Polyline Fitting of Planar Points under Min-Sum Criteria

Boris Aronov[*] Tetsuo Asano[†] Naoki Katoh [‡] Kurt Mehlhorn [§]
Takeshi Tokuyama [¶]

October 29, 2004

Abstract

Fitting a curve of a certain type to a given set of points in the plane is a basic problem in statistics and has numerous applications. We consider fitting a polyline with k joints under the min-sum criteria with respect to L_1- and L_2-metrics, which are more appropriate measures than uniform and Hausdorff metrics in statistical context. We present efficient algorithms for the 1-joint versions of the problem, and fully polynomial-time approximation schemes for the general k-joint versions.

1 Introduction

Curve fitting aims to approximate a given set of points in the plane by a curve of a certain type. This is a fundamental problem in statistics, and has numerous applications. In particular, it is a basic operation in *regression analysis. Linear regression* approximates a point set by a line, while *non-linear regression* approximates it by a non-linear function from a given family.

In this paper, we consider the case where the points are fitted by a polygonal curve (*polyline*) with k joints, see Figure 1. This is often referred to as *polygonal approximation* or *polygonal fitting* problem. It is used widely. For example, it is commonly employed in scientific and business analysis to represent a data set by a polyline with a small number of joints. The best representation is the polyline minimizing the error of approximation. Error is either defined as the maximum (vertical) distance of any input point from the polyline (*min-max-optimization*) or the sum of vertical distances (*min-sum-approximation*). In either case, distance is measured in some norm. We follow common practice and restrict ourselves to norms L_1 and L_2.

Min-max-approximation by a polyline is well studied. In one popular formulation which one minimizes the *maximum* of the vertical distance (called the *uniform metric* or *Chebyschev error function*) from the points to the curve. Hakimi and Schmeichel gave a $O(n^2 \log n)$ time algorithm for this problem [11]; the time complexity was later improved to $O(n^2)$ [26] and then to $O(n \log n)$ [10]. Another popular approach is to minimize the Hausdorff measure that is the maximum of the Euclidean distances between the points and the output curve. This problem can also be solved in polynomial time [22]. These problems are closely related to *curve simplification*, in which the input is a polyline with n edges rather than a set of n points; this question arises in geographic

[*]Polytechnic University, Brooklyn, NY 11201, USA; aronov@cis.poly.edu. Supported in part by NSF ITR Grant CCR-00-81964. Part of the work was carried out while B.A. was visiting JAIST.

[†]Japan Advanced Institute of Science and Technology, Tatsunokuchi, 923-1292, Japan; t-asano@jaist.ac.jp. Partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (B).

[‡]Kyoto University, Kyoto, 606-8501, Japan; naoki@archi.kyoto-u.ac.jp.

[§]Max-Planck-Institut für Informatik, D-66123, Saarbrücken, Germany; mehlhorn@mpi-sb.mpg.de.

[¶]Tohoku University, Sendai, 980-8579, Japan; tokuyama@dais.is.tohoku.ac.jp.

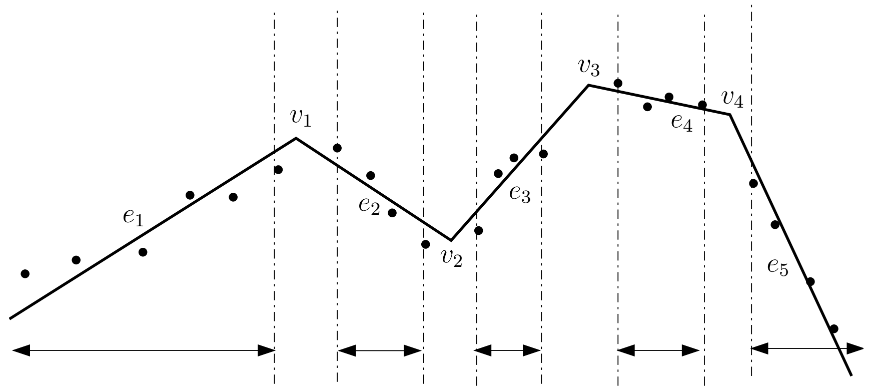

Figure 1: A 4-joint polyline fitting a set of points.

information systems (see the survey [27]) and has received much attention in computational geometry [5, 12, 15, 21].

The minimize-the-maximum (*min-max*) formulation is useful in pattern recognition applications. However, in applications to statistics, its serious deficiency is its extreme sensitivity to the presence and location of outliers. Even a single outlier can drastically change the output, while outliers, real or imagined, are common in statistical data. For this reason, minimize-the-sum-of-errors (*min-sum*) methods are considerably more popular in statistics: The most basic one is the least-squares method that minimizes the sum of the squares of vertical distances between the input points and the output curve. In this paper, we call it the L_2*-fitting* problem (the term *least-squares fitting* is commonly used as well), and *regression line* in statistics usually refers to the L_2-fitting line. If the output curve is either a straight line or a low-degree algebraic curve, it is quite easy to compute the optimal L_2-fitting. Another criterion is L_1*-minimization*, in which we minimize the sum of vertical distances from the candidate curve to the points being fitted. L_1 fitting is more resilient to outliers than L_2 fitting; however, it is usually more expensive (or complicated) to compute the optimal solution. For linear regression, the L_1-optimal line can be computed in linear time [13], but it requires sophisticated computational techniques. Several other formulations have been proposed for further reducing the effect of outliers on the linear regression. Repeated median regression is a well-known example, and efficient solutions are known for several other criteria [16].

In this paper, we focus on the L_1- and L_2*-fitting* problems when the desired curve is a *k-joint polyline*; in other words, it is a continuous piecewise-linear x-monotone curve with $k+1$ linear components. We assume that a coordinate system is fixed, and the input points are sorted with respect to their x-coordinate values. To the authors' knowledge, the computational complexity of the optimal k-joint problem under either of these minimization criteria has not been previously investigated. More specifically, it seems that an efficient solution of the L_1-fitting problem extending the result of Imai *et al.* [13] is theoretically challenging even for the 1-joint problem.

In this paper, we begin by considering the 1-joint problem. We give algorithms of complexity $O(n)$ and $\tilde{O}(n^{4/3})$ time for the L_2 and L_1 criteria, respectively.[1] The L_2-fitting algorithm is simple and practical, whereas the L_1-fitting algorithm depends on using a semi-dynamic range search data structure and parametric search. For general k, we present two approximation schemes. Let z_{opt} be the minimum fitting error for a k-joint polyline and let ϵ be a positive constant. We give a polynomial-time approximation scheme (PTAS) to compute a $\lfloor(1+\epsilon)k\rfloor$-joint fitting whose error is at most z_{opt} and we describe a fully polynomial-time approximation scheme (FPTAS) to compute a k-joint polyline with $(1+\epsilon)z_{\mathrm{opt}}$ fitting error, and consequently show that the problems

[1]We write $f(n) = \tilde{O}(g(n))$ if there exists an absolute constant $c \geq 0$ such that $f(n) = O(g(n)\log^c n)$.

cannot be strongly NP-hard, although their NP-hardness remains open.

