Learned Index Structures

CS 263

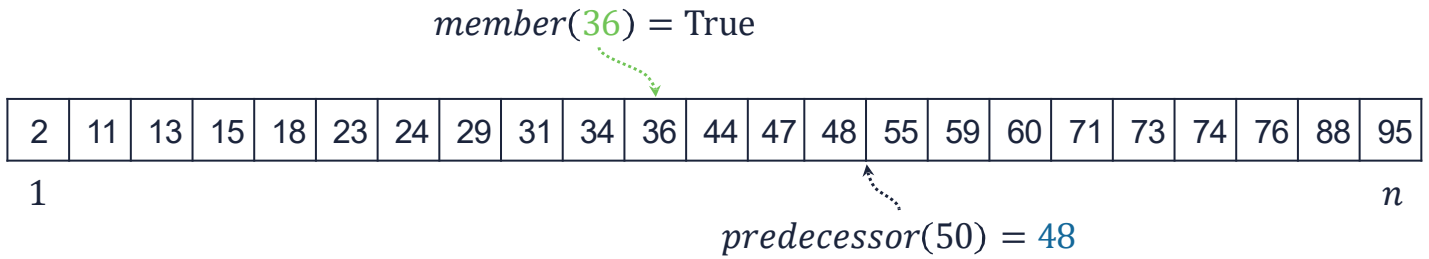
Michael T. Goodrich

University of California, Irvine

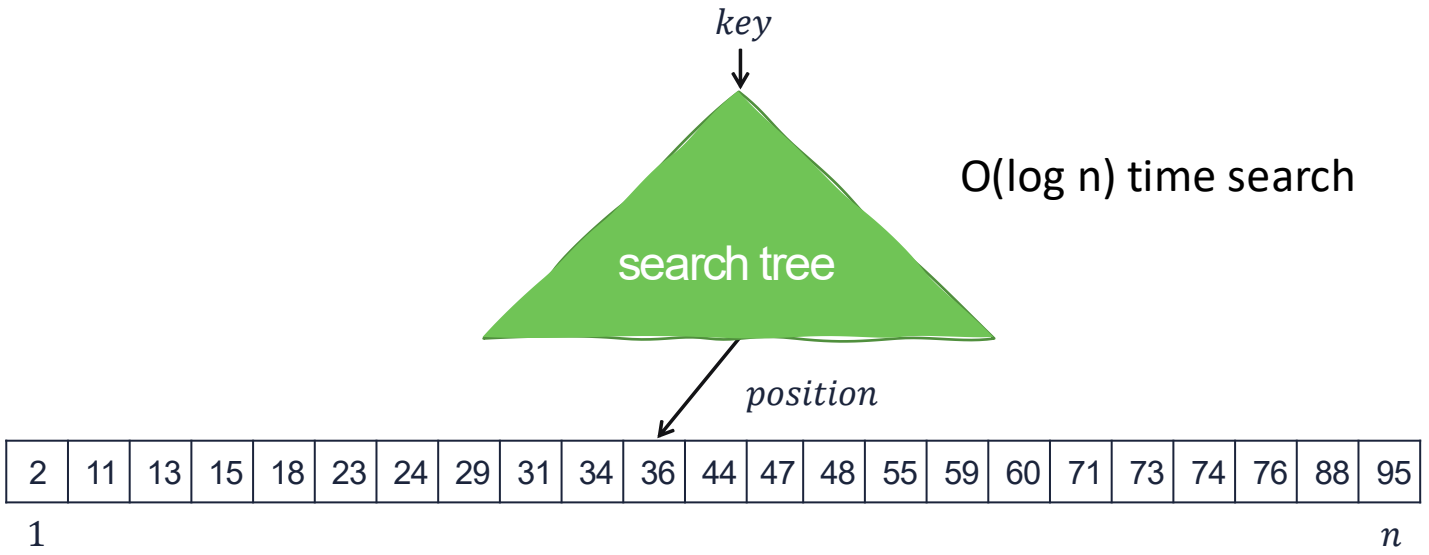
Some slides adapted from content by Paolo Ferragina, Fabrizio Lillo, Giorgio Vinciguerra, Carola Wenk, Joseph O'Rourke

A classical search problem in computer science

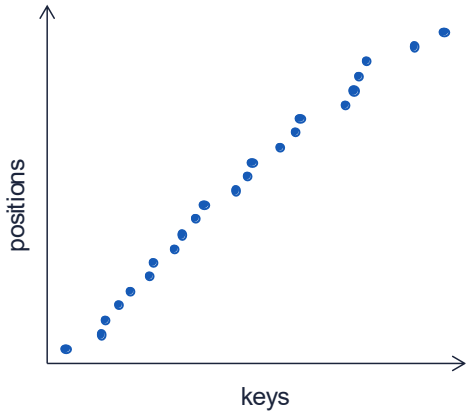
- Given a set of n sorted input keys (e.g. integers)
- Implement membership and predecessor queries
- Range queries in databases, conjunctive queries in search engines, IP lookup in routers…



