Discrete Comput Geom (2008) 39: 17–37
DOI 10.1007/s00454-007-9019-9

Computing the Detour and Spanning Ratio of Paths, Trees, and Cycles in 2D and 3D

Pankaj K. Agarwal · Rolf Klein · Christian Knauer · Stefan Langerman · Pat Morin · Micha Sharir · Michael Soss

Received: 19 April 2005 / Revised: 1 June 2006 /
Published online: 3 October 2007

Abstract The detour and spanning ratio of a graph G embedded in $\mathbb{E}^d$ measure how well G approximates Euclidean space and the complete Euclidean graph, respectively. In this paper we describe $O(n \log n)$ time algorithms for computing the detour and spanning ratio of a planar polygonal path. By generalizing these algorithms, we

This research was partly funded by CRM, FCAR, MITACS, and NSERC. P.A. was supported by NSF under grants CCR-00-86013 EIA-99-72879, EIA-01-31905, and CCR-02-04118, by ARO grants W911NF-04-1-0278 and DAAD19-03-1-0352, and by a grant from the U.S.-Israeli Binational Science Foundation. R.K. was supported by DFG grant Kl 655/14-1. M.S. was supported by NSF Grants CCR-97-32101 and CCR-00-98246, by a grant from the U.S.-Israeli Binational Science Foundation (jointly with P.A.), by a grant from the Israeli Academy of Sciences for a Center of Excellence in Geometric Computing at Tel Aviv University, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

Some of these results have appeared in a preliminary form in [2, 20].

P.K. Agarwal (✉)
Department of Computer Science, Duke University, Durham, NC 27708-0129, USA
e-mail: pankaj@cs.duke.edu

R. Klein
Institut für Informatik I, Universität Bonn, Römerstraße 164, 53117 Bonn, Germany
e-mail: rolf.klein@uni-bonn.de

C. Knauer
Institut für Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany
e-mail: knauer@inf.fu-berlin.de

S. Langerman
FNRS, Département d'Informatique, Université Libre de Bruxelles, ULB CP212, Boulevard du Triomphe, 1050 Bruxelles, Belgium
e-mail: sl@cgm.cs.mcgill.ca

P. Morin
School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, ON K1S 5B6, Canada
e-mail: morin@cs.carleton.ca

obtain $O(n \log^2 n)$-time algorithms for computing the detour or spanning ratio of planar trees and cycles. Finally, we develop subquadratic algorithms for computing the detour and spanning ratio for paths, cycles, and trees embedded in $\mathbb{E}^3$, and show that computing the detour in $\mathbb{E}^3$ is at least as hard as Hopcroft's problem.

1 Introduction

Suppose we are given an embedded connected graph $G = (V, E)$ in $\mathbb{E}^d$. Specifically, V consists of points in $\mathbb{E}^d$ and E consists of closed straight line segments whose endpoints are in V. For any two points p and q in $\bigcup_{e \in E} e$, let $d_G(p, q)$ be the shortest path between p and q along the edges of G. The *detour* between p and q in G is defined as

$$\delta_G(p, q) = \frac{d_G(p, q)}{\|pq\|}$$

where $\|pq\|$ denotes the Euclidean distance between p and q. The *detour of* G is defined as the maximum detour over all pairs of points in $\bigcup_{e \in E} e$, i.e.,

$$\delta(G) = \sup_{p \neq q} \delta_G(p, q).$$

The challenge is in computing the detour quickly. Several cases of this generic problem have been studied in the last few years. One variant results from restricting the points p, q in the above definition to a smaller set. For example, the *spanning ratio* or *stretch factor* of G is defined as the maximum detour over all pairs of *vertices* of G, i.e.,

$$\sigma(G) = \sup_{\substack{p \neq q \\ p,q \in V}} \delta_G(p, q).$$

Such restrictions influence the nature of the problem considerably. In this paper we study both, detour and spanning ratio.

The case of G being a planar polygonal chain is of particular interest. Alt et al. [6] proved that if the detour of two planar curves is at most κ, then their Fréchet distance is at most $\kappa + 1$ times their Hausdorff distance. The Fréchet and Hausdorff distances are two commonly used similarity measures for geometric shapes [5]. Although the Hausdorff distance works well for planar regions, the Fréchet distance is more suitable to measure the similarity of two curves [5]. However, the Fréchet distance is

M. Sharir
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
e-mail: michas@tau.ac.il

M. Sharir
Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA

M. Soss
Foreign Exchange Strategy Division, Goldman Sachs, New York, USA
e-mail: soss@cs.mcgill.ca

much harder to compute [6]. A relationship between the two measures suggests that one could use the Hausdorff distance when the detours of the two given curves are bounded and small. This is the only known condition (apart from convexity) under which a linear relationship between the two measures is known.

Analyzing on-line navigation strategies also often involves estimating the detour of curves [8, 17]. Sometimes the geometric properties of curves allow us to infer upper bounds on their detour [4, 18, 24], but these results do not lead to efficient computation of the detour of the curve.

Related Work Recently, researchers have become interested in computing the detour and spanning ratio of embedded graphs. The spanning ratio of a graph G embedded in $\mathbb{E}^d$ can be obtained by computing the shortest paths between all pairs of vertices of G. Similarly, the detour of G can be determined by computing the detour between every pair of edges $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$. Although this seems to involve infinitely many pairs of points, this problem is of constant size: For each pair of points (p, q) in $e_1 \times e_2$, the *type* of the shortest connecting path $d_G(p, q)$ is determined by the two endpoints of e_1 and e_2 contained in this path. In the 2-dimensional rectangular parameter space of all positions of p and q on e_1 and e_2, classification by type induces at most four regions that are bounded by a constant number of line segments. For each region, the maximization problem can be solved in time $O(1)$, after having computed the shortest paths between all pairs of vertices of G. This approach, however, requires $\Omega(n^2)$ and $\Omega(m^2)$ time for computing the spanning ratio and detour, respectively, where n denotes the number of vertices and m is the number of edges. Surprisingly, these are the best known results for these problems for arbitrary crossing-free graphs in $\mathbb{E}^2$. Even if the input graph G is a simple path in $\mathbb{E}^2$, no subquadratic-time algorithm has previously been known for computing its detour or spanning ratio.

Narasimhan and Smid [23] study the problem of approximating the spanning ratio of an arbitrary geometric graph in $\mathbb{E}^d$. They give an $O(n \log n)$-time algorithm that computes an $(1 - \varepsilon)$-approximate value of the spanning ratio of a path, cycle, or tree embedded in $\mathbb{E}^d$. More generally, they show that the problem of approximating the spanning ratio can be reduced to answering $O(n)$ approximate shortest-path queries after $O(n \log n)$ preprocessing.

Ebbers-Baumann et al. [10] have studied the problem of computing the detour of a planar polygonal chain G with n vertices. They have established several geometric properties, the most significant of which (restated in Lemma 2.1) is that the detour of G is always attained by two mutually visible points p, q, one of which is a vertex of G. Using these properties, they develop an ε-approximation algorithm that runs in $O((n/\varepsilon) \log n)$ time. However, the existence of a subquadratic exact algorithm has remained elusive.

New Results In this paper we present randomized algorithms with $O(n \log n)$ expected running time that compute the exact spanning ratio or detour of a polygonal path with n vertices embedded in $\mathbb{E}^2$. These are the first subquadratic-time algorithms for finding the exact spanning ratio or detour, and they solve open problems posed in at least two papers [10, 23]. Our algorithm for the spanning ratio is worst-case optimal, as shown in [23], and we suspect that the algorithm for the detour is

also optimal, although we are not aware of a published $\Omega(n \log n)$ lower bound. By extending these algorithms, we present $O(n \log^2 n)$ expected time randomized algorithms for computing the detour and spanning ratio of planar cycles and trees. We can also obtain deterministic versions of our algorithms. They are more complicated and a bit slower—they run in $O(n \log^c n)$ time, for some constant c.

We also consider the problem of computing the detour and spanning ratio of 3-dimensional polygonal chains, and show that the first problem can be solved in randomized expected time $O(n^{16/9+\varepsilon})$, for any $\varepsilon > 0$ (where the constant of proportionality depends on ε), and the second problem can be solved in randomized expected time $O(n^{4/3+\varepsilon})$, for any $\varepsilon > 0$. Using the same extensions as in the planar case, this leads to subquadratic time algorithms for 3-dimensional trees and cycles. We also show that it is unlikely that an $o(n^{4/3})$-time algorithm exists for computing the detour of 3-dimensional chains, since this problem is at least as hard as Hopcroft's problem, for which a lower bound of $\Omega(n^{4/3})$, in a special model of computation, is given in [12].

Preliminary versions of this work appeared in [2, 20]; the 2-dimensional algorithm described in [20] is significantly different from the one presented here.