Intuitively, why are the problems we consider in this paper more difficult than some related questions? We have mentioned that the uniform metric fitting problem can be solved efficiently. The key point is that its corresponding decision problem to determine whether there exists a k-joint polyline with a uniform error less than a given value w is geometrically a *stabbing* problem. The k-joint path must go through n vertical line segments of length $2w$ centered at the input points, and one can continuously move a feasible polyline so that each link becomes extremal in geometric sense, that is, goes through a pair of endpoints of vertical segments. Hence, we can design a polynomial-time algorithm to find the optimal path based on dynamic programming. The L_1- and L_2-fitting problems seem to be more subtle: We do not have reformulation of the corresponding decision problems in terms of stabbing.

2 Preliminaries

A k-joint polyline is an alternating sequence $P = (e_1, \mathbf{v}_1, e_2, \mathbf{v}_2, \ldots, e_k, \mathbf{v}_k, e_{k+1})$ of line segments (*links*) and joint vertices (*joints*), where e_s and e_{s+1} share the endpoint $\mathbf{v}_s$, for $s = 1, 2, \ldots, k$, and e_1 and e_{k+1} are infinite rays. We denote the link e_s on line $y = a_s x - b_s$ by (a_s, b_s) if the interval of the values of x corresponding to the link is understood. A joint $\mathbf{v}_s$ is represented by the pair (u_s, v_s) of its coordinate values. Thus, the connectivity and monotonicity of the polyline can be guaranteed by requiring that $v_s = a_s u_s - b_s = a_{s+1} u_s - b_{s+1}$, for $s = 1, 2, \ldots, k+1$, and $u_1 < \ldots < u_k$.

We now formulate the problem of fitting a k-joint polyline to an n-point set. Given a set of points $S = \{p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)\}$ with $x_1 < x_2 < \ldots < x_n$ and an integer k, and setting $u_0 = -\infty$ and $u_{k+1} = \infty$ for convenience, find a polyline $P = ((a_1, b_1), (u_1, v_1), (a_2, b_2), (u_2, v_2), \ldots, (u_k, v_k), (a_{k+1}, b_{k+1}))$ minimizing one of the following three quantities for L_1-, L_2-, and uniform metric fitting, respectively:

$$L_1\colon \sum_{s=1}^{k+1} \sum_{u_{s-1} < x_i \le u_s} |a_s x_i - b_s - y_i|, \tag{1}$$

$$L_2\colon \sum_{s=1}^{k+1} \sum_{u_{s-1} < x_i \le u_s} (a_s x_i - b_s - y_i)^2, \tag{2}$$

$$\text{Uniform metric}\colon \max_{s=1,\ldots,k+1} \Big\{ \max_{u_{s-1} \le x_i \le u_s} |a_s x_i - b_s - y_i| \Big\}. \tag{3}$$

For $k = 0$, the problems are linear regression problems. The L_2-linear regression is well known as the *Gaussian least-squares method.* Once we compute $\mathrm{A}_n = \sum_{i=1}^n x_i$, $\mathrm{B}_n = \sum_{i=1}^n y_i$, $\mathrm{C}_n = \sum_{i=1}^n x_i^2$, $\mathrm{D}_n = \sum_{i=1}^n x_i^2$, and $\mathrm{E}_n = \sum_{i=1}^n x_i y_i$ in linear time, we can construct an optimal fitting line $y = ax - b$ by considering the partial derivatives of the objective function and solving a 2×2 system of linear equations. The linear regression problem with respect to the uniform error is to find a pair of parallel lines at the minimum vertical distance that contain all the given points between them. This can be done by applying the *rotating caliper method* that computes antipodal pairs of points on the convex hull of the point set. For an x-sorted point set this can be done in $O(n)$ time [23]. The L_1-linear regression problem is more involved; however, a linear-time algorithm has been devised by Imai *et al.* [13] based on Megiddo's prune-and-search paradigm.

3 Fitting a 1-joint polyline

We consider the problem of fitting a 1-joint polyline to a set of points. We proceed in two steps. We first assume that the joint vertex lies in a fixed interval $[x_q, x_{q+1}]$ and later eliminate this assumption. Let $S_1(q) = \{p_1, p_2, \dots, p_q\}$ and $S_2(q) = \{p_{q+1}, \dots, p_n\}$. Our objective polyline consists of two links lying on lines $\ell_1\colon y = a_1x - b_1$ and $\ell_2\colon y = a_2x - b_2$, respectively. We call a tuple (a_1, b_1, a_2, b_2) *feasible* if the two lines $y = a_1x - b_1$ and $y = a_2x - b_2$ meet at a point whose x-coordinate $u = \frac{b_1 - b_2}{a_1 - a_2}$ lies in the interval $[x_q, x_{q+1}]$. Our goal here is to find a feasible tuple (a_1, b_1, a_2, b_2) representing a 1-joint polyline minimizing

$$\sum_{i=1}^{q} |a_1x_i - b_1 - y_i| + \sum_{i=q+1}^{n} |a_2x_i - b_2 - y_i| \quad \text{and} \tag{4}$$

$$\sum_{i=1}^{q} (a_1x_i - b_1 - y_i)^2 + \sum_{i=q+1}^{n} (a_2x_i - b_2 - y_i)^2, \tag{5}$$

for L_1- and L_2-fitting, respectively. Minimizing (4) is equivalent to, provided $a_1 \neq a_2$, minimizing $\sum_{i=1}^{n} w_i$ subject to

$$\begin{aligned} -w_i &\le a_1x_i - b_1 - y_i \le w_i, \quad \text{for } i \le q, \\ -w_i &\le a_2x_i - b_2 - y_i \le w_i, \quad \text{for } i \ge q+1, \quad \text{and} \\ x_q &\le \frac{b_1 - b_2}{a_1 - a_2} \le x_{q+1}, \end{aligned} \tag{6}$$

where the last line represents the feasibility condition.

Lemma 3.1 *For either L_1- or L_2-fitting criterion, the 1-joint problem for a fixed q reduces to solving two convex programming problems.*

Proof: Disregarding the feasibility constraint, the problem is clearly a quadratic programming problem for the L_2 case and a linear programming problem for the L_1 case. The feasibility constraint requiring that the lines ℓ_1 and ℓ_2 meet in the strip between x_q and x_{q+1} can be expressed by different linear constraints depending on whether $a_1 \le a_2$. Thus, we can decompose the (L_1 or L_2) problem into two subproblems. If $a_1 \le a_2$, the lines meet in the strip if and only if ℓ_1 is not below ℓ_2 at x_q and is not above it at x_{q+1}. Thus, the additional constraint becomes

$$x_q(a_2 - a_1) \le b_2 - b_1 \le x_{q+1}(a_2 - a_1)\,. \tag{7}$$

In the opposite case, the additional constraint is

$$x_{q+1}(a_2 - a_1) \le b_2 - b_1 \le x_q(a_2 - a_1)\,. \tag{8}$$

Clearly, each subproblem is a convex programming problem, as claimed. □

From the above lemma, it is clear that the optimal 1-joint polyline can be computed by using linear/quadratic programming. However, we aim to design combinatorial algorithms for these problems. Indeed, we can classify the solution into two types: (a) An inequality in (7), (8) holds with equality. (b) All of the inequalities in (7), (8) are strict. We call the solution *fixed* in the former case and *free* otherwise. From the form of the expressions in (7), (8) we deduce the following simple observation.

