Algorithms for Isotonic Regression

Günter Rote

Freie Universität Berlin
Minimize $\sum_{i=1}^{n} h(|x_i - a_i|)$ subject to $x_1 \leq \cdots \leq x_n$
Objective functions

Minimize $\sum_{i=1}^{n} h(|x_i - a_i|)$ subject to $x_1 \leq \cdots \leq x_n$

- $h(z) = z$: $L_1$-regression
  - $\sum_{i=1}^{n} |x_i - a_i| \rightarrow \text{min}$

- $h(z) = z^2$: $L_2$-regression
  - $\sum_{i=1}^{n} (x_i - a_i)^2 \rightarrow \text{min}$

- $h(z) = z^p, p \rightarrow \infty$: $L_\infty$-regression
  - $\max_{1 \leq i \leq n} |x_i - a_i| \rightarrow \text{min}$

versions with weights $w_i > 0$:

- $\sum_{i=1}^{n} w_i |x_i - a_i|$
- $\sum_{i=1}^{n} w_i (x_i - a_i)^2$
- $\max_{1 \leq i \leq n} w_i |x_i - a_i|$

General form: $\sum_{i=1}^{n} h_i(x_i) \rightarrow \text{min} / \max_{1 \leq i \leq n} h_i(x_i) \rightarrow \text{min}$

(*) $h_i$ convex and piecewise “simple”
Overview

- The classical *Pool Adjacent Violators (PAV)* algorithm
- Dynamic programming
- More general constraints:

\[ x_i \leq x_j \text{ for } i \prec j \]

with a given partial order \( \prec \)
- In particular, \( L_{\text{max}} \) regression with a \( d \)-dimensional partial order
- Randomized optimization technique of Timothy Chan (1998)
Pool Adjacent Violators (PAV)

Minimize \( \sum_{i=1}^{n} h(|x_i - a_i|) \) subject to \( x_1 \leq \cdots \leq x_n \)
Pool Adjacent Violators (PAV)

Minimize \( \sum_{i=1}^{n} h(|x_i - a_i|) \) subject to \( x_1 \leq \cdots \leq x_n \)

1. Relax all \( x_i \leq x_{i+1} \)
Pool Adjacent Violators (PAV)

Minimize $\sum_{i=1}^{n} h(|x_i - a_i|)$ subject to $x_1 \leq \cdots \leq x_n$

1. Relax all $x_i \leq x_{i+1}$
Pool Adjacent Violators (PAV)

Minimize \( \sum_{i=1}^{n} h(|x_i - a_i|) \) subject to \( x_1 \leq \cdots \leq x_n \)

1. Relax all \( x_i \leq x_{i+1} \)

2. Find adjacent violators \( x_i > x_{i+1} \)
Pool Adjacent Violators (PAV)

Minimize \( \sum_{i=1}^{n} h(|x_i - a_i|) \) subject to \( x_1 \leq \cdots \leq x_n \)

1. Relax all \( x_i \leq x_{i+1} \)

2. Find adjacent violators \( x_i > x_{i+1} \)

3. Pool them: \( x_i = x_{i+1} \), and solve

\[ h(|x - a_{10}|) + h(|x - a_{11}|) \rightarrow \text{min} \]
Pool Adjacent Violators (PAV)

Minimize \( \sum_{i=1}^{n} h(|x_i - a_i|) \) subject to \( x_1 \leq \cdots \leq x_n \)

1. Relax all \( x_i \leq x_{i+1} \)
2. Find adjacent violators \( x_i > x_{i+1} \)
3. Pool them: \( x_i = x_{i+1} \), and solve

\[
h(|x - a_{10}|) + h(|x - a_{11}|) \rightarrow \text{min}
\]
Pool Adjacent Violators (PAV)

Minimize \( \sum_{i=1}^{n} h(|x_i - a_i|) \) subject to \( x_1 \leq \cdots \leq x_n \)

1. Relax all \( x_i \leq x_{i+1} \)
2. Find adjacent violators \( x_i > x_{i+1} \)
3. Pool them: \( x_i = x_{i+1} \), and solve
4. Repeat from step 2.
Pool Adjacent Violators (PAV)

