A deterministic algorithm for fitting a step function to a weighted point-set

Hervé Fournier* Antoine Vigneron†

November 6, 2012

Abstract

Given a set of n points in the plane, each point having a positive weight, and an integer $k > 0$, we present an optimal $O(n \log n)$-time deterministic algorithm to compute a step function with k steps that minimizes the maximum weighted vertical distance to the input points. It matches the expected time bound of the best known randomized algorithm for this problem. Our approach relies on Cole's improved parametric searching technique. As a direct application, our result yields the first $O(n \log n)$-time algorithm for computing a k-center of a set of n weighted points on the real line.

1 Problem formulation and previous works

A function $f : \mathbb{R} \to \mathbb{R}$ is called a *k-step function* if there exists a real sequence $a_1 < \cdots < a_{k-1}$ such that the restriction of f to each of the intervals $(-\infty, a_1)$, $[a_i, a_{i+1})$ for $1 \leqslant i \leqslant k-2$, and $[a_{k-1}, +\infty)$ is a constant. A weighted point in the plane is a triplet $p = (x, y, w) \in \mathbb{R}^3$ where $(x, y) \in \mathbb{R}^2$ represents the coordinates of p and $w > 0$ is a weight associated with p. We use $\mathrm{d}(p, f)$ to denote the weighted vertical distance $w \cdot |f(x) - y|$ between p and f. For a set of weighted points P, we define the distance $\mathrm{d}(P, f)$ between P and a step function f as:

$$\mathrm{d}(P, f) = \max\{\mathrm{d}(p, f) \mid p \in P\}.$$

Given the point-set P, our goal is to find a k-step function f that minimizes $d(P, f)$.

This histogram construction problem is motivated by databases applications, where one wants to find a compact representation of the dataset that fits into main memory, so as to optimize query processing [6]. The unweighted version, where $w_i = 1$ for all i, has been studied extensively, until optimal algorithms were found. (See our previous article [5] and references therein.)

The weighted case was first considered by Guha and Shim [6], who gave an $O(n \log n + k^2 \log^6 n)$-time algorithm. Lopez and Mayster [10] gave an $O(n^2)$-time algorithm, which is thus faster for small values of k. Then Fournier and Vigneron [5] gave an $O(n \log^4 n)$ algorithm, which was further improved to $O(\min(n \log^2 n, n \log n + k^2 \log \frac{n}{k} \log n \log \log n))$ by Chen and Wang [2]. Eventually, an optimal randomized $O(n \log n)$-time algorithm was obtained by Liu [9]. In this note, we present a deterministic counterpart to Liu's algorithm, which runs in $O(n \log n)$ time. This time bound is optimal as the unweighted case already requires $\Omega(n \log n)$ time [5].

*Univ Paris Diderot, Sorbonne Paris Cité, Institut de Mathématiques de Jussieu, UMR 7586 CNRS, F-75205 Paris, France. Email: fournier@math.univ-paris-diderot.fr.

†King Abdullah University of Science and Technology (KAUST), Geometric Modeling and Scientific Visualization Center, Thuwal 23955-6900. Saudi Arabia. Email: antoine.vigneron@kaust.edu.sa.

Our approach combines ideas from previous work on this problem [6, 7] with the improved parametric searching technique by Cole [4].

Our result has a direct application to the k-center problem on the real line: Given a set of n points $r_1 < \cdots < r_n \in \mathbb{R}$, with weights $w_1, \ldots, w_n$, the goal is to find a set of k centers $c_1, \ldots, c_k \in \mathbb{R}$ that minimizes the maximum over i of the weighted distance $w_i \mathrm{d}(r_i, \{c_1, \ldots, c_k\})$. Given such an instance of the weighted k-center problem, we construct an instance of our step-function approximation problem where the input points are $p_i = (i, r_i)$ for $i = 1, \ldots, n$, keeping the same weights $w_1, \ldots, w_n$. Then these two problems are equivalent: The y-coordinates of the k steps of an optimal step-function give an optimal set of k centers. So our algorithm also solves the weighted k-center problem on the line in $O(n \log n)$ time, improving on a recent result by Chen and Wang [3].