2 Polygonal Chains in the Plane

Let the graph $P = (V, E)$ be a simple polygonal chain in the plane with n vertices. That is, $V = \{p_0, \ldots, p_{n-1}\}$ is a set of n points in $\mathbb{E}^2$, and $E = \{[p_{i-1}, p_i] \mid i = 1, \ldots, n-1\}$. Throughout the paper, we write P when referring to the set $\bigcup_{e \in E} e$. We extend the definition of the detour from points to any two subsets A and B of P, by putting

$$\delta_P(A, B) = \sup_{\substack{a \in A, b \in B \\ a \neq b}} \delta_P(a, b),$$

which we call the *P-detour* between A and B. We also write $\delta_P(A) = \delta_P(A, A)$. Thus, $\delta(P) = \delta_P(P) = \delta_P(P, P)$ and $\sigma(P) = \delta_P(V, V)$. Since P will be fixed throughout this section, we will omit the subscript P from δ.

2.1 Overall Approach

Since computing the detour is more involved than computing the spanning ratio, we present below the algorithm for solving the detour problem. Certain modifications and simplifications, noted on the fly, turn the algorithm into one that computes the spanning ratio.

We first describe an algorithm for the decision problem for the detour: "Given a parameter $\kappa \geq 1$, determine whether $\delta(P) \leq \kappa$." Our algorithm makes crucial use of the following properties established in [10]. The proof of property (iii) is straightforward. It implies that the maximum detour is attained by a pair of co-visible points. Property (ii) ensures that one of them can be assumed to be a vertex. Together, (ii) and (iii) imply property (i).

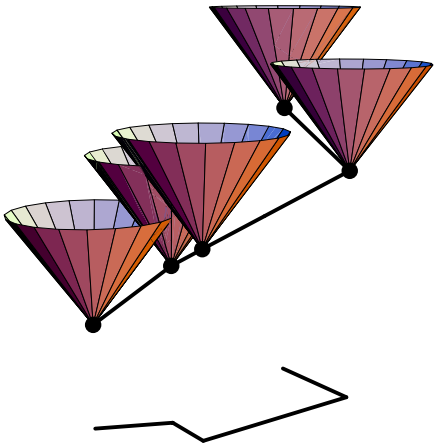

Fig. 1 Transforming P into a 3-dimensional chain

Lemma 2.1 (Ebbers-Baumann et al. [10])

(i) *Let V be the set of vertices in the polygonal chain P, and let $\kappa \geq 1$. There is a pair $(p, q) \in P \times P$ so that $\delta(p, q) > \kappa$ if and only if there is a pair $(p', q') \in P \times V$ so that $\delta(p', q') > \kappa$ and p' is visible from q'.*

(ii) *Assume that the detour attains a local maximum at two points, q, q' that are interior points of edges e, e' of P, correspondingly. Then the line segment qq' forms the same angle with e and e', and the detour of q, q' does not change as both points move, at the same speed, along their corresponding edges.*

(iii) *Let q, q' be two points on P, and assume that the line segment connecting them contains a third point, r, of P. Then $\max\{\delta(q, r), \delta(r, q')\} \geq \delta(q, q')$. Moreover, if the equality holds, then $\delta(q, r) = \delta(r, q') = \delta(q, q')$.*

We observe that a claim analogous to property (i) does not hold for the spanning ratio: while it is always attained by two vertices, by definition, these vertices need not be co-visible. As an immediate corollary of Lemma 2.1, we always have $\delta(P) = \delta(P, V)$. It thus suffices to describe an algorithm for the decision problem: *Given a parameter $\kappa \geq 1$, determine whether $\delta(P, V) \leq \kappa$*. We will then use a randomized technique by Chan [9] to compute the actual value of $\delta(P) = \delta(P, V)$.

2.2 Decision Algorithm

We orient P from p_0 to p_{n-1}. For a given parameter $\kappa \geq 1$, we describe an algorithm that determines whether for all pairs $(p, q) \in V \times P$, so that p lies before q, the inequality $\delta(p, q) \leq \kappa$ holds. By reversing the orientation of P and repeating the same algorithm once more, we can also determine whether for all pairs $(p, w) \in V \times P$ so that p lies after q the property $\delta(q, p) \leq \kappa$ is fulfilled.

For a point $p \in P$, we define the *weight* of p to be

$$\omega(p) = d_P(p_0, p)/\kappa.$$

Let C denote the cone $z = \sqrt{x^2 + y^2}$ in $\mathbb{E}^3$. We map each point $p = (p_x, p_y) \in V$ to the cone $C_p = C + (p_x, p_y, \omega(p))$. That is, we translate the apex of C (i.e., the origin)

to the point $\hat{p} = (p_x, p_y, \omega(p))$. If we regard C_p as the graph of a bivariate function, which we also denote by C_p, then for any point $q \in \mathbb{E}^2$, $C_p(q) = \|qp\| + \omega(p)$ holds. Let $\mathfrak{C} = \{C_p \mid p \in V\}$. We map a point $q = (q_x, q_y) \in P$ to the point $\hat{q} = (q_x, q_y, \omega(q))$ in $\mathbb{E}^3$. For any subchain π of P, we define $\hat{\pi} = \{\hat{q} \mid q \in \pi\}$.

Lemma 2.2 *For any point $q \in P$ and a vertex $p \in V$ that lies before q on P, $\delta(p, q) \le \kappa$ if and only if $\hat{q}$ lies below the cone C_p.*

Proof

$$\begin{aligned}
\delta(p,q) \le \kappa &\iff \frac{d_P(p,q)}{\|qp\|} \le \kappa \\
&\iff \frac{d_P(p_0,q) - d_P(p_0,p)}{\|qp\|} \le \kappa \\
&\iff \frac{d_P(p_0,q)}{\kappa} \le \|qp\| + \frac{d_P(p_0,p)}{\kappa} \\
&\iff \omega(q) \le \|qp\| + \omega(p) \\
&\iff \omega(q) \le C_p(q).
\end{aligned}$$

That is, $\delta(p, q) \le \kappa$ if and only if $\hat{q}$ lies below the cone C_p. □

Since the cones C_p are erected on the chain $\hat{P}$, the point $\hat{q}$, for any $q \in P$, always lies below all the cones erected on vertices appearing after q on P. Therefore, if we denote by V_q the set of all vertices $p \in V$ that precede q along P, Lemma 2.2 implies that $\delta(\{q\}, V_q) \le \kappa$ if and only if $\hat{q}$ lies on or below each of the cones in $\mathfrak{C}$, i.e., if and only if $\hat{q}$ lies on or below the lower envelope of $\mathfrak{C}$.

The minimization diagram of $\mathfrak{C}$, the projection of the lower envelope of $\mathfrak{C}$ onto the xy-plane, is the additive-weight Voronoi diagram $\mathrm{Vor}_\omega(V)$ of V, under the weight function ω. For a point $p \in V$, let $\mathrm{Vor}_\omega(p)$ denote the Voronoi cell of p in $\mathrm{Vor}_\omega(V)$. $\mathrm{Vor}_\omega(V)$ can be computed in $O(n \log n)$ time [13].

We first test whether $\mathrm{Vor}_\omega(p)$ is nonempty for every vertex $p \in V$. If not, we obtain a pair of vertices that attain a detour larger than κ, namely a vertex p that has an empty Voronoi cell, and a vertex q whose cone C_q passes below $\hat{p}$.

Note that if $\mathrm{Vor}_\omega(p)$ is empty for some vertex $p \in V$, then we also know that the spanning ratio of P is larger than κ. Conversely, if the spanning ratio is larger than κ, then some Voronoi cell $\mathrm{Vor}_\omega(p)$ must be empty. Thus, the decision procedure for the spanning ratio terminates after completing this step.

We can therefore assume, for the case of detour, that $\mathrm{Vor}_\omega(p)$ is nonempty for every vertex $p \in V$. To check whether $\hat{P}$ lies below the lower envelope of $\mathfrak{C}$, we proceed as follows. We partition P into a family E of maximal connected subchains so that each subchain lies within a single Voronoi cell of $\mathrm{Vor}_\omega(V)$. Since $\mathrm{Vor}_\omega(p)$ is nonempty for every vertex $p \in V$, p is the only vertex of P that lies in $\mathrm{Vor}_\omega(p)$. Therefore every subchain in E is either a segment or consists of two connected segments with p as their common endpoint. For each such segment $e \in E$, if e lies in $\mathrm{Vor}_\omega(p)$, we can determine in $O(1)$ time whether $\hat{e}$ lies fully below C_p. If this is true

for all segments, then $\hat{P}$ lies below $\mathfrak{C}$. The total time spent is $O(n)$ plus the number of segments. Unfortunately, the number of segments may be quadratic in the worst case, so we cannot afford to test them all.

