

# Minimizing the Sum of Piecewise Linear Convex Functions

Yu Cong

October 9, 2024

# Plan

---

1. Problems & Definitions
2. Properties
3. LP in Low Dimensions
4. Possible Improvements

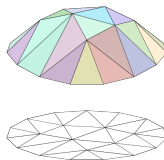
$$\min \sum f_i(a_i \cdot x - b_i)$$

## Problem

Given  $n$  piecewise linear convex functions  $f_1, \dots, f_n : \mathbb{R} \rightarrow \mathbb{R}$  of total  $m$  breakpoints, and  $n$  linear functions  $a_i \cdot x - b_i : \mathbb{R}^d \rightarrow \mathbb{R}$ , find  $\min_x \sum_i f_i(a_i \cdot x - b_i)$ .



(a) A 1D pwl function with 4 line segments and 3 breakpoints



(b) A 2D pwl concave function

$f_i(a_i \cdot x - b_i) : \mathbb{R}^d \rightarrow \mathbb{R}$  is also piecewise linear convex.

## General piecewise linear convex function in $\mathbb{R}^d$

### Definition (piecewise linear convex function in $\mathbb{R}^d$ )

$$g(x) = \max\{a_1^T x + b_1, \dots, a_L^T x + b_L\}$$

Every piecewise linear convex function in  $\mathbb{R}^d$  can be expressed in this form.<sup>1</sup>

However, observe that in our problem the piecewise linear convex function is not that general. It is a composition of a linear mapping and an 1D piecewise linear convex function.

---

<sup>1</sup>S.P. Boyd, L. Vandenberghe, **Convex optimization**, Cambridge University Press, Cambridge, UK ; New York, 2004.

$$f \circ l \not\equiv g$$

---

### Proof.

Consider a piecewise linear convex function  $g : \mathbb{R}^2 \rightarrow \mathbb{R}$ .  $g$  can be viewed as the maximum of a set of planes in  $\mathbb{R}^3$ .

Consider a series of points  $P = \{p_1, p_2, \dots, p_k\}$  on the 2D plane. After applying the linear mapping to  $P$ , we will get a sequence of numbers (points in 1D)  $P' = \{p'_1, p'_2, \dots, p'_k\}$ . We assume that  $P'$  is non-decreasing. Note that the value of  $g$  on  $P'$  is always unimodal since  $g$  is convex. However, the value of  $f$  on  $P$  may not be unimodal. Thus the composition of a linear mapping and a pwl convex function in 1D is not equivalent to pwl convex functions in high dimensions.  $\square$

# A linear time algorithm I

## Problem

*Given  $n$  piecewise linear convex functions  $f_1, \dots, f_n : \mathbb{R} \rightarrow \mathbb{R}$  of total  $m$  breakpoints, and  $n$  linear functions  $a_i \cdot x - b_i : \mathbb{R}^d \rightarrow \mathbb{R}$ , find  $\min_x \sum_i f_i(a_i \cdot x - b_i)$ .*

This can be solve in  $O(2^{2^d}(m + n))$  through Megiddo's Low dimension LP algorithm.<sup>2</sup>

Let  $n_i$  be the number of line segments in  $f_i$ . Note that

$$\sum_i n_i = m + n.$$

We can formulate the optimization problem as the following linear program,

## A linear time algorithm II

---

$$\begin{aligned} \min \quad & \sum_{i=1}^n f_i \\ \text{s.t.} \quad & f_i \geq \alpha_j(a_i \cdot x - b_i) - \beta_j \quad \forall i \in [n], \forall j \end{aligned}$$

where  $\alpha_j x - \beta_j$  is the  $j$ 'th line segment on  $f_i$ .

There will be  $m + n$  constraints in total.

---

<sup>2</sup>Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. J. ACM, 31(1):114–127, jan 1984.

# Megiddo's algorithm I

---

[https://people.inf.ethz.ch/gaertner/subdir/texts/own\\_work/chap50-fin.pdf](https://people.inf.ethz.ch/gaertner/subdir/texts/own_work/chap50-fin.pdf)

The dimension  $d$  (in our problem, the dimension of  $x$ ) is small while the number of constraints are huge. We need only  $d$  linearly independent tight constraints to identify the optimal solution  $x^*$ . Thus most of the constraints are useless.

