More Efficient Algorithms for Graph Orientations and Geometric Cover

PhD Defence

Edvin Berglin

Danmarks Grundforskningsfond
Danish National Research Foundation

MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

AARHUS UNIVERSITY

Topics of today

Topics of today

A Simple Greedy Algorithm for Dynamic Graph Orientation

E.B. and G.S. Brodal
International Symposium on Algorithms and Computation 2017

Topics of today

A Simple Greedy Algorithm for Dynamic Graph Orientation

E.B. and G.S. Brodal
International Symposium on Algorithms and Computation 2017

Applications of Incidence Bounds in Point Covering Problems

P. Afshani, E.B., I. van Duijn, J.S. Nielsen
Symposium on Computational Geometry 2016

A Simple Greedy Algorithm for Dynamic Graph Orientations

ISAAC'17, Phuket

joint work with Gerth Stølting Brodal

Danmarks Grundforskningsfond
Danish National Research Foundation

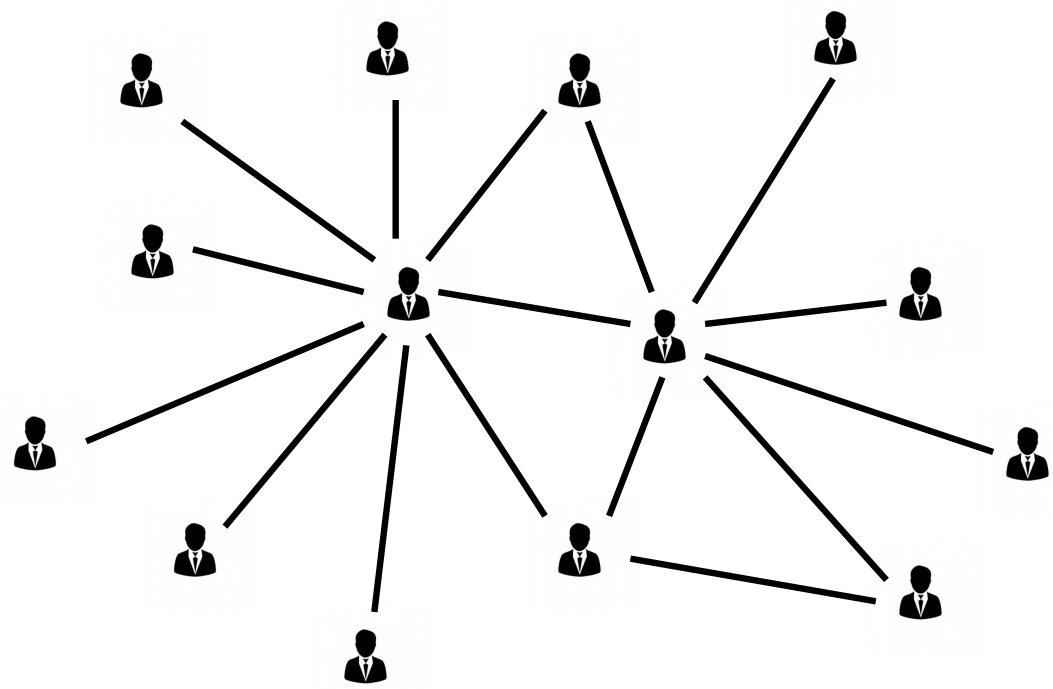
MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

AARHUS UNIVERSITY

How to keep track of your friends

All my friends:

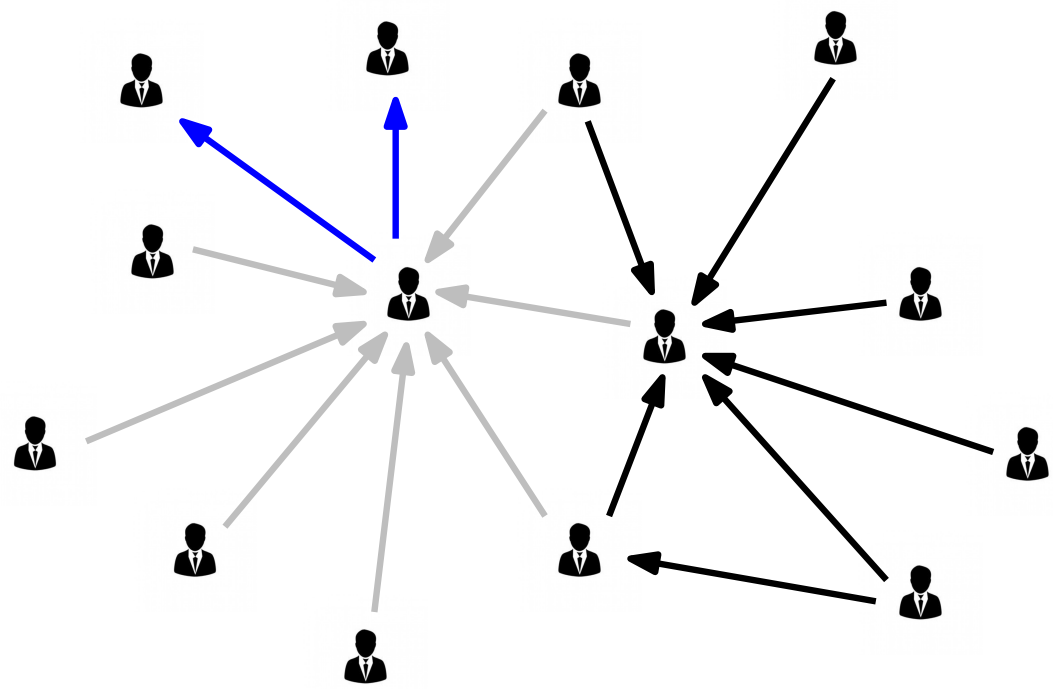
- Josh
- Peter
- Bob
- Mark
- Charles
- David
- Francisco
- Zeke
- Aaron



How to keep track of your friends

All my friends:

- Josh
- Peter
- Bob
- Mark
- Charles
- David
- Francisco
- Zeke
- Aaron

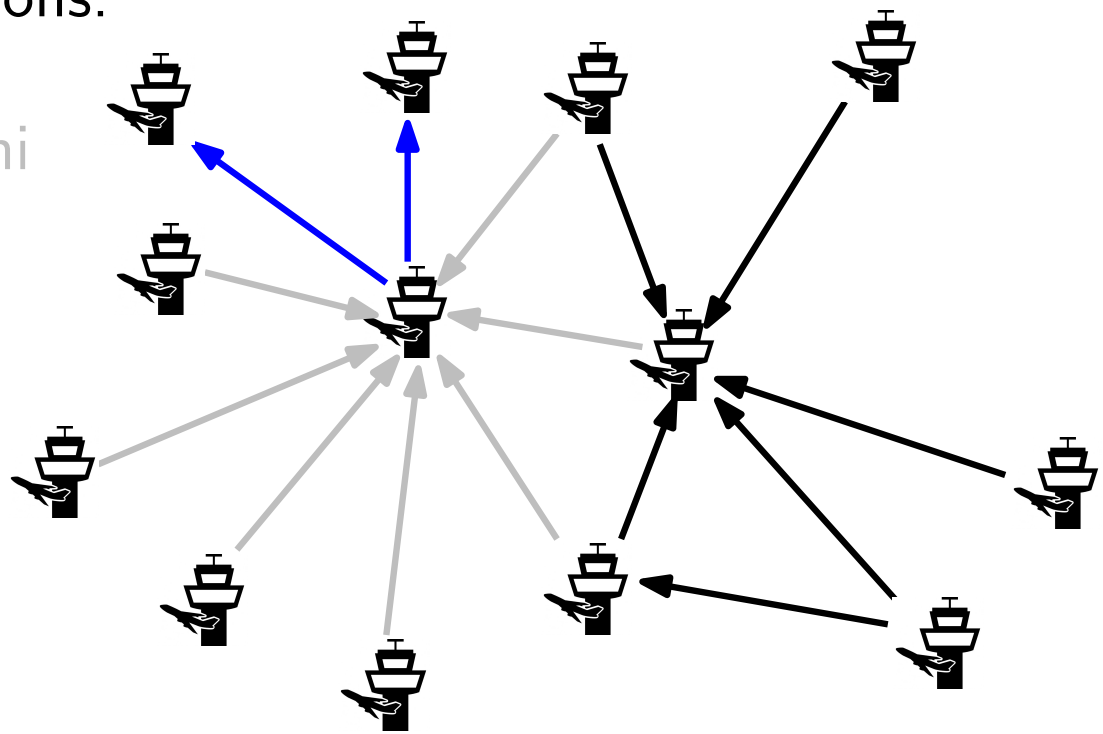


How to keep track of your friends

How to keep track of ~~your friends~~ flight connections

All my connections:

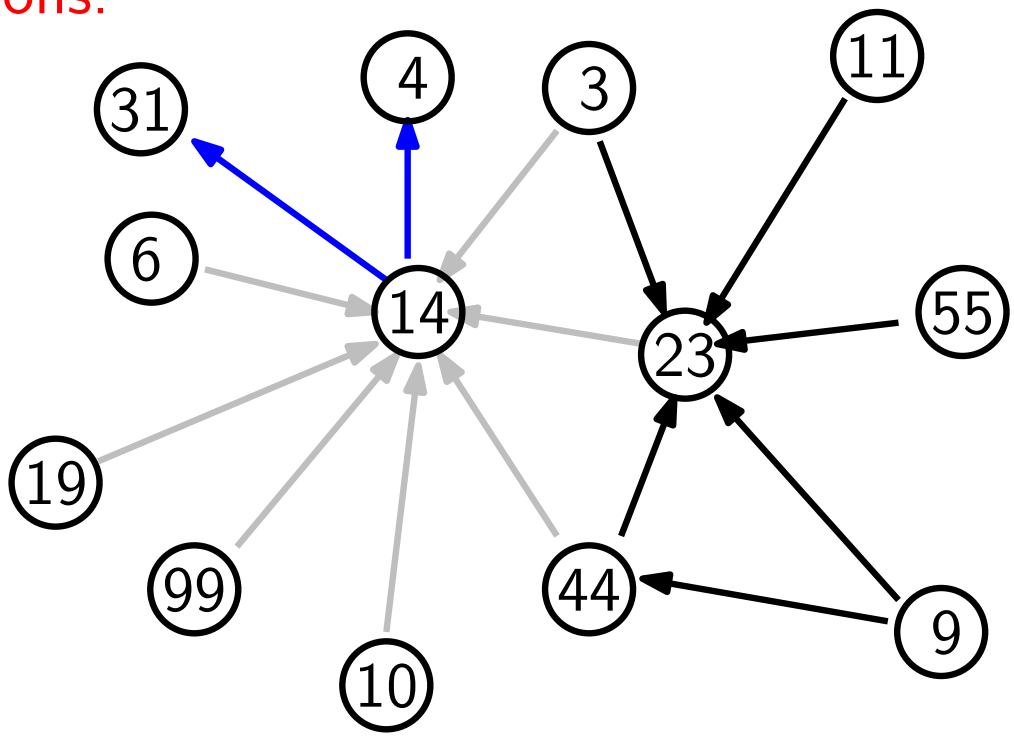
- Heathrow
- Indira Gandhi
- Kastrup
- McCarran
- JFK
- Schiphol
- Gardermoen
- Copernicus
- Keflavik



Graph orientations

All my connections:

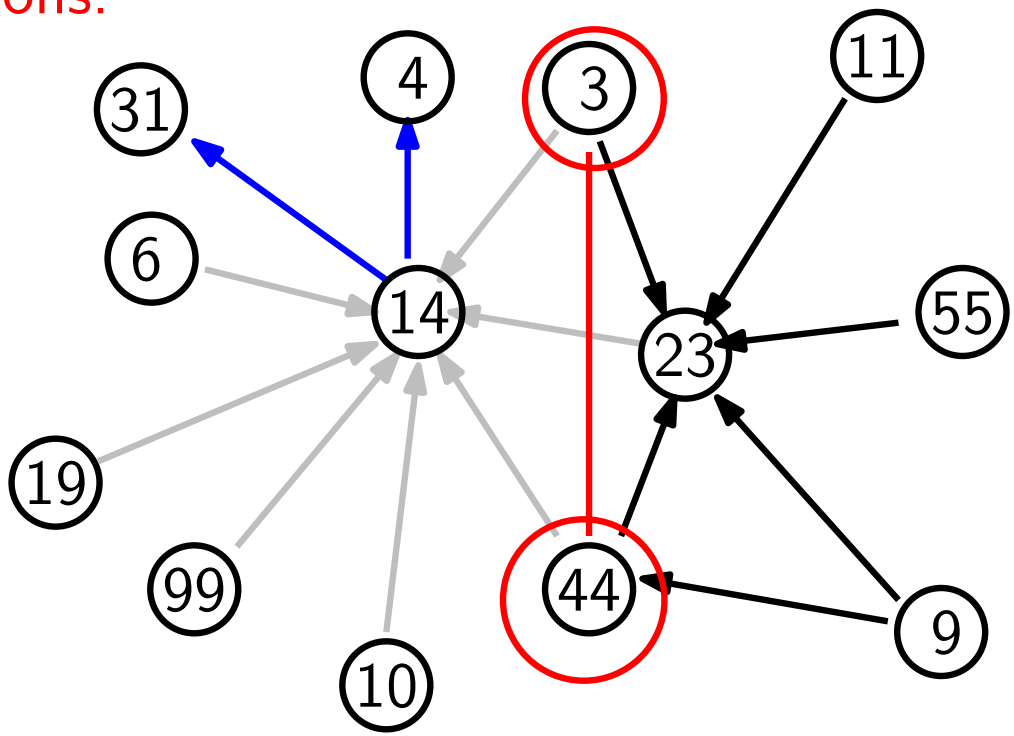
- 44
- 6
- 3
- 31
- 19
- 10
- 4
- 91
- 23



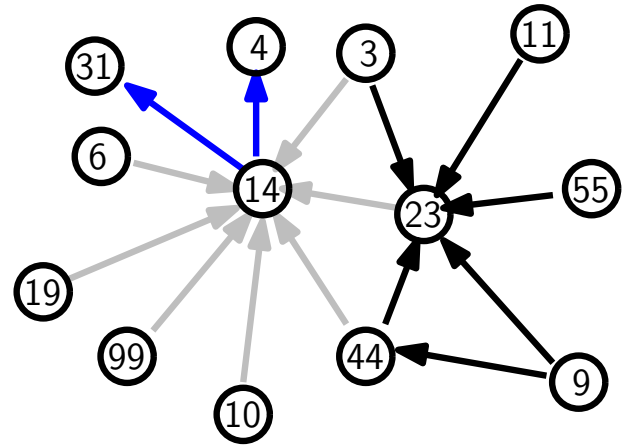
Graph orientations

All my connections:

- 44
- 6
- 3
- 31
- 19
- 10
- 4
- 91
- 23

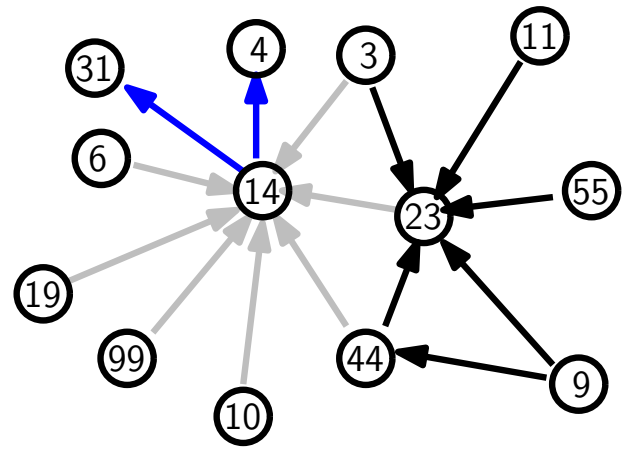


Definitions



Definitions

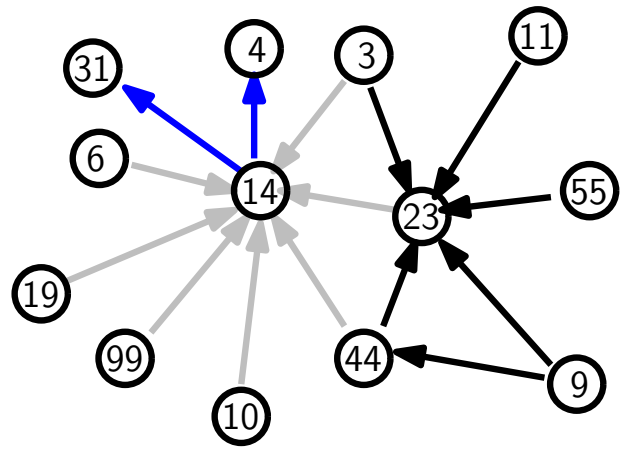
Graph with n *vertices* ○
and m *edges*. ／



Definitions

Graph with n *vertices* ○
and m *edges*. ／

Graph *orientation*: all edges are given a *direction*. ↗

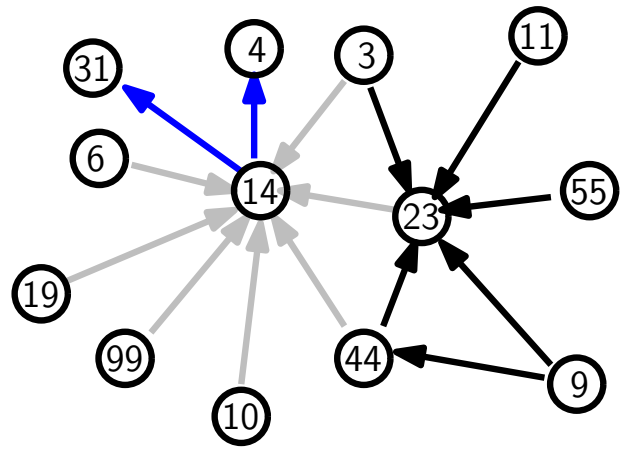


Definitions

Graph with n *vertices* and m *edges*.

Graph *orientation*: all edges are given a *direction*.

Out-degree of a vertex = number of *out-edges* from the vertex

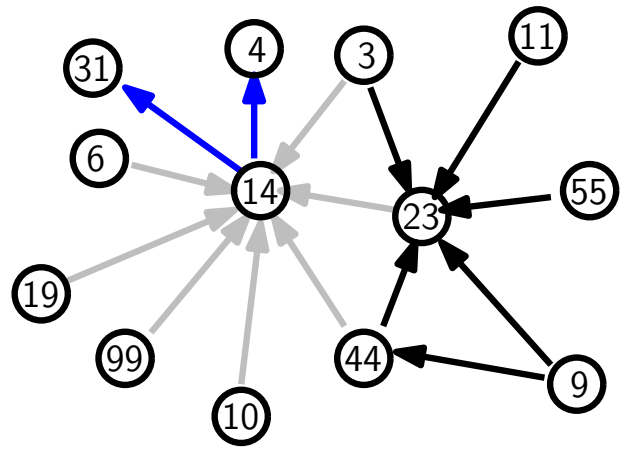


Definitions

Graph with n *vertices* and m *edges*.

Graph *orientation*: all edges are given a *direction*.

Out-degree of a vertex = number of *out-edges* from the vertex



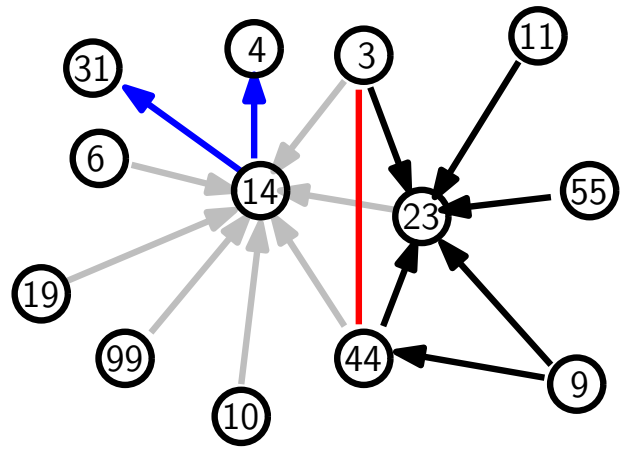
Situation: graph gets *updated* (edges inserted and deleted) in an *unpredictable* way.

Definitions

Graph with n *vertices* and m *edges*.

Graph *orientation*: all edges are given a *direction*.

Out-degree of a vertex = number of *out-edges* from the vertex



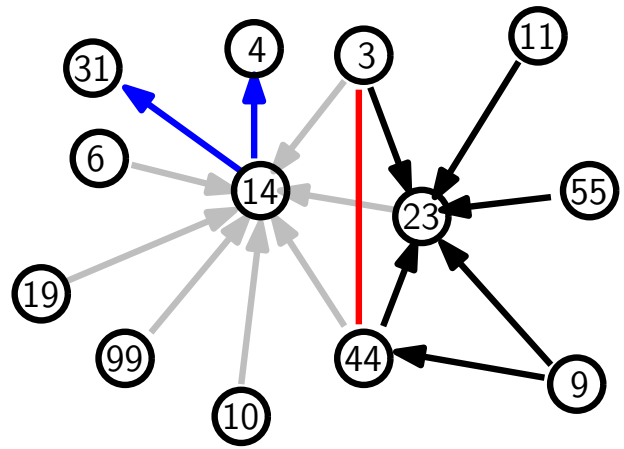
Situation: graph gets *updated* (edges inserted and deleted) in an *unpredictable* way.