We circumvent the problem of having to test all segments by using the observation (i) from Lemma 2.1 that it is sufficient to test all $q \in P$ that are visible from p. More precisely, let $\mathfrak{A}$ denote the planar subdivision obtained by overlaying $\mathrm{Vor}_\omega(V)$ with P. Each edge of $\mathfrak{A}$ is a portion of an edge of P or of $\mathrm{Vor}_\omega(V)$. For a vertex $p \in V$, let f_p denote the set of (at most *two*) faces of $\mathfrak{A}$ containing p, and let E_p denote the set of edges of $\mathfrak{A}$ that are portions of P and that bound the faces in f_p. The discussion so far implies the following lemma.

Lemma 2.3 *$\hat{P}$ lies below all the cones of $\mathfrak{C}$ if and only if $\bigcup\{\hat{e} \mid e \in E_p\}$ lies below all the cones of $\mathfrak{C}$.*

The algorithm thus proceeds as follows: We compute the Voronoi diagram $\mathrm{Vor}_\omega(V)$ in $O(n \log n)$ time [7]. By using the red-blue-merge algorithm of Guibas et al. [15] (see also [11, 25]), we compute the sets of faces f_p for all $p \in V$, which in turn gives us the sets E_p for all $p \in V$. By the Combination Lemma of Guibas et al. [15], $\sum_{p \in V} |E_p| = O(n)$, and the set $\{E_p \mid p \in V\}$ can be computed in $O(n \log n)$ time. Finally, for each edge $e \in E_p$, we determine whether $\hat{e}$ lies below C_p in $O(1)$ time. The overall running time of the algorithm is $O(n \log n)$.

As mentioned in the beginning, we next reverse the orientation of P and repeat the algorithm to determine whether for each vertex $p \in V$ lying after a point $q \in P$ the inequality $\delta(p, q) \leq \kappa$ holds. (Note that this reversal is not required in the decision procedure for the spanning ratio.) Putting everything together, we obtain the following.

Lemma 2.4 *Let P be as polygonal chain with n vertices embedded in $\mathbb{E}^2$, and let $\kappa \geq 1$ be a parameter. We can decide in $O(n \log n)$ time whether $\delta(P) \leq \kappa$ or $\sigma(P) \leq \kappa$.*

Let $W \subseteq V$ be a subset of vertices of P, and let Q be a subchain of P; set $m = |W| + |Q|$. Assuming that the weights of all vertices in W have been computed, the decision algorithm described above can be used to detect in $O(m \log m)$ time whether $\sigma(W, Q) \leq \kappa$. However, unlike $\delta(V, P)$, the detour of the entire chain P, $\delta(W, Q)$ need not be realized by a co-visible pair of points in $W \times Q$, so it is not clear how to detect in $O(m \log m)$ time whether $\delta(W, Q) \leq \kappa$. Instead we can make a weaker claim. Let $\delta^*(W, Q) = \sup_{(p,q) \in W \times Q} \delta(p, q)$, where the supremum is taken over all pairs of points such that the interior of the segment pq does not intersect the interior of an edge of Q. Obviously, $\delta^*(W, Q) \leq \delta(W, Q)$. Clearly, the above decision algorithm can detect in $O(m \log m)$ time whether $\delta^*(W, Q) \leq \kappa$. Lemma 2.1(iii) implies that if $\delta(W, Q) = \delta(P)$, then $\delta^*(W, Q) = \delta(W, Q)$, and in this special case we can detect in $O(m \log m)$ time whether $\delta(W, Q) \leq \kappa$. Hence, we obtain the following.

Corollary 2.5 *Let P be a polygonal chain with n vertices in $\mathbb{E}^2$. After $O(n)$ preprocessing, for a given subset W of vertices of P, a subchain Q of P, and a given parameter $\kappa \geq 1$, we can decide, in $O(m \log m)$ time, whether $\delta^*(W, Q) \leq \kappa$ or*

$\sigma(W, Q) \leq \kappa$, *where* $m = |W| + |Q|$. *Moreover, if* $\delta(W, Q) = \delta(P)$, *then we can also detect in* $O(m \log m)$ *time whether* $\delta(W, Q) \leq \kappa$.

2.3 Computing $\delta(P)$ and $\sigma(P)$

So far we have shown how to solve the decision problems associated with finding the detour and spanning ratio of a path. Now we apply a randomized technique of Chan [9], which does not affect the asymptotic running time of our decision algorithms, to compute the actual detour $\delta(P)$ or spanning ratio $\sigma(P)$. Suppose we have precomputed the weights of all vertices in P. Let W be a subset of vertices of P, and let Q be a subchain of P; set $m = |W| + |Q|$. We describe an algorithm that computes a pair $(\xi, \eta) \in W \times Q$ so that $\delta^*(W, Q) \leq \delta(\xi, \eta) \leq \delta(W, Q)$.

If $|W|$ or $|Q|$ is less than a prespecified constant, then we compute $\delta(W, Q)$ using a naive approach and report a pair (ξ, η) that attains it. Otherwise, we partition W into two subsets W_1, W_2 of roughly the same size, and partition Q into two subchains Q_1, Q_2 of roughly the same size. We have four subproblems (W_i, Q_j), $1 \leq i, j \leq 2$, at our hand. Note that

$$\delta(W, Q) = \max\{\delta(W_1, Q_1), \delta(W_2, Q_1), \delta(W_1, Q_2), \delta(W_2, Q_2)\}, \tag{1}$$

$$\delta^*(W, Q) \leq \max\{\delta^*(W_1, Q_1), \delta^*(W_2, Q_1), \delta^*(W_1, Q_2), \delta^*(W_2, Q_2)\}, \tag{2}$$

where (2) is an easy consequence of the visibility constraints in the definition of δ^*.

Following Chan's approach [9], we process the four subproblems in a random order and maintain a pair of points $(\xi, \eta) \in W \times Q$. Initially, we set (ξ, η) to be an arbitrary pair of points in $W \times Q$. While processing a subproblem (W_i, Q_j), for $1 \leq i, j \leq 2$, we first check in $O(m \log m)$ time whether $\delta^*(W_i, Q_j) > \delta(\xi, \eta)$, using Corollary 2.5. If the answer is yes, we solve the subproblem (W_i, Q_j) recursively and update the pair (ξ, η); otherwise, we ignore this subproblem. By (1), (2), and induction hypothesis, the algorithm returns a pair (ξ, η) such that $\delta^*(W, Q) \leq \delta(\xi, \eta) \leq \delta(W, Q)$. Moreover, if $\delta(W, Q) = \delta(P)$, then $\delta^*(W, Q) = \delta(W, Q)$, so the algorithm returns the value of $\delta(W, Q)$. Chan's analysis [9] (cf. proof of Lemma 2.1) shows that the expected running time of the algorithm on an input of size m is $O(m \log m)$. Hence, by invoking this algorithm on the pair (V, P), $\delta(V, P) = \delta(P)$ can be computed in $O(n \log n)$ expected time.

The case of the spanning ratio is handled in a similar and simpler manner, replacing (1) and (2) by

$$\sigma(W, Q) = \max\{\sigma(W_1, Q_1), \sigma(W_2, Q_1), \sigma(W_1, Q_2), \sigma(W_2, Q_2)\} \tag{3}$$

and applying Chan's technique using this relationship. Hence, we obtain the following main result of this section.

Theorem 2.6 *The detour or spanning ratio of a polygonal chain* P *with* n *vertices embedded in* $\mathbb{E}^2$ *can be computed in* $O(n \log n)$ *randomized expected time.*

Remark One can obtain an alternative *deterministic* solution that uses parametric search [22], and runs in time $O(n \log^c n)$, for some constant c. However, the resulting

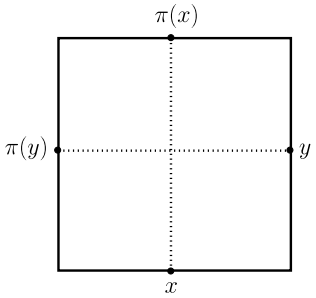

Fig. 2 *Dotted lines* indicate (the only two) pairs of points that attain the maximum detour

algorithm is considerably more involved on top of being slightly less efficient. We therefore omit its description.

We extend the definition of $\delta^*(\cdot,\cdot)$ to two disjoint subchains L and R of P as follows. Let V_L (resp. V_R) be the set of vertices in L (resp. R). Define $\delta^*(L,R) = \max\{\delta^*(V_L,R),\delta(V_R,L)\}$. Using the same argument as in the proof of Lemma 2.1, we can argue that if $\delta(L,R)=\delta(P)$, then $\delta(L,R)=\delta^*(L,R)$. The following corollary, which will be useful in the next section, is an obvious generalization of the above algorithm.

