Linear programming with limited workspace

Sándor Kisfaludi-Bak

Geometric algorithms with limited resources
Summer semester 2021

max planck institut
informatik

Overview

- Sorting with few passes
- A classic deterministic algorithm in $\mathbb{R}^2$
- Sublinear space LP (Chan–Chen '07)

Low-dim linear programming

Sorting in sublinear space

Theorem (Munro, Paterson 1980)
Given x and an unsorted array A, we can find the s smallest elements greater than x in A in a single pass, in $O(s)$ space and $O(n)$ time.
We can also sort in

- $O(n^2/s + n \log s)$ time
- $O(s)$ space
- with n/s passes.

Theorem (Munro, Paterson 1980)
A p-pass sorting algorithm needs $\Omega(n/p)$ space.

Linear Programming in low-dimensional space

Known LP algorithms

A: $n \times d$ matrix. (d variables, n cosntraints.)
$\max cx$ subject to $Ax \leq b$.

- Fourier-Motzkin elimination slow

Known LP algorithms

A: $n \times d$ matrix. (d variables, n cosntraints.)
$\max cx$ subject to $Ax \leq b$.

- Fourier-Motzkin elimination — slow
- Simplex method — fast in practice, slow in worst-case

Known LP algorithms

A: $n \times d$ matrix. (d variables, n cosntraints.)
$\max cx$ subject to $Ax \leq b$.

- Fourier-Motzkin elimination — slow
- Simplex method — fast in practice, slow in worst-case
- Ellipsoid method — slow in practice, poly time in worst-case

Known LP algorithms

A: $n \times d$ matrix. (d variables, n cosntraints.)
$\max cx$ subject to $Ax \leq b$.

- Fourier-Motzkin elimination — slow
- Simplex method — fast in practice, slow in worst-case
- Ellipsoid method — slow in practice, poly time in worst-case
- Interior point mehtods — poly time and practical

These use bit complexity!

Known LP algorithms

A: $n \times d$ matrix. (d variables, n cosntraints.)
$\max cx$ subject to $Ax \leq b$.

- Fourier-Motzkin elimination — slow
- Simplex method — fast in practice, slow in worst-case
- Ellipsoid method — slow in practice, poly time in worst-case
- Interior point mehtods — poly time and practical

These use bit complexity!

Open: poly LP solver for number of arithmetic operations. (e.g. Real RAM)

Best known by Clarkson, Matousek, Sharir, Welzl, Gärtner, Kalai (1996)

$$O(d^2 n) + 2^{O(\sqrt{d \log d})}$$

LP with 2 variables: halfplanes in $\mathbb{R}^2$

Given:

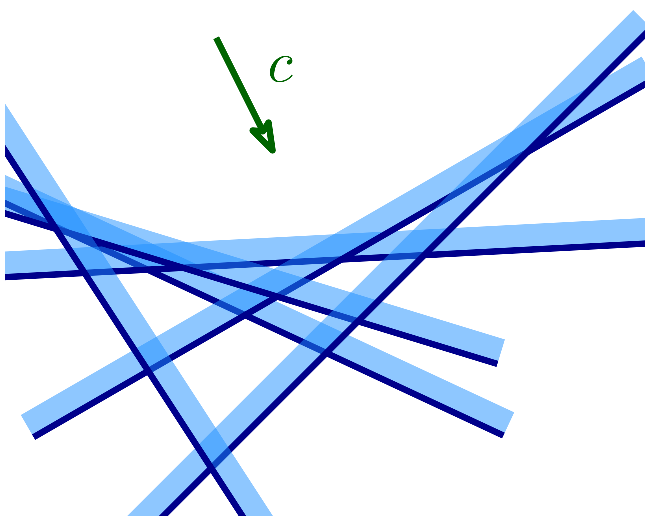
$$\max c_1 x + c_2 y \text{ subject to}$$
$$a_{11}x + a_{12}y \le b_1$$
$$a_{21}x + a_{22}y \le b_2$$
$$\cdots$$
$$a_{n1}x + a_{n2}y \le b_n$$

LP with 2 variables: halfplanes in $\mathbb{R}^2$

Given:

$$\max c_1 x + c_2 y \text{ subject to}$$
$$a_{11}x + a_{12}y \leq b_1$$
$$a_{21}x + a_{22}y \leq b_2$$
$$\dots$$
$$a_{n1}x + a_{n2}y \leq b_n$$

Given set H of n halfplanes, find extreme point in direction c.