Definitions

Graph with n *vertices* and m *edges*.

Graph *orientation*: all edges are given a *direction*.

Out-degree of a vertex = number of *out-edges* from the vertex



Situation: graph gets *updated* (edges inserted and deleted) in an *unpredictable* way.

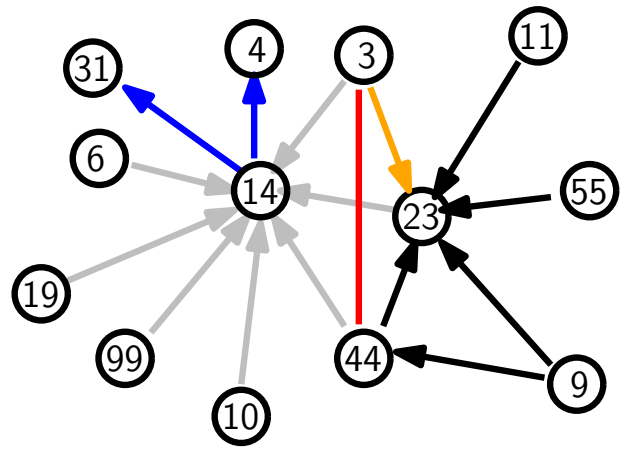
Task: *flip* a small number of edges to ensure all out-degrees remain low.

Definitions

Graph with n *vertices* and m *edges*.

Graph *orientation*: all edges are given a *direction*.

Out-degree of a vertex = number of *out-edges* from the vertex



Situation: graph gets *updated* (edges inserted and deleted) in an *unpredictable* way.

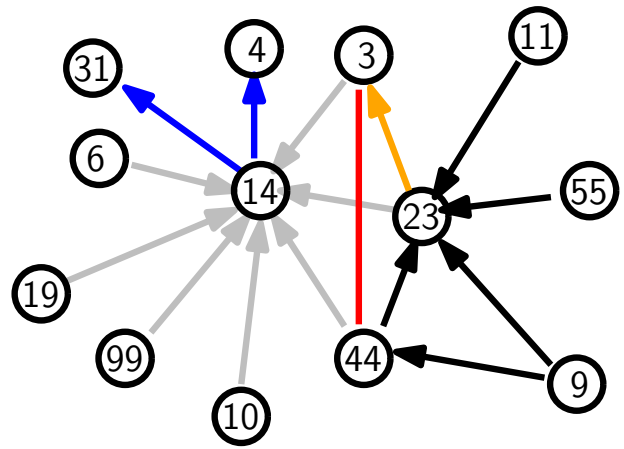
Task: *flip* a small number of edges to ensure all out-degrees remain low.

Definitions

Graph with n *vertices* and m *edges*.

Graph *orientation*: all edges are given a *direction*.

Out-degree of a vertex = number of *out-edges* from the vertex



Situation: graph gets *updated* (edges inserted and deleted) in an *unpredictable* way.

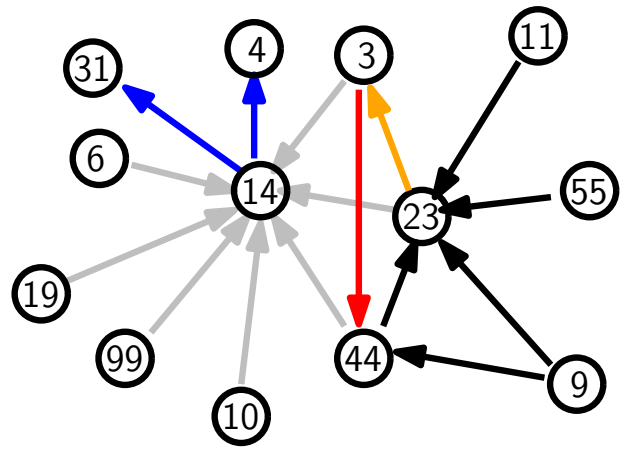
Task: *flip* a small number of edges to ensure all out-degrees remain low.

Definitions

Graph with n *vertices* and m *edges*.

Graph *orientation*: all edges are given a *direction*.

Out-degree of a vertex = number of *out-edges* from the vertex



Situation: graph gets *updated* (edges inserted and deleted) in an *unpredictable* way.

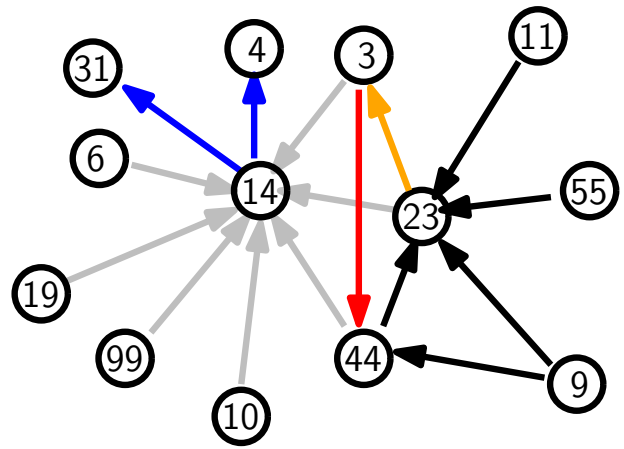
Task: *flip* a small number of edges to ensure all out-degrees remain low.

Definitions

Graph with n *vertices* and m *edges*.

Graph *orientation*: all edges are given a *direction*.

Out-degree of a vertex = number of *out-edges* from the vertex



Situation: graph gets *updated* (edges inserted and deleted) in an *unpredictable* way.

Task: *flip* a small number of edges to ensure all out-degrees remain low.

α = lowest possible out-degree (any number of flips)

Algorithm

Input: stream of updates, parameter k.

Algorithm

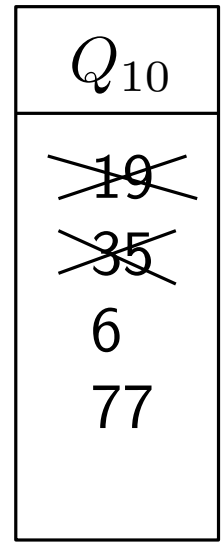
Input: stream of updates, parameter k.

Each vertex v stores its
out-edges in a queue Q_v.

Algorithm

Input: stream of updates, parameter k.

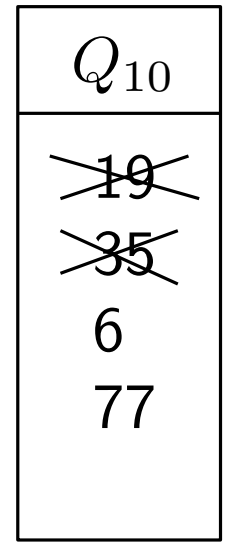
Each vertex v stores its out-edges in a queue Q_v.



Algorithm

Input: stream of updates, parameter k.

Each vertex v stores its out-edges in a queue Q_v.



On every insertion:

- add new edge in any direction
- repeat k times:
  - flip the first edge of the longest list

Algorithm

Input: stream of updates, parameter k.

Each vertex v stores its out-edges in a queue Q_v.

Q_{10}
~~19~~
~~35~~
6
77
45

On every insertion:

- add new edge in any direction
- repeat k times:
  - flip the first edge of the longest list

Algorithm

Input: stream of updates, parameter k.

Each vertex v stores its out-edges in a queue Q_v.

Q_{10}
~~19~~
~~35~~
~~6~~
77
45

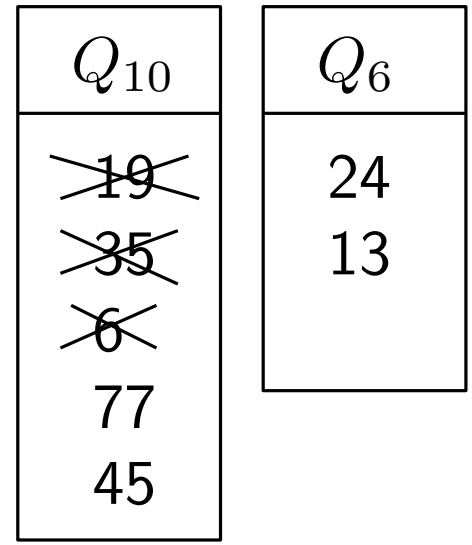
On every insertion:

- add new edge in any direction
- repeat k times:
  - flip the first edge of the longest list

Algorithm

Input: stream of updates, parameter k.

Each vertex v stores its out-edges in a queue Q_v.



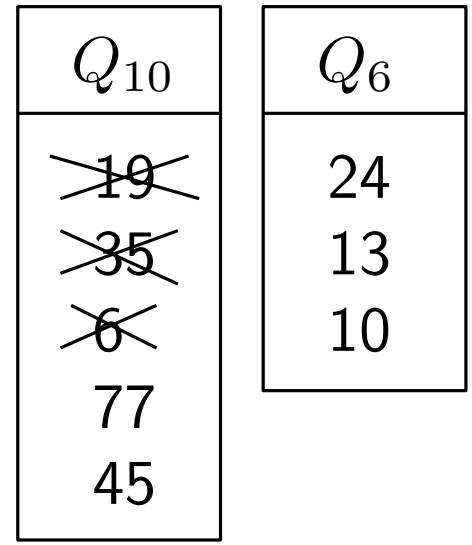
On every insertion:

- add new edge in any direction
- repeat k times:
  - flip the first edge of the longest list

Algorithm

Input: stream of updates, parameter k.

Each vertex v stores its out-edges in a queue Q_v.



On every insertion:

- add new edge in any direction
- repeat k times:
  - flip the first edge of the longest list

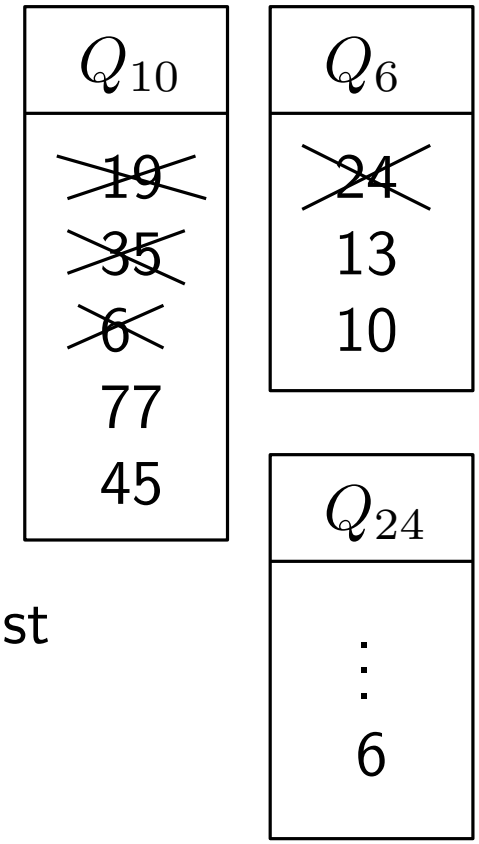
Algorithm

Input: stream of updates, parameter k.

Each vertex v stores its out-edges in a queue Q_v.

On every insertion:

- add new edge in any direction
- repeat k times:
  - flip the first edge of the longest list



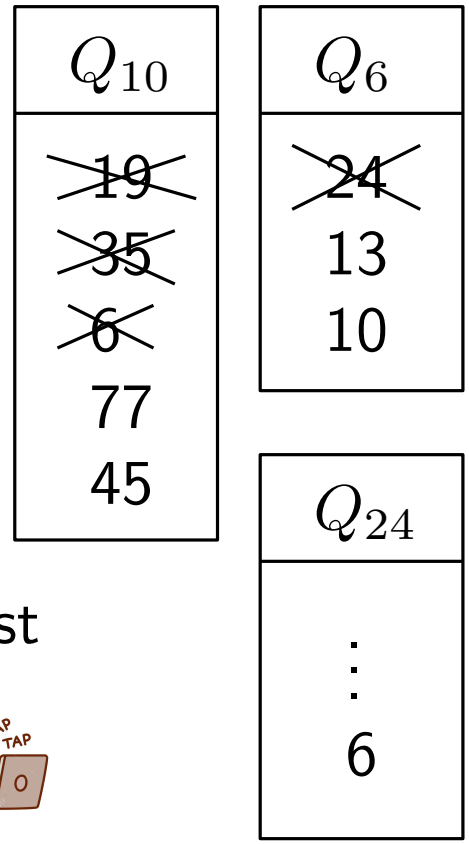
Algorithm

Input: stream of updates, parameter k.

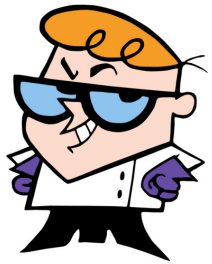
Each vertex v stores its out-edges in a queue Q_v.

On every insertion:

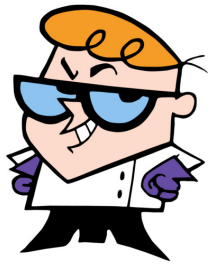
- add new edge in any direction
- repeat k times:
  - flip the first edge of the longest list



Quick analysis

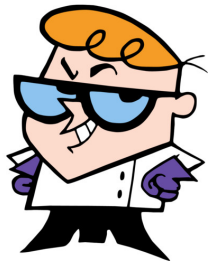


Quick analysis



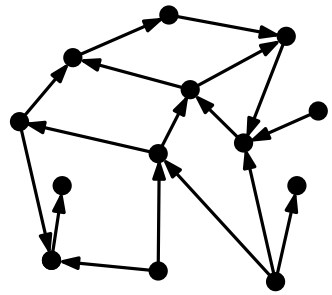
With at most x flips, what is the best you can do?

Quick analysis

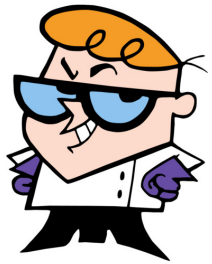


With at most x flips, what is the best you can do?

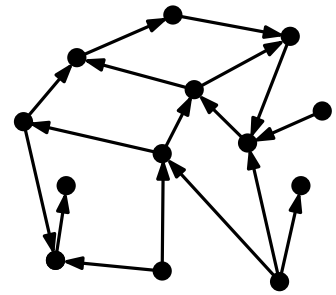
Easy! Every vertex will have out-degree at most δ.



Quick analysis



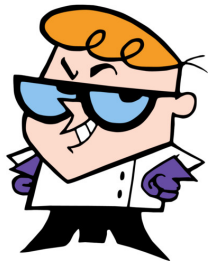
With at most x flips, what is the best you can do?



Easy! Every vertex will have out-degree at most δ.

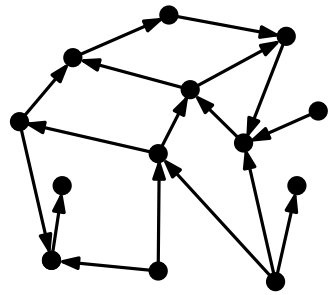
At least $2x + 2$ flips.

Quick analysis

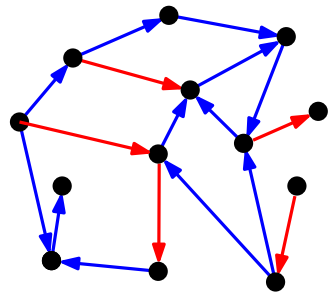


With at most x flips, what is the best you can do?

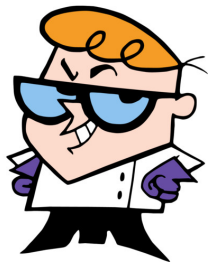
Easy! Every vertex will have out-degree at most δ.



At least $2x + 2$ flips.

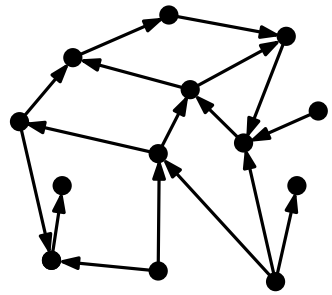


Quick analysis

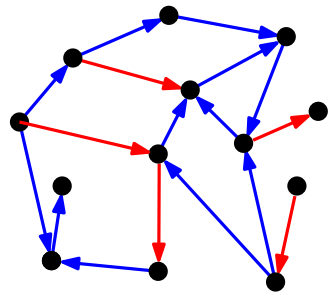


With at most x flips, what is the best you can do?

Easy! Every vertex will have out-degree at most δ.

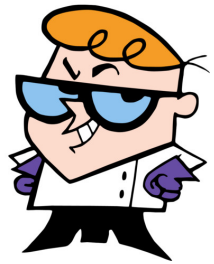


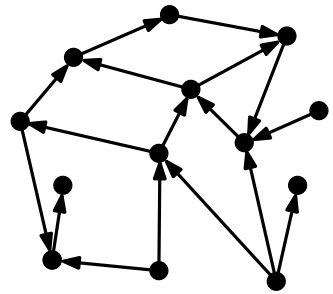
At least $2x + 2$ flips.

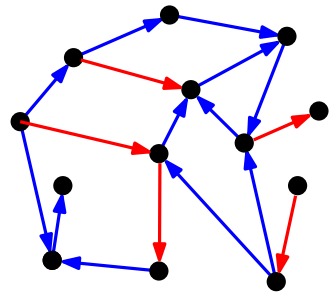


Goal: monkey's out-degrees are "not much worse than" δ.

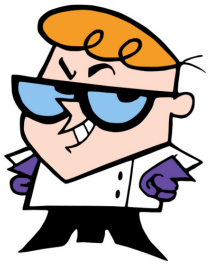
Quick analysis

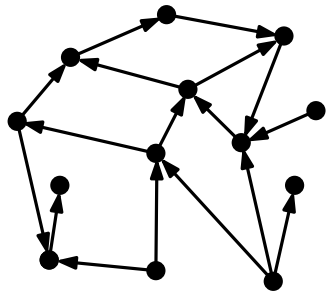


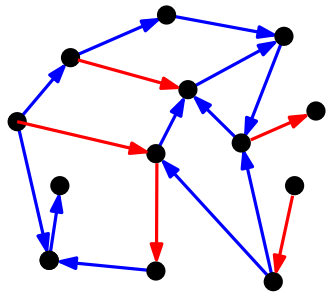




Quick analysis

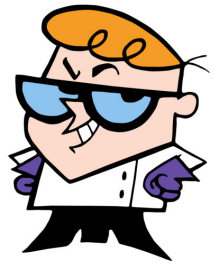


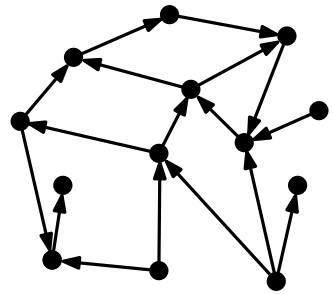


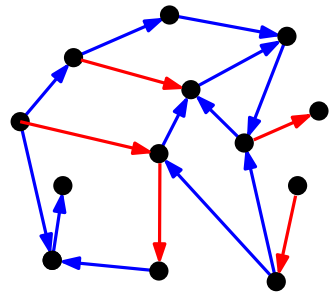


Associate a *value* to every edge: depends on its direction ("correct" or "wrong"), and on its *position in queue* (Q_v).

Quick analysis



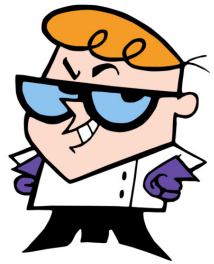


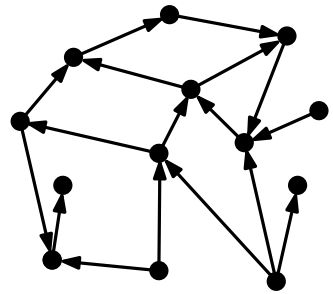


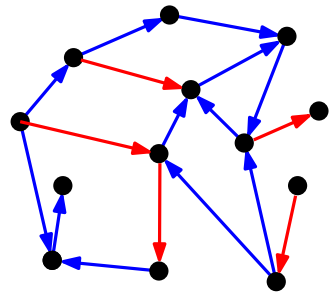
Associate a *value* to every edge: depends on its direction ("correct" or "wrong"), and on its *position in queue* (Q_v).

A vertex has "high" value $\iff$ it has many "wrong" out-edges.

Quick analysis





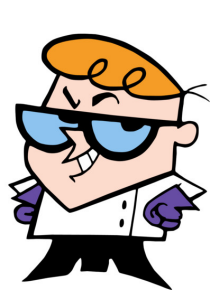


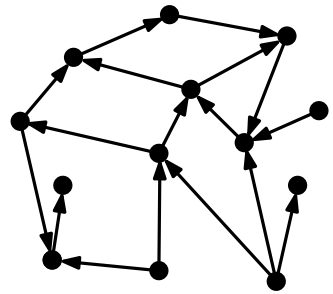
Associate a *value* to every edge: depends on its direction ("correct" or "wrong"), and on its *position in queue* (Q_v).