Corollary 2.7 *Let L and R be two disjoint subsets of a polygonal chain P in $\mathbb{E}^2$, with a total of n vertices, preprocessed to report weights in $O(1)$ time. Then $\sigma(L,R)$ can be computed in $O(n\log n)$ randomized expected time. We can also compute within the same time a pair $(p,q)\in L\times R$ such that $\delta^*(L,R)\le\delta(p,q)\le\delta(L,R)$. Moreover, if $\delta(L,R)=\delta(P)$, then $\delta(p,q)=\delta(L,R)$.*

As to lower bounds, it was shown by Narasimhan and Smid [23] that computing the spanning ratio of a planar polygonal chain requires $\Omega(n\log n)$ time if self-overlapping chains are allowed as input. Grüne [14] has shown that the same lower bound holds if the input is restricted to polygonal chains that are monotonic, hence simple. It is unknown whether the $\Omega(n\log n)$ lower bound also holds for computing the detour of a polygonal curve.

3 Planar Cycles and Trees

In this section we show that the tools developed for planar paths can be used for solving the detour and spanning ratio problems on more complicated graphs. Again, we consider only the problem of computing the detour, because the resulting algorithms can easily be adapted (and simplified) so as to compute the spanning ratio.

3.1 Polygonal Cycles in the Plane

Let us now consider the case in which $P=(V,E)$ is a closed (simple) polygonal curve. This case is more difficult because there are two paths along P between any two points of P. As a result, the detour of P might occur at a pair of points neither

of which is a vertex of P. For example, the detour in a unit square occurs at the midpoints of two opposite edges; in this case the lengths of the two paths between the points must be equal.

For two points $p, q \in P$, let $P[p, q]$ denote the subsets of P from p to q in counterclockwise direction. We use here the notation $d_P(p, q)$ to denote the length of $P[p, q]$; thus, in general, $d_P(p, q) \neq d_P(q, p)$ and $d_P(p, q) + d_P(q, p)$ is the length $|P|$ of the entire curve P. For a point $p \in P$, let $\pi(p)$ denote the point on P such that $d_P(p, \pi(p)) = d_P(\pi(p), p) = |P|/2$; obviously, $\pi(\pi(p)) = p$. Let P_p denote the polygonal chain $P[p, \pi(p)]$.

Lemma 3.1 *Let p be a point on P, and let A, B be two portions of P_p, then $\delta_P(A, B) = \delta_{P_p}(A, B)$.*

This follows from the fact that the shortest path along P between any two points $a, b \in A \times B$ is contained in the polygonal chain P_p.

Now the P-detour between two points $p, q \in P$ is defined as

$$\delta_P(p, q) = \frac{\min\{d_P(p, q), d_P(q, p)\}}{\|pq\|},$$

and the detour of the whole of P is defined as

$$\delta(P) = \max_{\substack{p,q \in P \\ p \neq q}} \delta_P(p, q).$$

Lemma 3.2 *The detour $\delta(P)$ of P is attained by a pair of points $p, q \in P$, such that either one of them is a vertex of P, or $q = \pi(p)$.*

Proof Suppose $\delta(P) = \delta_P(p, q)$, where neither p nor q is a vertex, and $q \neq \pi(p)$. Suppose $|P|/2 - d_P(p, q) = a > 0$. We extend, on either end, $P[p, q]$ by subpaths $P[p', p]$ and $P[q, q']$ of P, each of length $a/2$, and thereby obtain a polygonal subchain $P' = P[p', q'] \subset P$ of length $|P|/2$. Since a shortest path in P between any two points of P' is contained in P', we have

$$\delta(P) = \delta_P(p, q) \leq \delta(P') \leq \delta(P).$$

Thus, the maximum detour of P' is attained at p and q. By Lemma 2.1(ii), the detour does not change as we simultaneously move p toward p' and q toward q' at equal speed, along their edges in P'. This motion continues until one of the two points reaches a vertex of P'—which must be a vertex of P, too—or both endpoints $p', q' = \pi(p')$ of P' are reached. □

By using a rotating-caliper approach, we can compute $\max_{p \in P} \delta_P(p, \pi(p))$ in $O(n)$ time, so we focus on the case in which one of the points attaining the detour is a vertex of P. We present a different divide-and-conquer algorithm, which will use the algorithm described in Sect. 2.2 repeatedly. We can preprocess P in $O(n)$ time, so that, for any two points $p, q \in P$, $d_P(p, q)$ can be computed in $O(1)$ time.

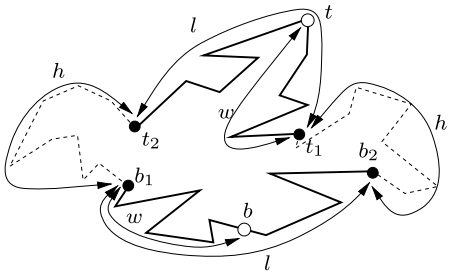

Fig. 3 An instance of the recursive problem; $d_P(t_1,t_2)=d_P(b_1,b_2)=l$, $d_P(t_2,b_1)=d_P(b_2,t_1)=h$, $|P|=2(l+h)$, $d_P(t_1,t)=d_P(b_1,b)=w$

Let t_1, t_2, b_1, b_2 be four points of P appearing in this counterclockwise order along P, so that the following condition is satisfied.

$$b_1=\pi(t_1) \quad \text{and} \quad b_2=\pi(t_2). \tag{4}$$

We observe that condition (4) implies $d_P(t_1,t_2)=d_P(b_1,b_2)$ and $d_P(t_2,b_1)=d_P(b_2,t_1)$. Let m,m' be the number of edges in $P[b_1,b_2]$ and $P[t_1,t_2]$, respectively. Define

$$\rho(t_1,t_2,b_1,b_2)=\delta_P(P[t_1,t_2],P[b_1,b_2]).$$

We describe a recursive algorithm that computes a pair of points $(p,q)\in P[b_1,b_2]\times P[t_1,t_2]$ such that $\delta(p,q)=\rho(t_1,t_2,b_1,b_2)$ if $\rho(t_1,t_2,b_1,b_2)=\delta(P)$. If $\rho(t_1,t_2,b_1,b_2)<\delta(P)$, it returns an arbitrary pair of points in $P[b_1,b_2]\times P[t_1,t_2]$.

If $\min\{m,m'\}=1$, then we can compute $\rho(t_1,t_2,b_1,b_2)$ in $O(m+m')$ time. Otherwise, suppose, without loss of generality, that $m'\geq m$, and let t be the middle vertex of $P[t_1,t_2]$ (i.e., the vertex for which each of $P[t_1,t]$, $P[t,t_2]$ has $m'/2$ edges), and let $b=\pi(t)$. It is easily seen that $b\in P[b_1,b_2]$ (by condition (4)). Clearly,

$$\rho(t_1,t_2,b_1,b_2)=\max\{\rho(t_1,t,b,b_2),\rho(t,t_2,b_1,b),\rho(t_1,t,b_1,b),\rho(t,t_2,b,b_2)\}.$$

Since $P[t_1,t]$ and $P[b,b_2]$ lie in $P[b,t]=P[\pi(t),t]$, using Corollary 2.7, we can compute in $O((m'+m)\log(m'+m))$ randomized expected time a pair $(p,q)\in P[t_1,t]\times P[b,b_2]$ so that $\delta(p,q)=\rho(t_1,t,b,b_2)$ if $\rho(t_1,t,b,b_2)=\delta(P)$. We can compute a similar pair in $P[t,t_2]\times P[b_1,b]$ within the same time bound. Each of the two 4-tuples (t_1,t,b_1,b) and (t,t_2,b,b_2) satisfies condition (4), and we solve the problem recursively for them. Among the pairs computed by the four subproblems, we return the one with the largest detour. The correctness of the algorithm is straightforward.