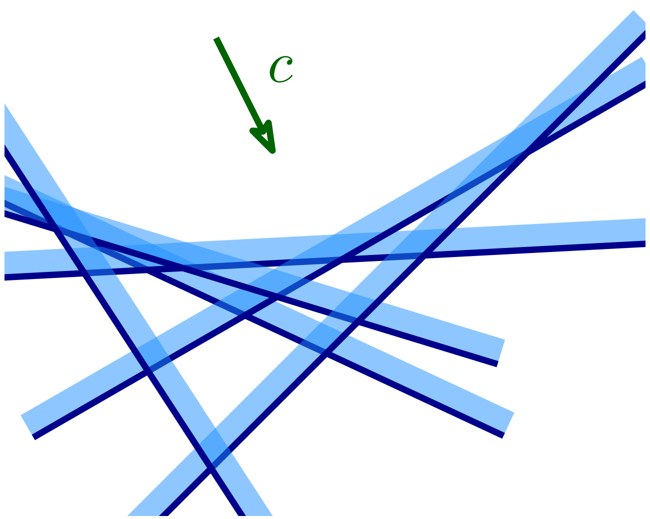


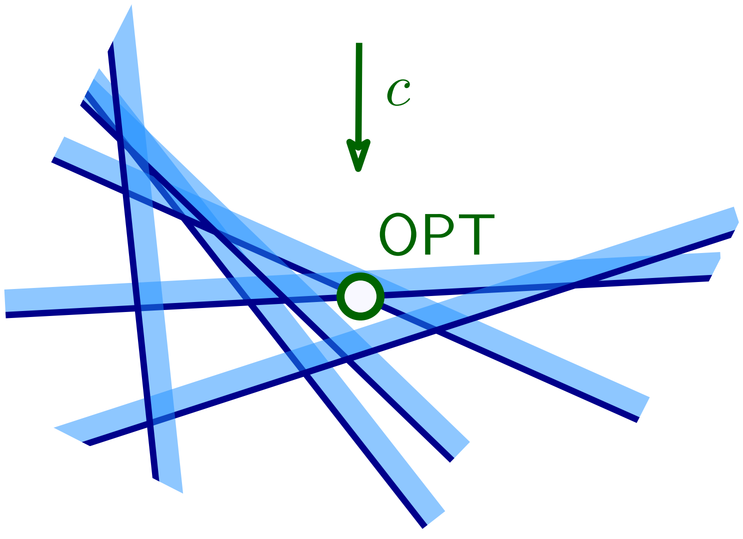
LP with 2 variables: halfplanes in $\mathbb{R}^2$

Given:

$$\max c_1 x + c_2 y \text{ subject to}$$
$$a_{11}x + a_{12}y \leq b_1$$
$$a_{21}x + a_{22}y \leq b_2$$
$$\dots$$
$$a_{n1}x + a_{n2}y \leq b_n$$

Given set H of n halfplanes, find extreme point in direction c.





Dual Graham's scan solves it in $O(n \log n)$

Intro/reminder on geometrc duality

Deterministic method: paired halfplanes

Lemma [Megiddo, Dyer 1984]
Assuming that $\bigcap_{h \in H} h \neq \emptyset$ is bounded from below, we can find OPT in $O(n)$ time.

Sublinear space low-dimensional LP

We prove:

Theorem (Chan–Chen 2007)
Fix $\delta > 0$. Given n half-planes in $\mathbb{R}^2$, the lowest point of their intersection can be computed in

- $O(\frac{1}{\delta}n^{1+\delta})$ time
- $O(\frac{1}{\delta}n^{\delta})$ space
- with $O(1/\delta)$ passes.

Sublinear space low-dimensional LP

We prove:

Theorem (Chan–Chen 2007)
Fix $\delta > 0$. Given n half-planes in $\mathbb{R}^2$, the lowest point of their intersection can be computed in

- $O(\frac{1}{\delta} n^{1+\delta})$ time
- $O(\frac{1}{\delta} n^{\delta})$ space
- with $O(1/\delta)$ passes.