Lemma 3.2 *If the solution is fixed, the joint is located on either of the two vertical lines $x = x_q$, $x = x_{q+1}$.*

If the joint is on the line $x = x_{q+1}$, we can regard it as a solution for the partition into $S_1(q+1) = S_1(q) \cup \{p_{q+1}\}$ and $S_2(q+1) = S_2(q) \setminus \{p_{q+1}\}$. Thus, for each partition, we essentially need to solve two subproblems: (1) the free problem and (2) the fixed problem where the joint is on the vertical line $x = x_q$. This leads to the following generic algorithm: For each partition of S into two intervals S_1 and S_2, we first consider the free problem ignoring the feasibility constraint, and check whether the resulting solution is feasible or not, i.e., we verify that the intersection point lies in the strip between p_q and p_{q+1}. If it is feasible, it is the best solution for the partition. Otherwise, we consider the fixed solution adding the constraint that the joint lie on $x = x_q$, and report the solution for the partition. After processing all $n-1$ possible partitions, we report the solution with the smallest error.

If it takes $O(f(n))$ time to process a subproblem for each partition, the total time complexity is $O(nf(n))$. For efficiency, we design a dynamic algorithm to process each partition so that $f(n)$ is reduced in the amortized sense.

3.1 The L_2 1-joint problem

We show how to construct an optimal L_2-fitting 1-joint polyline in linear time. We process the partitions $(S_1(q), S_2(q))$ starting from $q = 1$ to $q = n-1$, in order. We maintain the sums, variances, and covariances $\mathrm{A}_q = \sum_{i=1}^q x_i$, $\mathrm{B}_q = \sum_{i=1}^q y_i$, $\mathrm{C}_q = \sum_{i=1}^q x_i^2$, $\mathrm{D}_q = \sum_{i=1}^q y_i^2$, and $\mathrm{E}_q = \sum_{i=1}^q x_i y_i$ incrementally, at constant amortized cost. They also provide us with the corresponding values for $S_2(q)$ if we precompute those values for S, i.e., $\sum_{i=q+1}^n x_i = \mathrm{A}_n - \mathrm{A}_q$ etc.

For the free case, the objective function is separable, in the sense that the optimal solution can be identified by finding (a_1, b_1) minimizing $\sum_{i=1}^q (a_1 x_i - b_1 - y_i)^2$ and (a_2, b_2) minimizing $\sum_{j=q+1}^n (a_2 x_j - b_2 - y_j)^2$ independently. Each can be computed in $O(1)$ time from the values of $\mathrm{A}_q, \ldots, \mathrm{E}_q$ as explained in section 2. The feasibility check of the solution is done in $O(1)$ time by computing the intersection point of the corresponding pair of lines. It remains to solve the subproblems with the additional constraint that the joint is at $x = x_q$. Put

$$f(a_1, b_1, a_2, b_2) = \sum_{i=1}^q (a_1 x_i - b_1 - y_i)^2 + \sum_{j=q+1}^n (a_2 x_j - b_2 - y_j)^2, \tag{9}$$

$$g(a_1, b_1, a_2, b_2) = a_1 x_q - b_1 - a_2 x_q + b_2, \text{ and} \tag{10}$$

$$L(a_1, b_1, a_2, b_2) = f(a_1, b_1, a_2, b_2) - \lambda g(a_1, b_1, a_2, b_2), \tag{11}$$

so that $f(\cdot)$ is the function to be minimized and the joint constraint can be expressed as $g(\cdot) = 0$. Then, by the Kuhn-Tucker condition the optimal solution $Z_{\text{opt}} = (a_1^0, b_1^0, a_2^0, b_2^0)$ describing a best L_2-fitting 1-joint polyline for a fixed value of q has to satisfy

$$\left.\frac{\partial L}{\partial a_1}\right|_{Z_{\text{opt}}} = \left.\frac{\partial L}{\partial b_1}\right|_{Z_{\text{opt}}} = \left.\frac{\partial L}{\partial a_2}\right|_{Z_{\text{opt}}} = \left.\frac{\partial L}{\partial b_2}\right|_{Z_{\text{opt}}} = 0, \tag{12}$$

and

$$g(Z_{\text{opt}}) = 0. \tag{13}$$

This gives us a set of five linear equations that must be satisfied by the optimal parameter values of a_1, b_1, a_2, b_2 and the Lagrange multiplier λ. The coefficients can be expressed in terms of $x_q, \mathrm{A}_q, \ldots, \mathrm{E}_q$, and this system can be solved in constant time for each q. Thus, we have the following:

Theorem 3.3 *L_2-optimal 1-joint fitting can be computed in linear time.*

3.2 The L_1 1-joint problem

3.2.1 Semi-dynamic L_1 linear regression

We start with the problem of computing the optimal linear L_1-fitting (i.e., linear regression) of the input point set, i.e., we seek the line $\ell_{\text{opt}}\colon y = ax - b$ minimizing $\sum_{i=1}^{n} |ax_i - b - y_i|$.

The difficulty with the L_1-fitting problem is that, written in linear programming terms (as in (6)), it has $n+2$ variables, in contrast to the least-squares case where the problem is directly solved as a bivariate problem. Nonetheless, the problem can solved by a brute-force combinatorial algorithm in $O(n^3)$ time, since there are $O(n^2)$ possible linear dissections of the point set which can be enumerated in $\Theta(n^2)$ worst-case time by constructing the dual arrangement, and one can compute the optimal line in linear time once the dissection by the line is given (this algorithm can be easily sped up to constant or near-constant amortized time per dissection). Moreover, by Lemma 3.4, the optimal line bisects S into two equal-size subsets; in other words, the line is a halving line. Using this fact, Imai *et al.* [13] devised an optimal linear-time algorithm for computing ℓ_{opt} based on the multidimensional prune-and-search paradigm. In order to design an efficient algorithm for the 1-joint fitting problem, we consider a semi-dynamic version of the L_1 linear regression for a point set P with low amortized time complexity, where we dynamically maintain P with insertions and deletions under an assumption that P is always a subset of a fixed universe S of size n that is given from the outset. (In fact, for our application, it is sufficient to be able to start with $P = \emptyset$ and handle only insertions, and to start with $P = S$ and handle only deletions. Moreover, the order of insertions and deletions is known in advance. The data structure we describe below is more general.)