**For one constraint, how can we know where does  $x^*$  locate with respect to it?**

Through inquiries. Let  $a \cdot x \leq b$  be the constraint. Define 3 hyperplanes,  $a \cdot x = c$  where  $c \in \{b, b - \varepsilon, b + \varepsilon\}$ . Now solve three  $d - 1$  dimension linear programming. The largest of the three objective functions tells us where  $x^*$  lies with respect to the hyperplane.



## Megiddo's algorithm II

Finding the optimal solution  $x^*$  is therefore equivalent to the following problem,

### Problem (Multidimensional Search Problem)

*Suppose that there exists a point  $x^*$  which is not known to us, but there is a oracle that can tell the position of  $x^*$  relative to any hyperplane in  $\mathbb{R}^d$ . Given  $n$  hyperplanes, we want to know the position of  $x^*$  relative to each of them.*

**What about 1 dimension search?** A fastest way will be using the linear time median algorithm. We can find the median of  $n$  numbers and call the oracle to compare the median with  $x^*$ . Thus with  $O(n)$  time median finding and one oracle call, we find the relative position of  $n/2$  elements relative to  $x^*$ .

## Megiddo's algorithm III

---

If we can do similar things in  $\mathbb{R}^d$ , i.e., there is a method which makes  $A(d)$  oracle calls and determines at least  $B(d)$  fraction of relative positions, then we can apply this method  $\log_{\frac{1}{1-B(d)}} n$  times to find all relative positions.

Note that in 1 dimension,  $A(1) = 1$  and  $B(1) = 1/2$  (call oracle to compare  $x^*$  and the median). In  $\mathbb{R}^d$ , our oracle is the recursive inquiry.

A trivial method will be iterating on all hyperplanes and calling the oracle on each one, since there is no *median* of a set of hyperplanes in  $\mathbb{R}^d$ . The complexity recurrence is

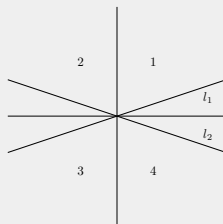
$$T(n, d) = n(3T(n-1, d-1) + O(nd))$$

Note that in this setting  $A(d) = 1$  and  $B(d) = 1/n$ .

## Megiddo's algorithm IV

Megiddo designed a clever method where  $A(d) = 2^{d-1}$  and  $B(d) = 2^{-(2^d-1)}$ .

### Lemma



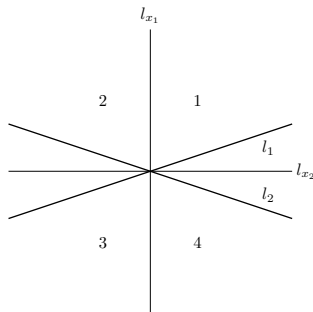
*Given two lines through the origin with slopes of opposite sign, knowing which quadrant  $x^*$  lies in allows us to locate it with respect to at least one of the lines.*

## Megiddo's algorithm V

Let  $l_H$  be the intersection of hyperplane  $H$  and  $x_1x_2$  plane.

Compute a partition  $S_1 \sqcup S_2 = \mathcal{H}$ .  $H \in S_1$  iff  $l_H$  has positive slope.

Otherwise  $l_H \in S_2$ . We further assume that  $|S_1| = |S_2| = n/2$ .



Now we have  $n/2$  pairs  $(H_1, H_2)$ , where  $H_i \in S_i$ . Let  $l_i$  be the intersection of  $H_i$  and  $x_1x_2$  plane. Let  $H_{x_i}$  be the linear combination of  $H_1$  and  $H_2$  s.t.  $x_i$  is eliminated.

By the previous lemma, calling oracle on  $l_{x_1}$  and  $l_{x_2}$  locate  $x^*$  with respect to at least one of  $H_1$  and  $H_2$ .

## Megiddo's algorithm VI

---

Input:  $S_1, S_2$  and the pairs.

- 1 recursively locate  $x^*$  respect to  $B(d-1)n/2$  hyperplanes( $H_{x_i}$ ) with  $A(d-1)$  oracle calls in  $S_1$ .
- 2 locate with respect to a  $B(d-1)$ -fraction of corresponding paired hyperplanes in  $S_2$ .
- 3 There are still  $(1 - B(d-1)^2)/2$ -fraction of hyperplanes for which we do not know the relative position with  $x^*$ . Run this algorithm on these hyperplanes.