Theorem (Chan–Chen 2007)
Given n half-spaces in $\mathbb{R}^d$ and $\delta > 0$, the lowest point of their intersection can be computed in

- $O_d(\frac{1}{\delta^{O(1)}} n)$ time
- $O_d(\frac{1}{\delta^{O(1)}} n^{\delta})$ space
- with $O(1/\delta^{d-1})$ passes.

Theorem (Chan–Chen 2007)
Given n half-spaces in $\mathbb{R}^d$, the lowest point of their intersection can be computed in $O_d(n)$ time and $O_d(\log n)$ space.

Towards sublinear space LP: filtering and listing

Given stream H of halfplanes, produce stream of vertical lines.

List(r, σ, H)
    while H not read through **do**
        $h_1, \ldots, h_r :=$ next r halfplanes from H
        Compute $I = h_1 \cap \cdots \cap h_r$
        Print vertical lines through vertices of I that fall in σ

Towards sublinear space LP: filtering and listing

Given stream H of halfplanes, produce stream of vertical lines.

List(r, σ, H)
while H not read through **do**
$h_1, \ldots, h_r :=$ next r halfplanes from H
Compute $I = h_1 \cap \cdots \cap h_r$
Print vertical lines through vertices of I that fall in σ

Given stream H of halfplanes, produce stream of halfplanes.

Filter(r, σ, H)
while H not read through **do**
$h_1, \ldots, h_r :=$ next r halfplanes from H
Compute $I = h_1 \cap \cdots \cap h_r$
Print halfplanes involved in $\partial(I \cap \sigma)$

Towards sublinear space LP: filtering and listing

Given stream H of halfplanes, produce stream of vertical lines.

List(r, σ, H)
while H not read through **do**
$h_1, \ldots, h_r :=$ next r halfplanes from H
Compute $I = h_1 \cap \cdots \cap h_r$
Print vertical lines through vertices of I that fall in σ

Given stream H of halfplanes, produce stream of halfplanes.

Filter(r, σ, H)
while H not read through **do**
$h_1, \ldots, h_r :=$ next r halfplanes from H
Compute $I = h_1 \cap \cdots \cap h_r$
Print halfplanes involved in $\partial(I \cap \sigma)$

List and Filter work in one pass, in $O(r)$ space and $O(n \log r)$ time.

Sublinear time LP in $\mathbb{R}^2$

Parameter: r

Invariant: solution is in σ_i and defined by halfplanes in H_i

Pseudocode

LP(r, σ, H)

    $\sigma_0 := \mathbb{R}^2$

    for $i = 0, 1, \ldots$ **do**

        if $|H_i| = O(1)$ **then**

            return brute force solution for H_i

        Divide σ_i into r slabs with roughly same # of lines from $List_{r,\sigma_i}(H_i)$

        Decide which subslab has the solution, let that be σ_{i+1}.

        $H_{i+1} := Filter_{r,\sigma_{i+1}}(H_i)$

Preserves invariant ✓

Pseudocode

LP(r, σ, H)

  $\sigma_0 := \mathbb{R}^2$

  for $i = 0, 1, \ldots$ **do**

    if $|H_i| = O(1)$ **then**

      return brute force solution for H_i

    Divide σ_i into r slabs with roughly same # of lines from $List_{r,\sigma_i}(H_i)$

    Decide which subslab has the solution, let that be σ_{i+1}.

    $H_{i+1} := Filter_{r,\sigma_{i+1}}(H_i)$

Preserves invariant ✓

Issue: H_i can't be stored. We need to recompute it every time.

Pseudocode

LP(r, σ, H)
  $\sigma_0 := \mathbb{R}^2$
  for $i = 0, 1, \ldots$ **do**
    if $|H_i| = O(1)$ **then**
      return brute force solution for H_i
    Divide σ_i into r slabs with roughly same # of lines from $List_{r,\sigma_i}(H_i)$
    Decide which subslab has the solution, let that be σ_{i+1}.
    $H_{i+1} := Filter_{r,\sigma_{i+1}}(H_i)$

Preserves invariant ✓

Issue: H_i can't be stored. We need to recompute it every time.