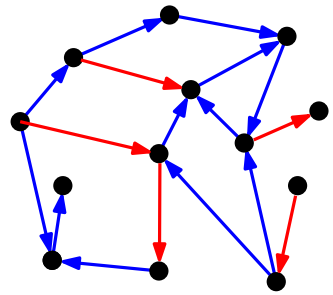
A vertex has "high" value $\iff$ it has many "wrong" out-edges.

The value on a vertex can change, but with at least $2x + 2$ flips, the *sum* of all values does not change.

Quick analysis







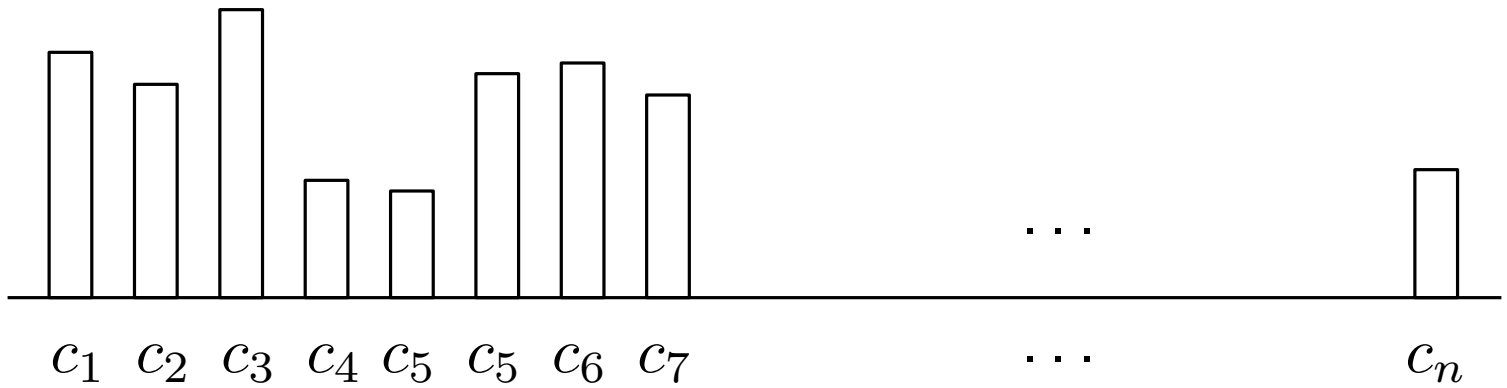
Associate a *value* to every edge: depends on its direction ("correct" or "wrong"), and on its *position in queue* (Q_v).

A vertex has "high" value $\iff$ it has many "wrong" out-edges.

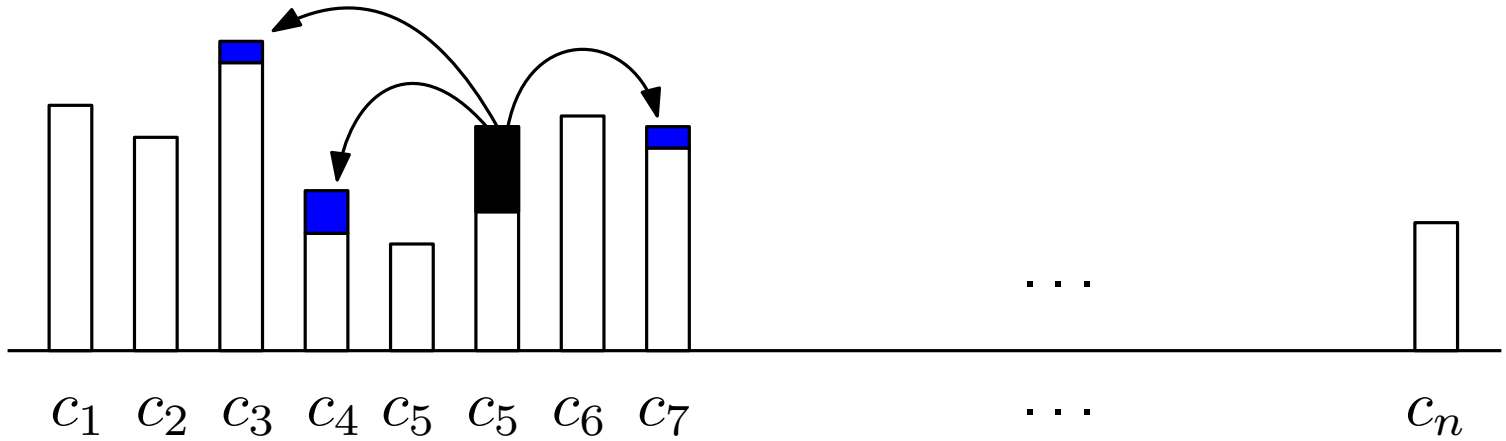
The value on a vertex can change, but with at least $2x + 2$ flips, the *sum* of all values does not change.

Forget about the graph – look *only* at values!

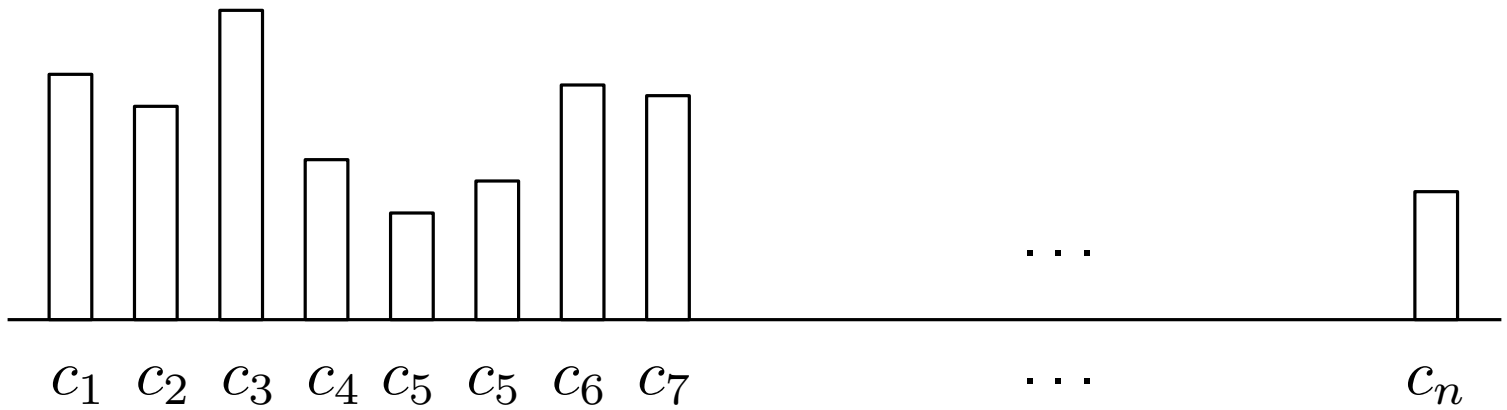
Counter game



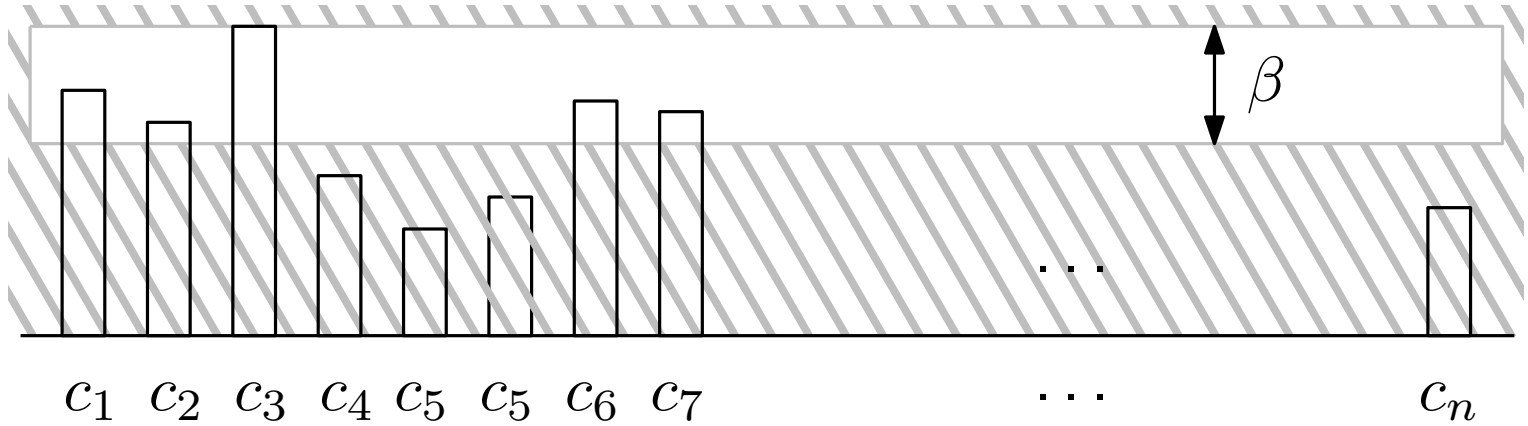
Counter game



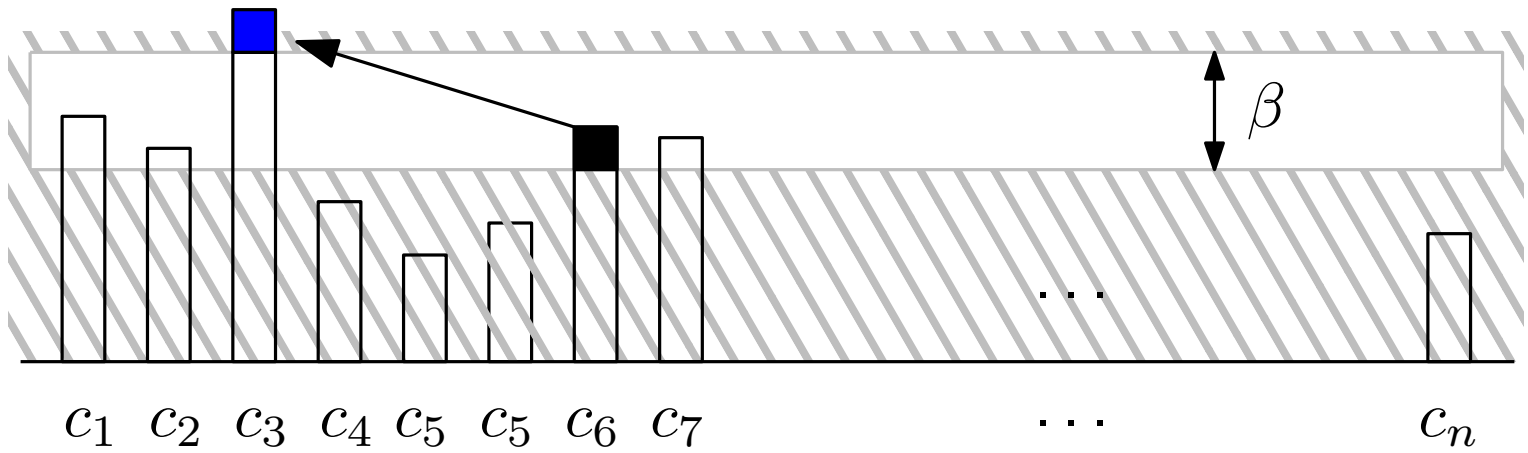
Counter game



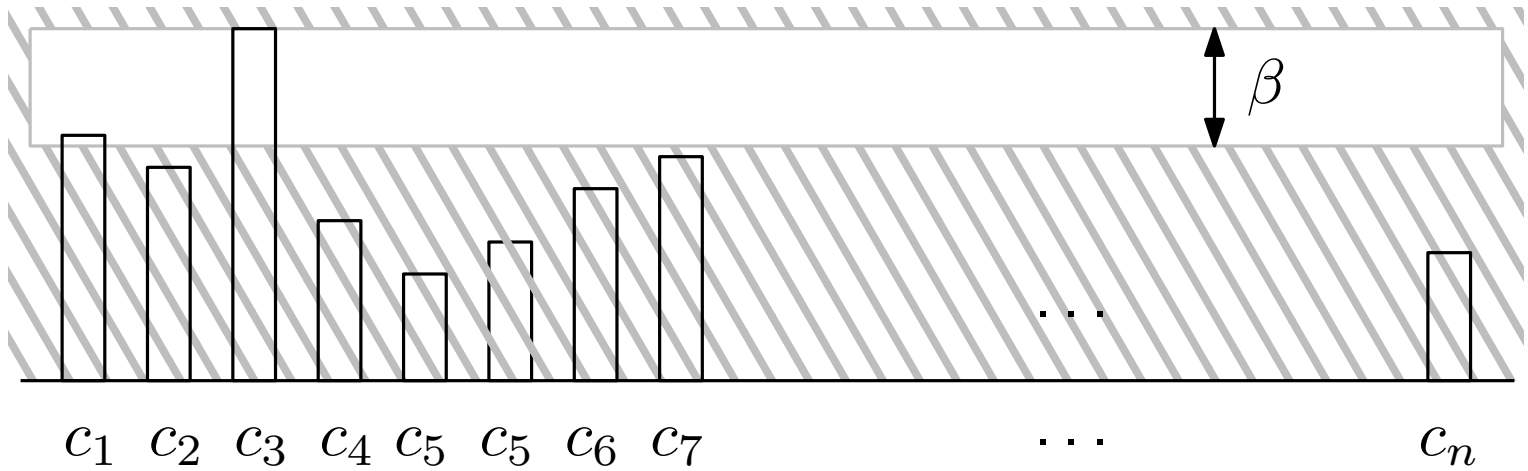
Counter game



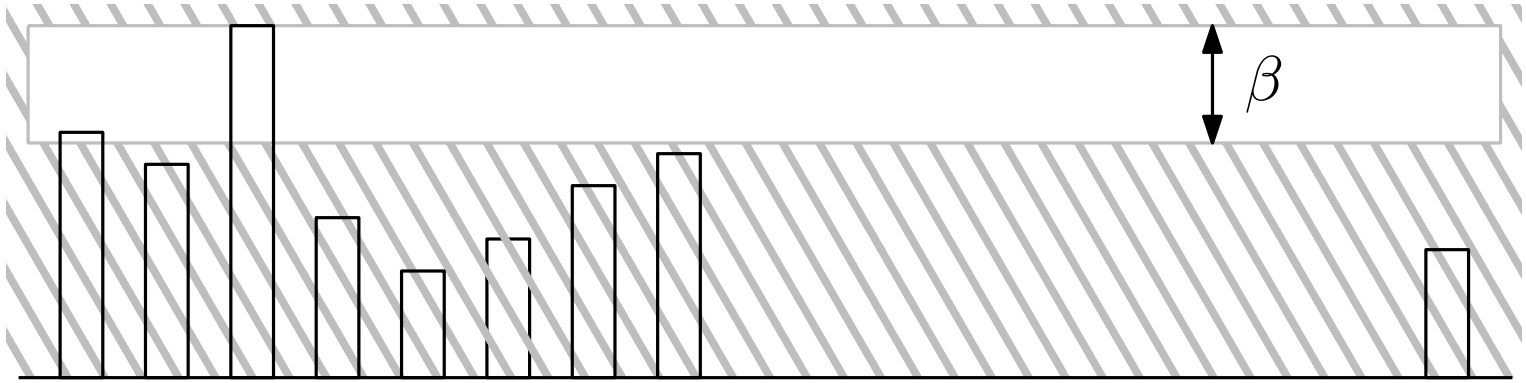
Counter game



Counter game

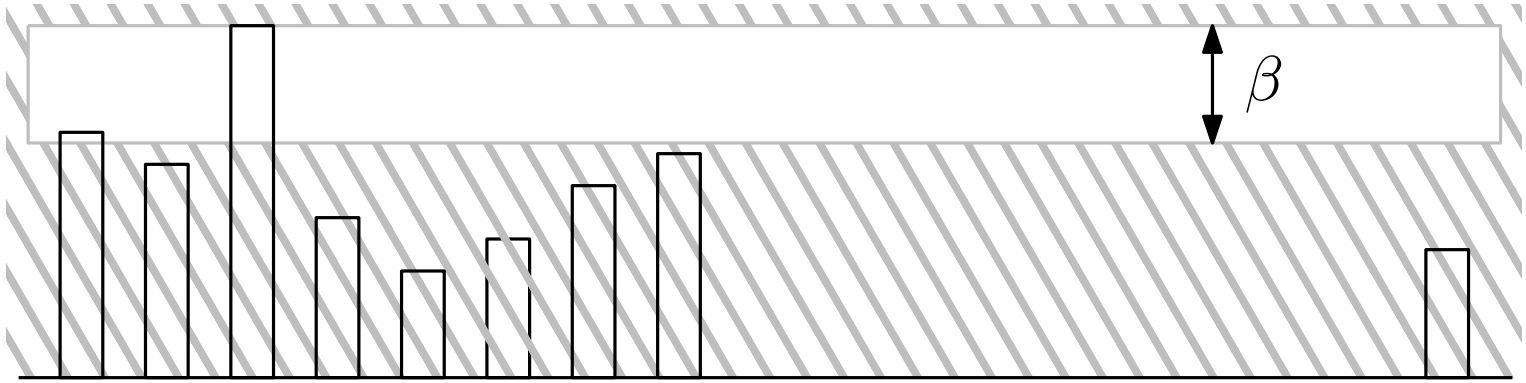


Counter game



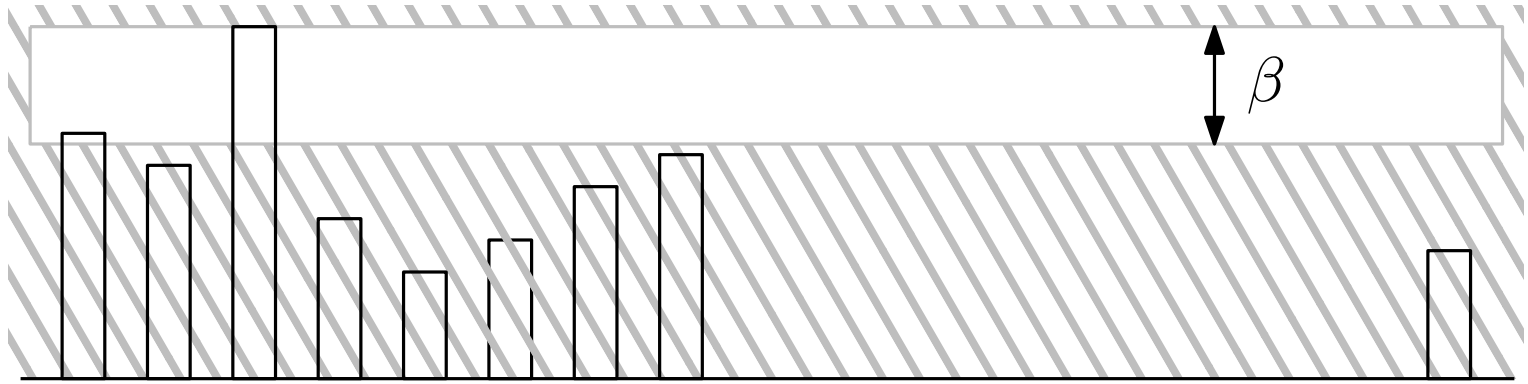
1: $\beta \approx 6\delta$ sufficient to simulate how values are moved around by the monkey.

Counter game



1: $\beta \approx 6\delta$ sufficient to simulate how values are moved around by the monkey.

Counter game

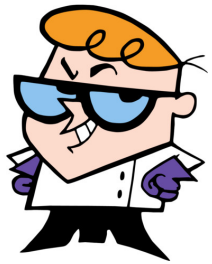


1: $\beta \approx 6\delta$ sufficient to simulate how values are moved around by the monkey.

2: An adversary cannot increase the value of any counter by more than $\beta \log n \approx 6\delta \log n$.

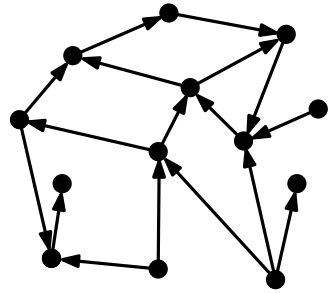
$$\log \underbrace{1000000} \approx 7$$

Quick analysis



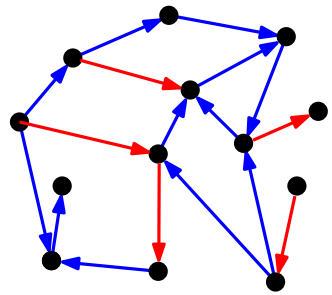
With at most x flips, what is the best you can do?

Easy! Every vertex will have out-degree at most δ.

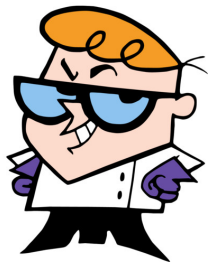


At least $2x + 2$ flips.

Proven: every vertex has out-degree at most $\approx 6\delta \log n$.