Minimize $\sum_{i=1}^{n} h(|x_i - a_i|)$ subject to $x_1 \leq \cdots \leq x_n$

1. Relax all $x_i \leq x_{i+1}$

2. Find adjacent violators $x_i > x_{i+1}$

3. Pool them: $x_i = x_{i+1}$, and solve

4. Repeat from step 2.
Pool Adjacent Violators (PAV)

Minimize $\sum_{i=1}^{n} h(|x_i - a_i|)$ subject to $x_1 \leq \cdots \leq x_n$

1. Relax all $x_i \leq x_{i+1}$
2. Find adjacent violators $x_i > x_{i+1}$
3. Pool them: $x_i = x_{i+1}$, and solve
4. Repeat from step 2.
Pool Adjacent Violators (PAV)

Minimize $\sum_{i=1}^{n} h(|x_i - a_i|)$ subject to $x_1 \leq \cdots \leq x_n$

1. Relax all $x_i \leq x_{i+1}$

2. Find adjacent violators $x_i > x_{i+1}$

3. Pool them: $x_i = x_{i+1}$, and solve

4. Repeat from step 2.
Pool Adjacent Violators (PAV)

Minimize $\sum_{i=1}^{n} h(|x_i - a_i|)$ subject to $x_1 \leq \cdots \leq x_n$

1. Relax all $x_i \leq x_{i+1}$
2. Find adjacent violators $x_i > x_{i+1}$
3. Pool them: $x_i = x_{i+1}$, and solve
4. Repeat from step 2.
Pool Adjacent Violators (PAV)

Minimize $\sum_{i=1}^{n} h(|x_i - a_i|)$ subject to $x_1 \leq \cdots \leq x_n$

1. Relax all $x_i \leq x_{i+1}$

2. Find adjacent violators $x_i > x_{i+1}$

3. Pool them: $x_i = x_{i+1}$, and solve

4. Repeat from step 2.
Pool Adjacent Violators (PAV)

Minimize $\sum_{i=1}^{n} h(|x_i - a_i|)$ subject to $x_1 \leq \cdots \leq x_n$

1. Relax all $x_i \leq x_{i+1}$

2. Find adjacent violators $x_i > x_{i+1}$

3. Pool them: $x_i = x_{i+1}$, and solve

4. Repeat from step 2.
Pool Adjacent Violators (PAV)

Minimize $\sum_{i=1}^{n} h(|x_i - a_i|)$ subject to $x_1 \leq \cdots \leq x_n$

1. Relax all $x_i \leq x_{i+1}$
2. Find adjacent violators $x_i > x_{i+1}$
3. Pool them: $x_i = x_{i+1}$, and solve
4. Repeat from step 2.
Subproblems

\[
\min \sum_{s \leq i \leq t} w_i |x - a_i| \quad \implies x^* = \text{weighted median of } a_s, \ldots, a_t
\]

\[
\min \sum_{s \leq i \leq t} w_i (x - a_i)^2 \quad \implies x^* = \text{weighted mean of } a_s, \ldots, a_t
\]

\[
x^* = \frac{\sum_{s \leq i \leq t} w_i a_i}{\sum_{s \leq i \leq t} w_i} \quad \text{in } O(1) \text{ time, after } O(n) \text{ preprocessing}
\]

Weighted isotonic \(L_2\) regression is solvable in \(O(n)\) time.
Subproblems

$$\min \sum_{s \leq i \leq t} w_i |x - a_i| \implies x^* = \text{weighted median of } a_s, \ldots, a_t$$

Ahuja and Orlin [2001]:
\(O(n \log n)\) algorithm based on PAV and scaling:
- Solve the problem for scaled (integer) data \(\overline{a}_i := 2 \lfloor a_i / 2 \rfloor\).
- Solution for original data \(a_i\) can be recovered in \(O(n)\) time.
- We can assume \(a_i \in \{1, 2, \ldots, n\}\), after sorting.

Quentin Stout [2008]: \(O(n \log n)\) PAV implementation
- median queries by mergeable trees (2-3-trees, AVL trees) extended with weight information

Rote [2012]: \(O(n \log n)\) by dynamic programming.
- A priority queue is sufficient
Dynamic programming

\[ f_k(z) := \min \left\{ \sum_{i=1}^{k} w_i \cdot |x_i - a_i| : x_1 \leq x_2 \leq \cdots \leq x_k = z \right\} \]

Rekursion:

\[ f_k(z) := \min\{ f_{k-1}(x) : x \leq z \} + w_k \cdot |z - a_k| \]
Dynamic programming

\[ f_k(z) := \min \left\{ \sum_{i=1}^{k} w_i \cdot |x_i - a_i| : x_1 \leq x_2 \leq \cdots \leq x_k = z \right\} \]

\( k = 0, 1, \ldots, n; \quad z \in \mathbb{R} \)