LP(r, σ, H)
  $\sigma_0 := \mathbb{R}^2$
  for $i = 0, 1, \ldots$ **do**
    if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**
      return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$
    Divide σ_i into r slabs with roughly same # of lines from
        $List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H))))$
    Decide which subslab has the solution, let that be σ_{i+1}

Pseudocode

LP(r, σ, H)
  $\sigma_0 := \mathbb{R}^2$
  for $i = 0, 1, \ldots$ **do**
    if $|H_i| = O(1)$ **then**
      return brute force solution for H_i
    Divide σ_i into r slabs with roughly same # of lines from $List_{r,\sigma_i}(H_i)$
    Decide which subslab has the solution, let that be σ_{i+1}.
    $H_{i+1} := Filter_{r,\sigma_{i+1}}(H_i)$

Preserves invariant ✓

Issue: H_i can't be stored. We need to recompute it every time.

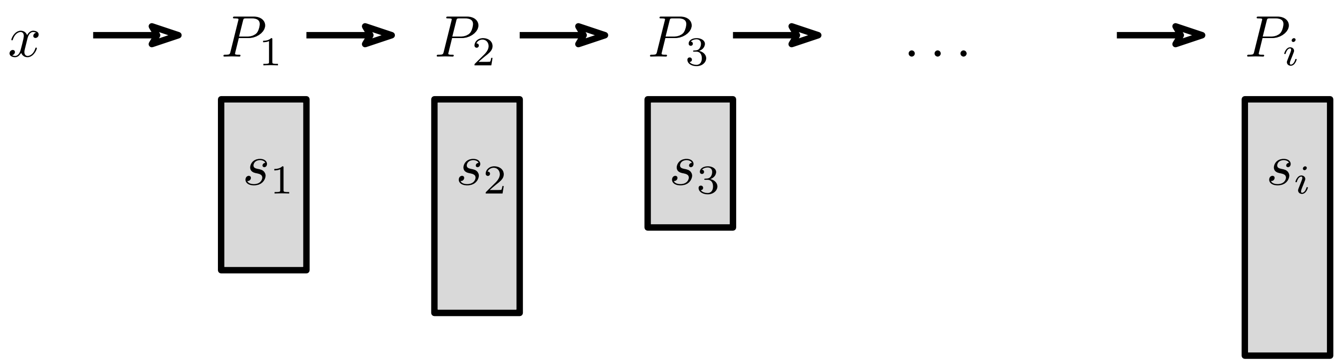
LP(r, σ, H)
  $\sigma_0 := \mathbb{R}^2$
  for $i = 0, 1, \ldots$ **do**
    if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**
      return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$
    Divide σ_i into r slabs with roughly same # of lines from
      $List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H))))$
    Decide which subslab has the solution, let that be σ_{i+1}

One pass, maintain $r-1$ minima at inner slab walls

Space-efficient pipeline of streams

How to execute $P_i(P_{i-1}(\ldots(P_1(x))))$
if P_j are single-pass processes with worksapce s_j and time t_j?

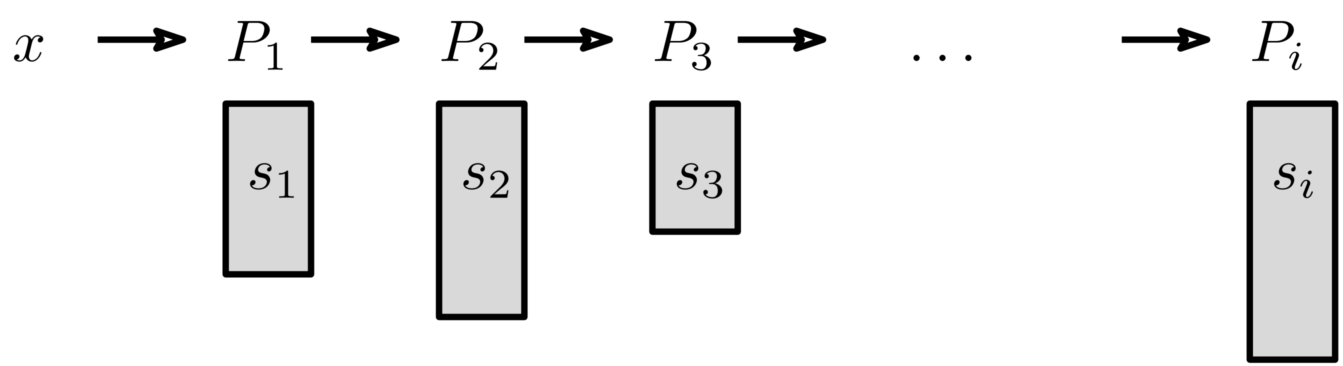
Pipeline:



Space-efficient pipeline of streams

How to execute $P_i(P_{i-1}(\ldots(P_1(x))))$
if P_j are single-pass processes with worksapce s_j and time t_j?

Pipeline:

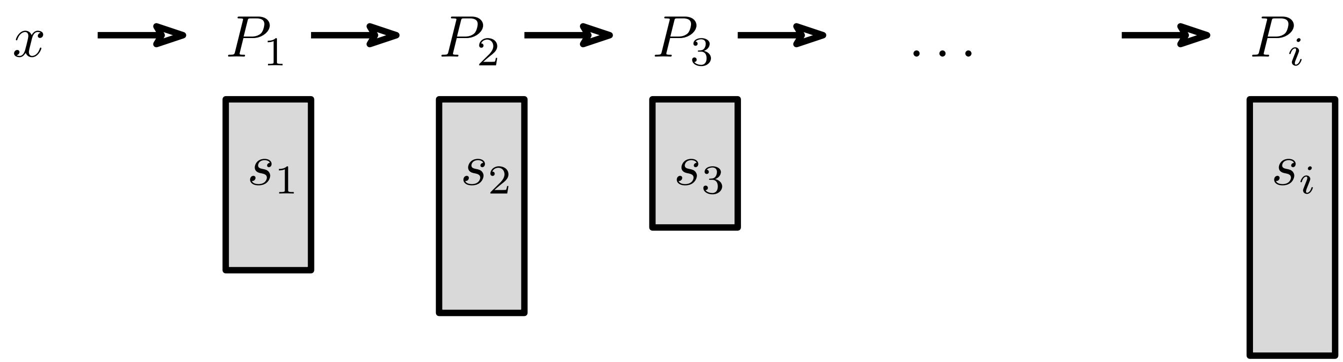


Each P_j is either waiting for input, or ready to excute.
Init: all waiting for input

Space-efficient pipeline of streams

How to execute $P_i(P_{i-1}(\ldots(P_1(x))))$
if P_j are single-pass processes with worksapce s_j and time t_j?

Pipeline:



Each P_j is either waiting for input, or ready to excute.
Init: all waiting for input

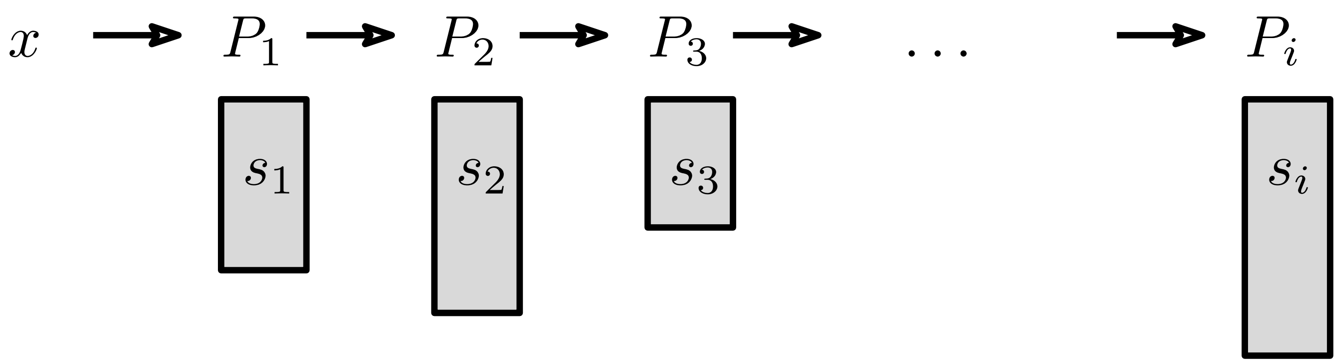
Simulation:

- if all P_j are waiting for input, execute P_1
- otherwise, pick largest j ready to execute, and execute one step.