Let m_1 be the number of edges in $P[b_1,b]$. Then $P[b,b_2]$ contains at most $m-m_1+1$ edges. Let $T(m',m)$ denote the maximum expected time of computing $\rho(t_1,t_2,b_1,b_2)$, with the relevant parameters m' and m. Then we obtain the following recurrence:

$$T(m',m)\leq T\left(\frac{m'}{2},m_1\right)+T\left(\frac{m'}{2},m-m_1+1\right)+O((m'+m)\log(m'+m)),$$

for $m'\geq m$,

with a symmetric inequality for $m \geq m'$, and $T(m', 1) = O(m')$, $T(1, m) = O(m)$. The solution to the above recurrence is easily seen to be

$$T(m', m) = O((m' + m)\log^2(m' + m)).$$

Returning to the problem of computing $\delta(P)$, we choose a vertex $v \in P$. Let $P_1 = P[v, \pi(v)]$ and $P_2 = P[\pi(v), v]$. Then

$$\begin{aligned}\delta(P) &= \max\Big\{\max_{x,y\in P_1} \delta_P(x, y), \max_{x,y\in P_2} \delta_P(x, y), \delta_P(P_1, P_2)\Big\} \\ &= \max\{\delta(P_1), \delta(P_2), \rho(v, \pi(v), \pi(v), v)\}.\end{aligned}$$

The last equality follows from the fact that the 4-tuple $(v, \pi(v), \pi(v), v)$ satisfies (4). We can compute $\delta(P_1), \delta(P_2)$ in $O(n \log n)$ randomized expected time, using Theorem 2.6. Next we invoke the above algorithm on the 4-tuple $(v, \pi(v), \pi(v), v)$. We return the maximum of these values. If $\rho(v, \pi(v), \pi(v), v) = \delta(P)$, then the above recursive algorithm computes $\rho(v, \pi(v), \pi(v), v)$. Hence, the total expected time spent in computing $\delta(P)$ is $O(n \log^2 n)$.

The same method also applies to the computation of the spanning ratio of P, and we thus obtain:

Theorem 3.3 *The detour or spanning ratio of a polygonal cycle P with n edges in $\mathbb{E}^2$ can be computed in $O(n \log^2 n)$ randomized expected time.*

3.2 Planar Trees

Let $T = (V, E)$ be a tree embedded in $\mathbb{E}^2$. With a slight abuse of notation, we will use T to denote the embedding of the tree as well. We describe a randomized algorithm for computing $\delta(T)$. Without loss of generality, assume T is rooted at a vertex v_0 so that if we remove v_0 and the edges incident upon v_0, each component in the resulting forest has at most $n/2$ vertices; v_0 can be computed in linear time; refer to Fig. 4. We partition the children of v_0 into two sets A and B. Let T_A (resp., T_B), denote the tree induced by v_0 and all vertices having ancestors in A (resp., B). The partition A, B is chosen so that

$$\frac{1}{4}n \leq \|T_A\|, \|T_B\| \leq \frac{3}{4}n.$$

Since no descendent of v_0 is the root of a subtree with size more than $n/2$, such a partition can be found with a linear-time greedy algorithm.

We recursively compute $\delta(T_A)$ and $\delta(T_B)$. Let $\kappa^* = \max\{\delta(T_A), \delta(T_B)\}$. If $\delta(T_A, T_B) > \kappa^*$, then we need to compute $\delta(T_A, T_B)$. The following lemma, whose proof is identical to that of Lemma 2.1 given in [10], will be useful.

Lemma 3.4 *Let T_A and T_B be two subtrees of T, and let V_A (resp. V_B) be the set of vertices in T_A (resp. T_B). There exists a pair of points $(p, q) \in (V_A \times T_B) \cup (V_B \times T_A)$ such that $\delta(p, q) = \delta(T_A, T_B)$. Moreover, if $\delta(T_A, T_B) = \delta(T)$ then p is visible from q with respect to $T_A \cup T_B$.*

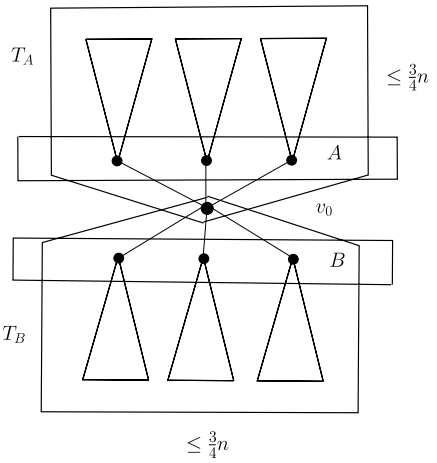

Fig. 4 Partitioning T into subtrees T_A and T_B

By Lemma 3.4, it suffices to compute $\delta(V_A, T_B)$ and $\delta(V_B, T_A)$, where V_A and V_B are the sets of vertices in T_A and T_B, respectively. As in Sect. 2, we first describe a decision algorithm that determines whether $\delta(T_A, T_B) \leq \kappa$ for some parameter $\kappa \geq \kappa^*$. We define the weight $\omega(p)$ of a point $p \in T$ to be

$$\omega(p) = \frac{d_T(p, v_0)}{\kappa}.$$

Let C be the cone $z = \sqrt{x^2 + y^2}$. To determine whether $\delta(V_A, T_B) \leq \kappa$, we map each point $u = (u_x, u_y) \in V_A$ to the cone $C_u = C + (u_x, u_y, -\omega(u))$, and map each point $v = (v_x, v_y) \in T_B$ to the point $\hat{v} = (v_x, v_y, \omega(v))$. Let $\hat{T}_B = \{\hat{v} \mid v \in T_B\}$ be the resulting tree embedded in $\mathbb{E}^3$. Following the same argument as in Lemma 2.2, we can argue that, for any $(u, v) \in V_A \times T_B$, $\delta(u, v) \leq \kappa$ if and only if $\hat{v}$ lies below the cone C_u. If $\delta(T_A, T_B) > \kappa \geq \kappa^*$, then $\delta(T_A, T_B) = \delta(T)$ and, by Lemma 3.4, there is a co-visible pair of points in $V_A \times T_B$ whose detour is greater than κ. So we can restrict our attention to co-visible pairs in $V_A \times T_B$. Using this observation and Lemma 3.4, we can determine whether $\delta(V_A, T_B) \leq \kappa$, in $O(n \log n)$ time, by the same approach as in Sect. 2. Similarly, we can determine whether $\delta(V_B, T_A) \leq \kappa$ in $O(n \log n)$ time.

Finally, returning to the problem of computing $\delta(T)$, we first use the decision algorithm to determine whether $\delta(T_A, T_B) > \kappa^*$. If the answer is no, we return κ^* and a pair of points, both from T_A or both from T_B, realizing this detour. Otherwise, $\delta(T) = \delta(T_A, T_B)$. Since each of T_A, T_B can be decomposed into two subtrees, each of size at most $3/4$ the size of T_A or T_B, respectively, we can plug this decision algorithm into Chan's technique, with the same twist as in Sect. 2, to obtain an algorithm that computes $\delta(V_A, T_B)$ in $O(n \log n)$ randomized expected time.

Putting everything together, the expected running time of the above algorithm is given by the recurrence

$$T(n) = T(n - k + 1) + T(k) + O(n \log n),$$

with $n/4 \leq k \leq 3n/4$. The recurrence solves to $O(n \log^2 n)$. (As in the case of chains, we need one preliminary global pass that computes the distances along T from v_0 to each of the vertices.)

The algorithm for computing the spanning ratio proceeds in a similar but simpler manner, as in the case of chains, and has the same randomized expected running time bound. We thus conclude the following.

Theorem 3.5 *The detour or spanning ratio of a planar tree with n vertices can be computed in $O(n \log^2 n)$ randomized expected time.*

4 Polygonal Chains, Cycles, and Trees in $\mathbb{E}^3$

Let P be a polygonal chain with n vertices embedded in $\mathbb{E}^3$. We describe subquadratic algorithms for computing the detour and spanning ratio of P, and a reduction showing that the problem of computing the detour is at least as hard as Hopcroft's problem.

4.1 Computing the Spanning Ratio

We begin with the simpler problem of computing the spanning ratio $\sigma(P)$ of P. We solve this problem by adapting the technique for computing spanning ratios in the plane, as described in Sect. 2. Specifically, consider the decision problem, where we want to determine whether $\sigma(P) \leq \kappa$. We take the set V of vertices of P, and map each $p \in V$ to the point $\hat{p} = (p, \omega(p)) \in \mathbb{R}^4$, where $\omega(p) = d_P(p_0, p)/\kappa$ and p_0 is the starting point of P. We take the cone

$$C : x_4 = \sqrt{x_1^2 + x_2^2 + x_3^2},$$

and define, for each $p \in V$, the cone C_p to be $\hat{p} + C$. As in the planar case, $\sigma(P) \leq \kappa$ if and only if each point $\hat{p}$, for $p \in V$, lies *on* the lower envelope of $\mathfrak{C} = \{C_q \mid q \in V\}$.