Classical Index Structures

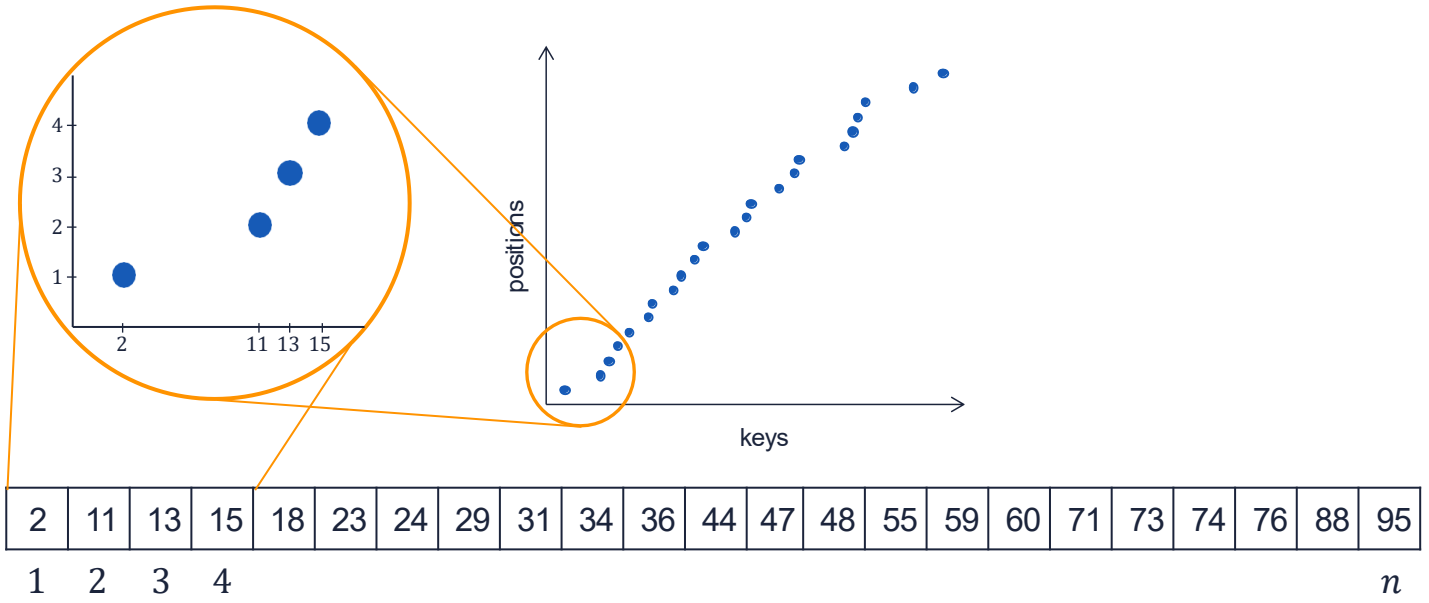


Input data as pairs $(key, position)$

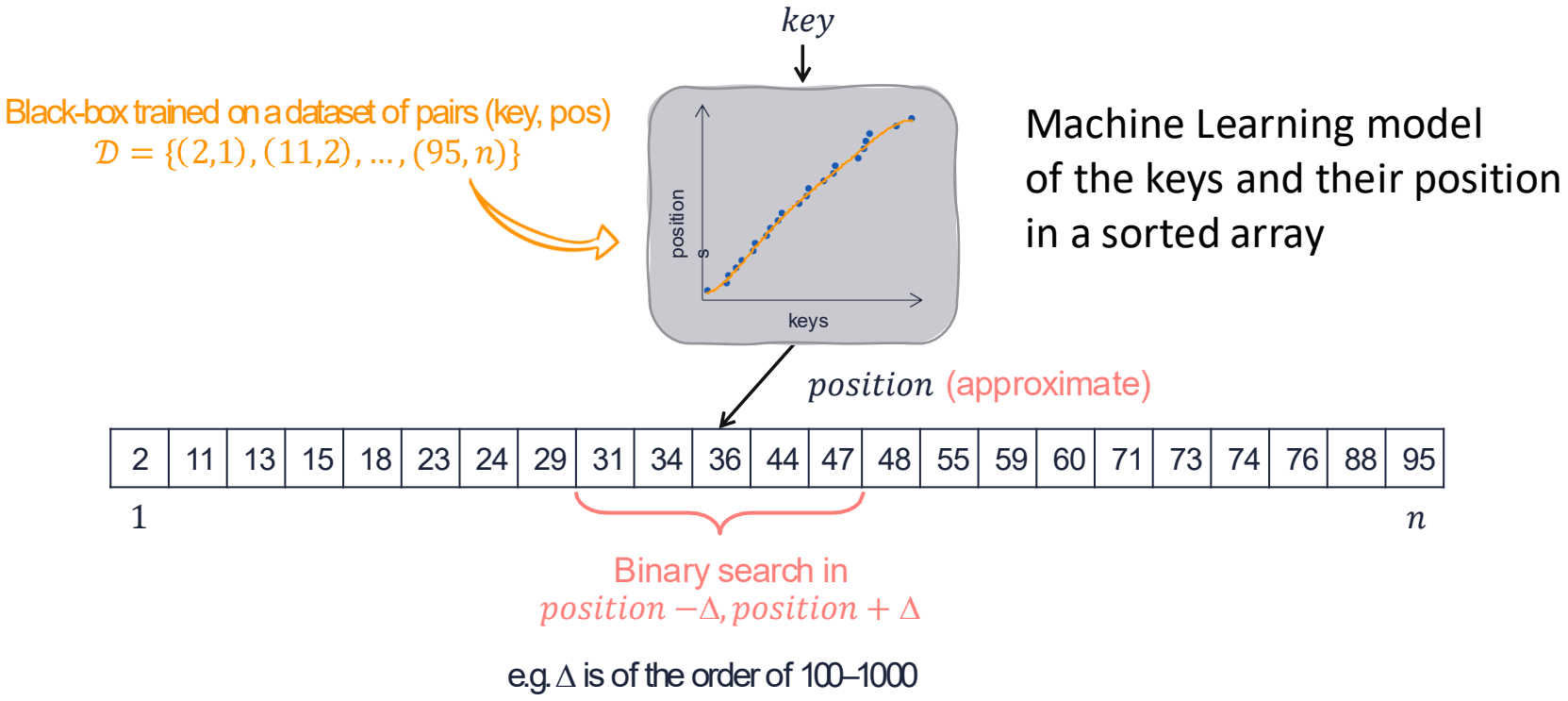


2	11	13	15	18	23	24	29	31	34	36	44	47	48	55	59	60	71	73	74	76	88	95
1																						n