Consider the dual space, with $p_i = (x_i, y_i)$ transformed to the dual line $Y = f_i(X)$ where $f_i(X) = x_i X - y_i$. The line $y = ax - b$ is transformed to the point (a, b) in the dual space. The kth *level* of the arrangement $\mathcal{A} = \mathcal{A}(S^*)$ of the set S^* of dual lines is the trajectory of the kth largest value among $f_i(X)$.[2] We call the $\lceil n/2 \rceil$th level the *median level.*

Lemma 3.4 (Imai *et al.* [13]) *If the optimal L_1-fitting line is given by $y = a_{\text{opt}}x - b_{\text{opt}}$, its dual point $(a_{\text{opt}}, b_{\text{opt}})$ is on the median level if n is odd, and between the $\frac{n}{2}$th level and the $(\frac{n}{2}+1)$th level if n is even.*

Now, given X-value t, consider the point $(t, f_i(t))$ for each $i = 1, 2, \ldots, n$, and let $F(t)$ be the sum of the $\lfloor n/2 \rfloor$ largest values in $\{f_i(t)\colon i = 1, 2, \ldots, n\}$ and $G(t)$ be the sum of the $\lfloor n/2 \rfloor$ smallest values in the same set. Put $H(t) = F(t) - G(t)$. $H(t)$ gives the L_1 fitting error of the dual line of any point (t, y) on the median level (or between the two median levels if n is even). Thus, by Lemma 3.4, $H(t)$ is minimized at $t = a_{\text{opt}}$.

Lemma 3.5 *$F(t)$ is a convex function, while $G(t)$ is concave. As a consequence, $H(t)$ is also convex. $H(t)$ has either slope 0 at $t = a_{\text{opt}}$ or its slope changes from negative to positive at $t = a_{\text{opt}}$.*

Proof: The convexity follows directly from the fact that, in any line arrangement, the portions of the lines lying on or below (resp. on or above) any fixed level k can be decomposed into k non-overlapping concave (resp. convex) chains; see, for example, [3]. □

Suppose a fixed universe S^* of lines is given. We need a data structure that maintains a subset $P^* \subseteq S^*$ and supports the following operations:

Median-location query For a query value t, return the point on the $\lfloor n/2 \rfloor$th highest line at $X = t$.

[2] We use an asterisk to denote geometric dual of a point, line, or a set of lines/points.

Slope-sum query For a query point $p = (t, y)$, return the sum of the slopes of lines below p at $X = t$.

Height-sum query For a query value $p = (t, y)$, return the sum of the Y-coordinates of the lines below p at $X = t$. The height-sum query is reduced to a slope-sum query plus a constant-term-sum query that reports the sum of the constant terms of the equations representing lines.

Update A line in S^* is added to or removed from P^*.

Suppose a data structure supporting such queries on a set $P^* \subseteq S^*$ of lines in $O(\tau(n))$ time is available, where $n = |S^*|$. Then we can query the slopes of F and G at t, and hence compute the slope of H at t in $O(\tau(n))$ time. Because of convexity of H, we have the following:

Lemma 3.6 *Given t, we can decide whether $t < a_{\text{opt}}$, $t > a_{\text{opt}}$, or $t = a_{\text{opt}}$ in $O(\tau(n))$ time.*

Thus, we can perform binary search to find a_{opt}. We show below how to make this search strongly polynomial. Once we know a_{opt}, we determine b_{opt} by the median-location query at $t = a_{\text{opt}}$.

3.2.2 Semi-dynamic data structure for the queries

We show how to realize semi-dynamic median-location query and sum-queries. As a preliminary step, we describe a semi-dynamic data structure for vertical ray queries, i.e., queries of the form: Given a vertical upward ray starting at (t, z) determine the number of lines in P^* intersected by the ray, the sum of their slopes, and the sum of their constant terms. A dual line $Y = x_i X - y_i$ is above (t, z) iff the primal point (x_i, y_i) is above the line $y = tx - z$. Thus our queries are reduced to half-space queries in the primal plane. We use the partition-tree data structure of Matoušek [4, 17, 19]. It supports half-space queries on sets with n points in time $O(\sqrt{n})$, linear space, and preprocessing time $O(n \log n)$.

We build a partition tree $\mathcal{T}(S)$ on the set S of points dual to the lines in S^* (in fact, these are the points to which a line is being fitted). A standard construction proceeds as follows: With each node v of the partition tree we associate a point set $S(v) \subseteq S$ and a triangle $\Delta(v) \supset S(v)$, where $S(v) \subset S(\text{parent}(v))$ at any node v other than the root and $S(v) = S$ at the root. In addition we also store at v the size $|S(v)|$ of $S(v)$ and the sums $\xi(S(v)) = \sum_{p_i \in S(v)} x_i$ and $\chi(S(v)) = \sum_{p_i \in S(v)} y_i$ of the slopes and constant terms of the corresponding dual lines. Since the point sets $S(v)$, over all children v of a node w in the tree, by definition of a partition tree, partition the set $S(w)$, and $|S(v)|$ is at most a fraction of $|S(w)|$, this tree has linear size and logarithmic depth. For our purposes, we modify the partition tree to obtain a new tree $\mathcal{T}(S, P)$ where the same $\Delta(v)$ as in $\mathcal{T}(S)$ is associated with every node v, but v stores $P(v) = S(v) \cap P$, $\xi(P(v))$ and $\chi(P(v))$ instead of the corresponding values for $S(v)$. This data structure enables us to execute the half-plane range query in P, and thus the vertical ray query in P^*.

Our data structure is semi-dynamic. When P changes, with a point p being added or removed, what we need to update is just values $|P(v)|$, $\xi(P(v))$, and $\chi(P(v))$ for each node v where p is relevant. Since the sets $S(v)$ for all nodes v at a fixed level of the partition tree form a partition of S, only one node must be updated at each level; to facilitate the update one might associate with each point $p \in S$ a list of length $O(\log n)$ containing the nodes v of the tree with $p \in S(v)$. Thus, the update can be performed in $O(\log n)$ time. This ends the description of the semi-dynamic vertical ray query data structure. Our sum-queries can be done by using the vertical ray query.

We next turn to the median-location query data structure. For a given t, let $m(t) = (t, y(t))$ be the intersection of the vertical line $X = t$ and the median level of the dual arrangement $\mathcal{A}(S^*)$. We can use the vertical query data structure to compare any given η with $y(t)$. We perform a vertical ray query to find the number of lines above (t, η). If it is less than $\lfloor n/2 \rfloor$, $y(t) < \eta$; otherwise $y(t) \geq \eta$. This suggests computing $y(t)$ by some kind of binary search. If we had the sorted list of intersections between the vertical line $X = t$ and the lines in S^* available, we could perform a binary search on L by using $O(\log n)$ ray queries. However, it takes $O(n \log n)$ time to compute the list, which is too expensive since we aim for a sublinear query time. Instead, we construct a data structure which can simulate the binary search without explicitly computing the sorted list.