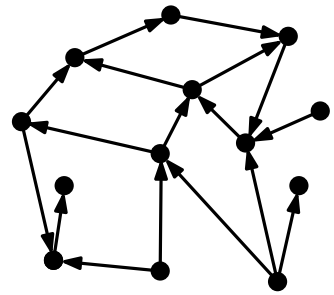


Quick analysis



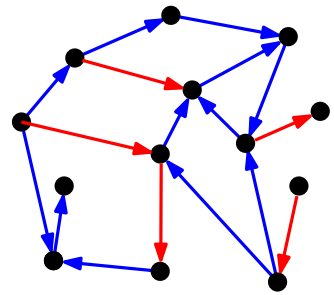
With at most x flips, what is the best you can do?

Easy! Every vertex will have out-degree at most δ.



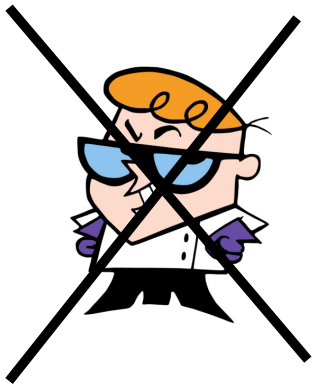
At least $2x + 2$ flips.

Proven: every vertex has out-degree at most $\approx 6\delta \log n$.



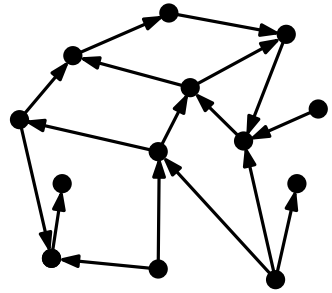
Even though monkey has no idea about x, δ, β, correct/wrong, counter games, adversaries...

Quick analysis



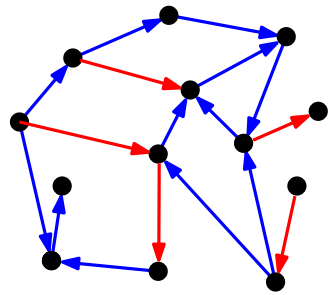
With at most x flips, what is the best you can do?

Easy! Every vertex will have out-degree at most δ.



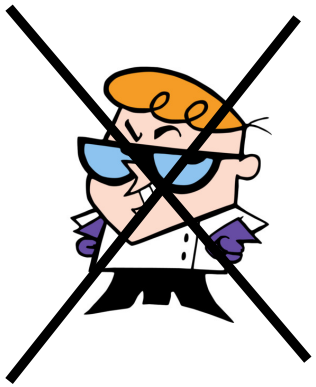
At least $2x + 2$ flips.

Proven: every vertex has out-degree at most $\approx 6\delta \log n$.



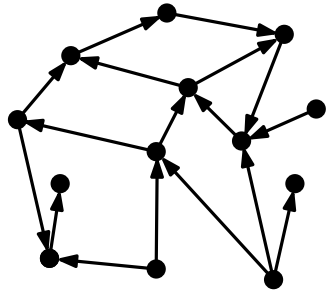
Even though monkey has no idea about x, δ, β, correct/wrong, counter games, adversaries...

Quick analysis



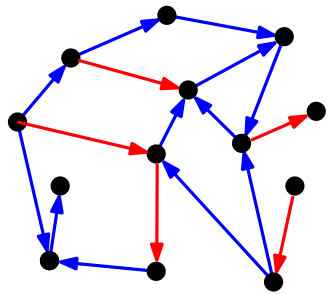
With at most x flips, what is the best you can do?

~~Easy! Every vertex will have out-degree at most δ.~~



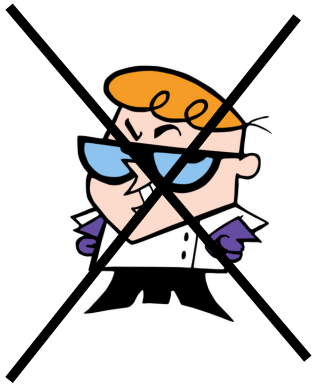
At least $2x + 2$ flips.

Proven: every vertex has out-degree at most $\approx 6\delta \log n$.



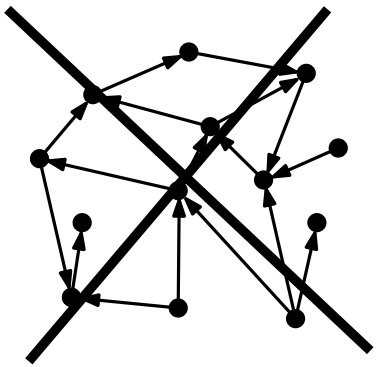
Even though monkey has no idea about x, δ, β, correct/wrong, counter games, adversaries...

Quick analysis



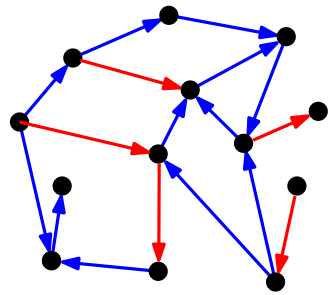
With at most x flips, what is the best you can do?

~~Easy! Every vertex will have out-degree at most δ.~~



At least $2x + 2$ flips.

Proven: every vertex has out-degree at most $\approx 6\delta \log n$.



Even though monkey has no idea about x, δ, β, correct/wrong, counter games, adversaries...

Quick analysis

Proven: with at least $2x + 2$ flips, every vertex has out-degree at most $\approx 6\delta \log n$.

What is x and δ?

Quick analysis

Proven: with at least $2x + 2$ flips, every vertex has out-degree at most $\approx 6\delta \log n$.

What is x and δ?

"I can get out-degree 2α with $\log n$ flips."
Brodal & Fagerberg (1999)

$\Rightarrow$ we get out-degree $\approx 12\alpha \log n$ with at least $2 \log n + 2$ flips.

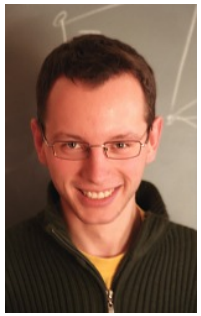
Quick analysis

Proven: with at least $2x + 2$ flips, every vertex has out-degree at most $\approx 6\delta \log n$.

What is x and δ?

"I can get out-degree 2α with $\log n$ flips."
Brodal & Fagerberg (1999)

$\Rightarrow$ we get out-degree $\approx 12\alpha \log n$ with at least $2 \log n + 2$ flips.



"I can get out-degree $2\alpha \log n$ with 1 flip."
Kowalik (2007)

$\Rightarrow$ we get out-degree $\approx 12\alpha(\log n)^2$ with at least 4 flips.

Quick analysis

Proven: with at least $2x + 2$ flips, every vertex has out-degree at most $\approx 6\delta \log n$.

What is x and δ?

"I can get out-degree 2α with 1 flip." (future)
$\Rightarrow$ we get out-degree $\approx 12\alpha \log n$ with 4 flips.

Quick analysis

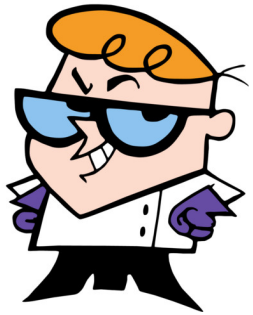
Proven: with at least $2x + 2$ flips, every vertex has out-degree at most $\approx 6\delta \log n$.

What is x and δ?

"I can get out-degree 2α with 1 flip." (future)
$\Rightarrow$ we get out-degree $\approx 12\alpha \log n$ with 4 flips.

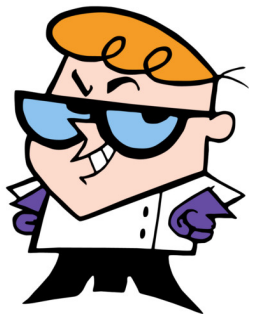
If so, the monkey *already* gets this (better) out-degree with 4 flips – we just don't know it yet.

Additional results

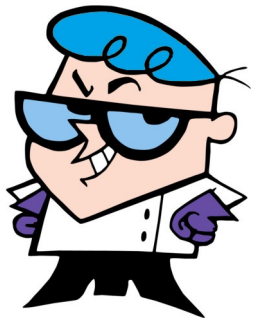


I can maintain out-degree δ with at most x flips *on average*.

Additional results

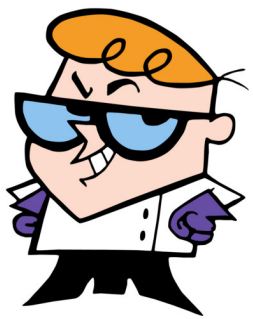


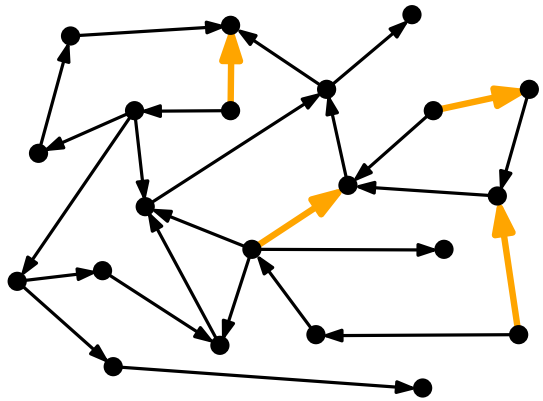
I can maintain out-degree δ with at most x flips *on average*.

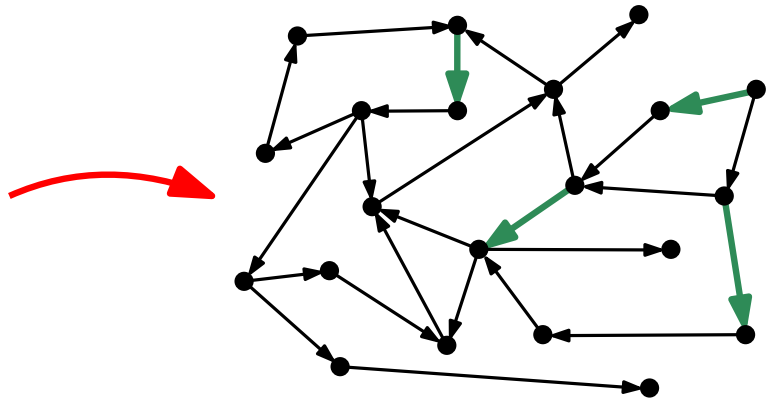


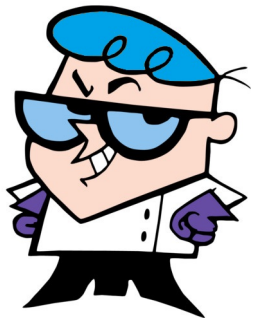
I can maintain out-degree δ with at most x flips on average.
I decide which edges to flip in *a nice way*.

Additional results

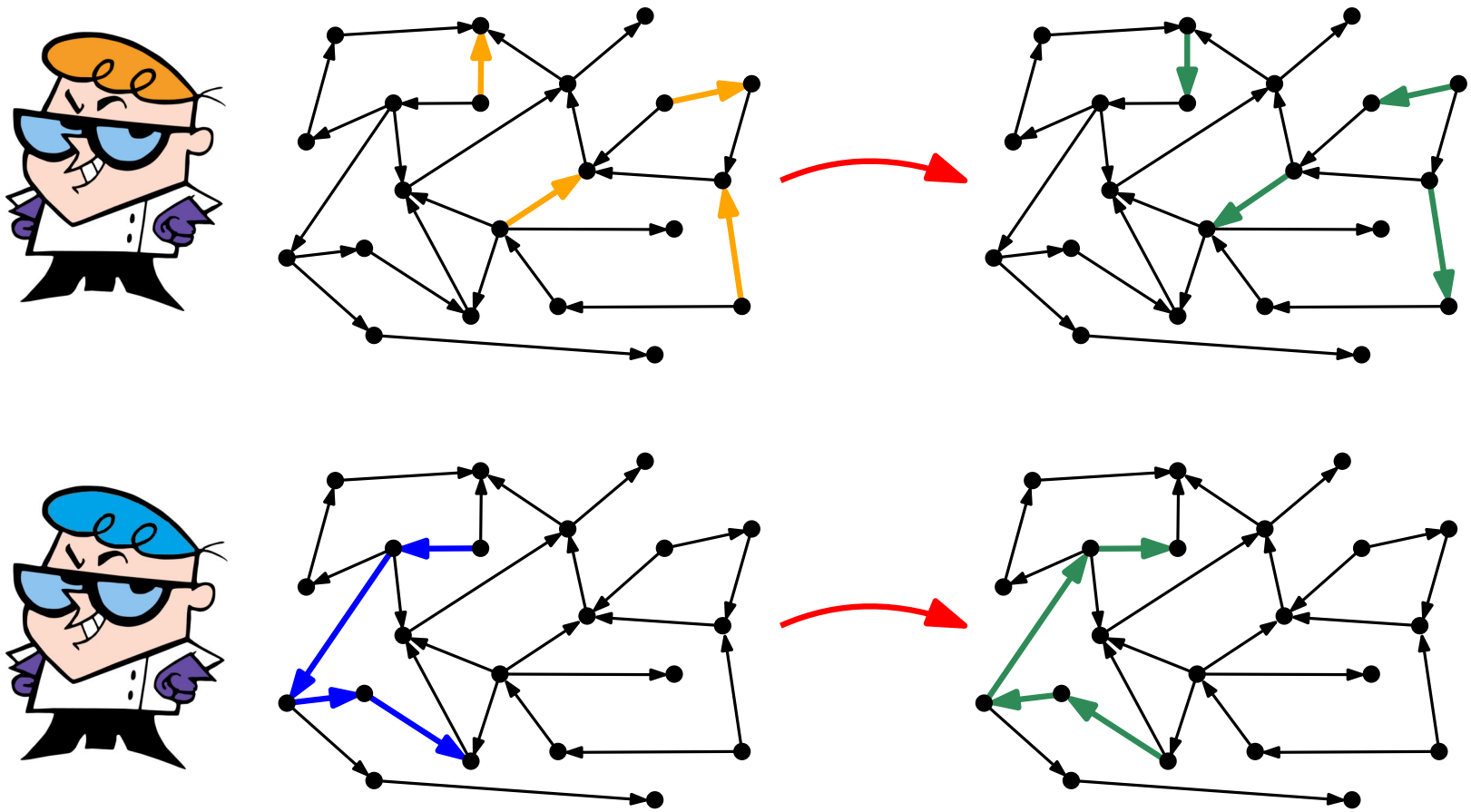




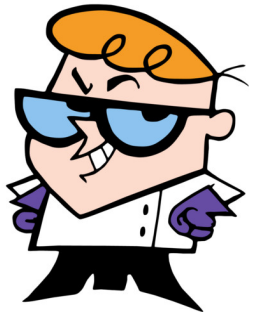




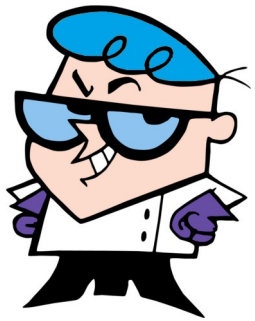
Additional results



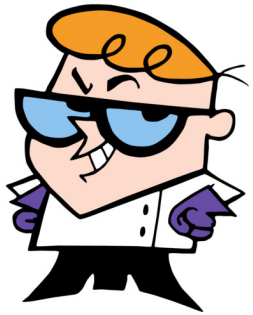
Additional results



"I want to be as smart as Dexter"

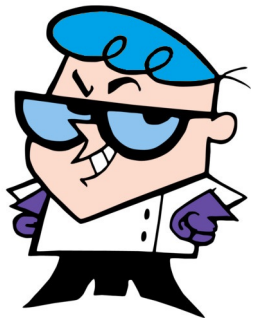


Additional results

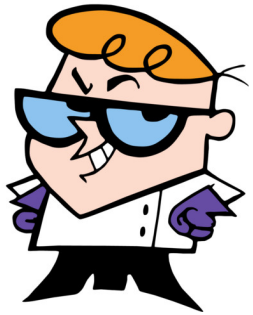


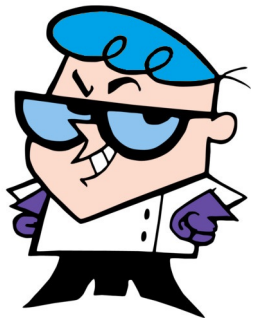
"I want to be as smart as Dexter"

"I want to find a limit on how smart Dexter can be"



Additional results



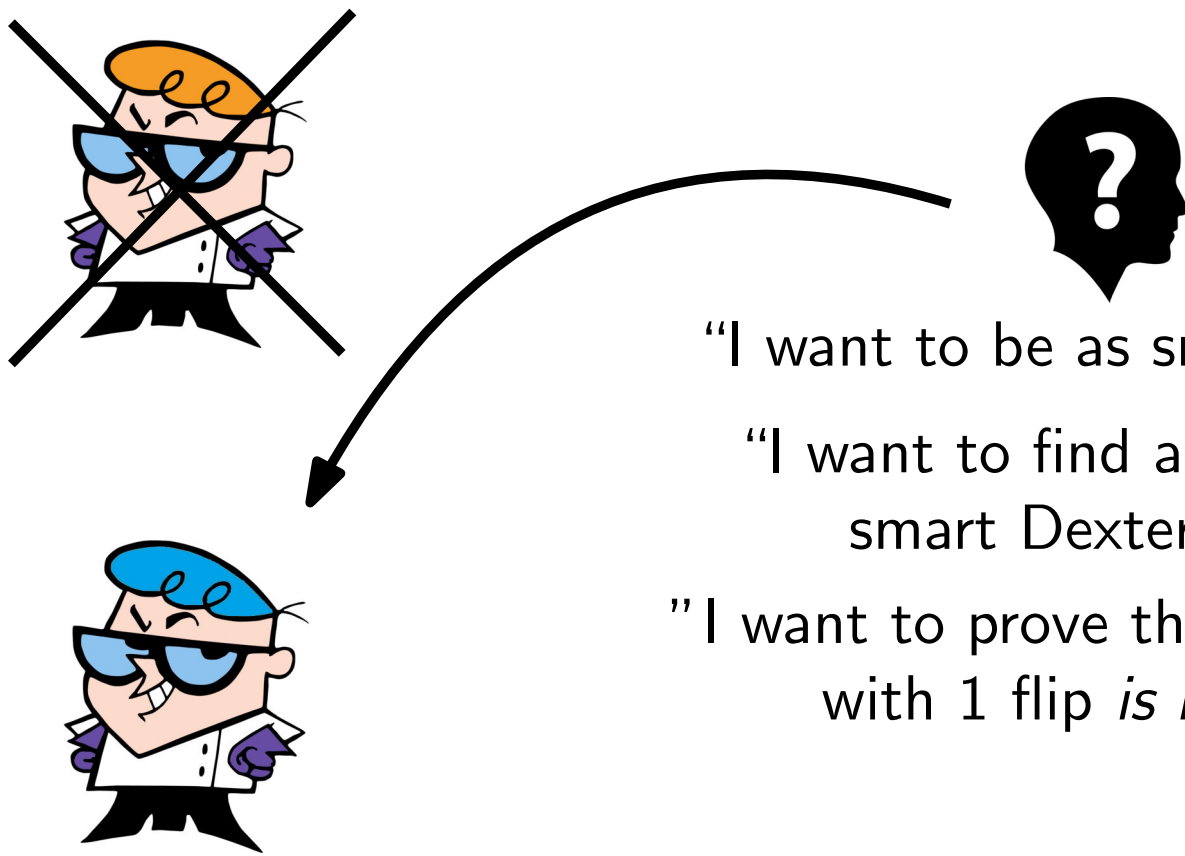


"I want to be as smart as Dexter"

"I want to find a limit on how smart Dexter can be"

"I want to prove that out-degree 2α with 1 flip *is impossible*"

Additional results



"I want to be as smart as Dexter"

"I want to find a limit on how smart Dexter can be"

"I want to prove that out-degree 2α with 1 flip *is impossible*"

Applications of incidence bounds in point covering problems

SoCG'16, Boston

joint work with Peyman Afshani, Ingo van Duijn, Jesper Sindahl Nielsen

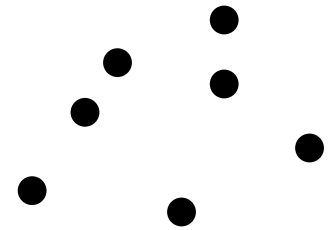
Danmarks Grundforskningsfond
Danish National Research Foundation

MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

AARHUS UNIVERSITY

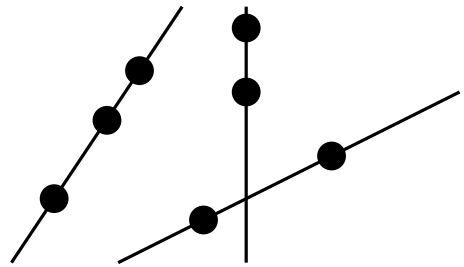
Problem definitions

Line Cover: given set of points P, draw k (or min #) lines so that every point is on a line.



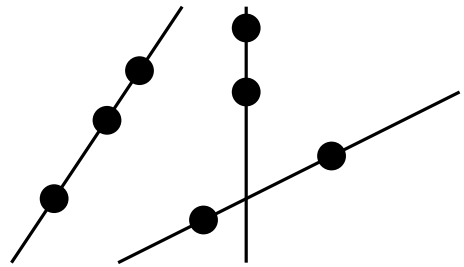
Problem definitions

Line Cover: given set of points P, draw k (or min #) lines so that every point is on a line.