Rekursion:

\[ f_k(z) := \min \left\{ f_{k-1}(x) : x \leq z \right\} + w_k \cdot |z - a_k| \]

\[ g_{k-1}(z) \]

Transform \( f_{k-1} \rightarrow g_{k-1} \rightarrow f_k \)
Recursion step 1

Transform $f_{k-1}$ to $g_{k-1}(z) := \min\{ f_{k-1}(x) : x \leq z \}$
Recursion step 1

Transform \( f_{k-1} \) to \( g_{k-1}(z) := \min\{ f_{k-1}(x) : x \leq z \} \)

Remove the increasing parts right of the minimum \( p_{k-1} \) and replace them by a horizontal part.
Recursion step 2

Transform $g_{k-1}$ to $f_k(z) = g_{k-1}(z) + w_k \cdot |z - a_k|

- Add two convex piecewise-linear functions
Recursion step 2

Transform $g_{k-1}$ to $f_k(z) = g_{k-1}(z) + w_k \cdot |z - a_k|$

- Add two convex piecewise-linear functions

Lemma.

- $f_k$ is a piecewise-linear convex function.
- The breakpoints are located at a subset of the points $a_i$.
- The leftmost piece has slope $-\sum_{i=1}^k w_i$.
- The rightmost piece has slope $w_k$. 

Piecewise-linear functions

\[ y = f(x) \]

\[ y = s'x + t' \]

\[ y = s''x + t'' \]

\[ y = sx + t \]

\( f \) has a breakpoint at position \( x_0 \) with value \( s'' - s' \).

Represent \( f \) by the rightmost slope \( s \) and the set of breakpoints (position+value). (The rightmost intercept \( t \) is not needed.)
Piecewise-linear functions

\[ y = f(x) \]

Transformation \( f_{k-1} \rightarrow g_{k-1} \):

while \( s - (\text{value of rightmost breakpoint}) \geq 0 \):
  remove rightmost breakpoint
  update \( s \)
update \( s \) to 0
Piecewise-linear functions

\[ y = f(x) \]

\[ y = s'x + t' \]
\[ y = s''x + t'' \]
\[ y = sx + t \]

Transformation

\[ g_{k-1} \rightarrow f_k(z) = g_{k-1}(z) + w_k \cdot |z - a_k| \]

- add \( w_k \) to \( s \).
- add a breakpoint at position \( a_k \) with value \( 2w_k \).
Piecewise-linear functions

\[ y = f(x) \]

Transformation \( g_{k-1} \rightarrow f_k(z) = g_{k-1}(z) + w_k \cdot |z - a_k| \)

- add \( w_k \) to \( s \).
- add a breakpoint at position \( a_k \) with value \( 2w_k \).
Piecewise-linear functions

\[ y = f(x) \]

Transformation \( f_{k-1} \rightarrow g_{k-1} \): 

while \( s - (\text{value of rightmost breakpoint}) \geq 0 \): 

remove rightmost breakpoint

update \( s \)

update \( s \) to 0
Piecewise-linear functions

\[ y = f(x) \]

Transformation \( f_{k-1} \rightarrow g_{k-1} \):

while \( s - \text{(value of rightmost breakpoint)} \geq 0 \):

- remove rightmost breakpoint
- update \( s \)

update \( s \) to 0
Piecewise-linear functions

\[ y = f(x) \]

Transformation \( f_{k-1} \rightarrow g_{k-1} \):

while \( s - (\text{value of rightmost breakpoint}) \geq 0 \):
  remove rightmost breakpoint
  update \( s \)
  update \( s \) to 0

Only access to the rightmost breakpoint is required.

→ priority queue ordered by position.
The algorithm

\[ Q := \emptyset; \quad // \text{priority queue of breakpoints ordered by the key position} \]
\[ s := 0; \]
\[ \text{for } k = 1, \ldots, n \text{ do} \]
\[ Q.\text{add}(\text{new breakpoint with } \text{position} := a_k, \text{value} := 2w_k); \]
\[ s := s + w_k; \]
\[ \text{loop} \]
\[ B := Q.\text{findmax}; \quad // \text{rightmost breakpoint } B \]
\[ \text{if } s - B.\text{value} < 0 \text{ then exit loop}; \]
\[ ; \]
\[ s := s - B.\text{value}; \]
\[ Q.\text{deletemax}; \]
\[ p_k := B.\text{position}; \]
\[ B.\text{value} \]
\[ s := 0; \quad // \text{We have computed } g_k. \]
\[ \text{loop} \]
\[ x_n := p_n; \]
\[ \text{for } k = n - 1, n - 2, \ldots, 1 \text{ do } x_k := \min\{x_{k+1}, p_k\}; \]
General objective functions

\[ \text{Minimize } \sum_{i=1}^{n} h_i(x_i) \]