Lemma 3.7 *We can construct a randomized data structure in time $O(n \log n)$ such that, given t, we can compute $y(t)$ in $\tilde{O}(\sqrt{n})$ time. The query time bound holds for every vertical line $X = t$ with high probability.*

Proof: We fix a small constant $\epsilon > 0$, and randomly select $cn^{1-\epsilon}$ lines from $\Psi_0 = S^*$, to have a set Ψ_1 of lines, where c is a suitable constant. From the results of Clarkson and Shor [9], if the constant c is sufficiently large, with high probablity every vertical segment intersecting no line of Ψ_1 intersects at most $n^\epsilon \log n$ lines of S^*. In other words, Ψ_1 is the dual of an $(n^{\epsilon-1} \log n)$-net of S. Similarly, we construct Ψ_{i+1} from Ψ_i such that Ψ^*_{i+1} is an $(\frac{n^\epsilon \log n}{|\Psi_i|})$-net of Ψ^*_i if $|\Psi_i| > n^\epsilon \log n$. Thus, we have a filtration $\Psi_0 \supset \Psi_1 \supset \ldots \supset \Psi_k$, and $|\Psi_k| \leq n^\epsilon$. The number k of layers is a function of ϵ and c only, so the construction takes $O(n)$ time.

Additionally, we construct a dual range-searching data structure for Ψ_i such that for a query vertical interval I we can report all lines in Ψ_i meeting I in $O(\sqrt{n} + K)$ time, where K is the number of reported lines. In primal space a vertical interval corresponds to a strip bounded by two parallel lines and hence we may use partition trees as described above to implement reporting queries. The preprocessing time is $O(n \log n)$.

Now, our algorithm for finding $y(t)$ is as follows: Given t, we first compute all the intersections between $X = t$ and the lines of Ψ_k, sort them, and perform binary search for $y(t)$ on them. Each step of the search requires a vertical ray query and hence time $O(\sqrt{n})$. As the result of the binary search, we obtain a vertical interval I containing $y(t)$ such that no line of Ψ_k crosses the interior of I. By using the dual range-searching data structure, we extract, in time $O(\sqrt{n} + K)$, the set of K lines in Ψ_{k-1} intersecting I; $K = O(n^\epsilon \log n)$ with high probability. Proceeding recursively, we obtain $y(t)$, since at the last level of the filtration we arrive at an interval I' containing $y(t)$, with no line of $\Psi_0 = S^*$ crossing its interior. The total time is $O(n^\epsilon \log^2 n + \sqrt{n} \log n) = \tilde{O}(\sqrt{n})$.
□

At this point, we have a $\tilde{O}(\sqrt{n})$ realization of the semi-dynamic query data structure, i.e., $\tau(n) = \sqrt{n}$. We finally come to the strongly polynomial method for determining a_{opt}. We use parametric search [24]. We use a parallel version of the ray-query algorithm, i.e., the parallel traversal of the partition tree, for the guide algorithm (see [14]). Since the depth of a partition tree is $O(\log n)$, the parallel time complexity of the ray query is $O(\log n)$. Thus, the parallel time complexity of sum queries is $O(\log^2 n)$ using $O(\tau(n))$ processors. Therefore, using standard parametric search paradigm, we can compute the optimal L_1 linear fitting in $\tilde{O}(\tau(n))$

More specifically, we review how parametric search applies to our situation in the following subsection. We remark that we do not employ parametric search to compute $y(t)$ for a fixed t, since it is not always possible to use it in a nested fashion, and there are technical difficulties in applying multi-dimensional parametric search paradigm [20, 25] to our problem.

3.2.3 A Review of Parametric Search:

Parametric search identifies a real number a^*. It has two ingredients:

- A decision procedure $D(t)$ of one real parameter. The procedure compares t and a^*, and return 0 for $t < a^*$ and 1 to $t \geq a^*$. The time complexity for the decision procedure is $O(T_D)$.

- A master program $M(t)$ with value in $[0,1]$. The master program is a parallel program which takes time T_M with p processors if the value of t is given, although it is simulated on a sequential machine. We assume that at each step of the master program, decision is done as follows: A real number (called threshold value) is computed, such that they subdivide the interval $(-\infty, \infty)$ into two subintervals. The threshold values is independent of choice of t, and the decision is determined by the subinterval containing t. Thus, in each round of the parallel program, we have at most p threshold values that subdivides $(-\infty, \infty)$ into at most $(p+1)$ subintervals, and all the decisions in the round is determined by the subinterval containing t.

Parametric search simulates the execution of $M(t)$ for $t = a^*$. We maintain an interval I containing a^* and a set Q of real numbers. Initially, $I = (-\infty, \infty)$ and $Q = \emptyset$. In each round, we simulate p parallel tasks. For each task, the decision is determined if I is contained in one of the subintervals defined by the threshold values. Otherwise, we insert all the threshold values contained in I to Q. Then, we compute the median m of Q, and compare m with a^* using the call of $D(m)$. If $m \geq a^*$, we replace I by $I \cap (-\infty, m]$, otherwise by $I \cap (m, \infty)$. The elements of Q that are not included in the updated I are deleted from Q. We iterate this process until I has no element of Q in its interior. Then, we can determine all the decisions in the current round, and proceed to the next round. Thus, we complete the simulation in $O(pT_M + T_M T_D \log p)$ time.

The decision procedure: In our case a^* (called a_{opt} in the paper) is the slope of the optimal L_1-line. The decision procedure works as follows. Let t be the input.

1. It first determines the y-coordinate $y(t)$ of the median level of the dual arrangement at t. For this end is uses a filtration of the dual arrangement. At level i (we call it Ψ_i) of the filtration, we have a vertical segment s spanned by two lines of Ψ_i and not properly intersected by any line of Ψ_i.

   (a) We first determine the lines of Ψ_{i+1} intersecting s. This is a range query, where the range is the wedge defined by the lines dual to the endpoints of s; more precisely, this range actually a strip between two parallel lines in primal space. The running time is $O(\sqrt{n} + K)$ where K is the number of lines intersected. K is small ($O(n^\epsilon)$ by net properties.

   What kind of comparisons are we making here? We have a line $tX + b$ and we check whether this line intersects a triangle. This can be rewritten as a comparisons of t with a real number.

   (b) Once it has determined the intersected lines, it sorts the intersections and performs binary search on them. For each point, we need to count the number of lines above it. This is again a range query. So with $\log n$ queries of the type "number of lines above a point", we have reduced s to s' spanned by two adjacent lines of Ψ_{i+1}.

   The comparison of two lines at the vertical line t is non-trivial. We compare $at + b$ with $a't + b'$. This can be rewritten as a comparison of t with a real number.

2. Once we have found $y(t)$, we use a single slope query and a single constant term query to determine the slope of $H(t)$ at t. This amounts to a comparison with a^*.

The Master Program: The master program is simply the parallel version of the decision procedure. It uses $\sqrt{n}$ processors. The range queries are easily parallelized because they amount to walking down up to $\log n$ paths in the partition tree. Thus the parallel time of a range query is $\log n$. Step 1.b amounts to a parallel sort (time $\log n$) plus a binary search (time $\log n$). Each step of the binary search requires a range query. Step 2 is also a single range query.

Thus the parallel time is $O(\log^2 n)$ with $\sqrt{n}$ processors.

Some Intuition: It is instructive to see what the parametric search does geometrically. In each Ψ_i it determines a trapezoid T with two vertical walls and two non-vertical edges. The non-vertical edges lie on adjacent lines in Ψ_i. The median level of the full arrangement intersects the vertical walls and does not intersect the non-vertical edges. Also a^* lies in the x-span I of the trapezoid. The interval I is the interval maintained in the parametric search.