Input data as pairs $(key, position)$



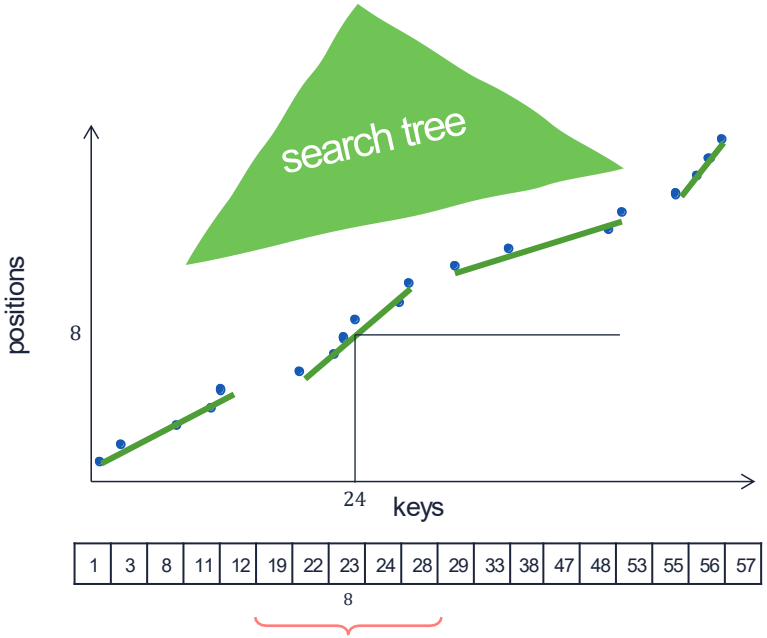
Learned index structures



PGM-index

[Ferragina and Vinciguerra, PVLDB 2020]

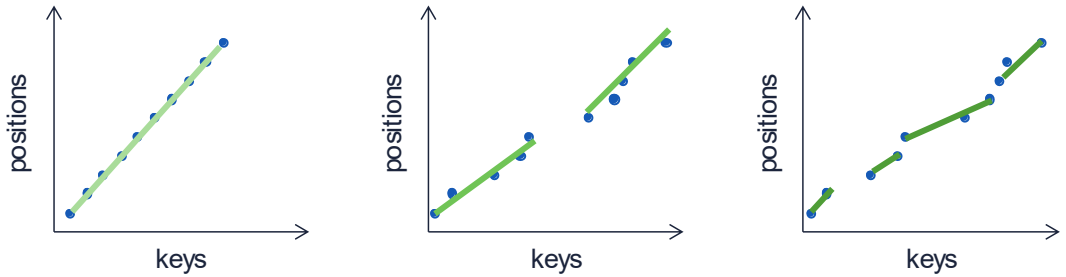
1. Fix a max error Δ, e.g. so that keys in $[pos - \Delta, pos + \Delta]$ fit in a cache-line
2. Find the smallest Piecewise Linear Δ-Approximation (PLA), e.g., with λ pieces
3. Store triples $(first\,key, slope, intercept)$ for each segment



We can do search
in O(log Δ + log λ) time

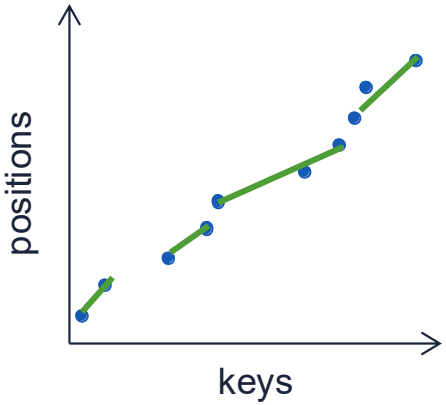
New Idea: Exponential Search to Find the Balance Point for Δ and λ

- Suppose we have an O(n) – time algorithm for finding the minimum piecewise-linear approximation (PLA) for a given value of Δ.
- We can then do an exponential search to find the balance point for log Δ and log λ.
- Start with $\Delta = 2$, and keep squaring until log Δ and log λ cross.



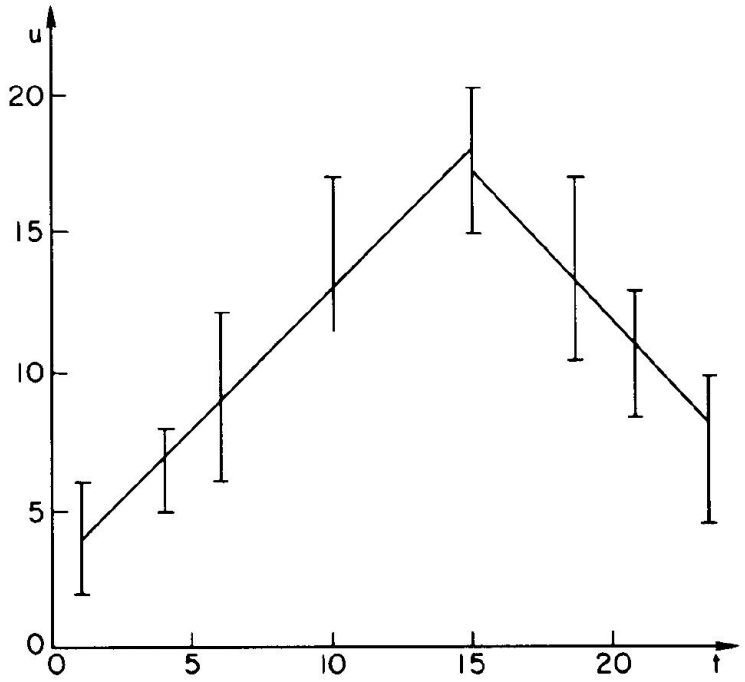
Piecewise Linear Approximation

- Given a sorted set of keys and an error term Δ, we need to find a piecewise linear approximation for this set of keys with the minimum number of line segments in O(n) time that has error Δ.



PLA Algorithm

- We use a classic algorithm due to O'Rourke.
  - Model the input as key-rank (k,r) points.
  - Expand each (k,r) as a vertical segment from (k,r-Δ) to (k,r+Δ).
  - We want a piecewise linear approximation that intersects each vertical line segment.