Let $p = (a_1, a_2, a_3)$ be a point in V, and let $\omega(p) = a_4$. A point $\xi = (\xi_1, \xi_2, \xi_3, \xi_4)$ lies below the cone

$$C_p : x_4 - a_4 = \sqrt{(x_1 - a_1)^2 + (x_2 - a_2)^2 + (x_3 - a_3)^2}$$

if and only if the point

$$\varphi(\xi) = (\xi_1, \xi_2, \xi_3, \xi_4, \xi_4^2 - \xi_1^2 - \xi_2^2 - \xi_3^2)$$

in $\mathbb{E}^5$ lies in the halfspace

$$h_p : x_5 \leq -2a_1x_1 - 2a_2x_2 - 2a_3x_3 + 2a_4x_4 + (a_1^2 + a_2^2 + a_3^2 - a_4^2).$$

Therefore a point $\xi \in \mathbb{E}^4$ lies in the lower envelope of $\mathfrak{C}$ if and only if $\varphi(\xi)$ lies in the convex polyhedron $\bigcap_{p \in V} h_p$. Hence, the problem of determining whether $\sigma(P) \leq \kappa$ reduces to locating n points in a 5-dimensional convex polyhedron defined by the intersection of n halfspaces. This problem can be solved in $O(n^{4/3+\varepsilon})$ time using a data structure for halfspace-emptiness queries [1]. Using Chan's technique, as in the planar case, we can compute $\sigma(P)$ itself within the same asymptotic time bound. Finally, as for the planar case, the algorithm can be extended to compute the spanning ratio of a polygonal cycle or tree embedded in $\mathbb{E}^3$. That is, we have shown:

Theorem 4.1 *The spanning ratio of a polygonal chain, cycle, or tree with n vertices embedded in $\mathbb{E}^3$ can be computed in randomized expected time $O(n^{4/3+\varepsilon})$, for any $\varepsilon > 0$.*

4.2 Computing the Detour

We next consider the problem of computing the detour $\delta(P)$ of P. Here the algorithm becomes considerably more involved and less efficient, albeit still subquadratic. As in some of the preceding algorithms, we use a divide-and-conquer approach to compute $\delta(P)$. That is, we partition P into two connected portions, P_1, P_2, each consisting of $n/2$ edges, recursively compute $\delta(P_1)$ and $\delta(P_2)$, and then compute explicitly the detour between P_1 and P_2, as follows. Let o be the common endpoint of P_1 and P_2. For any point x in P, let $\omega(x) = d_P(o, x)$ be the arc length of P (that is, either of P_1 or of P_2) between o and x. For any $x \in P_1$, $y \in P_2$, we have

$$\delta_P(x, y) = \frac{\omega(x) + \omega(y)}{\|xy\|}.$$

For a pair of edges $e \in P_1$ and $e' \in P_2$, define, as above,

$$\delta(e, e') = \delta_P(e, e') = \max_{x \in e, x' \in e'} \delta_P(x, x');$$

as in Sect. 2, we drop the subscript P in the function δ. Then

$$\delta(P) = \max\Big\{\delta(P_1), \delta(P_2), \max_{e \in P_1, e' \in P_2} \delta(e, e')\Big\}.$$

Let A, B denote the set of edges of P_1 and P_2, respectively. It suffices to compute the third term,

$$\delta(A, B) = \max_{a \in A, b \in B} \delta(a, b).$$

Unlike the planar case, the detour of P is not necessarily attained at a vertex of P (for example, there P might contain two long edges that orthogonally pass near each other at a very small distance, and the detour could then be obtained between the two points that realize the distance between the segments). This makes the 3-dimensional algorithm considerably more complicated, and less efficient, than its 2-dimensional counterpart. Consider first the decision problem, in which we wish to determine whether $\delta(A, B) \le \kappa$, for some given $\kappa \ge 1$.

For an edge $e \in A \cup B$, let e^+ denote the ray that emanates from the endpoint, z^+, of e closer to o along P and that contains e; see Fig. 5. Similarly, let e^- denote the ray emanating from the point z^- of e farther from o and containing e. We extend the definition of $\omega(\cdot)$ for points on the rays e^+, e^- even though these points might not lie on P. For a point $x \in e^+$ (resp., $x \in e^-$), we define $\omega(x) = \omega(z^+) + \|z^+x\|$ (resp., $\omega(x) = \omega(z^-) - \|xz^-\|$). Note that these definitions of ω are consistent with the earlier definition, in the sense that all of them assume the same value for the points on e. We can now define $\delta(\cdot, \cdot)$ for points lying on the rays supporting the edges of P_1 and P_2. Namely, for a given pair a, b, where a, b are either edges of P or the rays supporting the edges, $\delta(a, b) = \max_{x \in a, y \in b}(\omega(x) + \omega(y))/\|xy\|$.

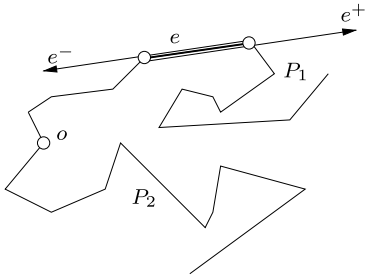

Fig. 5 Decomposition of P and rays e^+, e^-

Lemma 4.2 *Let $a \in A$ and $b \in B$ be a pair of edges. The following four conditions are equivalent*:

(i) $\delta(a, b) > \kappa$;
(ii) $\delta(a^+, b) > \kappa$ *and* $\delta(a^-, b) > \kappa$;
(iii) $\delta(a, b^+) > \kappa$ *and* $\delta(a, b^-) > \kappa$;
(iv) $\delta(a^+, b^+) > \kappa$, $\delta(a^+, b^-) > \kappa$, $\delta(a^-, b^+) > \kappa$, *and* $\delta(a^-, b^-) > \kappa$.

Proof Let a^* (resp., b^*) be the line supporting the edge a (resp., b) oriented in the direction of the ray a^+ (resp., b^+). Parametrize the lines a^* and b^* by the signed distances along these lines from appropriate respective initial points $\xi \in a, \eta \in b$, and denote these distances by t and s, respectively. Regard $a^* \times b^*$ as the parametric ts-plane. Let u, v denote the positively oriented unit vectors along a^* and b^*, respectively. For $x = \xi + tu \in a^*$ and $y = \eta + sv \in b^*$, the condition $\delta(x, y) > \kappa$ can be written as:

$$\delta(x, y) = \frac{\omega(\xi) + \omega(\eta) + t + s}{\|(\xi - \eta) + tu - sv\|} > \kappa,$$

or

$$\kappa\|(\xi - \eta) + tu - sv\| - \omega(\xi) - \omega(\eta) - t - s < 0. \tag{5}$$

The left-hand side of (5) is a *convex* function on the st-parametric plane, being the difference of a convex function and a linear function. The lemma is then an easy consequence of this convexity property. Indeed (i) implies (ii)–(iv) because $a = a^+ \cap a^-$ and $b = b^+ \cap b^-$. For the converse implications, consider the implication (ii) $\Rightarrow$ (i). Suppose that $\delta(x^+, y^+) > \kappa$ for $x^+ \in a^+$, $y^+ \in b$ and $\delta(x^-, y^-) > \kappa$ for $x^- \in a^-$, $y^- \in b$. By construction, $x^+x^- \cap a \neq \emptyset$. Moreover, by convexity of (5), $\delta(x', y') > \kappa$ for all $x' \in x^+x^-$, $y' \in y^+y^-$, thereby implying that $\delta(a, b) > \kappa$. Similar arguments imply that (iii) or (iv) implies (i). □

Using Lemma 4.2(iv) and the standard random-sampling technique [16], we construct a four-level data structure to decide whether $\delta(A, B) > \kappa$. The first level constructs a complete bipartite decomposition for the set $\{(a, b) \in A \times B \mid \delta(a^+, b^+) > \kappa\}$. The second level processes each bipartite clique $A_i \times B_i$ in the decomposition, and represents the set $\{(a, b) \in A_i \times B_i \mid \delta(a^-, b^+) > \kappa\}$ as the union of complete bipartite subgraphs. The third level then refines further this decomposition, to collect pairs that also satisfy $\delta(a^+, b^-) > \kappa$, and the fourth level finally tests whether $\delta(a^-, b^-) > \kappa$ for any of the surviving pairs.

We compute the first-level decomposition of $\{(a,b) \in A \times B \mid \delta(a^+, b^+) > \kappa\}$, as follows. (Similar procedures are then applied at each of the three other levels of the data structure.) For each edge $a \in A$, we map the ray a^+ to a point $\zeta(a^+) = (\zeta_1, \ldots, \zeta_6)$ in $\mathbb{R}^6$, where $(\zeta_1, \zeta_2, \zeta_3)$ are the coordinates of the endpoint z^+ of a^+, (ζ_4, ζ_5) is an appropriate parametrization of the orientation of a^+, and $\zeta_6 = \omega(z^+)$. A similar parametrization will be used for the rays a^-. Next, we map each edge $b \in B$ to a surface $\gamma(b^+)$ that represents the locus of all rays a^+ for which $\delta(a^+, b^+) = \kappa$. Since δ increases as the parameter ζ_6 increases and each 5-tuple $(\zeta_1, \ldots, \zeta_5)$ defines a unique ray in $\mathbb{E}^3$, it follows that $\gamma(b^+)$ is the graph of a totally defined 5-variate function and $\delta(a^+, b^+) > \kappa$ (resp., $\delta(a^+, b^+) < \kappa$) if and only if $\zeta(a^+)$ lies above (resp., below) $\gamma(b^+)$. We can thus regard the problem at hand as that of collecting, in compact form, all pairs $(\zeta(a^+), \gamma(b^+))$ for which $\zeta(a^+)$ lies above $\gamma(b^+)$. Abusing the notation slightly, set $|A| = n$ and $|B| = m$.