In step 1a, we narrow this trapezoid by moving the vertical walls. This will make sure that no lines of Ψ_{i+1} enters the reduced trapezoid through its non-vertical sides. In other words all vertical segments connecting the two non-vertical sides of the trapezoid intersect the same lines of Ψ_{i+1}.

Next we come to sorting the intersections (step 1b) with the generic vertical segment spanned by T. Step 1b will further move in the vertical walls until the trapezoid contains no intersections between lines in Ψ_{i+1}. Each search step will move one of the non-vertical walls. Also in each search step we do a range search and this may further move in the vertical walls (since the median level must stay within the trapezoid).

3.2.4 Time-space tradeoff

To speed up the query time $\tau(n)$ and thus the overall algorithm, we generalize the data structure to allow it to use super-linear storage based on Matoušek's construction [18]. If we can use $O(m)$ space for $n < m < n^2$, we first select $r = O(m/n)$ points from S and construct a dual cutting, i.e., a decomposition of the dual plane into cells, such that each cell C is intersected by at most n/r lines dual to points of S; the number of cells required is $O(r^2)$ and the computation time is $O(nr)$.

Let $S(C)$ be the set of lines intersecting C. We construct a point-location data structure on the cutting. For each cell C, we store the cumulative statistics (the sum of slopes etc.) for the set of lines passing below C, and construct the partition tree for $S(C)$. The query time of each tree is $\tilde{O}(\sqrt{n/r})$. When P changes, we need to update the data stored in each of the $O(r^2)$ cells of the cutting, and also the $O(r)$ partition trees corresponding to sets containing the updated point. Thus, update time is $O(r^2 + r\log n)$.

Update time can be sped up by not storing the statistics for each cell explicitly, but rather retrieving them when needed at a cost of $O(\log r)$. This reduces the time needed for an update to $O(r\log r)$ and the total time of all updates to $O(nr\log n)$ as shown below. Thus, if we set $r = n^{1/3}$, the update time and query time are both $\tilde{O}(n^{1/3})$. The space and preprocessing time is $\tilde{O}(n^{4/3})$. The parallel time complexity is not affected by the space-time trade-off.

In the following, we explain how to reduce the update time to $O(r\log r)$. The issue is that we have r^2 cells in a cutting, each with an attached data structure (partition tree) and with some constant-size data (total number of lines passing below/above and sums of their slope and constant terms); call the later *global counts*. Let ℓ be the line inserted to or deleted from P in the update. The partition trees are easy to update when a line is inserter/removed and there are only $O(r)$ of them to modify; indeed, only the partition trees stored at cells intersected by ℓ, and there are $O(r)$ such cells because of the zone theorem [7] of an arrangement.

On the other hand, the line ℓ contributes to r^2 global counts; thus we need $O(r^2)$ time if we directly update them. Instead, we store the global counts inplicitly, such that we can compute them on the fly, when we need them (in about logarithmic time).

We consider the adjacency graph of the cells of the cutting, and take an Eulerian path σ on it. The size of the path is $O(r^2)$. If a cell appears on σ several time, pick one occurrence and call it REAL, make the remaining ones DUMMY. Now number the cells (by REAL occurrence) along σ. Build a balanced binary tree B on top of σ. Each internal node v corresponds to a collection of consecutive nodes on σ. Let us consider the number of lines in P such that they (1) pass above all the cells correspond to the collection, and (2) do not pass above all the cells corresponding to the correction for the parent node of v. We call the number the ABOVE count of v. Symmetrically, we define the BELOW count. The tree B is static, but each node v will store with it the ABOVE and BELOW counts that are dynamically maintained.

Initialize all the counts to 0 when $P = \emptyset$. Now say we want to add a line ℓ to P (deletion is symmetric). Line ℓ intersects $O(r)$ cells of the cutting. We can afford to compute which cells these are, explicitly (and we do, to update the partition tree information). This gives $m = O(r)$ cell numbers. Sort them, adding a $-\infty$ and $+\infty$. Call them $c_0 = -\infty, c_1, c_2, ..., c_m, c_{m+1} = +\infty$. Each interval (c_i, c_{i+1}), if non-empty (i.e. if $c_{i+1} > c_i + 1$), corresponds to a connected subpath of σ lying completely to one side of ℓ. We can test in constant time which side it is by checking one of the cells. Now decompose this interval into a logarithmic number of canonical ones (corresponding to nodes of this balanced tree) and increment ABOVE/BELOW counts at the $O(\log r)$ nodes. Do this for each of the $O(r)$ subpaths. Total update time is thus reduced to $O(r \log r)$.

3.2.5 Algorithm for L_1 1-joint fitting

Finally, we describe the algorithm to find the L_1-optimal 1-joint polyline fitting a set S of n points in the plane. Recall that there are two different types of solutions:

Type 1 There is an index q such that the 1-joint polyline consists of the optimal L_1-fitting line of $S_1(q) = \{p_1, p_2, \dots, p_q\}$ and that of $S_2(q) = \{p_{q+1}, p_{q+2}, \dots, p_n\}$.

Type 2 There is an index q such that the joint lies on the vertical line $x = x_q$.

If the optimal solution is of type 1, we compute an optimal L_1-fitting line for $S_1(q)$ and $S_2(q)$ separately, for every $q = 1, 2, \dots, n$, by using the semi-dynamic algorithm with S as the universe. If we use quasi-linear space $\tilde{O}(n)$, the time complexity is $\tilde{O}(n^{1.5})$, and if we use $O(n^{4/3})$ space, the time complexity is $\tilde{O}(n^{4/3})$.

Otherwise, the optimal solution is of type 2. For each q, we guess the y-coordinate value η of the joint vertex (x_q, η). Then, we can compute the best line, in the sense of L_1 fitting, approximating $S_1(q)$ going through the (for now, fixed) joint by using almost the same strategy as in section 3.2.1. Indeed, it suffices to determine the slope of this line. In the dual space, we just need to compute a point $p = (a(p), b(p))$ on the line $Y = x_q X - \eta$ such that $\sum_{i=1}^{q} |a(p)x_i - b(p) - y_i|$ is minimized. We observe that the above function is convex if it is regarded as a function of a, and hence $\theta(p) = \theta^+(p) - \theta^-(p)$ is monotone and changes the sign at p, where $\theta^+(p)$ ($\theta^-(p)$) is the sum of slopes of lines above p (resp. below p). Thus, we can apply binary search by using slope-sum query, and this binary search can be performed in $O(\log n)$ steps by using the filtration as described in Lemma 3.7.

Moreover, because of the convexity of the objective function, once we know the optimal solution for a given η_0, we can determine whether the global optimal value η is greater than η_0 or not by using the height-sum query. Indeed, when we infinitesimally slide η_0, the gain (or loss) of the objective function can computed from the slope sums and height sums of dual lines associated with each of the sets of points lying above, below, and on the current polyline (for each of $S_1(q)$ and $S_2(q)$).