This gives the recurrence

$$T(n, d) \leq 3 \cdot 2^{d-1} T(n, d-1) + T((1 - 2^{1-2^d})n, d) + O(nd)$$

with solution  $T(n, d) = O(2^{2^d} n)$ .

## Zemel's conversion

---

$$\begin{aligned} \min \quad & \sum_{i=1}^n f_i \\ \text{s.t.} \quad & f_i \geq \alpha_j(a_i \cdot x - b_i) - \beta_j \quad \forall i \in [n], \forall j \end{aligned}$$

Our linear program has *dimension*  $n + d$ . **Zemel** showed that this kind of problem can be solved in linear time.

This is a *d-dimensional search problem* with  $n + d$  hyperplanes.

# Other algorithms for fixed dimension LP

simplex method	det.	$O(n/d)^{d/2+O(1)}$
Megiddo [24]	det.	$2^{O(2^d)}n$
Clarkson [9]/Dyer [14]	det.	$3^{d^2}n$
Dyer and Frieze [15]	rand.	$O(d)^{3d}(\log d)^d n$
Clarkson [10]	rand.	$d^2n + O(d)^{d/2+O(1)} \log n + d^4 \sqrt{n} \log n$
Seidel [26]	rand.	$d!n$
Kalai [19]/Matoušek, Sharir, and Welzl [23]	rand.	$\min\{d^2 2^d n, e^{2\sqrt{d \ln(n/\sqrt{d})} + O(\sqrt{d} + \log n)}\}$
combination of [10] and [19, 23]	rand.	$d^2n + 2^{O(\sqrt{d \log d})}$
Hansen and Zwick [18]	rand.	$2^{O(\sqrt{d \log((n-d)/d)})}n$
Agarwal, Sharir, and Toledo [4]	det.	$O(d)^{10d}(\log d)^{2d}n$
Chazelle and Matoušek [8]	det.	$O(d)^{7d}(\log d)^d n$
Brönnimann, Chazelle, and Matoušek [5]	det.	$O(d)^{5d}(\log d)^d n$
<del>this paper</del> Chan	det.	$O(d)^{d/2}(\log d)^{3d}n$

Figure: Algorithms for LP in low dimensions <sup>3</sup>

**Can we use faster fixed dimension LP algorithms to get better complexity?**

<sup>3</sup>table stolen from <https://dl.acm.org/doi/10.1145/3155312>

## LP-type problem I

Algorithms for low dim LP are actually solving a more abstract problem.

### Definition (LP-type problem)

Given a set  $S$  and a function  $f : S \rightarrow \mathbb{R}$ .  $f$  satisfies two properties:

- Monotonicity:  $\forall A \subseteq B \subseteq S, f(A) \leq f(B) \leq f(S)$ .
- Locality:  $\forall A \subseteq B \subseteq S$  and  $\forall x \in S$ , if  $f(A) = f(B) = f(A \cup \{x\})$ , then  $f(A) = f(B \cup \{x\})$ .

Linear programs(minimization) are LP-type problems.

$B \subseteq S$  is a basis if  $\forall B' \subsetneq B, f(B') < f(B)$ . A set of 'useful' constraints in a linear program is a basis.

The combinatorial dimension is the size of the largest basis.

If a LP problem has low dimension, then its combinatorial dimension is low. **What about the converse?**



## LP-type problem II

---

$$\begin{aligned} \min \quad & \sum_{i=1}^n f_i \\ \text{s.t.} \quad & f_i \geq \alpha_j(a_i \cdot x - b_i) - \beta_j \quad \forall i \in [n], \forall j \\ & \dots \end{aligned}$$

### Does our LP has low combinatorial dimension?

No. A basis contains at least  $n$  constraints since otherwise some  $f_i$  is unbounded.

### Problem

*Is it possible to formulate the pwl convex minimization problem as an LP-type problem with low combinatorial dimension?*

## Aggregate the pwl convex functions

---

The sum of pwl convex functions are still pwl convex.

If we can compute  $F = \sum f_i$  in  $O(m)$  and the number of line segments on  $F$  is also  $O(m)$ , then the corresponding LP will have low combinatorial dimension.

$$\begin{aligned} \min \quad & F \\ \text{s.t.} \quad & F \geq \alpha_j \cdot x - \beta_j \quad \forall j \\ & \dots \end{aligned}$$

However, this is not possible for general pwl convex functions in  $\mathbb{R}^d$ .<sup>4</sup>

---

<sup>4</sup>see this [blog post](#) for detail.