2 An optimal deterministic algorithm

We consider an input set of weighted points $P = \{(x_i, y_i, w_i) \mid 1 \leqslant i \leqslant n\}$, and an integer $k > 0$. Let ε^* denote the optimal distance from P to a k-step function, that is,

$$\varepsilon^* = \min\{\mathrm{d}(P, f) \mid f \text{ is a } k\text{-step function}\}.$$

Karras et al. [7] made the following observation:

Lemma 1 *Given a set of n weighted points sorted with respect to their x-coordinate, an integer $k > 0$ and a real $\varepsilon > 0$, one can decide in time $O(n)$ if $\varepsilon < \varepsilon^*$.*

The above lemma is obtained by a greedy method, going through the points from left to right and creating a new step whenever necessary. More than k steps are created along this process if and only if $\varepsilon < \varepsilon^*$. A consequence is that once ε^* is known, an optimal k-step function can be built in linear time by running this algorithm on $\varepsilon = \varepsilon^*$.

A second observation, made by Guha and Shim [6], is the following. The distance of a point $p = (x_i, y_i, w_i)$ to the constant function c is equal to $\mathrm{d}(p, c) = w_i \cdot |c - y_i|$. Hence, for a (non empty) subset $Q \subseteq P$ of the input points, the distance $\min\{\mathrm{d}(Q, f) \mid f \text{ is a constant function}\}$ between Q and the closest constant function is given by the minimum y-coordinate of the points in the region U_Q defined as:

$$U_Q = \bigcap_{(x_i, y_i, w_i) \in Q} \{(x, y) \mid y \geqslant w_i \cdot |x - y_i|\}.$$

In other words, the distance between Q and the closest 1-step function is the y-coordinate of the lowest vertex in the upper envelope U_Q of the lines with equation $y = \pm w_i(x - y_i)$ corresponding to the points $(x_i, y_i, w_i) \in Q$. (There is only one lowest vertex as the slopes $\pm w_i$ are nonzero.)

An immediate consequence is the following. For $i \in \{1, \ldots, n\}$, let ℓ_{2i-1} be the line defined by the equation $y = w_i(x - y_i)$, and ℓ_{2i} the line defined by $y = -w_i(x - y_i)$. Let $L = \{\ell_1, \ldots, \ell_{2n}\}$. We denote by $\mathcal{A}(L)$ the arrangement of these lines. (See Figure 1.)

Lemma 2 *The optimal distance ε^* from a set of weighted points P to a k-step function is the y-coordinate of a vertex of $\mathcal{A}(L)$.*

The deterministic algorithm presented here will be obtained by performing a search on the vertices of $\mathcal{A}(L)$, calling the decision procedure of Lemma 1 only $O(\log n)$ times, and with an overall extra time $O(n \log n)$. We achieve it by applying Cole's improved parametric searching technique [4]:

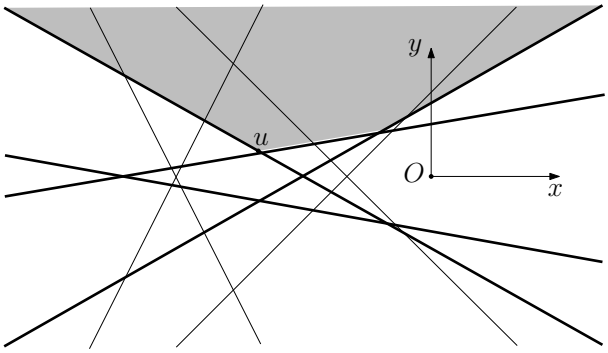

Figure 1: The arrangement $\mathcal{A}(L)$ and the upper envelope (shaded) of a subset $Q \subset L$ (bold).The y-coordinate of the lowest point u in U_Q gives the minimum distance from Q to a 1-step function.

Theorem 3 (Cole [4]) *Consider the problem of sorting an array $A[1, \ldots, n]$ of size n. Assume the following two conditions hold:*

(i) There is an $O(n)$ time algorithm to test if $A[i] \leqslant A[j]$.

(ii) There exists a linear order $\preceq$ on the set $\{(i, j) \mid 1 \leqslant i < j \leqslant n\}$ such that

$$(i, j) \preceq (i', j') \;\Rightarrow\; \big(A[i] \leqslant A[j] \Rightarrow A[i'] \leqslant A[j']\big)$$

and such that we can decide if $(i, j) \preceq (i', j')$ in $O(1)$ time.

Then, the array A can be sorted in $O(n \log n)$ time.

We briefly explain Cole's method. Recall that a sorting network [8] for n elements is a sequence $L_1, \ldots, L_d$, each L_i being a set of comparisons on disjoint inputs in $\{1, \ldots, n\}$. On an input array $A[1, \ldots, n]$, the network operates as follows: for each level p from 1 to d, the comparisons in L_p are performed in parallel, and the two elements of A corresponding to each comparison are swapped if they appear in the wrong order. If the sorting network is correct, the elements of A are output in sorted order after the last level.