Problem definitions

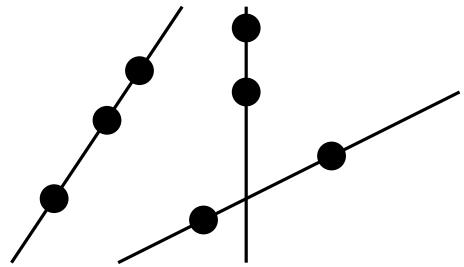
Line Cover: given set of points P, draw k (or min #) lines so that every point is on a line.



NP-complete, APX-hard, FPT

Problem definitions

Line Cover: given set of points P, draw k (or min #) lines so that every point is on a line.

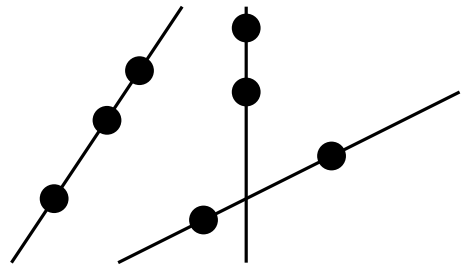


NP-complete, APX-hard, FPT

Curve Cover: given P and family of curves $\mathcal{C}$, draw k curves...

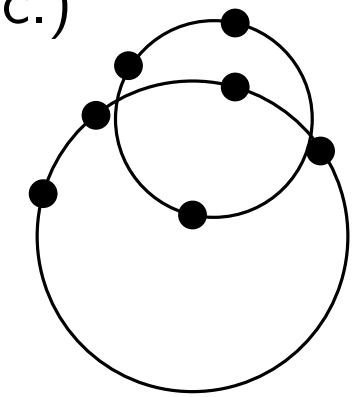
Problem definitions

Line Cover: given set of points P, draw k (or min #) lines so that every point is on a line.



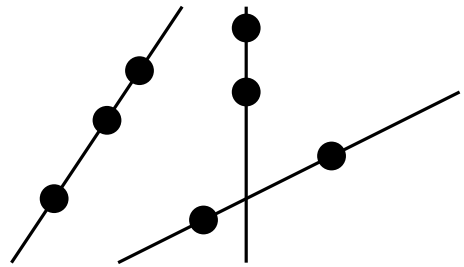
NP-complete, APX-hard, FPT

Curve Cover: given P and family of curves $\mathcal{C}$, draw k curves... (circles, hyperbolas, polynomials, splines, etc.)



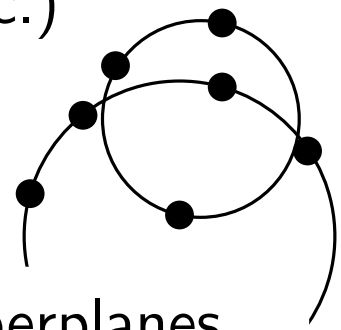
Problem definitions

Line Cover: given set of points P, draw k (or min #) lines so that every point is on a line.



NP-complete, APX-hard, FPT

Curve Cover: given P and family of curves $\mathcal{C}$, draw k curves... (circles, hyperbolas, polynomials, splines, etc.)



Hyperplane Cover: given P in $\mathbb{R}^d$, draw k hyperplanes...

Known results

Algorithm	Running Time (in $\mathcal{O}^*$)
• Naïve Solution	$\binom{n^2}{k} \approx (n^2/k)^k$
• Kernelisation	
• Previous best	
• Best non-parametrized	

Kernel

Instance: points P, budget k.

Goal: *either* quickly solve instance, *or* bound $|P| \leq \mathrm{poly}(k)$.

Kernel

Instance: points P, budget k.

Goal: *either* quickly solve instance, *or* bound $|P| \leq \mathrm{poly}(k)$.

Obs: If there are $k+1$ collinear points, every solution includes the line through them.

Kernel

Instance: points P, budget k.

Goal: *either* quickly solve instance, *or* bound $|P| \leq \mathrm{poly}(k)$.

Obs: If there are $k+1$ collinear points, every solution includes the line through them.

Polytime kernelization algorithm:

1. Find $k+1$ collinear pts, remove them and decrement k.

Kernel

Instance: points P, budget k.

Goal: *either* quickly solve instance, *or* bound $|P| \leq \mathrm{poly}(k)$.

Obs: If there are $k+1$ collinear points, every solution includes the line through them.

Polytime kernelization algorithm:

1. Find $k+1$ collinear pts, remove them and decrement k.
2. Exhaustively repeat step 1 until only $\leq k$ collinear points.

Kernel

Instance: points P, budget k.

Goal: *either* quickly solve instance, *or* bound $|P| \leq \mathrm{poly}(k)$.

Obs: If there are $k+1$ collinear points, every solution includes the line through them.

Polytime kernelization algorithm:

1. Find $k+1$ collinear pts, remove them and decrement k.
2. Exhaustively repeat step 1 until only $\leq k$ collinear points.
3. If $|P| > k^2$, reject instance.

Kernel

Instance: points P, budget k.

Goal: *either* quickly solve instance, *or* bound $|P| \leq \mathrm{poly}(k)$.

Obs: If there are $k+1$ collinear points, every solution includes the line through them.

Polytime kernelization algorithm:

1. Find $k+1$ collinear pts, remove them and decrement k.
2. Exhaustively repeat step 1 until only $\leq k$ collinear points.
3. If $|P| > k^2$, reject instance.

Result: $|P| \leq k^2$, no $(k+1)$-rich line.

Known results

Algorithm	Running Time (in $\mathcal{O}^*$)
• Naïve Solution	$\binom{n^2}{k} \approx (n^2/k)^k$
• Kernelisation	
• Previous best	
• Best non-parametrized	

Known results

Algorithm	Running Time (in $\mathcal{O}^*$)
• Naïve Solution	$\binom{n^2}{k} \approx (n^2/k)^k$
• Kernelisation	$\binom{(k^2)^2}{k} \approx k^{3k}$
• Previous best	
• Best non-parametrized	

Known results

Algorithm	Running Time (in $\mathcal{O}^*$)
• Naïve Solution	$\binom{n^2}{k} \approx (n^2/k)^k$
• Kernelisation	$\binom{(k^2)^2}{k} \approx k^{3k}$
• Previous best	$(k/1.35)^k$, Wang et al. '10
• Best non-parametrized	

Known results

Algorithm	Running Time (in $\mathcal{O}^*$)
• Naïve Solution	$\binom{n^2}{k} \approx (n^2/k)^k$
• Kernelisation	$\binom{(k^2)^2}{k} \approx k^{3k}$
• Previous best	$(k/1.35)^k$, Wang et al. '10
• Best non-parametrized	2^n, 2^{k^2} on kernel ☹

Known results

Algorithm	Running Time (in $\mathcal{O}^*$)
• Naïve Solution	$\binom{n^2}{k} \approx (n^2/k)^k$
• Kernelisation	$\binom{(k^2)^2}{k} \approx k^{3k}$
• Previous best	$(k/1.35)^k$, Wang et al. '10
• Best non-parametrized	2^n, 2^{k^2} on kernel ☹

*but competitive if $n \approx k \log k$

Known results

Algorithm	Running Time (in $\mathcal{O}^*$)
• Naïve Solution	$\binom{n^2}{k} \approx (n^2/k)^k$
• Kernelisation	$\binom{(k^2)^2}{k} \approx k^{3k}$
• Previous best	$(k/1.35)^k$, Wang et al. '10
• Best non-parametrized	2^n, 2^{k^2} on kernel ☹

*but competitive if $n \approx k \log k$

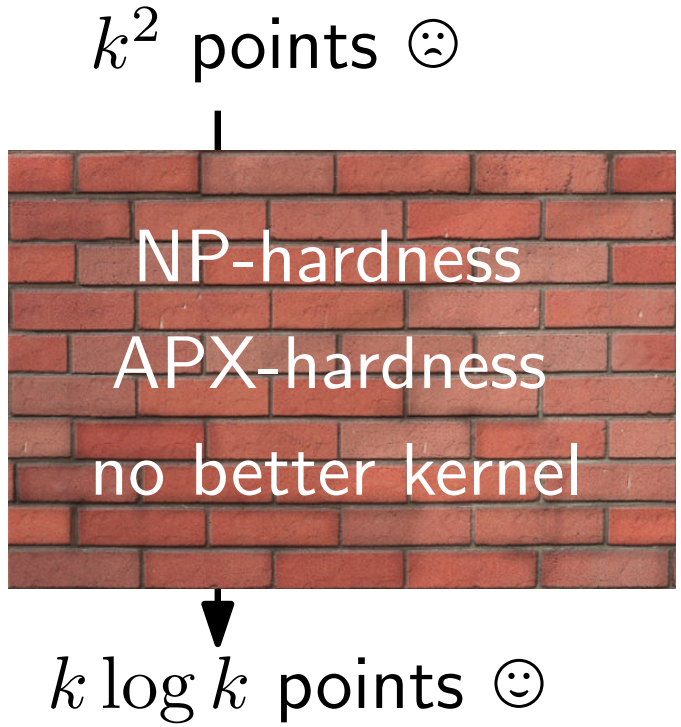
Kratsch et al. '14: No $\mathcal{O}(k^{2-\varepsilon})$ kernel ☹.

Our hero

k^2 points ☹

$k \log k$ points ☺

Our hero


k^2 points ☹
NP-hardness
APX-hardness
no better kernel
$k \log k$ points ☺

Our hero

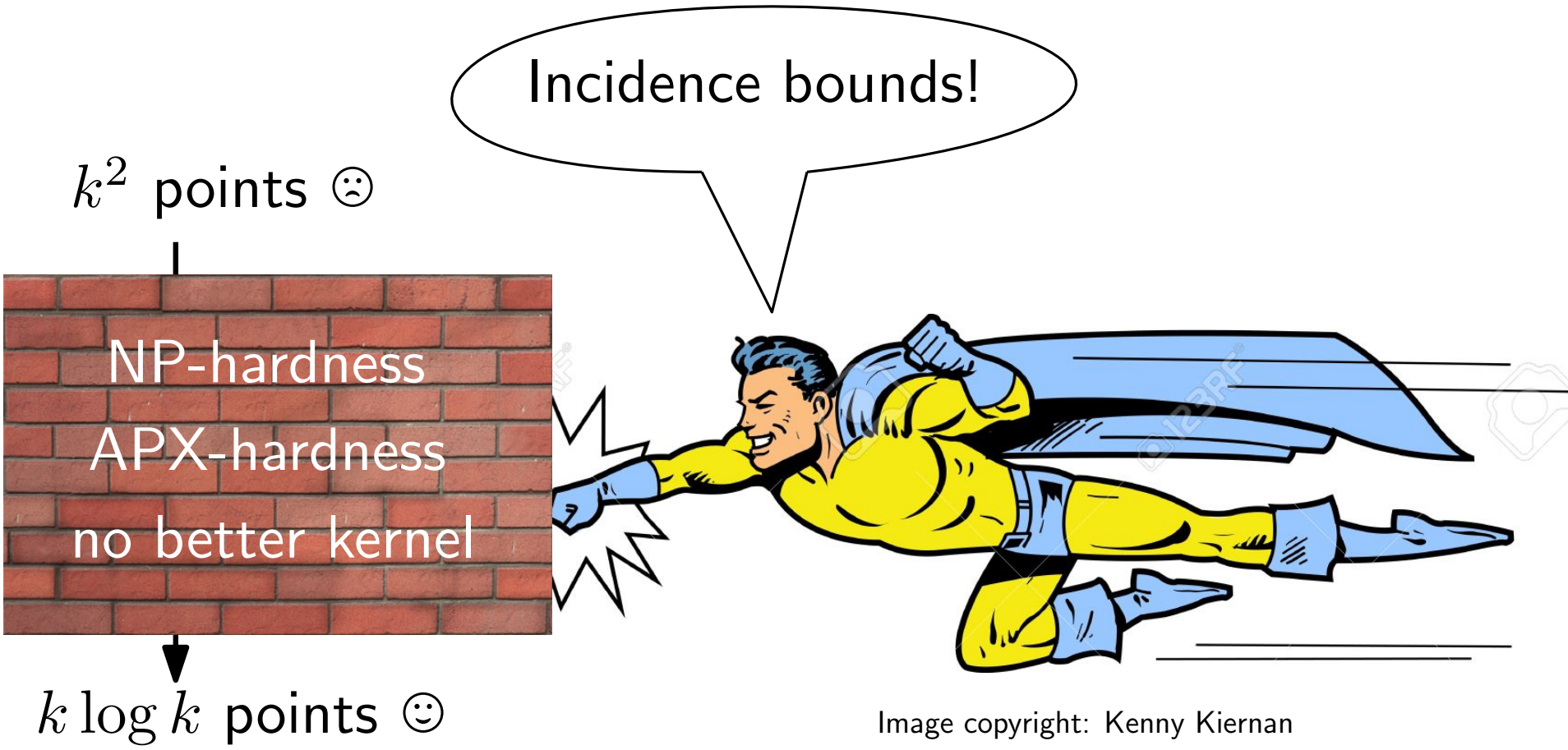
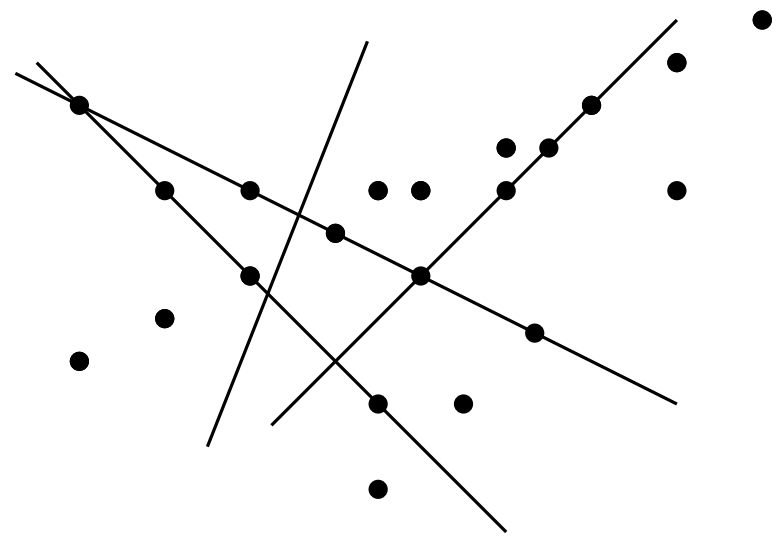


Image copyright: Kenny Kiernan

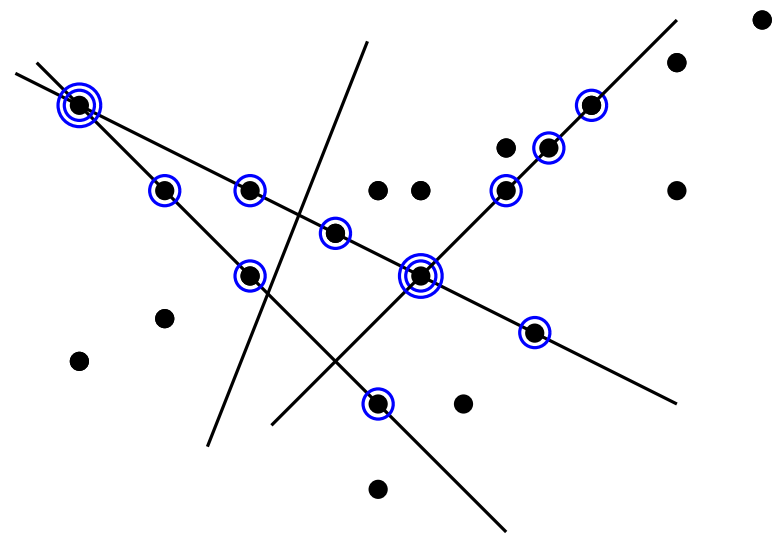
Incidence bounds

Arrangement of n points P and m lines L.



Incidence bounds

Arrangement of n points P and m lines L.

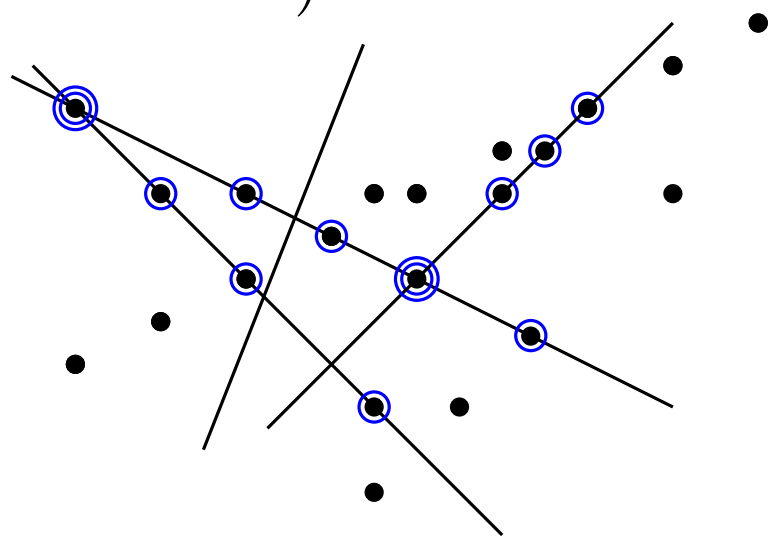


Incidence bounds

Arrangement of n points P and m lines L.

Szemerédi&Trotter '83:

#incidences $I(P, L) = \mathcal{O}\big((nm)^{2/3} + n + m\big)$.

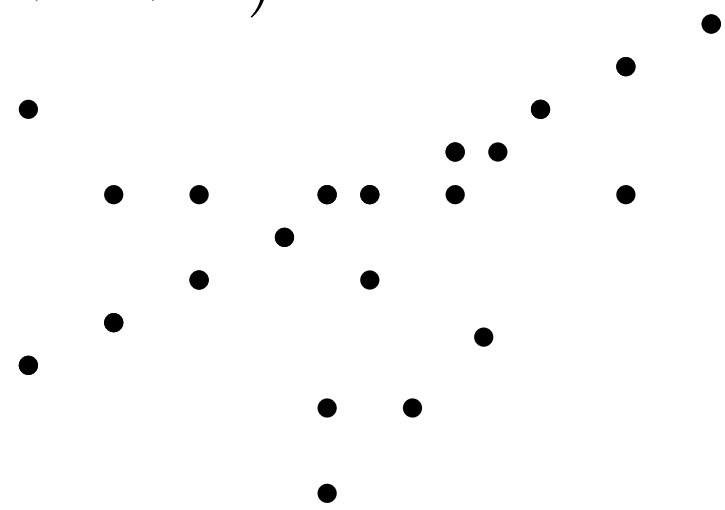


Incidence bounds

Arrangement of n points P and m lines L.

Szemerédi&Trotter '83:

#incidences $I(P, L) = \mathcal{O}\big((nm)^{2/3} + n + m\big)$.



Corollary:

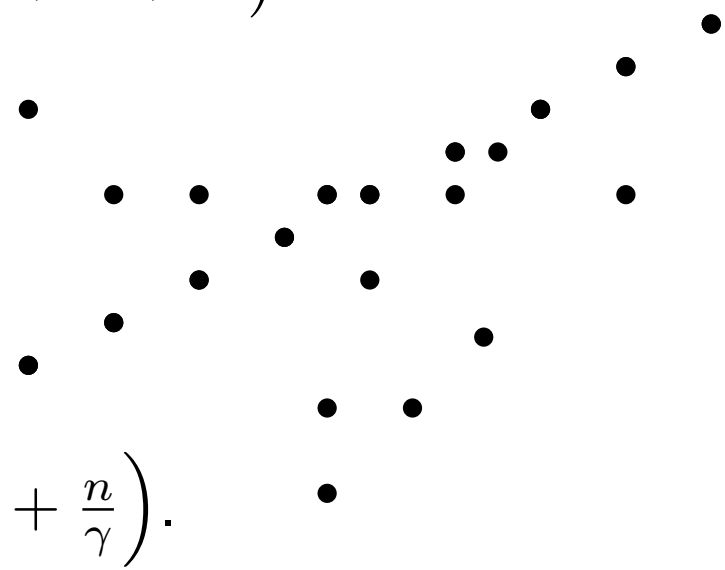
n points P.

Incidence bounds

Arrangement of n points P and m lines L.

Szemerédi&Trotter '83:

#incidences $I(P, L) = \mathcal{O}\big((nm)^{2/3} + n + m\big)$.



Corollary:

n points P.

of γ-rich candidates $m = \mathcal{O}\left(\frac{n^2}{\gamma^3} + \frac{n}{\gamma}\right)$.