Each \( h_i \) is convex and piecewise “simple”:
- Summation of pieces in constant time
- Minimum on a sum of pieces in constant time

Examples:
- \( L_3 \) norm: \( h_i(x) = \begin{cases} (x - a_i)^3, & x \geq a_i \\ -(x - a_i)^3, & x \leq a_i \end{cases} \Rightarrow O(n \log n) \) time
- \( L_4 \) norm: \( h_i(x) = (x - a_i)^4 \Rightarrow O(n) \) time
  (no breakpoints! A stack suffices.)
- Linear + sinusoidal pieces: \( -w_i \cos(x - a_i) \)
General partial orders

Minimize \( \sum_{i=1}^{n} h_i(x_i) \) or \( \max_{1 \leq i \leq n} h_i(x_i) \)

subject to

\( x_i \leq x_j \) for \( i \prec j \)

for a given partial order \( \prec \).

PAV can be extended to tree-like partial orders.

weighted \( L_1 \)-regression for a DAG with \( m \) edges in \( O(nm + n^2 \log n) \) time.

[Angelov, Harb, Kannan, and Wang, SODA’2006]

... and many other results
General partial orders

Minimize \( \sum_{i=1}^{n} h_i(x_i) \) or \( \max_{1 \leq i \leq n} h_i(x_i) \)

subject to
\[ x_i \leq x_j \text{ for } i \prec j \]

for a given partial order \( \prec \).

PAV can be extended to tree-like partial orders.

weighted \( L_1 \)-regression for a DAG with \( m \) edges in \( O(nm + n^2 \log n) \) time.

[Angelov, Harb, Kannan, and Wang, SODA’2006]

... and many other results
Weighted $L_\infty$ regression

Minimize $\ z := \max_{1 \leq i \leq n} h_i(x_i) \ \text{subject to } x_i \leq x_j \ \text{for } i < j.$

$h_i(x) = w_i|x - a_i|$

or $h_i(x) = \text{any function which increases from a minimum into both directions}$
Weighted $L_\infty$ regression

Minimize \( z := \max_{1 \leq i \leq n} h_i(x_i) \) subject to \( x_i \leq x_j \) for \( i < j \).

\[ h_i(x) = w_i |x - a_i| \]

or \( h_i(x) = \) any function which increases from a minimum into both directions

\[ h_i(x) \leq \varepsilon \iff \ell_i(\varepsilon) \leq x \leq u_i(\varepsilon) \]
Checking feasibility

Is the optimum value \( z^* \leq \varepsilon ? \) \[ z = \max_i h_i(x_i) \]

Find values \( x_i \) with
\[
\ell_i(\varepsilon) \leq x_i \leq u_i(\varepsilon) \quad \text{for all } i, \quad \text{and} \]
\[
x_i \leq x_j \quad \text{for } i < j. \tag{1}
\]

Find the smallest values \( x_i = x_{i}^\text{low} \) with
\[
\ell_i(\varepsilon) \leq x_i \quad \text{for all } i, \quad \text{and} \]
\[
x_i \leq x_j \quad \text{for } i < j. \tag{2}
\]

Result:
\[
x_j^\text{low} = \max\{\ell_j, \max\{\ell_i \mid i < j\}\}
\]

Calculate in topological order:
\[
x_j^\text{low} := \max\{\ell_i, \max\{x_i^\text{low} \mid i \text{ predecessor of } j\}\}
\]

\( z^* \leq \varepsilon \) iff \( x_i^\text{low} \leq u_i \) for all \( i \quad \implies \quad O(m + n) \) time
Characterizing feasibility

\[ z^* \leq \varepsilon \text{ iff for every pair } i, j \text{ with } i \prec j: \]

\[ \ell_i(\varepsilon) \leq u_j(\varepsilon) \]

**Theorem.**

Define for every pair \( i, j \):

\[ z_{ij} := \min\{ \varepsilon \mid \ell_i(\varepsilon) \leq u_j(\varepsilon) \} \]

Then \( z^* = \max\{ z_{ij} \mid i \prec j \} \).
$d$-dimensional orders

$n$ points $p = (p_1, p_2, \ldots, p_d) \in \mathbb{R}^d$

Product order (domination order):

$$p \prec q \iff (p_1 \leq q_1) \land (p_2 \leq q_2) \land \cdots \land (p_d \leq q_d)$$
**d-dimensional orders**