The algorithm from Theorem 3 works as follows. First build a sorting network of depth $O(\log n)$ in deterministic $O(n \log n)$ time [1, 11]. During the course of the sorting algorithm, each comparison in the network is marked with one of the following labels: *resolved*, *active* or *inactive*. In the beginning, the comparisons at the first level L_1 of the network are marked active, while all others are inactive. The weight $1/4^p$ is assigned to each active node at level p. Repeat the following until all nodes are resolved:

– Compute the weighted median (i_m, j_m) of all active comparisons with respect to the order $\preceq$ defined in (ii);

– Decide if $A[i_m] \leqslant A[j_m]$ with the algorithm from (i). This solves a weighted half of the active comparisons. Swap the corresponding element of A when in the wrong order, and mark these comparisons as *resolved*. Mark all inactive comparisons having their two inputs resolved as *active*.

It can be proved that at most $O(n)$ nodes are active at any step. Since the weighted median can be computed in linear time, each step is performed in $O(n)$ time. Moreover, it can be showed that the algorithm terminates in at most $O(\log n)$ steps. So overall, this procedure sorts an array of size n in $O(n \log n)$ time.

We are now ready to give our algorithm for fitting a step function to a weighted point set:

Theorem 4 *Given a set P of n weighted points in the plane and an integer $k > 0$, a k-step function f minimizing $\mathrm{d}(P, f)$ can be computed in $O(n \log n)$ deterministic time.*

Proof: Let $\theta : \mathbb{R} \to \{0, 1\}$ be the mapping defined by $\theta(\varepsilon) = 0$ if $\varepsilon < \varepsilon^*$ and $\theta(\varepsilon) = 1$ otherwise. First we sort the points of P with respect to their x-coordinate. Given ε, this allows to compute $\theta(\varepsilon)$ in time $O(n)$, using the decision algorithm of Lemma 1.

We denote by $\pi : \mathbb{R}^2 \to \mathbb{R}$ the projection onto y-coordinate axis. Recall the definition of the lines $L = \{\ell_1, \ldots, \ell_{2n}\}$. For $y \in \mathbb{R}$, let $\ell_i(y)$ be the unique $x \in \mathbb{R}$ such that $(x, y) \in \ell_i$; it is well-defined since no line is parallel to the x-axis. By Lemma 2, it holds that

$$\varepsilon^* = \min\{\pi(v) \mid v \text{ vertex of } \mathcal{A}(L),\ \theta(\pi(v)) = 1\}.$$

Although ε^* is not known, we shall sort the set $\{\ell_i(\varepsilon^*) \mid 1 \leqslant i \leqslant 2n\}$ using Cole's method.

Note that L could be of cardinality smaller than $2n$ if some lines are identical. We discard identical lines and order them with respect to their order at $-\infty$. That is, $L = \{\ell_1, \ldots, \ell_m\}$ and for all $0 < i < m$, it holds that $\ell_i(y) < \ell_{i+1}(y)$ when $y \to -\infty$. Let us check that conditions (i) and (ii) of Theorem 3 hold.

Condition (i): Given $i < j$, we want to decide if $\ell_i(\varepsilon^*) \leqslant \ell_j(\varepsilon^*)$. If lines ℓ_i and ℓ_j are parallel, the ordering on the lines ensures that $\ell_i(y) < \ell_j(y)$ for all y and in particular for ε^*. Otherwise, we compute $y_0 = \pi(\ell_i \cap \ell_j)$ in time $O(1)$, then decide if $y_0 < \varepsilon^*$ in time $O(n)$ by Lemma 1. If $y_0 < \varepsilon^*$, then $\ell_i(\varepsilon^*) > \ell_j(\varepsilon^*)$; otherwise $\ell_i(\varepsilon^*) \leqslant \ell_i(\varepsilon^*)$.

Condition (ii): For $i < j$, let us define $\tilde{\pi}(\ell_i, \ell_j) = \pi(\ell_i \cap \ell_j)$ if lines ℓ_i and ℓ_j intersect, and $\tilde{\pi}(\ell_i, \ell_j) = +\infty$ otherwise. (Or equivalently $\tilde{\pi}(\ell_i, \ell_j) = \sup\{y \in \mathbb{R} \mid \ell_i(y) < \ell_j(y)\}$.) Let $\preceq$ be the order on the set $\{(i, j) \mid 1 \leqslant i < j \leqslant m\}$ defined by:

$$(i, j) \preceq (i', j') \text{ if and only if } \tilde{\pi}(\ell_i, \ell_j) \leqslant \tilde{\pi}(\ell_{i'}, \ell_{j'}).$$