Point-Line Duality

Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^2$ be a set of n points. Now define a set $P^* = \{p_1^*, \dots, p_n^*\}$ of n lines as follows:

Primal plane

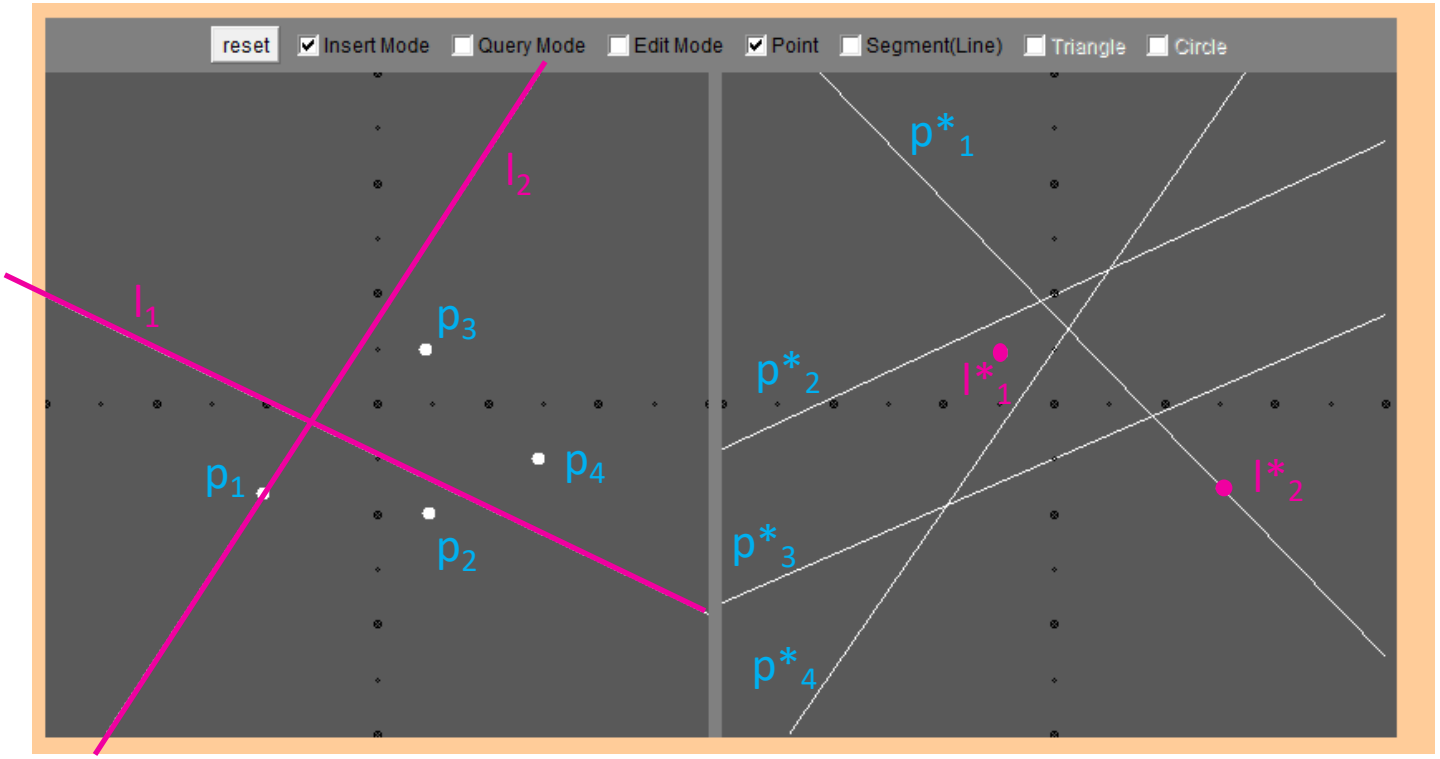
Point: $p = (p_x, p_y)$

Line: $l: y = mx + b$

Dual plane

Line: $p^*: y = p_x x - p_y$

Point: $l^* = (m, -b)$

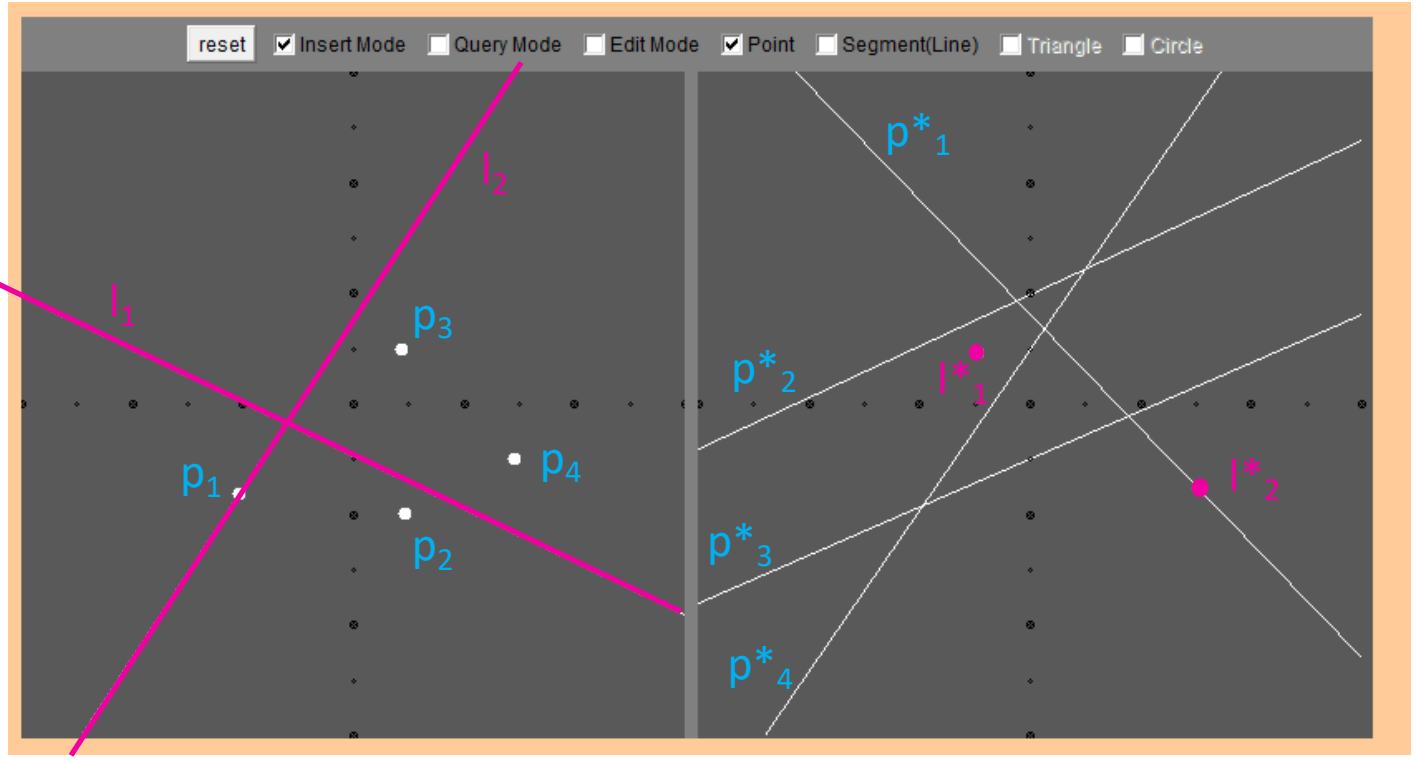


Primal plane	Dual plane
Point: $p = (p_x, p_y)$	Line: $p^*: y = p_x x - p_y$
Line: $l: y = mx + b$	Point: $l^* = (m, -b)$