$n$ points $p = (p_1, p_2, \ldots, p_d) \in \mathbb{R}^d$

Product order (domination order):

$p < q \iff (p_1 \leq q_1) \land (p_2 \leq q_2) \land \cdots \land (p_d \leq q_d)$

The digraph of the order is not explicitly given. It might have quadratic size.
$L_\infty$ regression for $d$-dimensional orders

Stout [2011]: $O(n \log^{d-1} n)$ space and $O(n \log^d n)$ time

Embed the order in a DAG with more vertices but fewer edges. divide-and-conquer strategy, recursive in the dimension.
$L_\infty$ regression for $d$-dimensional orders

Stout [2011]: $O(n \log^{d-1} n)$ space and $O(n \log^d n)$ time

Embed the order in a DAG with more vertices but fewer edges. divide-and-conquer strategy, recursive in the dimension.
\( L_\infty \) regression for \( d \)-dimensional orders

Stout [2011]: \( O(n \log^{d-1} n) \) space and \( O(n \log^d n) \) time

Embed the order in a DAG with more vertices but fewer edges. A divide-and-conquer strategy, recursive in the dimension.
$L_\infty$ regression for $d$-dimensional orders

Stout [2011]: $O(n \log^{d-1} n)$ space and $O(n \log^d n)$ time.

Embed the order in a DAG with more vertices but fewer edges.

divide-and-conquer strategy, recursive in the dimension.
$L_\infty$ regression for $d$-dimensional orders

Stout [2011]: $O(n \log^{d-1} n)$ space and $O(n \log^d n)$ time

Embed the order in a DAG with more vertices but fewer edges. divide-and-conquer strategy, recursive in the dimension.
L∞ regression for d-dimensional orders

Stout [2011]: $O(n \log^{d-1} n)$ space and $O(n \log^d n)$ time

Embed the order in a DAG with more vertices but fewer edges. 
divide-and-conquer strategy, recursive in the dimension.

Recurse in each half.
→ $O(n \log n)$ in 2 dimensions.

small Manhattan networks
[ Gudmundsson, Klein, Knauer, and Smid 2007 ]

$O(n \log n)$ is optimal in 2 dimensions: Horton sets
$L_\infty$ regression for $d$-dimensional orders

Stout [2011]: $O(n \log^{d-1} n)$ space and $O(n \log^d n)$ time

Embed the order in a DAG with more vertices but fewer edges. divide-and-conquer strategy, recursive in the dimension.

Rote [2013]: $O(n)$ space and $O(n \log^{d-1} n)$ expected time

Recurse in each half.
$\rightarrow O(n \log n)$ in 2 dimensions.

small Manhattan networks [Gudmundsson, Klein, Knauer, and Smid 2007]

$O(n \log n)$ is optimal in 2 dimensions: Horton sets
\( L_\infty \) regression for \( d \)-dimensional orders

Rote [2013]: \( O(n) \) space and \( O(n \log^{d-1} n) \) expected time

- feasibility checking without explicitly constructing the DAG
- randomized optimization by Timothy Chan’s technique (saves a log-factor.)

\[
x_j^{\text{low}} = \max\{\ell_j, \max\{\ell_i \mid p_i \prec p_j\}\}
\]

More general operation \( \text{update}(A, B) : (A, B \subseteq \{1, \ldots, n\}) \)

for all \( j \in B \): \( x_j^{\text{new}} = \max\{x_j^{\text{old}}, \max\{x_i^{\text{old}} \mid i \in A, p_i \prec p_j\}\}\)

equivalent formulation:

for all \( i \in A, j \in B \) (in any order):

\[
\text{if } p_i \prec p_j \text{ then set } x_j := \max\{x_j, x_i\}.
\]
Partitioning the *update* operation

**procedure** `update(A, B)`:

Split $A \cup B$ along the last coordinate;

- `update(A^-, B^-);`
- `update(A^-, B^+);`
- `update(A^+, B^-);`
- `update(A^+, B^+);`
Partitioning the *update* operation

**procedure** $update(A, B)$:
Split $A \cup B$ along the last coordinate;
$update(A^-, B^-)$;
$update(A^-, B^+)$;
$\underline{update(A^+, B^-)}$;
$update(A^+, B^+)$;
Partitioning the *update* operation

\[
A^-, A^+, B^-, B^+
\]

\[
1, 2, \ldots, d - 1
\]

```
procedure update(A, B):
    Split \( A \cup B \) along
    the last coordinate;
    update(A^-, B^-);
    update(A^-, B^+);
    update(A^+, B^-);
    update(A^+, B^+);
```