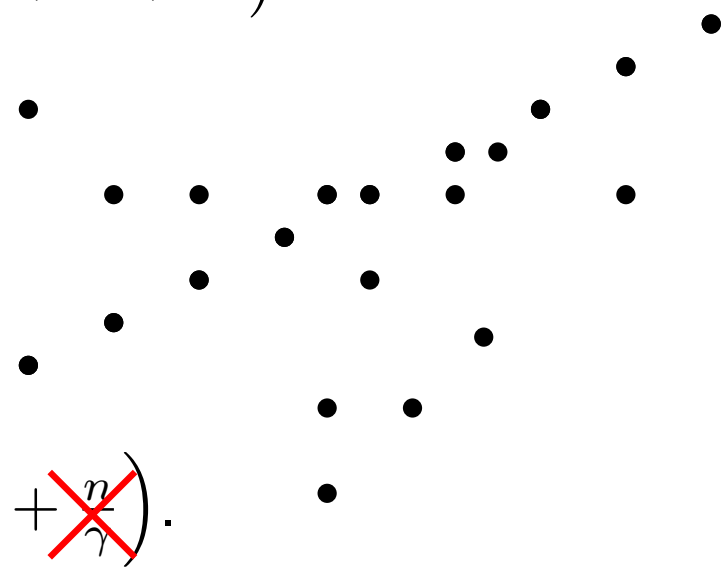
Pf: m γ-rich candidates $\Rightarrow$ at least $m\gamma = \mathcal{O}(\dots)$ incidences.

Incidence bounds

Arrangement of n points P and m lines L.

Szemerédi&Trotter '83:

#incidences $I(P, L) = \mathcal{O}\big((nm)^{2/3} + n + m\big)$.



Corollary:

n points P.

of γ-rich candidates $m = \mathcal{O}\Big(\frac{n^2}{\gamma^3} + \cancel{\frac{n}{\gamma}}\Big)$.

Pf: m γ-rich candidates $\Rightarrow$ at least $m\gamma = \mathcal{O}(\dots)$ incidences.

Constructing our algorithm

Kernel: $|P| \leq k^2$, no $(k+1)$–rich candidate.

S&T: few "rich" candidates (some high richness $\gamma_1 < k+1$).

Constructing our algorithm

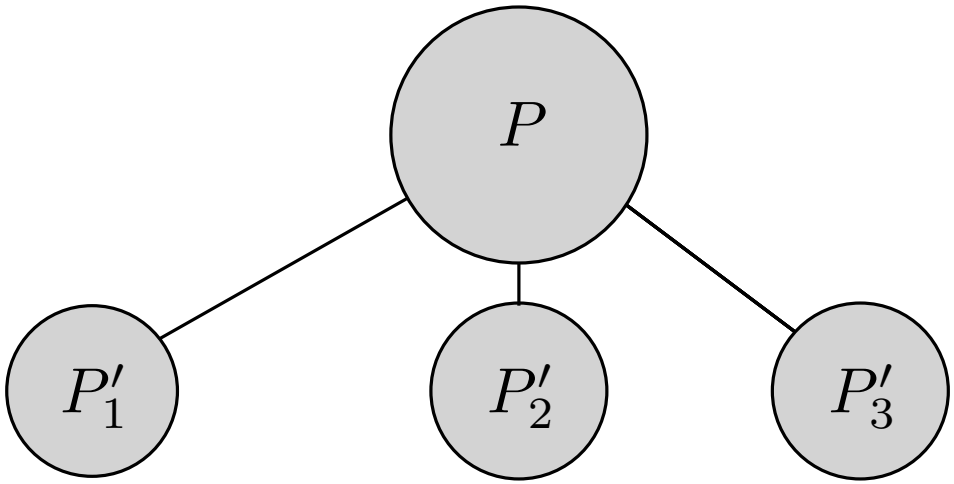
Kernel: $|P| \leq k^2$, no $(k+1)$–rich candidate.

S&T: few "rich" candidates (some high richness $\gamma_1 < k+1$).

Suppose we know solution S contains k_1 such "rich" lines.

Branch in $\binom{\text{few}}{k_1}$ ways, make very good progress ☺.

Constructing our algorithm



branch $\binom{\text{few}}{k_1}$ ways
kill many points

Constructing our algorithm

Kernel: $|P| \leq k^2$, no $(k+1)$–rich candidate.

S&T: few "rich" candidates (some high richness $\gamma_1 < k+1$).

Suppose we know solution S contains k_1 such "rich" lines.

Branch in $\binom{\text{few}}{k_1}$ ways, make very good progress ☺.

Constructing our algorithm

Kernel: $|P| \leq k^2$, no $(k+1)$–rich candidate.

S&T: few "rich" candidates (some high richness $\gamma_1 < k+1$).

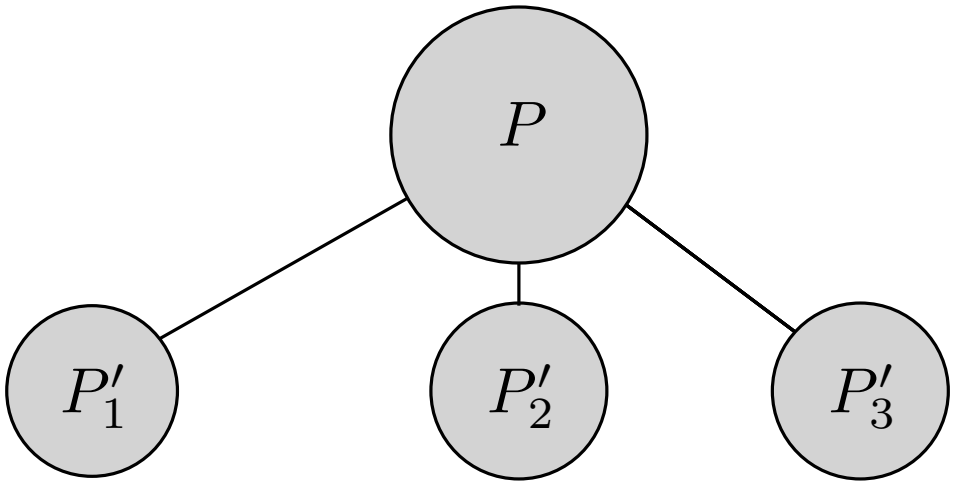
Suppose we know solution S contains k_1 such "rich" lines.

Branch in $\binom{\text{few}}{k_1}$ ways, make very good progress ☺.

Know S contains k_2 not-as-rich lines (fairly high $\gamma_2 < \gamma_1$).

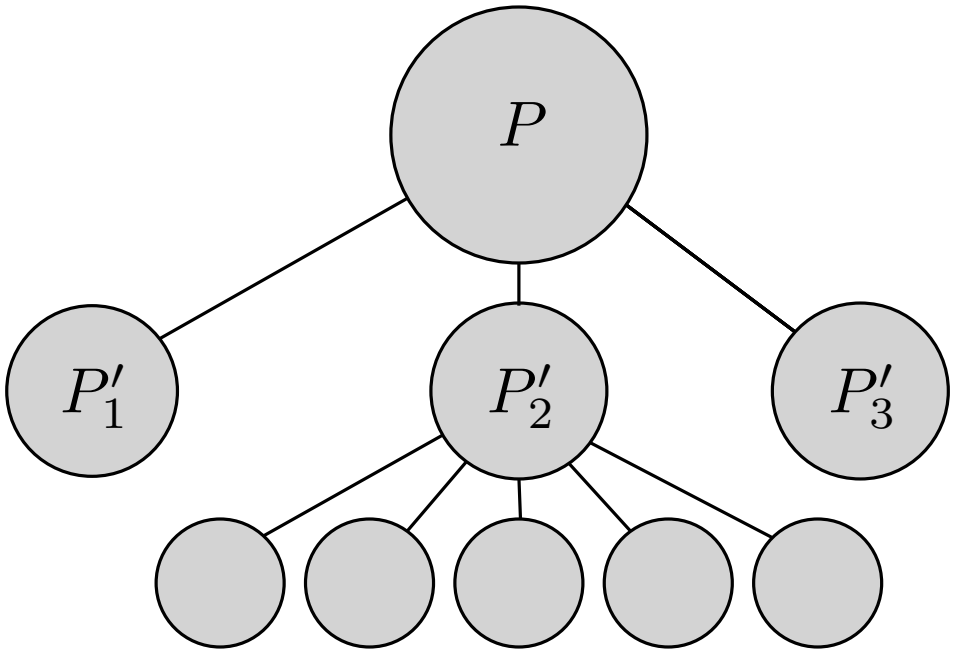
S&T: lower richness $\Rightarrow$ more candidates.

Constructing our algorithm



branch $\binom{\text{few}}{k_1}$ ways
kill many points

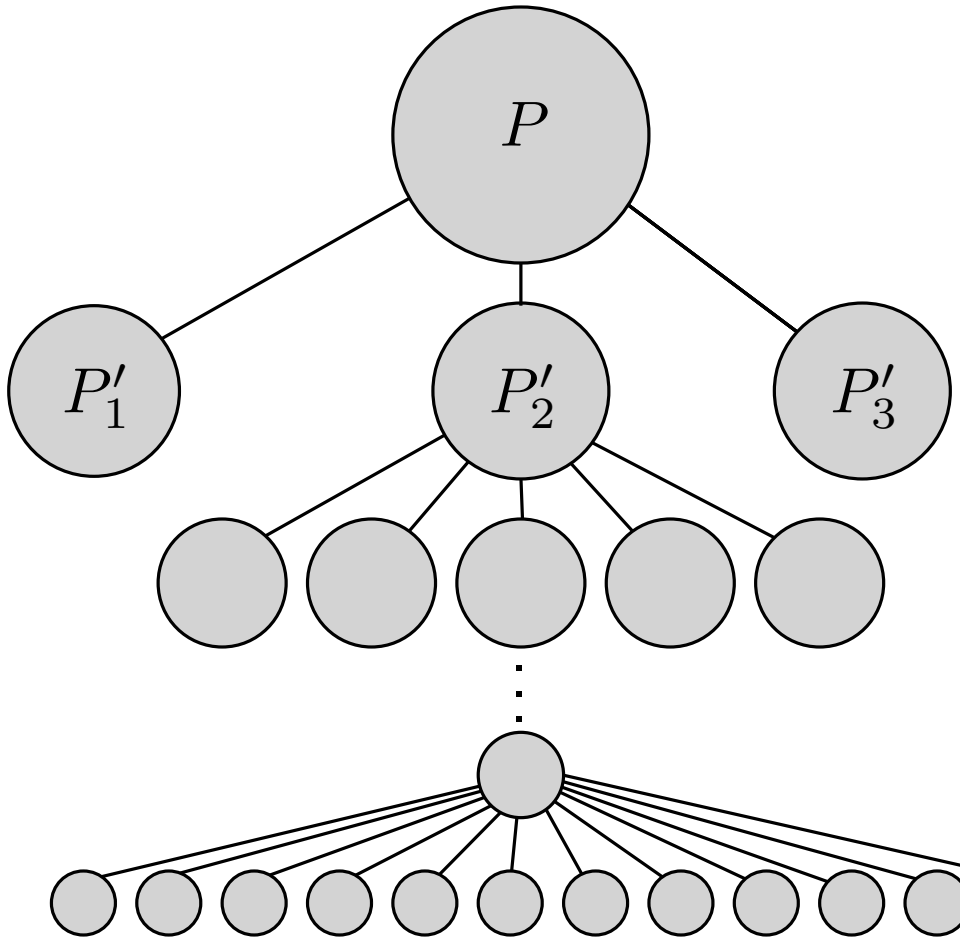
Constructing our algorithm



branch $\binom{\text{few}}{k_1}$ ways
kill many points

branch $\binom{\text{slighly more}}{k_2}$ ways
kill slightly fewer points

Constructing our algorithm



branch $\binom{\text{few}}{k_1}$ ways
kill many points

branch $\binom{\text{slighly more}}{k_2}$ ways
kill slightly fewer points

branch $\binom{\text{very many}}{k_i}$ ways
kill very few points

Constructing our algorithm

Kernel: $|P| \leq k^2$, no $(k+1)$–rich candidate.

S&T: few "rich" candidates (some high richness $\gamma_1 < k+1$).

Suppose we know solution S contains k_1 such "rich" lines.

Branch in $\binom{\text{few}}{k_1}$ ways, make very good progress ☺.

Know S contains k_2 not-as-rich lines (fairly high $\gamma_2 < \gamma_1$).

S&T: lower richness $\Rightarrow$ more candidates.

Constructing our algorithm

Kernel: $|P| \leq k^2$, no $(k+1)$–rich candidate.

S&T: few "rich" candidates (some high richness $\gamma_1 < k+1$).

Suppose we know solution S contains k_1 such "rich" lines.

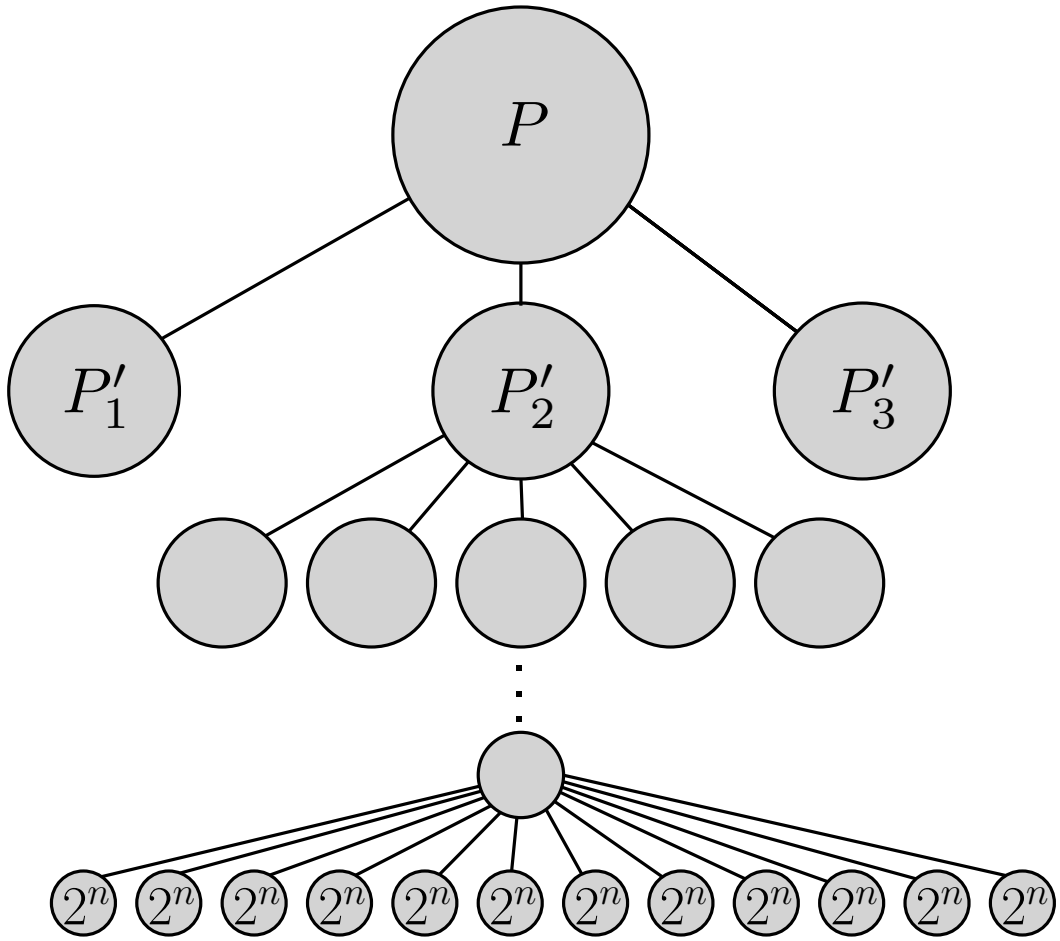
Branch in $\binom{\text{few}}{k_1}$ ways, make very good progress ☺.

Know S contains k_2 not-as-rich lines (fairly high $\gamma_2 < \gamma_1$).

S&T: lower richness $\Rightarrow$ more candidates.

But to cover P' with k' *poor* lines, $|P'|$ must be low (relative to k')! Switch to non-parametrized algorithm.

Constructing our algorithm



branch $\binom{\text{few}}{k_1}$ ways
kill many points

branch $\binom{\text{slighly more}}{k_2}$ ways
kill slightly fewer points

branch $\binom{\text{very many}}{k_i}$ ways
kill very few points

Constructing our algorithm

Kernel: $|P| \leq k^2$, no $(k+1)$–rich candidate.

S&T: few "rich" candidates (some high richness $\gamma_1 < k+1$).

Suppose we know solution S contains k_1 such "rich" lines.

Branch in $\binom{\text{few}}{k_1}$ ways, make very good progress ☺.

Know S contains k_2 not-as-rich lines (fairly high $\gamma_2 < \gamma_1$).

S&T: lower richness $\Rightarrow$ more candidates.

But to cover P' with k' *poor* lines, $|P'|$ must be low (relative to k')! Switch to non-parametrized algorithm.

Constructing our algorithm

Kernel: $|P| \leq k^2$, no $(k+1)$–rich candidate.

S&T: few "rich" candidates (some high richness $\gamma_1 < k+1$).

Suppose we know solution S contains k_1 such "rich" lines.

Branch in $\binom{\text{few}}{k_1}$ ways, make very good progress ☺.

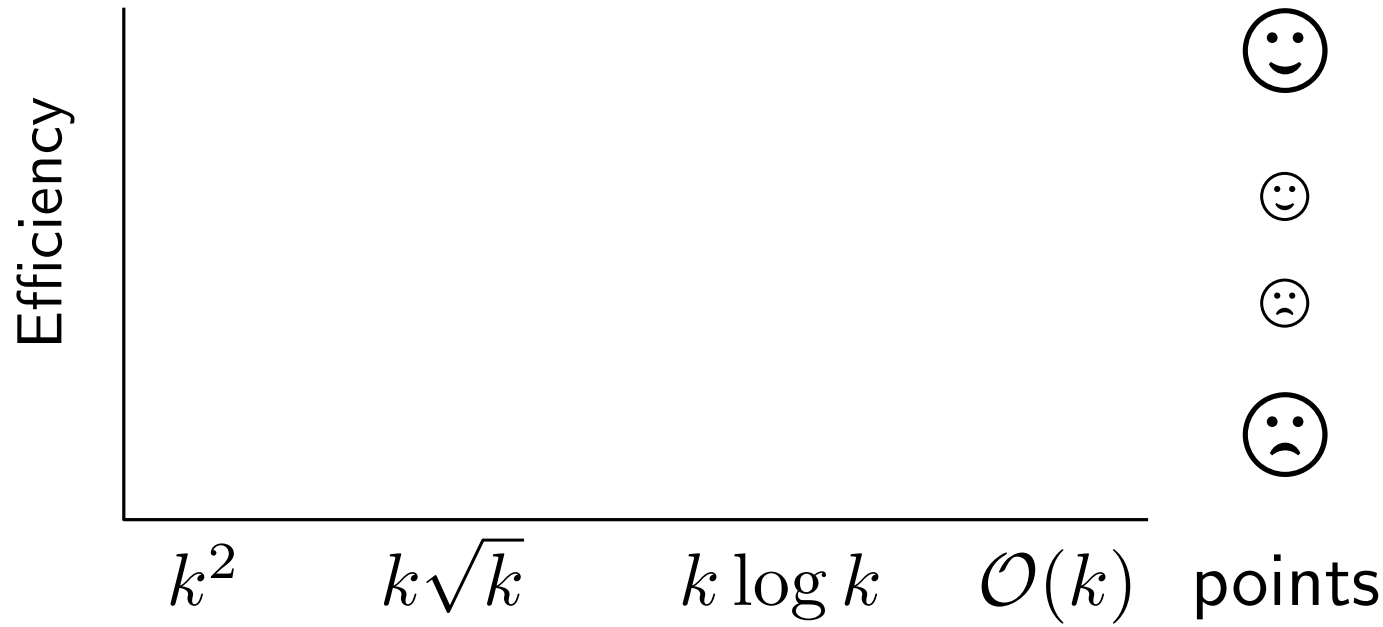
Know S contains k_2 not-as-rich lines (fairly high $\gamma_2 < \gamma_1$).

S&T: lower richness $\Rightarrow$ more candidates.