Properties

Primal

Dual



- $(p^*)^* = p$
- $p \in l \Leftrightarrow l^* \in p^*$ *incidence-preserving*
- p lies above $l \Leftrightarrow l^*$ lies above p^*

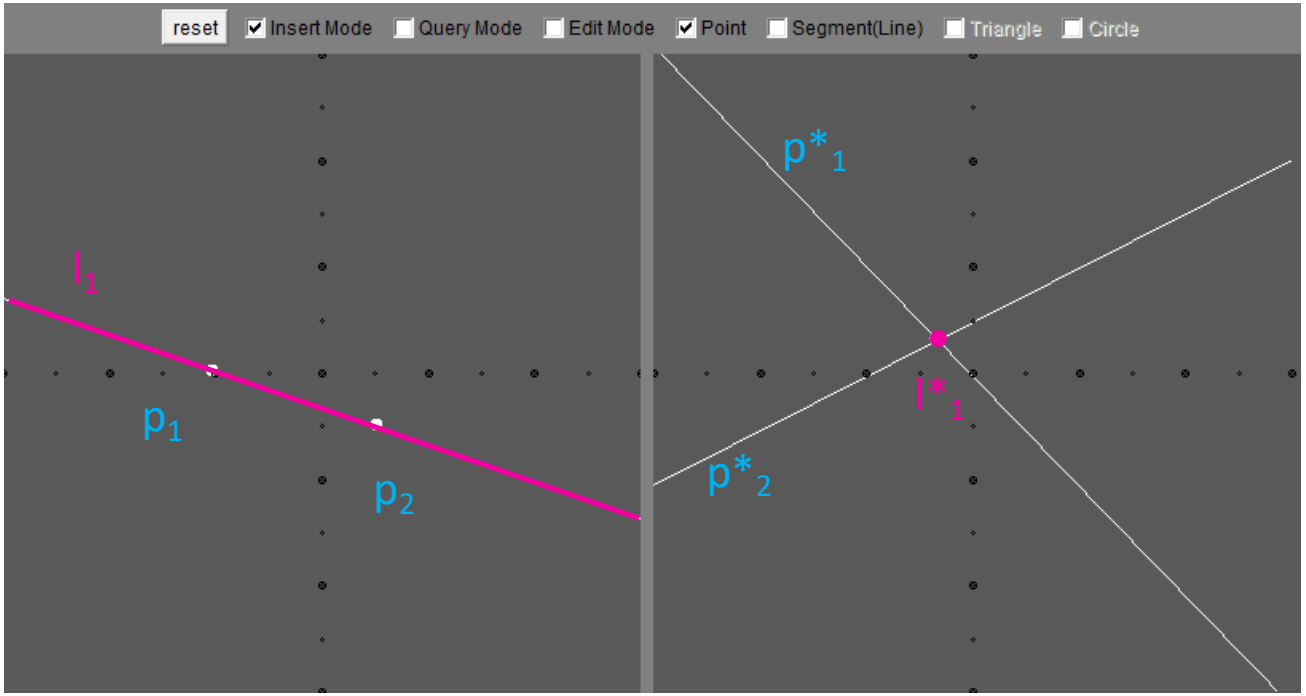
- $p_1 \in l_2 \Leftrightarrow l^*_2 \in p^*_1$
- p_3 is above $l_1 \Leftrightarrow l^*_1$ is above p^*_3

Primal plane	Dual plane
Point: $p = (p_x, p_y)$	Line: $p^*: y = p_x x - p_y$
Line: $l: y = mx + b$	Point: $l^* = (m, -b)$