A \((d-1)\)-dimensional order!
Partitioning the *update* operation

**procedure update** \((A, B)\):

Split \(A \cup B\) along
the last coordinate;

- \(update(A^-, B^-);\)
- \(update(A^-, B^+);\)
- \(update(A^+, B^-);\)
- \(update(A^+, B^+);\)

a \((d-1)\)-dimensional order!
Partitioning the *update* operation

**procedure** update\((A, B)\):  
Split \(A \cup B\) along 
the last coordinate;  
\(update(A^-, B^-)\);  
\(update(A^-, B^+)\);  
\(\underline{update(A^+, B^-)}\);  
\(update(A^+, B^+)\);

a \((d-1)\)-dimensional order!

**procedure** update\(_k\)(\(A, B\)):  
Split \(A \cup B\) along 
the \(k\)-th coordinate;  
\(update_k(A^-, B^-)\);  
\(update_k(A^-, B^+)\);  
\(update_{k-1}(A^-, B^+)\);  
\(update_k(A^+, B^+)\);
Partitioning the *update* operation (2)

**procedure** $update_k(A, B)$:
- Split $A \cup B$ into two equal parts along the $k$-th coordinate;
- $update_k(A^-, B^-)$;
- $update_{k-1}(A^-, B^+)$;
- $update_k(A^+, B^+)$;

Initially sort along all coordinates in $O(n \log n)$ time.
→ Splitting takes linear time.

Initial call: $update_d(P, P)$ with $P = \{1, \ldots, n\}$

Base case: $update_1(A, B)$ is a linear scan. Takes $O(n)$ time.
Induction: $update_k(A, B)$ in $O(n \log^{k-1} n)$ time. ($n = |A \cup B|$)

Remark: $update_d(A, B)$ is always called with $A = B$.
$update_k(A, B)$ for $k < d$ is always called with $A \cap B = \emptyset$. 

<table>
<thead>
<tr>
<th>$k$</th>
<th>$update_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$update_1(A, B)$</td>
</tr>
<tr>
<td>$k$</td>
<td>$update_k(A, B)$</td>
</tr>
<tr>
<td>$d$</td>
<td>$update_d(A, B)$</td>
</tr>
</tbody>
</table>
Randomized optimization technique

- The problem is decomposable: \( z^* = \max \{ z_{ij} \mid i < j \} \)

Define \( z(P) := \max \{ z_{ij} \mid i, j \in P, i < j \} \)

If \( P = P_1 \cup P_2 \cup P_3 \), then

\[
z(P) = \max \{ z(P_1 \cup P_2), z(P_1 \cup P_3), z(P_2 \cup P_3) \}
\]

(Similar problems: Diameter, closest pair)

- We can check feasibility: Is \( z(P) \leq \varepsilon \)?

**Lemma.** (Chan 1998)

\[\implies \text{The solution can be computed in the same expected time as checking feasibility.}\]
Randomized maximization

Permute subproblems $S_1, S_2, \ldots, S_r$ into random order.

$z^* := -\infty$;

for $k = 1, \ldots, r$ do

if $z(S_k) > z^*$ then (*)

$z^* := z(S_k)$; (**)

Proposition.

The test (*) is executed $r$ times, and the computation (**) is executed in expectation at most

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{r} = H_r \leq 1 + \ln r$$

times.
$L_\infty$ regression in $d$ dimensions

Partition $P$ into 10 equal subsets $P_1 \cup \cdots \cup P_{10}$ (for example along the $d$-axis).

Form 45 subproblems $(P_i, P_j)$ for $1 \leq i \leq j \leq 10$ and permute them into random order.

$z^* := -\infty$;

for $k = 1, \ldots, 45$ do

Let $(P_i, P_j)$ the $k$-th subproblem.

Feasibility check: Is $z(P_i \cup P_j) \leq z^*$? \hspace{1cm} (*)

if not then compute $z(P_i \cup P_j)$ recursively and set $z^* := z(P_i \cup P_j)$; \hspace{1cm} (**)

$T(n) = O(\text{FEAS}(n)) + H_{45} \cdot T(2n/10) \leq O(n \log^{d-1} n) + 4.395 \cdot T(n/5) = O(n \log^{d-1} n)$