Space-efficient pipeline of streams

How to execute $P_i(P_{i-1}(\ldots(P_1(x))))$
if P_j are single-pass processes with worksapce s_j and time t_j?

Pipeline:



Each P_j is either waiting for input, or ready to excute.
Init: all waiting for input

Simulation:

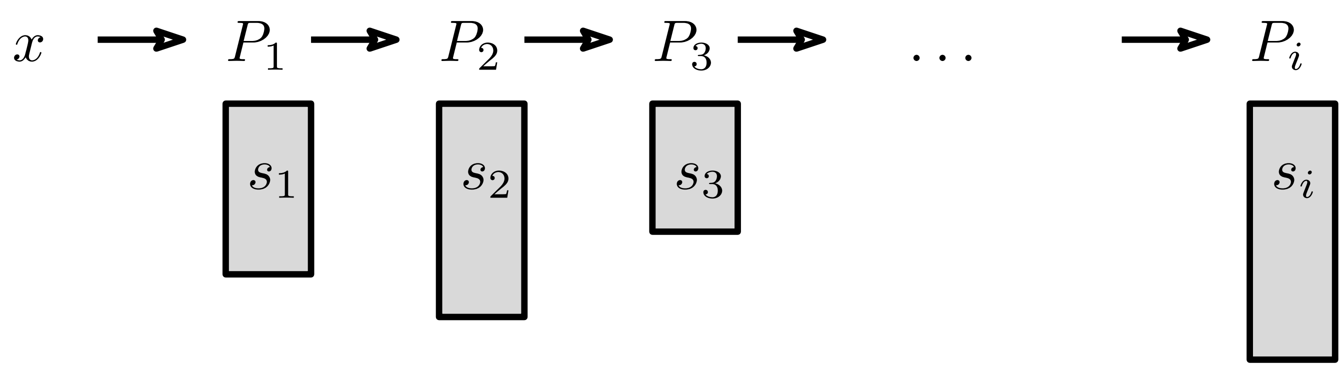
- if all P_j are waiting for input, execute P_1
- otherwise, pick largest j ready to execute, and execute one step.

Space: $\sum_j s_j + O(1)$

Space-efficient pipeline of streams

How to execute $P_i(P_{i-1}(\ldots(P_1(x))))$
if P_j are single-pass processes with worksapce s_j and time t_j?

Pipeline:



Each P_j is either waiting for input, or ready to excute.
Init: all waiting for input

Simulation:

- if all P_j are waiting for input, execute P_1
- otherwise, pick largest j ready to execute, and execute one step.

Space: $\sum_j s_j + O(1)$ Time (mini-hw): $O(\sum_j t_j)$

Time and space needs

Filter(r, σ, H)

$\sigma_0 := \mathbb{R}^2$

for $i = 0, 1, \ldots$ **do**

if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**

return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$

Divide σ_i into r slabs:

$\mathbf{ApproxQuant_r}\left(List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H))))\right)$

Decide which subslab has the solution, let that be σ_{i+1}

Time and space needs

Filter(r, σ, H)

$\sigma_0 := \mathbb{R}^2$

for $i = 0, 1, \ldots$ **do**

if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**

return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$

Divide σ_i into r slabs:

$\mathbf{ApproxQuant_r}\left(List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H))))\right)$

Decide which subslab has the solution, let that be σ_{i+1}

Let $n_i = |H_i|$. There are $\log_r(n)$ iterations, $O(\log_r n)$ passes.

Time and space needs

Filter(r, σ, H)

$\sigma_0 := \mathbb{R}^2$

for $i = 0, 1, \ldots$ **do**

if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**

return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$

Divide σ_i into r slabs:

$\mathbf{ApproxQuant_r}(List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))))$

Decide which subslab has the solution, let that be σ_{i+1}

Let $n_i = |H_i|$. There are $\log_r(n)$ iterations, $O(\log_r n)$ passes.