Properties

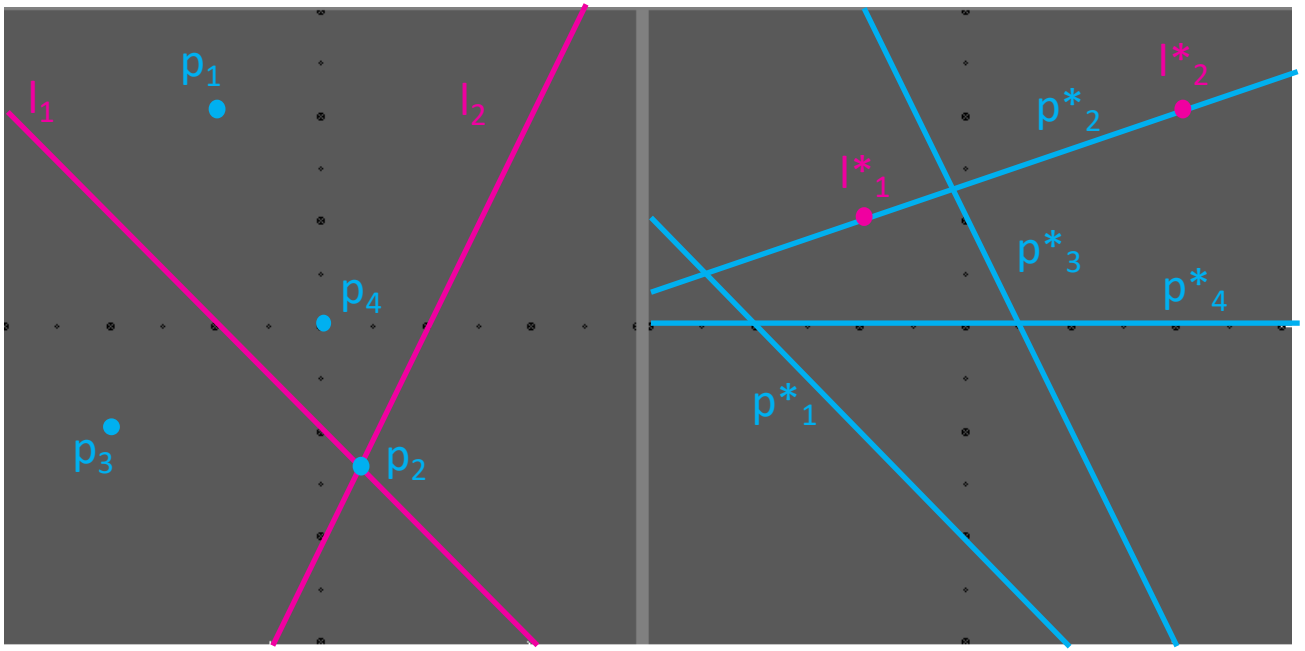
Primal

Dual



- Points p_1 and p_2 lie on line l_1
- Lines p_1^* and p_2^* contain point l_1^*

Point-Line Duality Puzzle



p_1 lies above l_1 $\Leftrightarrow$ l_1^* lies above p_1^*

p_2 lies on l_1 $\Leftrightarrow$ l_1^* lies on p_2^*

p_2 lies on l_2 $\Leftrightarrow$ l_2^* lies on p_2^*

p_3 lies below l_1 $\Leftrightarrow$ l_1^* lies below p_3^*

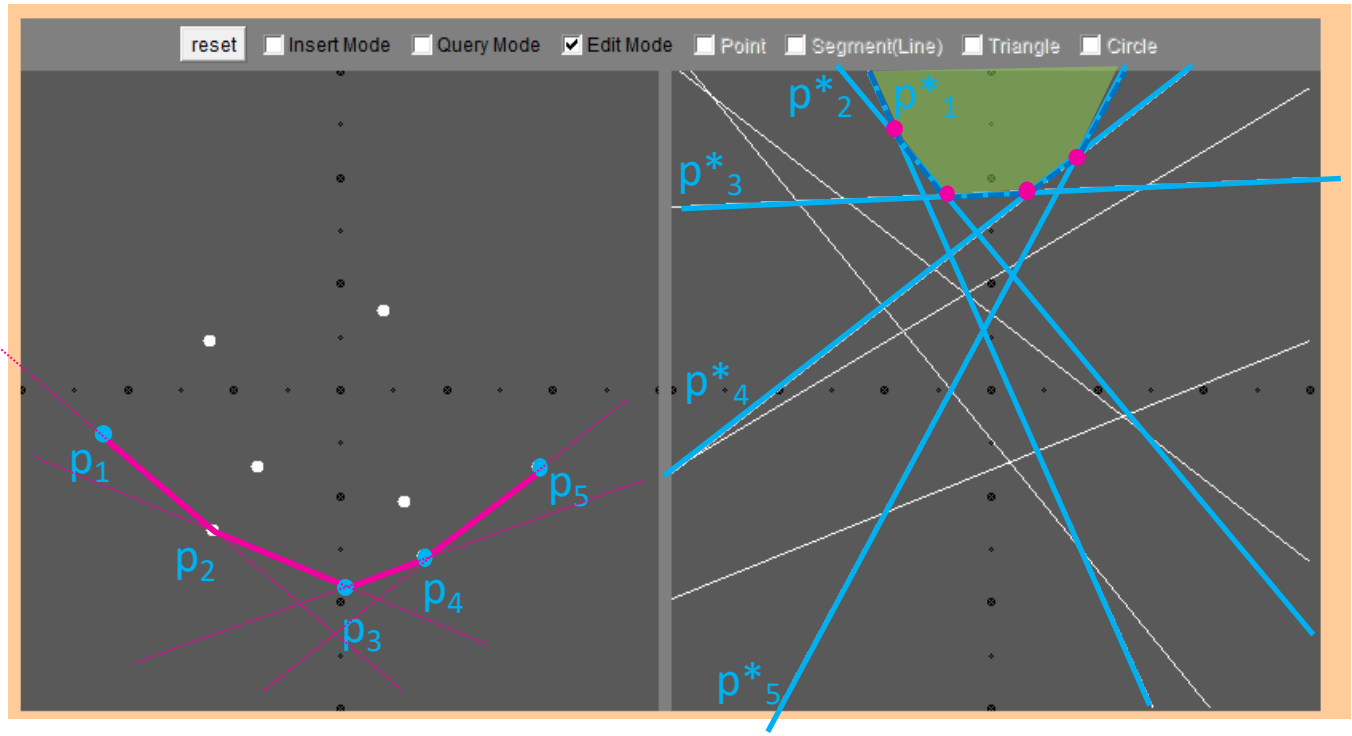
p_4 lies above l_2 $\Leftrightarrow$ l_2^* lies above p_4^*

Lower Convex Hull ≅ Intersecting Halfspaces

Primal plane	Dual plane
Point: $p = (p_x, p_y)$	Line: $p^*: y = p_x x - p_y$
Line: $l: y = mx + b$	Point: $l^* = (m, -b)$

Primal

Dual



lower convex hull = halfplane intersection (of upper halfplanes)

- LCH = p_1, p_2, p_3, p_4, p_5
- UE = p^*_1, p^*_2, p^*_3, p^*_4, p^*_5

Using Point-Line Duality for PLA

- Each vertical line segment defines constraints on each line that can intersect the segment (k,r-D) to (k,r+D). This defines a strip between two parallel lines in the dual space.