We fix a sufficiently large constant r, draw a random sample R of $cr \log r$ edges of B, where c is a sufficiently large constant independent of r, and compute the vertical decomposition $\mathfrak{A}^{\|}$ of the arrangement $\mathfrak{A}$ of the surfaces $\{\gamma(b^+) \mid b \in R\}$. It is easily verified that these surfaces are all semi-algebraic of constant description complexity. Hence, we can apply the result of Koltun [19], to conclude that $\mathfrak{A}^{\|}$ has $O(r^{8+\varepsilon})$ cells, for any $\varepsilon > 0$. For each cell $\tau \in \mathfrak{A}^{\|}$, let $A_\tau = \{e \in A \mid \zeta(e^+) \in \tau\}$, let $B_\tau \subseteq B$ be the set of edges b for which the surface $\gamma(b^+)$ crosses τ, and let $B^*_\tau \subseteq B$ be the set of edges b for which the surface $\gamma(b^+)$ lies completely below τ. The sets A_τ, B_τ can be computed in $O(m+n)$ time under an appropriate model of computation, in which we assume that the roots of a constant degree polynomial can be computed in $O(1)$ time; see [25].

Set $n_\tau = |A_\tau|$ and $m_\tau = |B_\tau|$. Obviously, $\sum_\tau n_\tau = n$ and $|B^*_\tau| \le m$. By the theory of random sampling [16, 25] (where we use the fact that the VC-dimension of the underlying range space is finite), $m_\tau \le m/r$ for all τ, with probability at least $1 - \eta$, where $\eta = \eta(r)$ is a constant that can be made arbitrarily small by choosing the value of r sufficiently large. If $m_\tau > m/r$ for a cell, we choose another random sample and restart the above step. Since the probability of this event is a sufficiently small constant, it does not affect the asymptotic expected running time of the algorithm and we can ignore this step. Moreover, by splitting the cells into subcells, if needed, we may also assume that $n_\tau \le n/r^8$ for each τ; the number of cells remains $O(r^{8+\varepsilon})$. By construction, $\delta(a^+, b^+) > \kappa$ for any pair $e \in A_\tau$ and $b \in B^*_\tau$. We use the second-level data structure, sketched below, to determine whether $\delta(A_\tau, B^*_\tau) > \kappa$. If m_τ or n_τ is less than a prespecified constant, then we use a naive procedure to determine whether $\delta(A_\tau, B_\tau) > \kappa$. Otherwise, we recursively determine (using the first-level data structure) whether $\delta(A_\tau, B_\tau) > \kappa$. For an edge $a \in A_\tau$ and for an edge $b \in B$ such that $\gamma(b^+)$ lies above τ, $\delta(a^+, b^+) < \kappa$, so there is no need to compare A_τ with such edges.

To exploit the symmetry in the condition $\delta(a^+, b^+) > \kappa$ between A and B, we next switch the roles of A_τ and B_τ, by mapping the rays b^+, for $b \in B_\tau$, to points in $\mathbb{R}^6$, and the rays a^+, for $a \in A_\tau$, to surfaces $\gamma(a^+)$, as above. We take a random sample of $cr \log r$ of these surfaces, and construct the vertical decomposition of their arrangement, as above. Repeating this for each cell τ, we end up with $O(r^{16+\varepsilon})$ subproblems, each involving at most n/r^9 segments of A and at most m/r^9 segments of B, which we proceed to solve recursively, using the first-level data structure. In

addition, we have subproblems involving pairs of sets of the form A_τ, B^*_τ, or $B_{\tau'}$, $A^*_{\tau'}$, which we pass to the second level of the structure.

The second-level structure is constructed in an analogous manner, with the only difference that we use the rays a^- instead of the rays a^+. Thus, starting with a pair of subsets A_τ, B_τ, we obtain a decomposition into $O(r^{16+\varepsilon})$ subproblems, each involving at most $|A_\tau|/r^9$ segments of A_τ and at most $|B_\tau|/r^9$ segments of B_τ, which we process recursively using the second-level structure, and a collection of other subproblems that we pass to the third level. The third level is again constructed in complete analogy, using the rays a^+ for the segments in A and the rays b^- for the segments in B. The fourth-level structure is constructed for the rays a^-, b^-, and is a little simpler than the preceding levels, in the sense that whenever we detect a cell that lies fully below a surface ($\gamma(a^-)$ or $\gamma(b^-)$), we stop and report that $\delta(A, B) > \kappa$. Otherwise, we continue the processing recursively, as in the preceding levels.

For $i = 1, \ldots, 4$ and for integers $m, n > 0$, let $T^{(i)}(n, m)$ denote the maximum running time of the ith level data structure on a set of n edges of P_1 and a set of m edges of P_2. Then

$$T^{(4)}(n, m) = O(r^{16+\varepsilon}) \cdot T^{(4)}\left(\frac{n}{r^9}, \frac{m}{r^9}\right) + O(m + n),$$

and

$$T^{(i)}(n, m) = O(r^{16+\varepsilon}) \cdot \left[T^{(i)}\left(\frac{n}{r^9}, \frac{m}{r^9}\right) + T^{(i+1)}(n, m)\right] + O(m + n),$$

for $i \leq 3$. The solutions to the above recurrences are easily seen to be $T^{(i)}(n, m) = O((mn)^{8/9+\varepsilon})$, for any $\varepsilon > 0$ and for each i.

Hence, we obtain the following.

Lemma 4.3 *Given a polygonal chain in $\mathbb{E}^3$, two disjoint subchains A and B of P with a total of m vertices, and a parameter $\kappa \geq 1$, we can determine, in $O(n^{16/9+\varepsilon})$ randomized expected time, whether $\delta(A, B) > \kappa$.*

As in the planar case, we can use the randomized technique of Chan [9] to compute the actual $\delta(A, B)$ within the same asymptotic expected running time bound. The algorithm extends to polygonal cycles and trees in $\mathbb{E}^3$.

In conclusion, we obtain the following.

Theorem 4.4 *The detour of a polygonal chain, cycle, or tree with n edges in $\mathbb{E}^3$ can be computed in randomized expected time $O(n^{16/9+\varepsilon})$, for any $\varepsilon > 0$.*

Remark We remark that it is also possible to use the parametric search technique [22], as in [3], to obtain a deterministic alternative solution. This however (a) results in a considerably more involved algorithm, and (b) requires us to derandomize the decision algorithm, i.e., its vertical decomposition step. This too is doable, but is considerably more complicated.

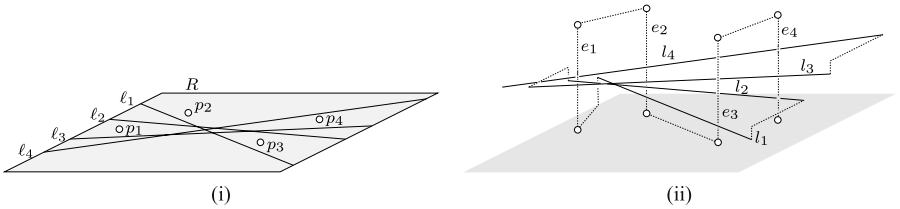

Fig. 6 Reducing Hopcrofts's problem to computing the detour of a 3-dimensional path. (**i**) An instance of Hopcroft's problem. (**ii**) Construction of the polygonal chain Π

4.3 Lower Bound

Finally, we show that computing the detour of a 3-dimensional path is as hard as Hopcroft's problem: Given a set $L = \{\ell_1, \ldots, \ell_n\}$ of n lines in $\mathbb{R}^2$ and a set $P = \{p_1, \ldots, p_n\}$ of n points in $\mathbb{R}^2$, determine whether any line of L contains any point of P. There is an abundance of evidence to suggest that Hopcroft's problem has an $\Omega(n^{4/3})$ lower bound [12]. The best known upper bound in any reasonable model of computation is $O(n^{4/3}2^{O(\log^* n)})$ [21].

To reduce an instance of Hopcroft's problem to that of computing the detour of a 3-dimensional path, we will first build a 3-dimensional path Π that is self-intersecting, i.e., has infinite detour, if and only if the answer to Hopcroft's problem is affirmative. Then we show how the proof can be modified to cover the case where we know *a priori* that the polygonal chains we are given as input do not self-intersect. The construction uses techniques presented in Erickson [12].