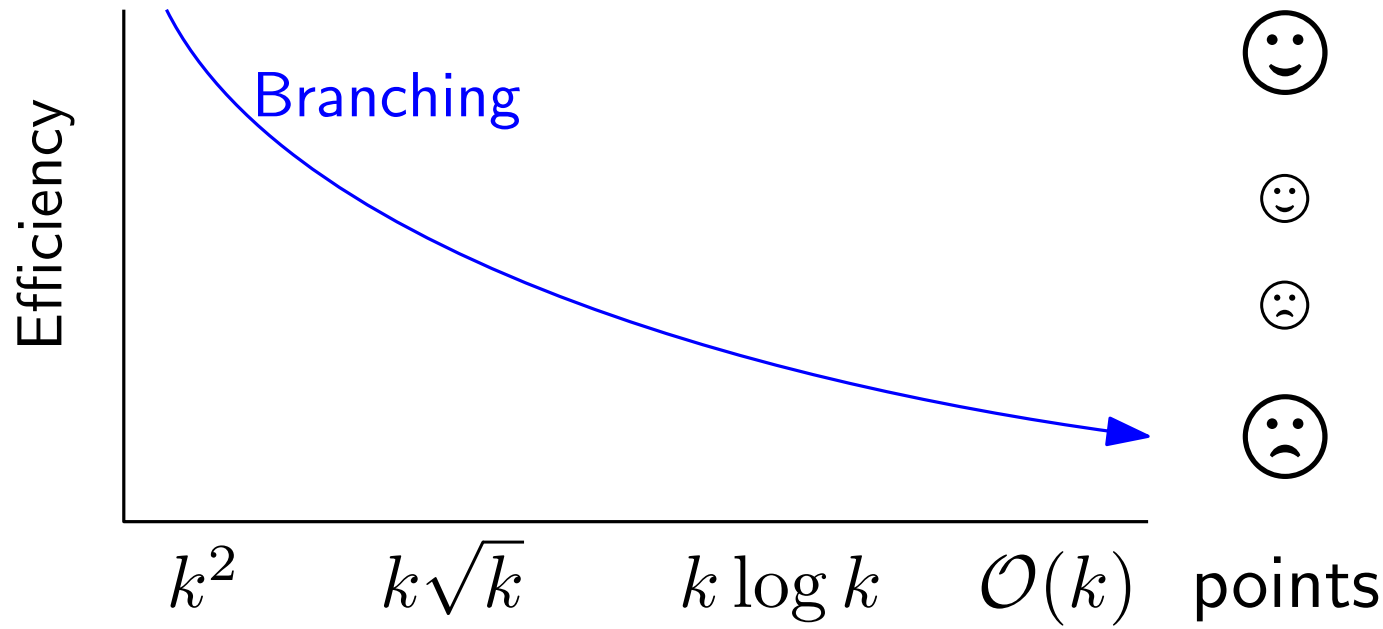
But to cover P' with k' *poor* lines, $|P'|$ must be low (relative to k')! Switch to non-parametrized algorithm.

Make progress towards small P' even if k_i low or 0: no *usable* γ_i-rich candidate, $|P'| \leq k'\gamma_i$ (same as kernel)

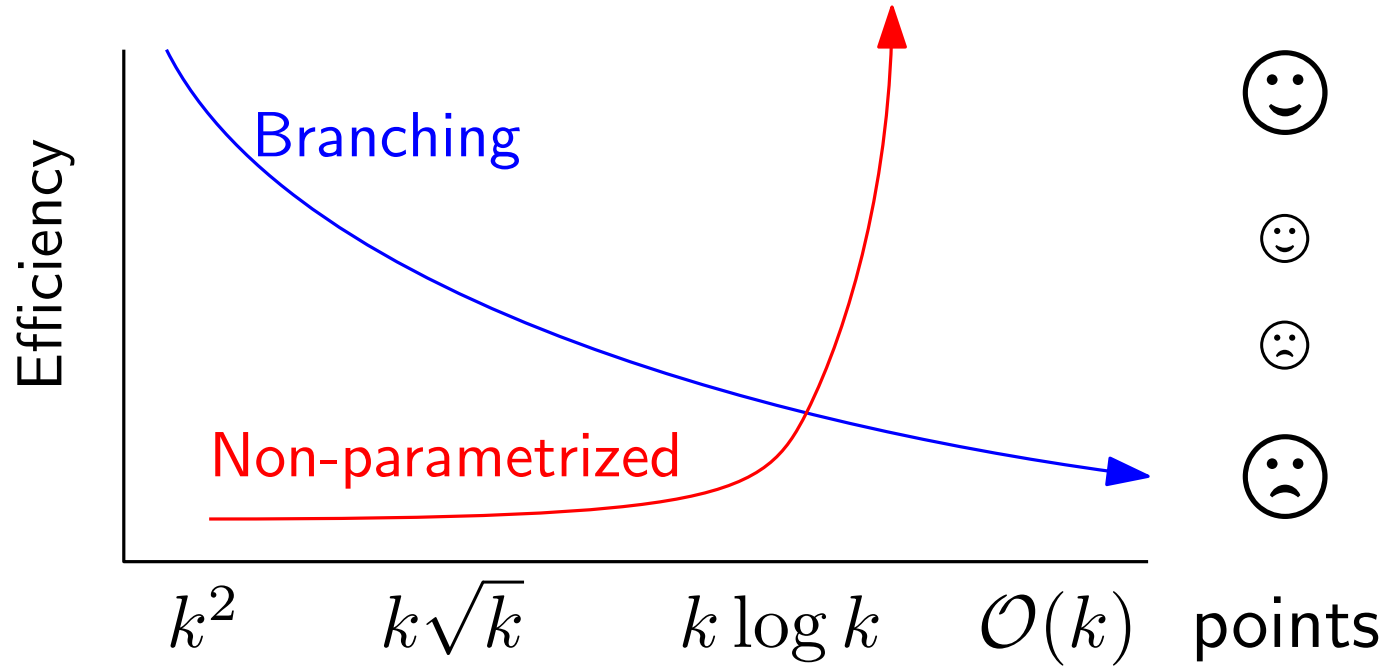
Constructing our algorithm



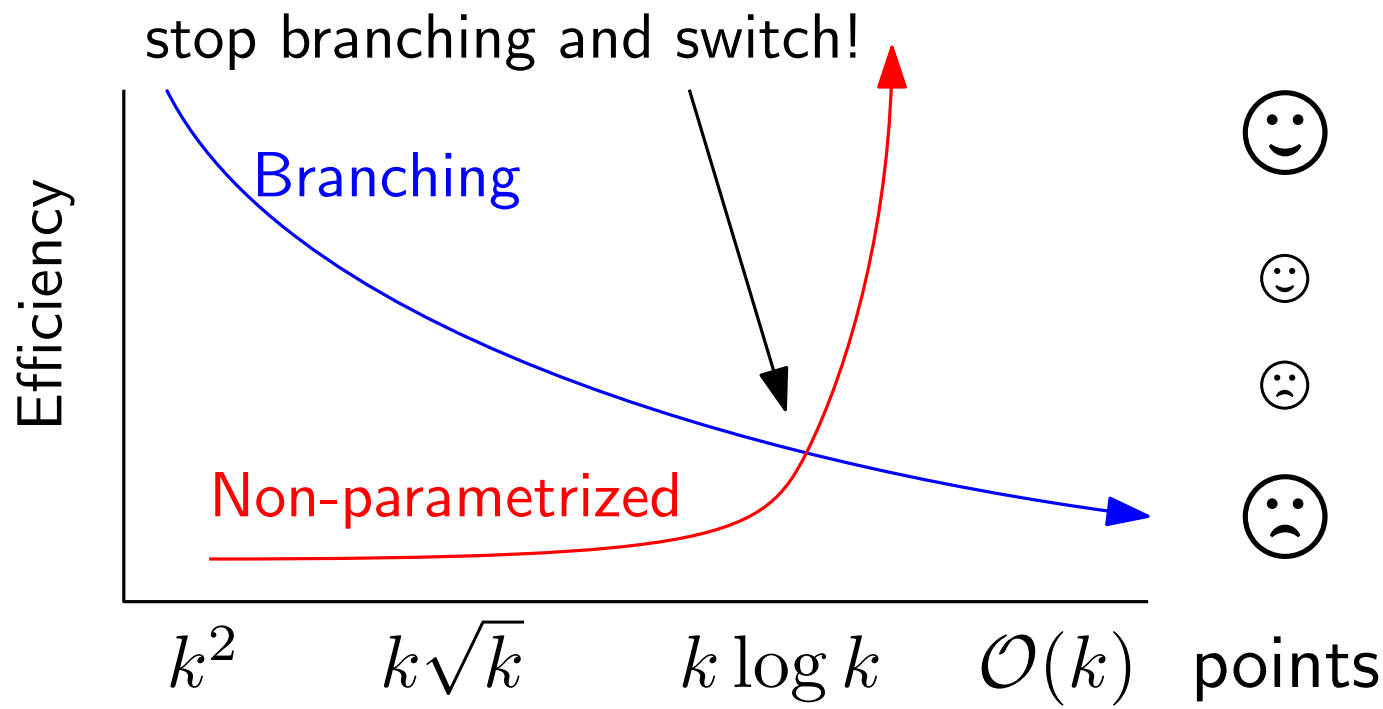
Constructing our algorithm



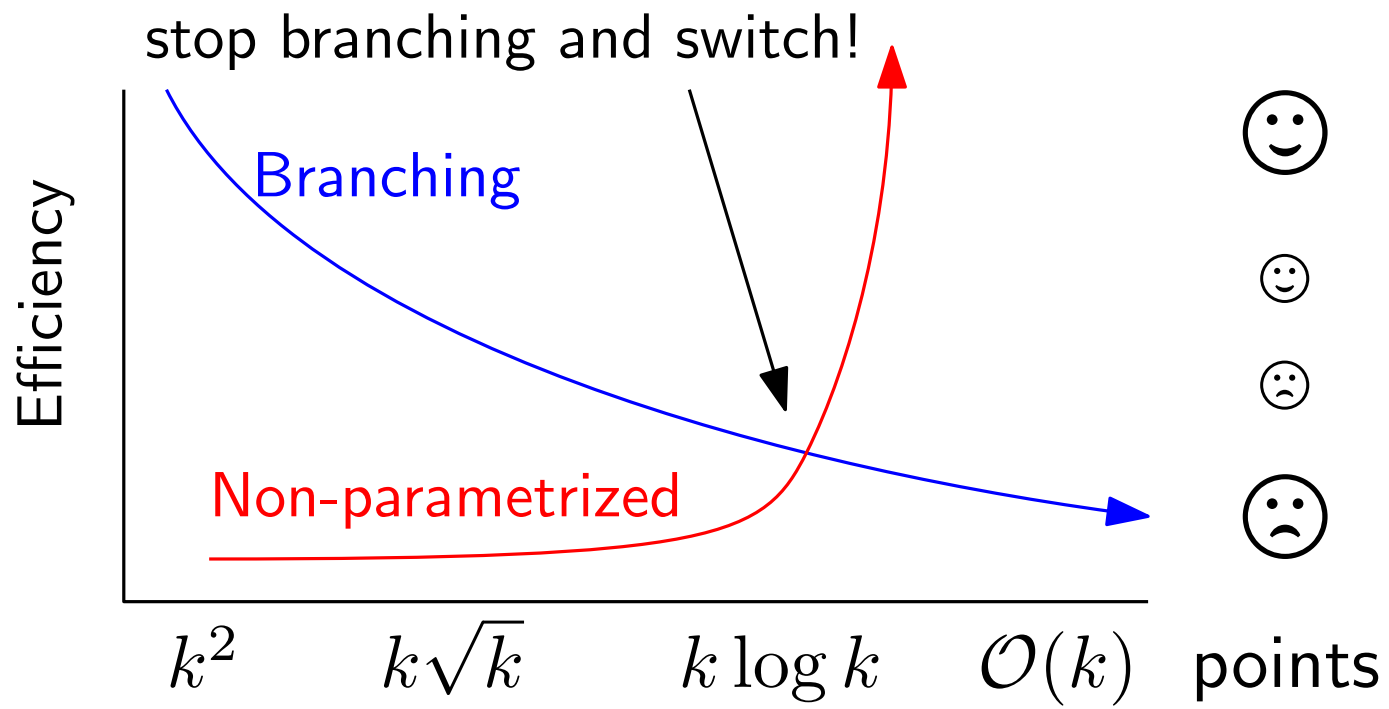
Constructing our algorithm



Constructing our algorithm

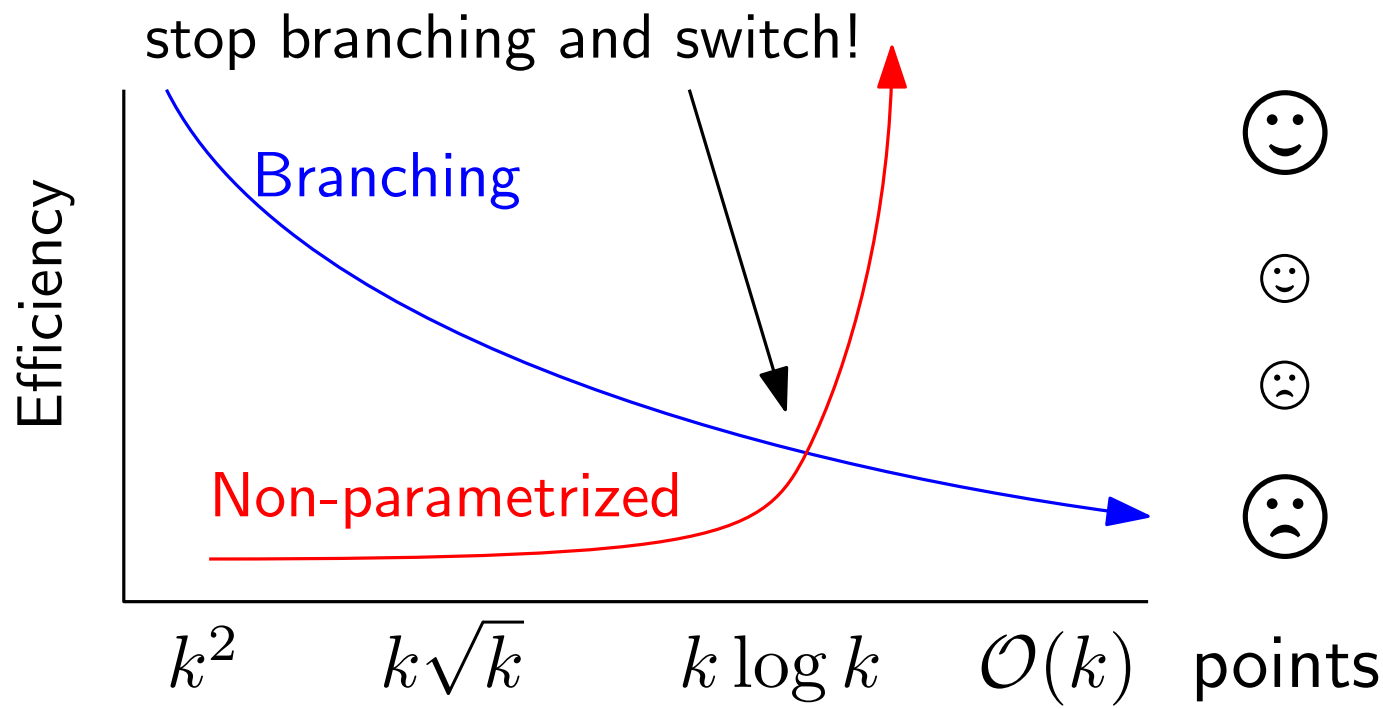


Constructing our algorithm



Total running time $=$ # leaves $\times$ $2^{\text{(problem size at switch)}}$

Constructing our algorithm



Total running time $=$ # leaves $\times$ $2^{\text{(problem size at switch)}}$
$= \mathcal{O}((ck/\log k)^k)$ for some constant c.

Curve Cover

Curve Cover

$\mathcal{C}$ is a family of (d, s)-curves if:

Curve Cover

$\mathcal{C}$ is a family of (d, s)-curves if:

1. Two distinct curves intersect in at most s points.
2. For any d points at most s curves pass through them.

Curve Cover

$\mathcal{C}$ is a family of (d, s)-curves if:

1. Two distinct curves intersect in at most s points.
2. For any d points at most s curves pass through them.

d degrees of freedom, multiplicity-type s.

Curve Cover

$\mathcal{C}$ is a family of (d, s)-curves if:

1. Two distinct curves intersect in at most s points.
2. For any d points at most s curves pass through them.

d degrees of freedom, multiplicity-type s.

Lines are (2,1)-curves.

Curve Cover

$\mathcal{C}$ is a family of (d, s)-curves if:

1. Two distinct curves intersect in at most s points.
2. For any d points at most s curves pass through them.

d degrees of freedom, multiplicity-type s.

Lines are (2,1)-curves.

Unit circles are (2,2)-curves in $\mathbb{R}^2$ but (3,2)-curves in $\mathbb{R}^3$.

Curve Cover

$\mathcal{C}$ is a family of (d, s)-curves if:

1. Two distinct curves intersect in at most s points.
2. For any d points at most s curves pass through them.

d degrees of freedom, multiplicity-type s.

Lines are (2,1)-curves.

Unit circles are (2,2)-curves in $\mathbb{R}^2$ but (3,2)-curves in $\mathbb{R}^3$.

Degree b polynomials are $(b+1, b)$-curves.

Curve Cover

$\mathcal{C}$ is a family of (d, s)-curves if:

1. Two distinct curves intersect in at most s points.
2. For any d points at most s curves pass through them.

d degrees of freedom, multiplicity-type s.

Lines are (2,1)-curves.

Unit circles are (2,2)-curves in $\mathbb{R}^2$ but (3,2)-curves in $\mathbb{R}^3$.

Degree b polynomials are $(b+1,b)$-curves.

Sine waves are not (d, s)-curves.

Curve Cover

Given n points P and family $\mathcal{C}$ of (d, s)-curves.

Pach&Sharir '98: # γ-rich candidates is $\mathcal{O}\left(\frac{n^d}{\gamma^{2d-1}} + \frac{n}{\gamma}\right)$.

Kernel: $|P| \leq sk^2$, no $(sk+1)$-rich candidate.

Curve Cover

Given n points P and family $\mathcal{C}$ of (d, s)-curves.

Pach&Sharir '98: # γ-rich candidates is $\mathcal{O}\left(\frac{n^d}{\gamma^{2d-1}} + \frac{n}{\gamma}\right)$.

Kernel: $|P| \leq sk^2$, no $(sk+1)$-rich candidate.

$\Rightarrow \mathcal{O}((ck/\log k)^{(d-1)k})$ time algorithm. c depends on d, s.

Curve Cover

Given n points P and family $\mathcal{C}$ of (d, s)-curves.

Pach&Sharir '98: # γ-rich candidates is $\mathcal{O}\left(\frac{n^d}{\gamma^{2d-1}} + \frac{n}{\gamma}\right)$.

Kernel: $|P| \leq sk^2$, no $(sk+1)$-rich candidate.

$\Rightarrow \mathcal{O}((ck/\log k)^{(d-1)k})$ time algorithm. c depends on d, s.

Beats previous bests:

- $\mathcal{O}((k/1.35)^{(d-1)k})$ for lines $(d = 2)$, Wang et al. '10
- $\mathcal{O}((k/1.38)^{(d-1)k})$ for conics $(d = 5)$, Tiwari '12
- $\mathcal{O}((k/1.15)^{(d-1)k})$ for parabolas $(d = 4)$, Tiwari '12
- $\mathcal{O}(k^{dk})$ for general (d, s)-curves, Langerman&Morin '05

Hyperplane Cover

Agarwal&Aronov '92: $\mathcal{O}\left(\frac{n^d}{\gamma^3}\right)$ candidates in $\mathbb{R}^d$.

Hyperplane Cover

Agarwal&Aronov '92: $\mathcal{O}\left(\frac{n^d}{\gamma^3}\right)$ candidates in $\mathbb{R}^d$.

Tight for general point sets and $d \geq 2$.

Hyperplane Cover

Agarwal&Aronov '92: $\mathcal{O}\left(\frac{n^{d}}{\gamma^{3}}\right)$ candidates in $\mathbb{R}^d$.

Tight for general point sets and $d \geq 2$.

Unusable for $d \geq 3$; need denominator $>$ numerator.

Hyperplane Cover

Agarwal&Aronov '92: $\mathcal{O}\left(\frac{n^{d}}{\gamma^{3}}\right)$ candidates in $\mathbb{R}^d$.

Tight for general point sets and $d \geq 2$.

Unusable for $d \geq 3$; need denominator $>$ numerator.

Worst-case constructions put very many pts on same line.

Algorithmically *easy* thanks to kernelization.

Hyperplane Cover

Agarwal&Aronov '92: $\mathcal{O}\left(\frac{n^{d}}{\gamma^{3}}\right)$ candidates in $\mathbb{R}^d$.

Tight for general point sets and $d \geq 2$.

Unusable for $d \geq 3$; need denominator $>$ numerator.

Worst-case constructions put very many pts on same line.

Algorithmically *easy* thanks to kernelization.

Need *specialized* incidence bound.

Hyperplane Cover

Agarwal&Aronov '92: $\mathcal{O}\left(\frac{n^{d}}{\gamma^{3}}\right)$ candidates in $\mathbb{R}^d$.

Tight for general point sets and $d \geq 2$.

Unusable for $d \geq 3$; need denominator $>$ numerator.

Worst-case constructions put very many pts on same line.

Algorithmically *easy* thanks to kernelization.

Need *specialized* incidence bound.

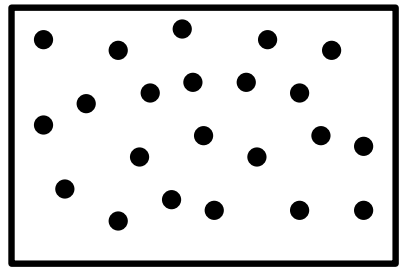
- Wait and hope for bounds on kernelized instances.
- Use other specialized bounds that already exist.