Thus, we can apply binary search for computing the optimal value of η. In order to construct a strongly polynomial algorithm, we apply parametric search. Note that given η, our algorithm has a natural parallel structure inherited from the range-searching algorithms, and runs in polylogarithmic time using $\tilde{O}(\tau(n))$ processors. Thus, the parametric search paradigm [24] is applicable here. Therefore, for a fixed q, the second case of the problem can be handled in $\tilde{O}(\tau(n))$ time. Thus, we have the following:

Theorem 3.8 *The optimal L_1-fitting 1-joint polyline is computed in $\tilde{O}(n^{1.5})$ randomized time using quasi-linear space, and $\tilde{O}(n^{4/3})$ randomized time using $O(n^{4/3})$ space.*

4 Fitting a k-joint polyline

The k-joint fitting problem is polynomial-time solvable if k is a fixed constant. We describe the algorithm in a non-deterministic fashion. We guess the partition of $x_1, \dots, x_n$ into k intervals each of which corresponds to a line segment in the polyline. Also, we guess whether each joint is free or fixed. We decompose the problem at the free joints and have a set of subproblems. In each subproblem, we add the linear constraints corresponding to the fixed condition (i.e., each joint is located on a guessed vertical line). Thus, each subproblem is a convex programming problem: a linear program for L_1, and a quadratic program for L_2. We solve each subproblem separately to obtain the solution of the whole problem. Note that this strategy works because of the convexity of each subproblem. There are $O((3n)^k)$ different choices of the guesses, thus we can be replace guessing by a brute-force search to have a polynomial-time deterministic algorithm if k is a constant.

For a general k, we do not know whether the problem is in the class P or not. Thus, we would like to consider approximation algorithms. One possible approach is to relax the requirement that number of joints be exactly k. We can design a PTAS for it.

Theorem 4.1 *Let z_{opt} be the optimal L_1 (or L_2) error of a k-joint fitting. Then, for any constant $\epsilon > 0$, we can compute a $\lfloor (1+\epsilon)k \rfloor$-joint fitting whose error is at most z_{opt} in polynomial time.*

Proof: We ignore continuity and approximate the points by using a piecewise-linear (not necessarily continuous) function with k linear pieces. This can be done by preparing the optimal linear regression for each subinterval of consecutive points of S, and then applying dynamic programming. We can restore the continuity by inserting at most k steep (nearly vertical) line segments. The resulting polyline has at most $2k$ joints and error at most z_{opt}. We can improve $2k$ to $\lfloor \frac{3k}{2} \rfloor$ by applying the 1-joint algorithm instead of linear regression algorithm, and further improve it to $\lfloor (1+\epsilon)k \rfloor$ by using the r-joint algorithm mentioned above for $r = \lceil \epsilon^{-1} \rceil$. $\square$

Another approach is to keep the number of joints at k and approximate the fitting error. We give a FPTAS for it. We only discuss the L_1 case, since the L_2 case is analogous. Let z_{opt} be the optimal L_1-error, and we aim to find a k-joint polyline whose error is at most $(1+\epsilon)z_{\text{opt}}$. We remark that if $z_{\text{opt}} = 0$, our solution is exactly the same as the solution for the uniform metric fitting problem, and thus we may assume $z_{\text{opt}} > 0$. Recall that the uniform metric fitting problem can be solved in $O(n \log n)$ time [10]. The following is a trivial but crucial observation:

Lemma 4.2 *Let z_∞ be the optimal error for the uniform metric k-joint fitting problem. Then, $z_\infty \le z_{\text{opt}} \le n z_\infty$.*

Proof: The sum of the errors in the uniform-metric–optimal polyline is at most nz_∞. Hence $nz_\infty \ge z_{\text{opt}}$. On the other hand, every k-joint polyline has a data point in S such that the vertical distance to the polyline is at least z_∞, so $z_{\text{opt}} \ge z_\infty$. $\square$

Our strategy is as follows: We call the n vertical lines through our input points the *column lines.* We give a set of *portal points* on each column line, and call a k-joint polyline a *tame polyline* if each of its links satisfies the condition that the line containing the link goes through a pair of portal points.

On each column line, the distance between its data point and the intersection point with the optimal polyline is at most z_{opt}, thus at most nz_∞. Thus, on the ith column line, we place the portals in the vertical range $[y_i - nz_\infty, y_i + nz_\infty]$. The portal points are placed symmetrically above and below y_i. The jth portal above y_i is located at the y-value $y_i + (1+\frac{\epsilon}{2})^{j-1}\delta$, where $\delta = \frac{z_\infty \epsilon}{2n}$ and $j = 1, 2, \dots, M$. We choose the largest M satisfying $(1+\frac{\epsilon}{2})^M \delta \le nz_\infty$, and hence $M = O(\epsilon^{-1}\log(n+\epsilon^{-1}))$. We also put portals at heights y_i and $y_i \pm nz_\infty$. In this way the number of portals in any column is at most $2M+3$.

We call a closed interval between adjacent portals in a column a *prime interval.*

Lemma 4.3 *There exists a tame polyline whose L_1 error is at most $(1+\epsilon)z_{\text{opt}}$.*

Proof: We start from the optimal polyline ℓ_{opt}, and deform it to obtain a tame polyline. We proceed sequentially, left to right. Consider the line containing the leftmost link of ℓ_{opt}. We continuously move the line to a tame line without crossing any portal point during the movement; if the line started off passing through a portal point, we rotate it around it; if the line started off passing through two, it is already tame. The right joint of the current link is accordingly moved to the intersection of the new line and the line containing the right neighbor link. It may happen that during this transformation a joint crosses a column line. However, the intersection points of the original and the deformed polylines with a column line are located in a common prime interval. We repeat this operation, proceeding from left to right, to obtain a tame polyline.

Now, consider the change of vertical distances between a point p_i and the two polylines. The polylines go through the same prime interval of the column line through p_i. An index i is called a near-index if the polylines goes through a prime interval containing p_i in its closure; otherwise it is called a far-index. For the near-indices, the summation of the errors caused by the new polyline is bounded by $n\delta = \frac{z_\infty \epsilon}{2}$. For each far-index, the errors caused by the new polyline at the column is bounded by $1 + \frac{\epsilon}{2}$ times the one caused by the old (i.e. optimal) polyline. Thus, the total error of the new polyline for all the far indices is at most $(1+\frac{\epsilon}{2})z_{\text{opt}}$. In total, the error of the new polyline is bounded by $(1+\frac{\epsilon}{2})z_{\text{opt}} + \frac{z_\infty \epsilon}{2} \leq (1+\epsilon)z_{\text{opt}}$. □

Thus, it suffices to compute the optimal tame polyline. There are Mn portals, and thus $N = O(M^2n^2)$ lines going through a pair of portals. Let $\mathcal{L}$ be the set of these lines. We design a dynamic programming algorithm. For the ith column, for each line $\ell \in \mathcal{L}$ and each $m \leq k$, we record the approximation error of the best m-joint tame polyline up to the current column whose (rightmost) link covering p_i is on ℓ. When we proceed to the $(i+1)$th column, each approximation error is updated. If there is an intersection between lines ℓ and ℓ' in the interval $(x_i, x_{i+1}]$, we consider the polylines that have the intersection as a possible joint. This can be done by copying the data for ℓ to ℓ' and vice versa incrementing the join number by one, and then keeping the smaller of the current and the new (copied) error for each of the pairs (ℓ, m) and (ℓ', m) for $m = 1, 2, \ldots, k$. Then, we add the distance from p_{i+1} to each polyline. Finally, we select the minimum error at the nth column, and retrieve the polyline by backtracking.