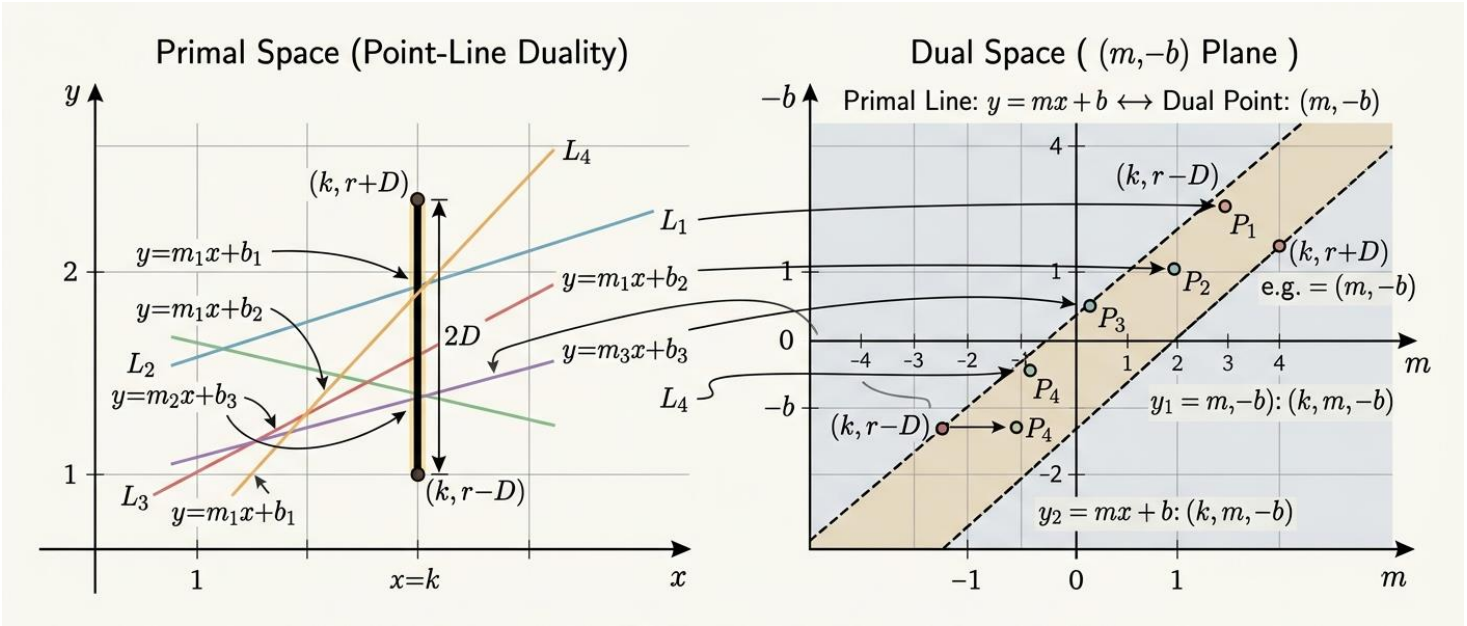
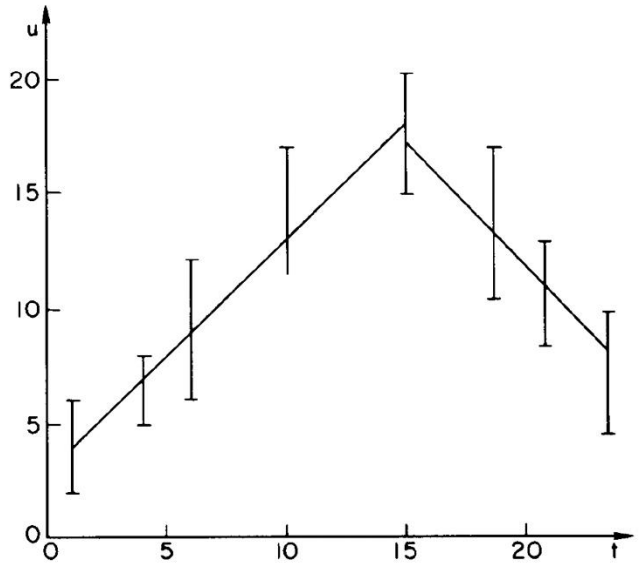


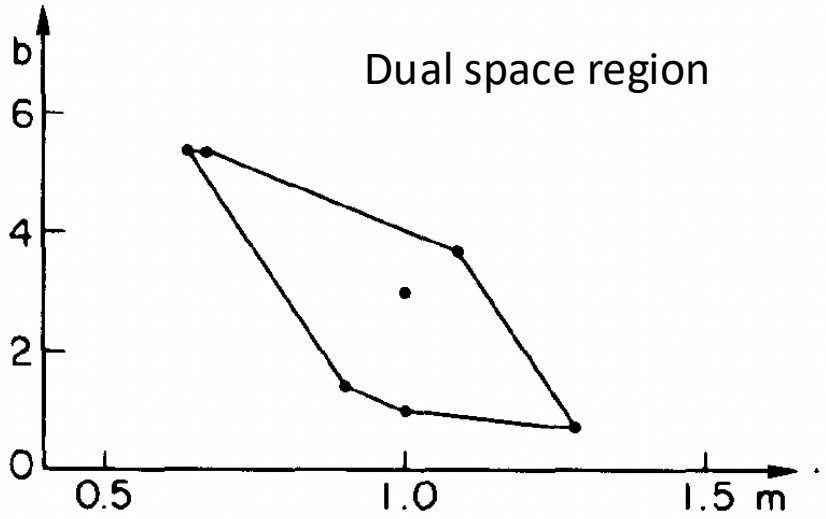
Image generated using Nano Banana Pro.

The Greedy Algorithm (High Level)

- Consider the vertical segments one at a time, maintaining a representation in dual space of all the lines that can intersect all the segments considered so far.



Primal space ranges



Dual space region

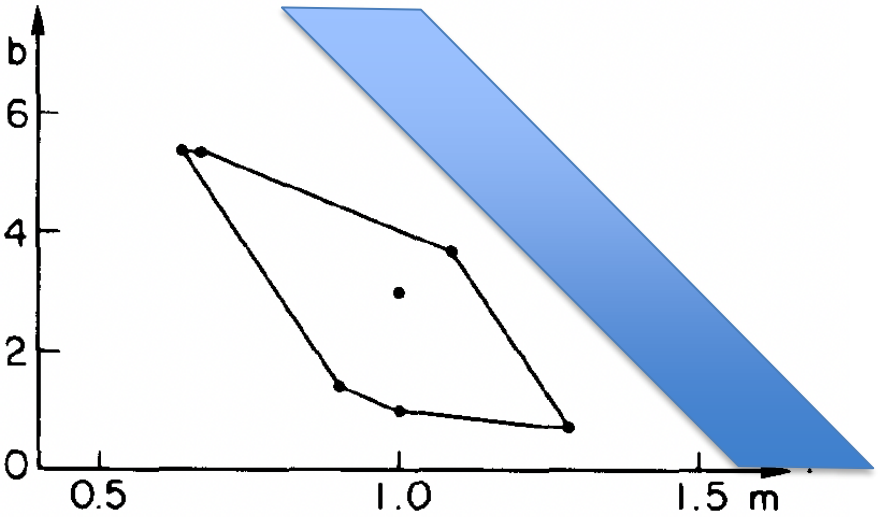
The convex polygon in m-b dual space corresponding to the first 5 data ranges of the left figure.
Every point inside the polygon represents a straight line that fits the 5 data ranges. The single interior corresponds to the straight line drawn in the figure.