Without loss of generality, we may assume that none of the given lines is y-vertical. We begin by sorting the lines in L in increasing order of their slopes and the points in P in increasing lexicographic order. Let $\langle \ell_1, \ldots, \ell_n \rangle$ be the resulting sequence of lines, and let $\langle p_1, \ldots, p_n \rangle$ be the resulting sequence of points. We compute a bounding rectangle R so that each line of L intersects the two y-vertical edges of R, and all the points of P, as well as all the intersection points of lines in L, lie inside R. These steps require $O(n \log n)$ time.

By construction, the ordering of L along the left edge of R in $-y$-direction is $\ell_1, \ldots, \ell_n$, and its ordering along the right edge of R is $\ell_n, \ldots, \ell_1$. For each $1 \le i \le n$, we lift the segment $R \cap \ell_i$ orthogonally to the plane $z = i$, to obtain a line segment l_i. Next, we transform each input point $p_j \in P$ to a line segment e_j that is parallel to the z-axis, whose endpoints are $(p_j, 0)$ and $(p_j, n+1)$; see Fig. 6.

This gives us a set of line segments so that the answer to Hopcroft's problem for the original lines and points is "yes" if and only if some segment l_i intersects some segment e_j. It remains to construct a polygonal chain that contains all these segments without introducing any additional crossings. To do this, we first form a chain containing all segments l_j. It starts at the left endpoint of l_1. The right endpoint of l_1 is connected to the right endpoint of l_2. This connection consists of two segments; the first one is parallel to the z-axis and leads from the plane $z = 1$ to the plane $z = 2$, and the second one, contained in $z = 2$, is parallel to the y-axis. Next, l_2 is traversed, and its left endpoint is connected to the left endpoint of l_3 in an analogous way. We continue until the last endpoint of l_n is reached. Clearly, the resulting chain is simple.

Next, we connect the segments $e_1, \ldots, e_n$ into a simple polygonal chain by connecting the upper endpoints of e_i to e_{i+1} if i is odd and the lower endpoints if i is even. This chain is clearly not self-intersecting since its xy-projection is monotone in the lexicographic order. Finally, we connect the left endpoint of l_1 in $z = 1$ to the free endpoint of e_1 in $z = 0$ by two additional segments. The resulting concatenation of the two chains has the desired property. See Fig. 6.

One might state the problem of computing the detour of a 3-dimensional chain in such a way that the input chains are known apriori not to have self-intersections. The above lower bound proof can be adapted to this situation in the following way. First, we move each of the original lines ℓ_i a distance of ϵ to the right, where ϵ is a formal infinitesimal, i. e., ϵ is positive, but smaller than any real number. Then we construct the polygonal chain in the same way as before. It will always be non-intersecting, but its detour is bigger than c/ϵ, for some appropriate constant $c > 0$, if and only if there was a point-line incidence in the original instance of Hopcroft's problem. Reductions using infinitesimals were formally shown to be correct, in the algebraic decision tree model, by Erickson [12].

In conclusion, we have shown:

Theorem 4.5 *An algorithm with running time $f(n)$ for computing the detour of 3-dimensional polygonal chains with n vertices implies an $O(n \log n + f(n))$ time algorithm for Hopcroft's problem.*

Remark It is interesting to note that we have almost matched this lower bound with the algorithm in Theorem 4.1 for computing the spanning ratio of P. We do not know whether the preceding construction can be extended to yield a lower bound argument for computing spanning ratios.

5 Conclusions

We have given $O(n \log n)$-time randomized algorithms for computing the detour and spanning ratio of planar polygonal chains. These algorithms lead to an $O(n \log^2 n)$-time algorithms for computing the detour and spanning ratio of planar trees and cycles. In three dimensions, we have given subquadratic algorithms for computing the detour and spanning ratio of polygonal chains, cycles, and trees. Previously, no subquadratic-time (exact) algorithms were known for any of these problems.

There are many open problems in this new area. The most obvious is: Which other classes of graphs admit subquadratic-time algorithms for computing their detour or spanning ratio? Also, it remains open to prove an $\Omega(n \log n)$ lower bound for computing the detour of a simple planar polygonal chain of n vertices; at present, such a bound is only known for computing the spanning ratio. Finally, it seems likely that the algorithm for computing the detour in $\mathbb{E}^3$ can be improved.

Acknowledgement We would like to thank Günter Rote for interesting discussions related to the problems studied in the paper.

References

1. Agarwal, P.K., Erickson, J.: Geometric range searching and its relatives. In: Chazelle, B., Goodman, J.E., Pollack, R. (eds.) Advances in Discrete and Computational Geometry. Contemporary Mathematics, vol. 223, pp. 1–56. American Mathematical Society, Providence (1999)
2. Agarwal, P.K., Klein, R., Knauer, C., Sharir, M.: Computing the detour of polygonal curves. Technical Report B 02-03, Freie Universität Berlin, Fachbereich Mathematik und Informatik (2002)
3. Agarwal, P.K., Sharir, M., Toledo, S.: Applications of parametric searching in geometric optimization. J. Algorithms **17**, 292–318 (1994)
4. Aichholzer, O., Aurenhammer, F., Icking, C., Klein, R., Langetepe, E., Rote, G.: Generalized self-approaching curves. Discrete Appl. Math. **109**, 3–24 (2001)
5. Alt, H., Guibas, L.J.: Discrete geometric shapes: Matching, interpolation, and approximation. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 121–153. Elsevier, Amsterdam (2000)
6. Alt, H., Knauer, C., Wenk, C.: Comparison of distance measures for planar curves. Algorithmica **38**, 45–58 (2004)
7. Aurenhammer, F., Klein, R.: Voronoi diagrams. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 201–290. Elsevier, Amsterdam (2000)
8. Bose, P., Morin, P.: Competitive online routing in geometric graphs. Theor. Comput. Sci. **324**, 273–288 (2004)
9. Chan, T.M.: Geometric applications of a randomized optimization technique. Discrete Comput. Geom. **22**(4), 547–567 (1999)
10. Ebbers-Baumann, A., Klein, R., Langetepe, E., Lingas, A.: A fast algorithm for approximating the detour of a polygonal chain. Comput. Geom. Theory Appl. **27**, 123–134 (2004)
11. Edelsbrunner, H., Guibas, L.J., Sharir, M.: The complexity and construction of many faces in arrangements of lines and of segments. Discrete Comput. Geom. **5**, 161–196 (1990)
12. Erickson, J.: New lower bounds for Hopcroft's problem. Discrete Comput. Geom. **16**, 389–418 (1996)
13. Fortune, S.J.: A sweepline algorithm for Voronoi diagrams. Algorithmica **2**, 153–174 (1987)
14. Grüne, A.: Umwege in Polygonen. Master's thesis, Institut für Informatik I, Universität Bonn (2002)
15. Guibas, L.J., Sharir, M., Sifrony, S.: On the general motion planning problem with two degrees of freedom. Discrete Comput. Geom. **4**, 491–521 (1989)
16. Haussler, D., Welzl, E.: Epsilon-nets and simplex range queries. Discrete Comput. Geom. **2**, 127–151 (1987)
17. Icking, C., Klein, R.: Searching for the kernel of a polygon: a competitive strategy. In: Proceedings of the 11th Annual Symposium on Computational Geometry, pp. 258–266 (1995)
18. Icking, C., Klein, R., Langetepe, E.: Self-approaching curves. Math. Proc. Camb. Philos. Soc. **125**, 441–453 (1999)
19. Koltun, V.: Almost tight upper bounds for vertical decompositions in four dimensions. J. ACM **51**, 699–730 (2004)
20. Langerman, S., Morin, P., Soss, M.: Computing the maximum detour and spanning ratio of planar chains, trees and cycles. In: Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS 2002). Lecture Notes in Computer Science, vol. 2285, pp. 250–261. Springer, Berlin (2002)
21. Matoušek, J.: Range searching with efficient hierarchical cuttings. Discrete Comput. Geom. **10**(2), 157–182 (1993)
22. Megiddo, N.: Applying parallel computation algorithms in the design of serial algorithms. J. ACM **30**(4), 852–865 (1983)
23. Narasimhan, G., Smid, M.: Approximating the stretch factor of Euclidean graphs. SIAM J. Comput. **30**(3), 978–989 (2000)
24. Rote, G.: Curves with increasing chords. Math. Proc. Camb. Philos. Soc. **115**, 1–12 (1994)
25. Sharir, M., Agarwal, P.K.: Davenport-Schinzel Sequences and Their Geometric Applications. Cambridge University Press, New York (1995)