Filter pipeline (plus List) in iteration i needs:

$O(ri) = O(r \log_r n)$ space, $O(n_0 \log r + \cdots + n_{i-1} \log r) = O(n \log r)$ time

Time and space needs

Filter(r, σ, H)

$\sigma_0 := \mathbb{R}^2$

for $i = 0, 1, \ldots$ **do**

if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**

return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$

Divide σ_i into r slabs:

$\mathbf{ApproxQuant_r}\left(List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H))))\right)$

Decide which subslab has the solution, let that be σ_{i+1}

Let $n_i = |H_i|$. There are $\log_r(n)$ iterations, $O(\log_r n)$ passes.

Filter pipeline (plus List) in iteration i needs:
$O(ri) = O(r \log_r n)$ space, $O(n_0 \log r + \cdots + n_{i-1} \log r) = O(n \log r)$ time

ApproxQuant needs $O(r \log^2 n_i)$ space and $O(n_i \log(r \log n_i))$ time. see later!

Time and space needs

Filter(r, σ, H)

$\sigma_0 := \mathbb{R}^2$

for $i = 0, 1, \ldots$ **do**

if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**

return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$

Divide σ_i into r slabs:

$\mathbf{ApproxQuant_r}(List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))))$

Decide which subslab has the solution, let that be σ_{i+1}

Let $n_i = |H_i|$. There are $\log_r(n)$ iterations, $O(\log_r n)$ passes.

Filter pipeline (plus List) in iteration i needs:
$O(ri) = O(r \log_r n)$ space, $O(n_0 \log r + \cdots + n_{i-1} \log r) = O(n \log r)$ time

ApproxQuant needs $O(r \log^2 n_i)$ space and $O(n_i \log(r \log n_i))$ time. see later!

Subslab selection needs $O(r)$ space and $O(nr)$ time.

Time and space needs

Filter(r, σ, H)
$\sigma_0 := \mathbb{R}^2$
for $i = 0, 1, \ldots$ **do**
if $|Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))| = O(1)$ **then**
return brute force solution for $Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))$
Divide σ_i into r slabs:
$\mathbf{ApproxQuant_r}(List_{r,\sigma_i}(Filter_{r,\sigma_i}(\ldots(Filter_{r,\sigma_1}(H)))))$
Decide which subslab has the solution, let that be σ_{i+1}

Let $n_i = |H_i|$. There are $\log_r(n)$ iterations, $O(\log_r n)$ passes.

Filter pipeline (plus List) in iteration i needs:
$O(ri) = O(r \log_r n)$ space, $O(n_0 \log r + \cdots + n_{i-1} \log r) = O(n \log r)$ time

ApproxQuant needs $O(r \log^2 n_i)$ space and $O(n_i \log(r \log n_i))$ time. see later!

Subslab selection needs $O(r)$ space and $O(nr)$ time.

Altogether: $O(r \log_r n + r \log^2 n)$ space and $O(nr \log_r n)$ time.

Approximate quantiles

Goal:
Given unsorted stream S of n numbers, output r entries $a_1 \leq \cdots \leq a_r$ so that the rank of a_i and a_{i+1} in the sorting of S differ by at most $O(n/r)$.

Approximate quantiles

Goal:
Given unsorted stream S of n numbers, output r entries $a_1 \leq \cdots \leq a_r$ so that the rank of a_i and a_{i+1} in the sorting of S differ by at most $O(n/r)$.

Fix $k > 0$ even, suppose n/k is power of 2.

$PartSort_k$: Repeatedly read next k elements, output them in sorted order.

Space: $O(k)$, Time: $O(\frac{n}{k} \cdot k \log k) = O(n \log k)$

Approximate quantiles

Goal:
Given unsorted stream S of n numbers, output r entries $a_1 \leq \cdots \leq a_r$ so that the rank of a_i and a_{i+1} in the sorting of S differ by at most $O(n/r)$.

Fix $k > 0$ even, suppose n/k is power of 2.

$PartSort_k$: Repeatedly read next k elements, output them in sorted order.

Space: $O(k)$, Time: $O(\frac{n}{k} \cdot k \log k) = O(n \log k)$

$SampleMerge_k$: Read in next $k + k$ elements $a_1, \ldots, a_k, b_1, \ldots, b_k$ (a and b are sorted). Merge the sequences $a_2, a_4, \ldots, a_k$ and $b_2, b_4, \ldots, b_k$. Output sorted merged sequence.