There are $O(N^2)$ intersections of lines, and it takes $O(k)$ time for each intersection for copying and updating. This requires $O(N^2k)$ work and dominates the running time. Since $N = O(n^2M^2) = O(n^2\epsilon^{-2}\log^2(n+\epsilon^{-1}))$, we have the following:

Theorem 4.4 *An $(1+\epsilon)$-approximation, i.e., a k-joint polyline with error $1+\epsilon$ times the optimal, for each of the L_1 and L_2 k-joint problems can be computed in $O(kn^4\epsilon^{-4}\log^4(n+\epsilon^{-1}))$ time.*

5 Concluding remarks

A major open problem is to determine the complexity class of the k-joint problem for L_1- and L_2-fitting. The corresponding L_1 or L_2 polyline approximation problem where the input is a curve is also interesting.

We remark that the curve simplification problem under the L_1-measure is to minimize the area between input and output polylines. In the restricted case where the vertex set or the set of lines supporting edges of the output polyline is required to be a subset of that of the input polyline, the problem is reduced to the k-link shortest path problem in a graph. In particular, if the input polyline is convex, this problem is related to matrix searching (see [2]). However, for the general case the authors are not aware of an efficient algorithm, and it is an interesting research problem. A related question for which polynomial-time algorithms have been constructed for L_1 and L_2 measures is approximating an x-monotone curve by a k-peak curve, i.e., a curve with at most k local maxima [6]. The k-joint problem seems to be more difficult than the k-peak problem because controlling continuity is in the former case is a challenge.

Acknowledgment: The authors would like to thank Jirí Matoušek for a stimulating discussion on convexity.

References

[1] P. K. Agarwal, S. Hal-Peled, N.H. Mustafa, and Y. Wang, Near-linear time approximation algorithm for curve simplification , *Proc. 2002 European Symp. Algorithms*, LNCS 2461, (2002) pp.29-41.

[2] A. Aggarwal, B. Schieber, and T. Tokuyama, “Finding a minimum-weight k-link path in graphs with the concave Monge property and applications,” *Discrete Comput. Geom.*, **12** (1994) 263–280.

[3] P. Agarwal, B. Aronov, T. Chan, M. Sharir, “On Levels in Arrangements of Lines, Segments, Planes, and Triangles,” *Discrete Comput. Geom.*, **19** (1998) 315–331.

[4] P. Agarwal and J. Matoušek, “Ray shooting and parametric search,” *SIAM J. Comput.*, **22** (1993) 794–806.

[5] P. K. Agarwal and K. R. Varadarajan, “Efficient algorithms for approximating polygonal chains,” *Discrete Comput. Geom.*, **23** (2000) 273–291.

[6] J. Chun, K. Sadakane, and T. Tokuyama, “Linear time algorithm for approximating a curve by a single-peaked curve,” *Proc. 14th Internat. Symp. Algorithms Comput. (ISAAC 2003)*, LNCS 2906, 2003, pp. 6–16.

[7] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, ETACS Monographs on Theoretical Computer Science 10, Springer-Verlag, 1987.

[8] D. Eu and G.T. Toussaint, “On approximating polygonal curves in two and three dimensions”, *CVGIP: Graphical Models And Image processing* **56** (1994) 231–246.

[9] K. L. Clarkson and P. W. Shor, “Application of Random Sampling in Computational Geometry,” *Discrete Comput. Geom.*, **4** (1989) 423–432.

[10] M. Goodrich, “Efficient piecewise-linear function approximation using the uniform metric,” *Discrete Comput. Geom.*, **14** (1995) 445–462.

[11] S. Hakimi and E. Schmeichel, “Fitting polygonal functions to a set of points in the plane,” *Graphical Models and Image Processing*, **53** (1991) 132–136.

[12] H. Imai and M. Iri: “Polygonal approximations of a curve - Formulations and algorithms,” *Computational Morphology*, Elsevier Science Publishers B.V. (North Holland), 1988, 71–86.

[13] H. Imai, K. Kato, and P. Yamamoto: “A linear-time algorithm for linear L_1 approximation of points,” *Algorithmica*, **4** (1989) 77–96.

[14] N. Katoh and T. Tokuyama, “Notes on computing peaks in k-levels and parametric spanning trees,” Proc. 17th ACM Symp. on Computational Geometry, 2001, pp. 241–248.

[15] Y. Kurozumi and W.A. Davis: “Polygonal approximation by the minimax method,” *Computer Graphics and Image Processing*, **19** (1982) 248–264.

[16] S. Langerman and W. Steiger, “Optimization in arrangements.” *Proc. Symp. Theor. Aspects Computer Science (STACS2003)*, LNCS 2607, 2003, pp. 50–61.

[17] J. Matoušek, “Efficient partition trees,” *Discrete Comput. Geom.*, **8** (1992) 315–334.

[18] J. Matoušek, “Range searching with efficient hierarchical cutting,” *Proc. 8th ACM Symp. on Comput. Geom.*, (1992) 276–287.

[19] J. Matoušek, “Geometric range searching,” *ACM Computing Surveys*, **26** (1994) 421–461.

[20] J. Matoušek and O. Schwarzkopf, “Linear optimization queries,” *Proc. 8th Annual ACM Symp. Comput. Geom.*, 1992, pp. 16–25.

[21] A. Melkman and J. O'Rourke: "On polygonal chain approximation," *Computational Morphology*, Elsevier Science Publishers B.V. (North Holland), 1988, 87–95.

[22] J. O'Rourke and G. Toussaint, "Pattern Recognition," Chapter 43 of *Handbook of Discrete and Computational Geometry* (eds. J. Goodman and J. O'Rourke), CRC Press, 1997.

[23] F. P. Preparata and M. I. Shamos, *Computational Geometry, an Introduction*, Springer-Verlag, New York, 1985.

[24] J. Salowe, "Parametric Search," Chapter 37 of *Handbook of Discrete and Computational Geometry* (eds. J. Goodman and J. O'Rourke), CRC Press, 1997.

[25] T. Tokuyama, "Minimax parametric optimization problems and multi-dimensional parametric searching," *Proc. 33rd Symp. Theory Comput.* (2001), pp. 75–83.

[26] D. P. Wang, N. F. Huang, H. S. Chao, and R. C. T. Lee, "Plane sweep algorithms for polygonal approximation problems with applications," *Proc. 4th Internat. Symp. Algorithms Comput. (ISAAC 2003)*, LNCS 762, 1993, pp. 515–522.

[27] Robert Weibel, "Generalization of Spatial Data: Principles and Selected Algorithms," *Algorithmic Foundations of Geographic Information Systems*, LNCS 1340, 1997, pp. 99–152.