Assume $(i, j) \preceq (i', j')$ and $\ell_i(\varepsilon^*) \leqslant \ell_j(\varepsilon^*)$. (See figure 2.) From the second condition, it holds

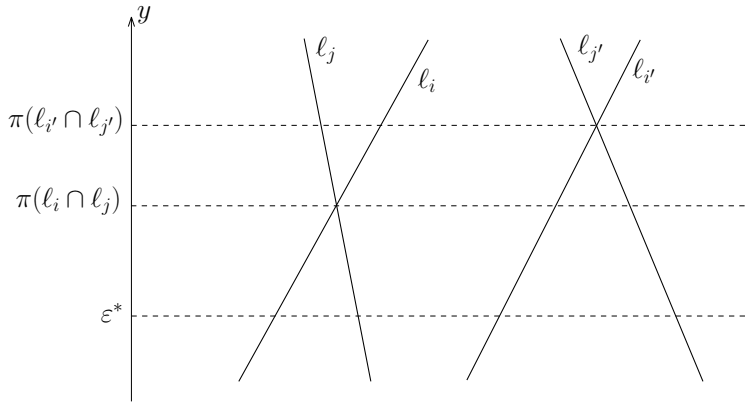

Figure 2: A case where $(i, j) \preceq (i', j')$ and $\ell_i(\varepsilon^*) \leqslant \ell_j(\varepsilon^*)$.

that $\varepsilon^* \leqslant \tilde{\pi}(\ell_i, \ell_j)$, then the first condition yields $\varepsilon^* \leqslant \tilde{\pi}(\ell_{i'}, \ell_{j'})$; it follows that $\ell_{i'}(\varepsilon^*) \leqslant \ell_{j'}(\varepsilon^*)$. Moreover, the order $\preceq$ can obviously be computed in $O(1)$ time.

Hence Theorem 3 allows to compute a permutation σ of $\{1, \ldots, m\}$ such that

$$\ell_{\sigma(1)}(\varepsilon^*) \leqslant \ell_{\sigma(2)}(\varepsilon^*) \leqslant \ldots \leqslant \ell_{\sigma(m)}(\varepsilon^*)$$

in time $O(n \log n)$. By Lemma 2, it holds that $\varepsilon^* \in \{\pi(\ell_{\sigma(i)} \cap \ell_{\sigma(i+1)}) \mid 0 < i < m\}$. After sorting this set, we perform a binary search using the linear-time decision algorithm, and thus we compute ε^* in $O(n \log n)$ time. At last, we run the decision algorithm on ε^*, which gives an optimal k-step function for P. $\square$

3 Concluding remarks

When the input points are given in unsorted order, our algorithm is optimal by reduction from sorting [5]. So an intriguing question is whether there exists an $o(n \log n)$-time algorithm when the input points are given in sorted order. For instance, in the unweighted case, a linear-time algorithm exists if the points are sorted according to their x-coordinates [5].

Our algorithm is mainly of theoretical interest, as Cole's parametric searching technique relies on a sorting network with $O(\log n)$ depth; all known constructions for such networks involve large constants. So it would be interesting to have a practical $O(n \log n)$-time deterministic algorithm.

References

[1] Miklós Ajtai, János Komlós, and Endre Szemerédi. Sorting in $c \log n$ parallel sets. *Combinatorica*, 3(1):1–19, 1983.

[2] Danny Z. Chen and Haitao Wang. Approximating points by a piecewise linear function: I. In *ISAAC*, pages 224–233, 2009.

[3] Danny Z. Chen and Haitao Wang. Efficient algorithms for the weighted k-center problem on a real line. In *ISAAC*, pages 584–593, 2011.

[4] Richard Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, 1987.

[5] Hervé Fournier and Antoine Vigneron. Fitting a step function to a point set. *Algorithmica*, 60(1):95–109, 2011.

[6] Sudipto Guha and Kyuseok Shim. A note on linear time algorithms for maximum error histograms. *IEEE Trans. Knowl. Data Eng.*, 19(7):993–997, 2007.

[7] Panagiotis Karras, Dimitris Sacharidis, and Nikos Mamoulis. Exploiting duality in summarization with deterministic guarantees. In *KDD*, pages 380–389, 2007.

[8] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley Professional, May 1998.

[9] Jin-Yi Liu. A randomized algorithm for weighted approximation of points by a step function. In *COCOA (1)*, pages 300–308, 2010.

[10] Mario A. Lopez and Yan Mayster. Weighted rectilinear approximation of points in the plane. In *LATIN*, pages 642–653, 2008.

[11] Mike Paterson. Improved sorting networks with $O(\log N)$ depth. *Algorithmica*, 5(1):65–92, 1990.