsilon_0)^{1/2} + \epsilon_1)^{1/2} + \cdots)^{1/2} + \epsilon_k - \exp(y)^{\frac{1}{2^k}}| \\ &\le |Z(y, \epsilon, k) - \exp(y)^{\frac{1}{2^k}}| \\ &\le |(k+1)\epsilon|. \end{aligned}$$

Thus, in order to satisfy Eq. (A.4), it suffices to bound each $\epsilon_i$, $1 \le i \le -\lceil \lg x \rceil$, by $2^{-s'-\lg(-\lceil \lg x \rceil + 1)}$.

Now let us consider the time complexity to compute $\exp(x)$, $x \in (0, 1/2]$:
(i) Computing $\lceil \lg x \rceil$ and $y$ takes $O(|\lg x|)$ time.
(ii) Computing $\exp(y)$ to absolute precision $s' + \lg(-\lceil \lg x \rceil + 1)$ (i.e., to precision $O(s + \lg(-\lg x))$) takes $O(\mu(s + \lg(-\lceil \lg x \rceil))\lg(s + \lg(-\lceil \lg x \rceil)))$ time by Proposition A.4.
(iii) By Eq. (A.2), we have to take square-root of $\exp(y)$ for a total of $-\lceil \lg x \rceil$ times. By the above analysis, each square-root should be computed to precision $O(s + \lg(-\lg x))$. As each exponent has size $O(1)$, this takes time $O(\mu(s + \lg(-\lg x))(-\lg x))$.

Therefore, in total, computing $\exp(x)$, $0 < x \le 1/2$, takes time $O(\mu(s + \lg(-\lg x))\ [-\lg x + \lg(s + \lg(-\lg x))])$.

The above arguments can easily be extended to the case $x < 0$. Concluding, we get the following result, implying Lemma 2:

**Lemma A.7** *Let $x$ be a floating point number with $l$-bit exponent and $m$-bit fraction. For $|x| \ge 1$, we can compute $\exp(x)$ to precision $s$ in time*

$$O(\mu(s + |x|)\log(s + |x|)).$$

*For $0 < |x| < 1$, we can compute $\exp(x)$ to precision $s$ in time*

$$O(\mu(s + \log l) \cdot (l + \log(s + \log l))).$$

*A.4. Computation of $E^F$*

In this subsection we will prove Lemma 3. Let $E$ and $F$ be two expressions such that $2^{-L_1} \leq F \leq 1$ and $-L_2 \leq \lg E \leq -2^{-L_2}$, where $L_1, L_2$ are positive integers. The goal is to evaluate $E^F$ to precision $s$ (i.e., to absolute precision $s' = s - (\lfloor \lg |E^F| \rfloor + 1) \leq s + L_2$). We use our previous results on evaluating $\ln(x)$ and $\exp(x)$ by computing $E^F = \exp(F \ln E)$, according to the straight line program in Section 4.

Let $Y$ be the result of this computation. Then

$$Y = \exp((F + \epsilon_1)(\ln(E + \epsilon_2) + \epsilon_3) + \epsilon_4) + \epsilon_5.$$

We have to bound $|\epsilon_1|, |\epsilon_2|, |\epsilon_3|, |\epsilon_4|, |\epsilon_5|$ such that

$$|Y - \exp(F \ln E)| < 2^{-s'}. \tag{A.5}$$

We evaluate

$$\begin{aligned}
&|Y - \exp(F \ln E)| \\
&\quad \leq |\exp((F + |\epsilon_1|)(\ln E + \ln(1 + |\epsilon_2|/E) + |\epsilon_3|) + |\epsilon_4|) + |\epsilon_5| - \exp(F \ln E)| \\
&\quad \leq |\exp((F + |\epsilon_1|)(\ln E + |\epsilon_2|/E + |\epsilon_3|) + |\epsilon_4|) - \exp(F \ln E)| + |\epsilon_5|.
\end{aligned}$$

Let $|\epsilon_5| < 2^{-s'-1}$. Then it remains to bound $|\epsilon_1|, |\epsilon_2|, |\epsilon_3|, |\epsilon_4|$ such that

$$|\exp((F + |\epsilon_1|)(\ln E + |\epsilon_2|/E + |\epsilon_3|) + |\epsilon_4|) - \exp(F \ln E)| < 2^{-s'-1}.$$

Simplifying this by using $\epsilon = \max\{|\epsilon_1|, |\epsilon_2|, |\epsilon_3|, |\epsilon_4|\}$, we get

$$\begin{aligned}
&|\exp((F + |\epsilon_1|)(\ln E + |\epsilon_2|/E + |\epsilon_3|) + |\epsilon_4|) - \exp(F \ln E)| \\
&\quad \leq |\exp(F \ln E + \epsilon F/E + \epsilon F + \epsilon \ln E + \epsilon^2/E + \epsilon^2 + \epsilon) - \exp(F \ln E)| \\
&\quad \leq |\exp(F \ln E)(1 + \epsilon 2^{L_2+8} - 1)| \\
&\quad \leq 2^{L_2+8}\epsilon.
\end{aligned}$$

Thus if $\epsilon < 2^{-s'-L_2-9}$, then Eq. (A.5) is satisfied.

Now let us consider the time complexity to compute $E^F$. Let us assume that computing $E$ and $F$ to precision $r$ takes $O(T_E(r))$ and $O(T_F(r))$ time, respectively.
(i) From the above analysis, we have to compute $E$ to absolute precision $s' + L_2 + 9 = s - (F|\lg E| + 1) + L_2 + 9$ (i.e., to precision $O(s + L_2)$). This takes $O(T_E(s + L_2))$ time.
(ii) Also from the above analysis, we have to compute $F$ to absolute precision $s' + L_2 + 9 = s - (F|\lg E| + 1) + L_2 + 9$ (i.e., to precision $O(s + L_2)$). This takes $O(T_F(s + L_2))$ time.
(iii) By Lemma A.5, computing $\ln E$ to absolute precision $s' + L_2 + 9$ (i.e., to precision $O(s + L_2)$) takes $O(\mu(s + L_2)\log(s + L_2))$ time.
(iv) After that, we multiply the approximate values for $F$ and $\ln E$ to absolute precision $s' + L_2 + 9$ (i.e., to precision $O(s + L_2)$). This takes $O(\mu(s + L_2) + L_1 + L_2)$

time by Lemma A.3.
(v) Finally, computing $Y$ to precision $s$ takes time $\max\{\mu(s+L_2)\log(s+L_2), \mu(s+\lg(L_1+L_2))((L_1+L_2)+\log(s+\lg(L_1+L_2))\}$ by Lemma A.7 (note that $-L_2 \leq F\ln E \leq -2^{-(L_1+L_2)}$).

We conclude:

**Lemma A.8** *Let $E, F$ be two expressions with $2^{-L_1} \leq F \leq 1$ and $-L_2 \leq \lg E \leq -2^{-L_2}$. If we can compute $E$ to precision $r$ in time $T_E(r)$ and $F$ to precision $r$ in time $T_F(r)$, then we can compute $E^F$ to precision $s$ in time*

$$\begin{aligned} O(\quad & T_E(s+L_2) + T_F(s+L_2) + \\ & \mu(s+L_2)\log(s+L_2) + \\ & \mu(s+\lg(L_1+L_2))((L_1+L_2)+\log(s+\lg(L_1+L_2))) \ ). \end{aligned}$$