Space: $O(k)$, Time: $O(n)$

Approximate quantiles

Goal:
Given unsorted stream S of n numbers, output r entries $a_1 \leq \cdots \leq a_r$ so that the rank of a_i and a_{i+1} in the sorting of S differ by at most $O(n/r)$.

Fix $k > 0$ even, suppose n/k is power of 2.

$PartSort_k$: Repeatedly read next k elements, output them in sorted order.

Space: $O(k)$, Time: $O(\frac{n}{k} \cdot k \log k) = O(n \log k)$

$SampleMerge_k$: Read in next $k + k$ elements $a_1, \ldots, a_k, b_1, \ldots, b_k$ (a and b are sorted). Merge the sequences $a_2, a_4, \ldots, a_k$ and $b_2, b_4, \ldots, b_k$. Output sorted merged sequence.

Space: $O(k)$, Time: $O(n)$

Idea:
Do $PartSort_k$, then repeatedly run $SampleMerge_k$ to get sample of size k.

$$\underbrace{SampleMerge_k(SampleMerge_k(\ldots}_{\log(n/k)}((PartSort_k(S)))\ldots)$$

Approximate quantiles as a pipeline

Lemma (assignment)
Let $a_1, \ldots, a_k$ be the result of $Sort_k$ and $\log(n/k)$ runs of $SampleMerge_k$.
Then the rank of a_i and a_{i+1} differ by at most $O(\frac{n}{k} \log n)$ for all $i \in [k-1]$.

Approximate quantiles as a pipeline

Lemma (assignment)
Let $a_1, \ldots, a_k$ be the result of $Sort_k$ and $\log(n/k)$ runs of $SampleMerge_k$.
Then the rank of a_i and a_{i+1} differ by at most $O(\frac{n}{k} \log n)$ for all $i \in [k-1]$.

Set $k = r \log n$.
$PostSelect_r$:

Approximate quantiles as a pipeline

Lemma (assignment)
Let $a_1, \ldots, a_k$ be the result of $Sort_k$ and $\log(n/k)$ runs of $SampleMerge_k$.
Then the rank of a_i and a_{i+1} differ by at most $O(\frac{n}{k} \log n)$ for all $i \in [k-1]$.

Set $k = r \log n$.
$PostSelect_r$:

$ApproxQuant_r$:

$$PostSelect_r(\underbrace{SampleMerge_k(SampleMerge_k(\ldots}_{\log(n/k)}((PartSort_k(S)))\ldots)$$

Time: $O(n \log k + n + n/2 + n/4 + \cdots + k + n \log k) = O(n \log(r \log n))$

Space: $O(k + k + \cdots + k + k) = O(k \log(n/k)) = O(r \log^2 n)$

Chan-Chen simple LP wrap-up

Theorem (Chan–Chen 2007)
Fix $\delta > 0$. Given n half-planes in $\mathbb{R}^2$, the lowest point of their intersection can be computed in

- $O(\frac{1}{\delta} n^{1+\delta})$ time
- $O(\frac{1}{\delta} n^{\delta})$ space
- with $O(1/\delta)$ passes.

Altogether: $O(r \log_r n + r \log^2 n)$ space and $O(nr \log_r n)$ time.

Chan-Chen simple LP wrap-up

Theorem (Chan–Chen 2007)
Fix $\delta > 0$. Given n half-planes in $\mathbb{R}^2$, the lowest point of their intersection can be computed in

- $O(\frac{1}{\delta} n^{1+\delta})$ time
- $O(\frac{1}{\delta} n^{\delta})$ space
- with $O(1/\delta)$ passes.

Altogether: $O(r \log_r n + r \log^2 n)$ space and $O(nr \log_r n)$ time.

Set $r = n^{\delta/2}$.

$$O\left(n^{\delta/2} \cdot \tfrac{2}{\delta} + n^{\delta/2} \log^2 n\right) = O(\tfrac{1}{\delta} n^{\delta}) \qquad O\left(n \cdot n^{\delta/2} \cdot \frac{\log n}{\log n^{\delta/2}}\right) = O(\tfrac{1}{\delta} n^{1+\delta})$$