The Greedy Step

- Consider the vertical segments one at a time, maintaining a representation in dual space of all the lines that can intersect all the segments considered so far.
- If a new set of constraints for a vertical segment causes the region to become empty, output a line approximating the segments considered so far and start over with the new segment.

constraints for the new segment

Dual space region

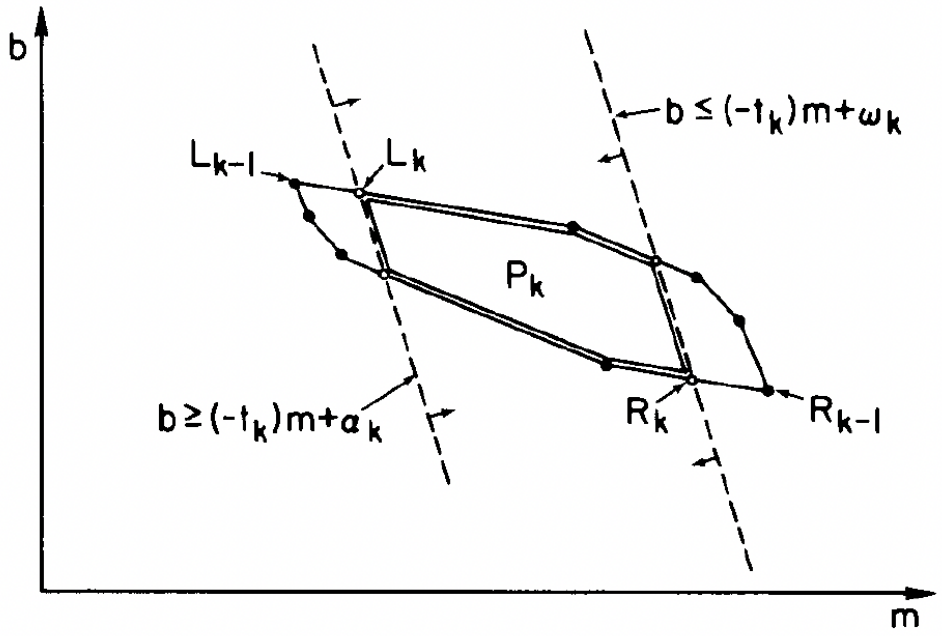


Why does this greedy strategy give the minimum number of pieces?

Maintaining the Constraint Polygon

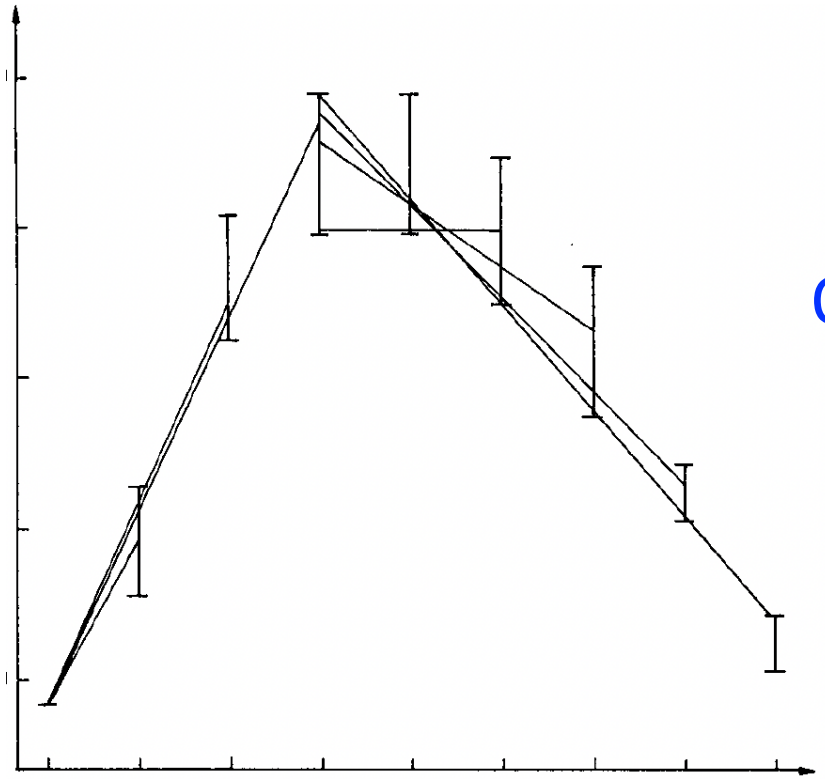
- We are maintaining a convex polygon of the common intersection of the strips, adding new strips one at a time.
- We can do the update in amortized constant time, by maintaining L_k and R_k, the leftmost and rightmost points of the constraint polygon.

Fig. 4. Polygon P_k is Formed by Intersecting Polygon P_{k-1} with the Two Half-Planes Derived from the kth Data Range. The search for the intersection points between polygon P_{k-1} and the right half-plane edge starts at R_{k-1} and moves towards L_{k-1} along the upper and lower polygon halves; for the left half-plane edge, the search proceeds from L_{k-1} to R_{k-1}. The points falling outside of the half-planes are discarded, and polygon P_k is what remains.



Primal Space View

- This will construct a sequence of candidate lines, resulting in the minimum number of pieces in the piecewise linear approximation



O(n) time in total

Future Work

- **There are a lot of possible problems to work on in the future**:
- What if we allow gaps in the positional array?
- What if we want to insert and/or delete keys from our data set?
- What if we want to build a learned index structure for external memory, where we need to minimize the number input/output operations?