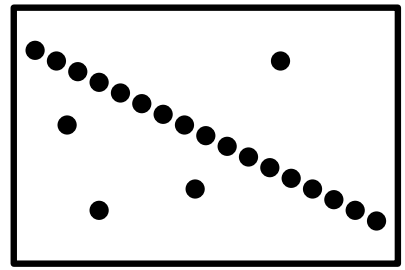
Plane Cover ($\mathbb{R}^3$)

Plane is δ-*degenerate* if $\leq \delta$ fraction of its pts on a line.

Plane Cover ($\mathbb{R}^3$)

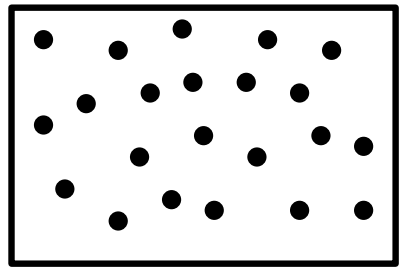
Plane is δ-*degenerate* if $\leq \delta$ fraction of its pts on a line.

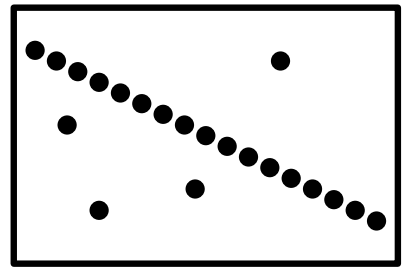
$\delta < 1/3$ 

$\delta > 4/5$ 

Plane Cover ($\mathbb{R}^3$)

Plane is δ-*degenerate* if $\leq \delta$ fraction of its pts on a line.

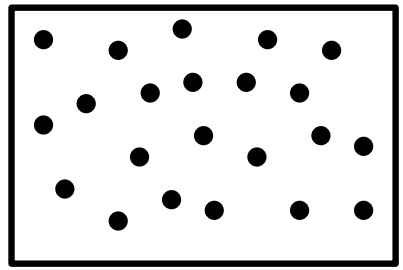
$\delta < 1/3$ 

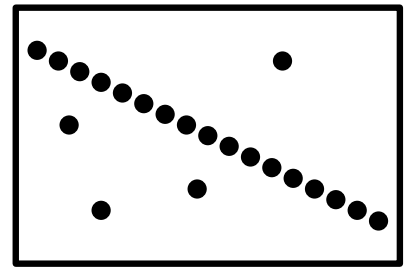
$\delta > 4/5$ 

Elekes&Tóth '05: $\mathcal{O}\left(\frac{1}{(1-\delta)^4} \cdot \frac{n^3}{\gamma^4}\right)$ γ-rich δ-deg candidates.

Plane Cover ($\mathbb{R}^3$)

Plane is δ-*degenerate* if $\leq \delta$ fraction of its pts on a line.

$\delta < 1/3$ 

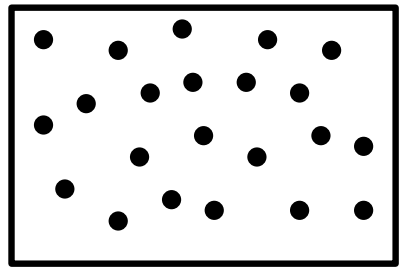
$\delta > 4/5$ 

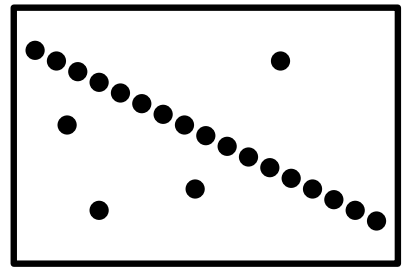
Elekes&Tóth '05: $\mathcal{O}\left(\frac{1}{(1-\delta)^4} \cdot \frac{n^3}{\gamma^4}\right)$ γ-rich δ-deg candidates.

If all candidates have low δ (e.g. $\leq 1/2$), no problem!

Plane Cover ($\mathbb{R}^3$)

Plane is δ-*degenerate* if $\leq \delta$ fraction of its pts on a line.

$\delta < 1/3$ 

$\delta > 4/5$ 

Elekes&Tóth '05: $\mathcal{O}\left(\frac{1}{(1-\delta)^4} \cdot \frac{n^3}{\gamma^4}\right)$ γ-rich δ-deg candidates.

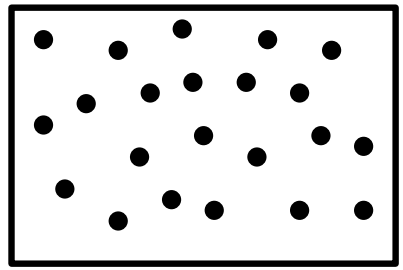
If all candidates have low δ (e.g. $\leq 1/2$), no problem!

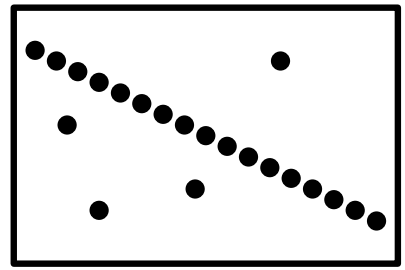
But if P contains high δ candidates, the solution might too.

Deal with these in a different way.

Dealing with too-degenerate plane

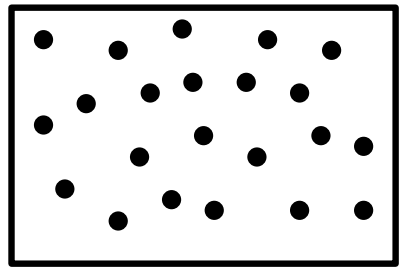
Plane is δ-*degenerate* if $\leq \delta$ fraction of its pts on a line.

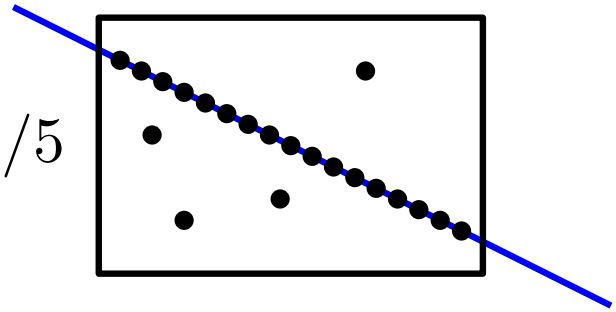
$\delta < 1/3$ 

$\delta > 4/5$ 

Dealing with too-degenerate plane

Plane is δ-*degenerate* if $\leq \delta$ fraction of its pts on a line.

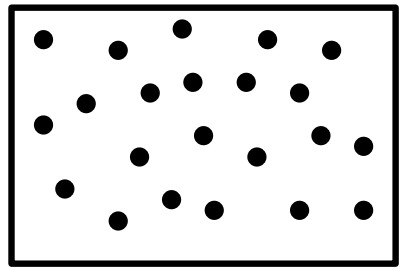
$\delta < 1/3$ 

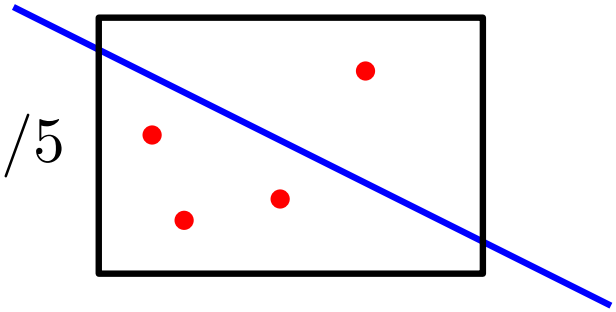
$\delta > 4/5$ 

For $\delta > 1/2$, *most* points on a line (the *degenerate line* ℓ).

Dealing with too-degenerate plane

Plane is δ-*degenerate* if $\leq \delta$ fraction of its pts on a line.

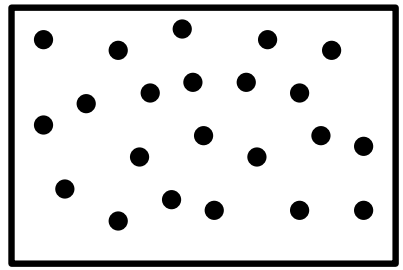
$\delta < 1/3$ 

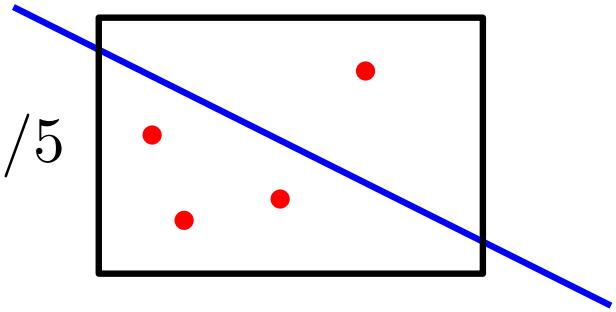
$\delta > 4/5$ 

For $\delta > 1/2$, *most* points on a line (the *degenerate line* ℓ).
Removing only the points on ℓ is "almost as good".
Leaves *ghost points*, but *few* of them.

Dealing with too-degenerate plane

Plane is δ-*degenerate* if $\leq \delta$ fraction of its pts on a line.

$\delta < 1/3$ 

$\delta > 4/5$ 

For $\delta > 1/2$, *most* points on a line (the *degenerate line* ℓ).

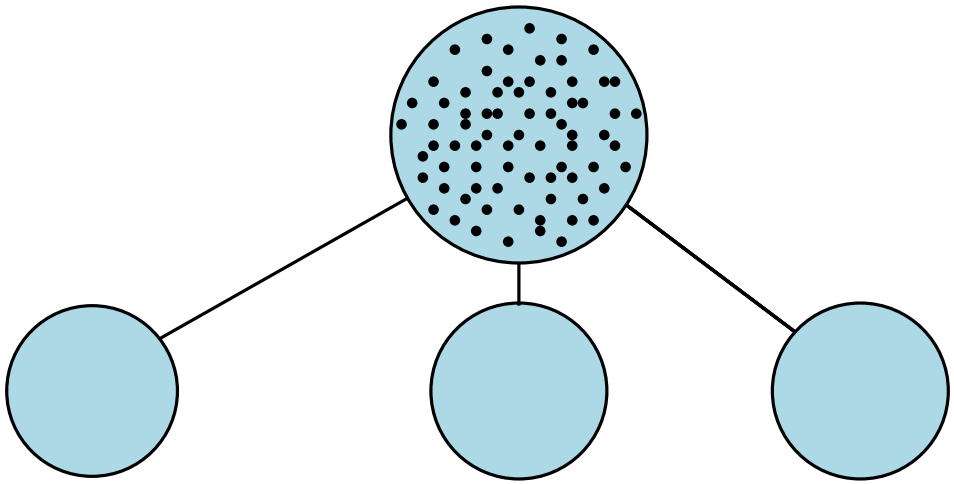
Removing only the points on ℓ is "almost as good".

Leaves *ghost points*, but *few* of them.

We can postpone dealing with them.

Dealing with too-degenerate plane

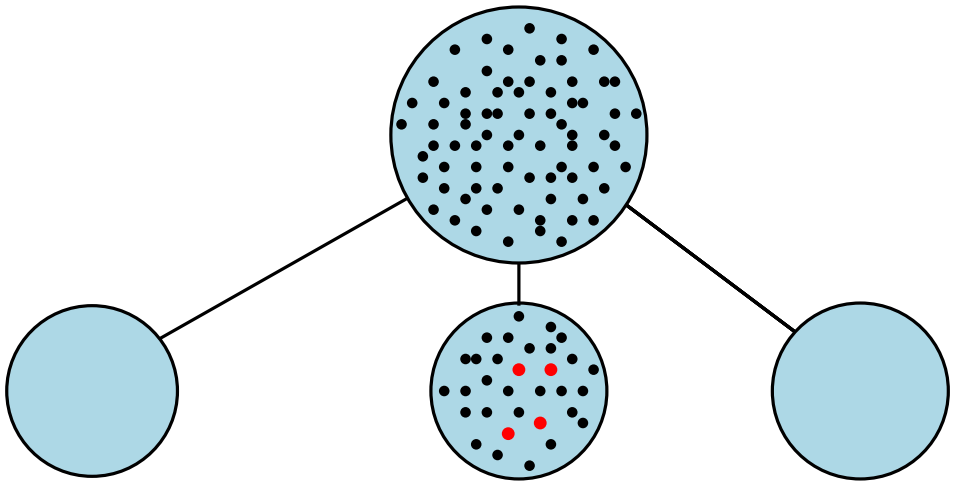
Dealing with too-degenerate plane



$k_i = l_i + h_i$

branch $\binom{\text{few}}{l_1}$ ways among $\gamma_i\delta_i$-rich lines, remember ℓ

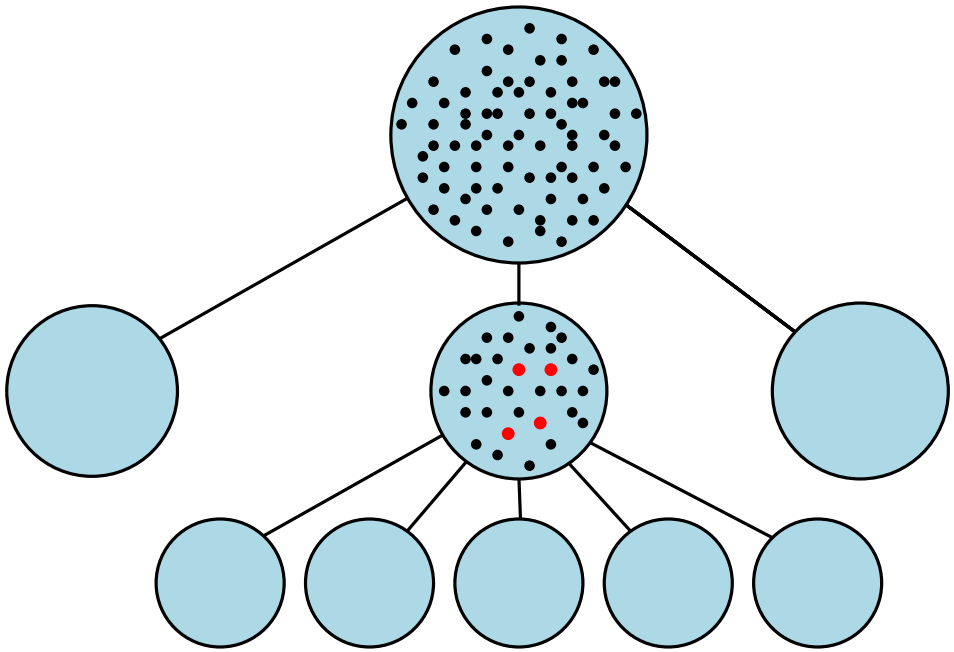
Dealing with too-degenerate plane



$k_i = l_i + h_i$

branch $\binom{\text{few}}{l_1}$ ways among $\gamma_i\delta_i$-rich lines, remember ℓ

Dealing with too-degenerate plane

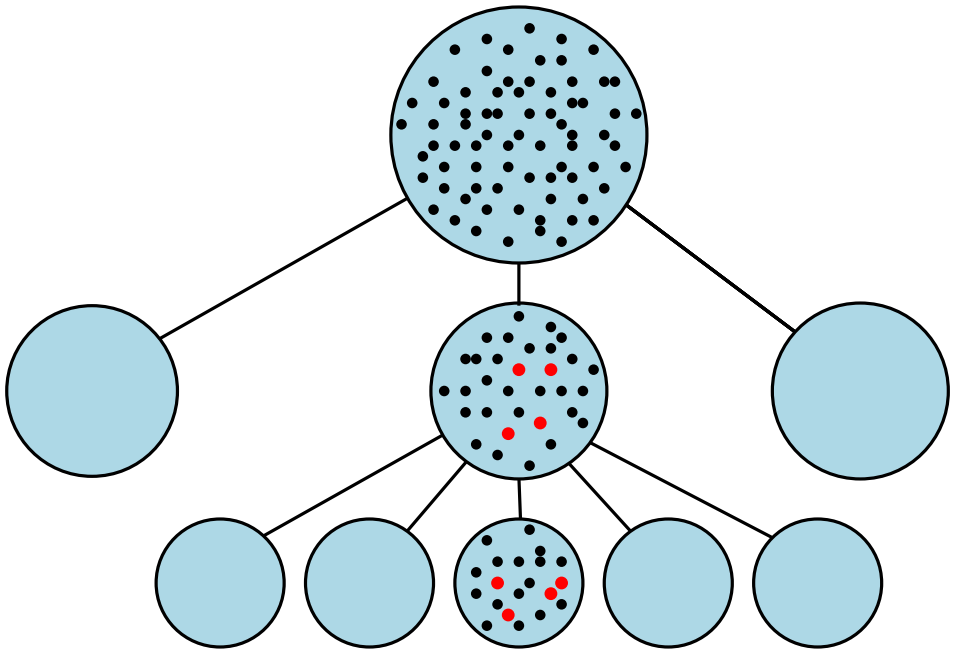


$k_i = l_i + h_i$

branch $\binom{\text{few}}{l_1}$ ways among $\gamma_i\delta_i$-rich lines, remember ℓ

branch $\binom{\text{few}}{h_1}$ ways among γ_i-rich δ_i-degenerate planes

Dealing with too-degenerate plane

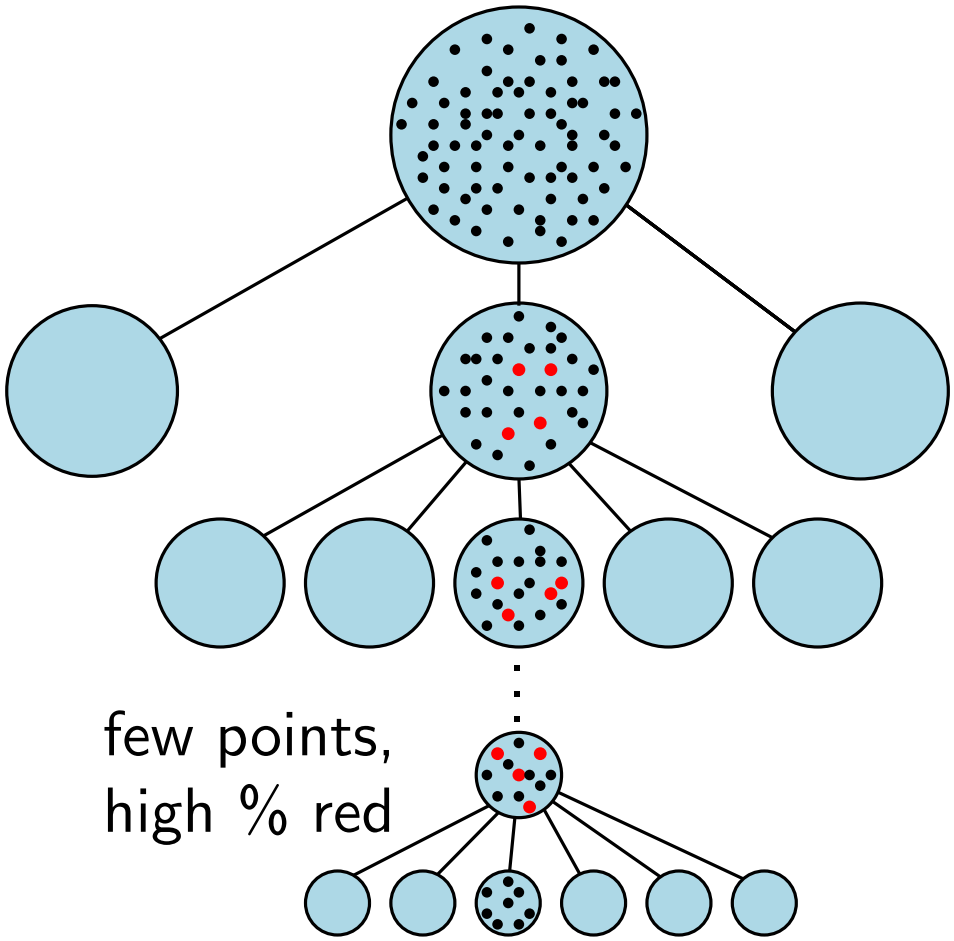


$k_i = l_i + h_i$

branch $\binom{\text{few}}{l_1}$ ways among $\gamma_i\delta_i$-rich lines, remember ℓ

branch $\binom{\text{few}}{h_1}$ ways among γ_i-rich δ_i-degenerate planes

Dealing with too-degenerate plane



few points,
high % red

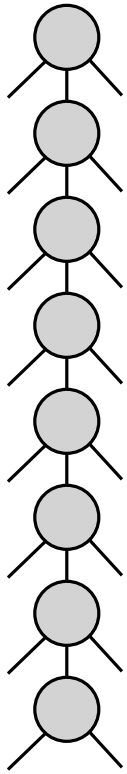
$k_i = l_i + h_i$

branch $\binom{\text{few}}{l_1}$ ways among $\gamma_i\delta_i$-rich lines, remember ℓ

branch $\binom{\text{few}}{h_1}$ ways among γ_i-rich δ_i-degenerate planes

pair l_i lines ℓ with points (forming planes)

Dealing with too-degenerate plane



l_1 lines

h_1 planes

l_2 lines

h_2 planes

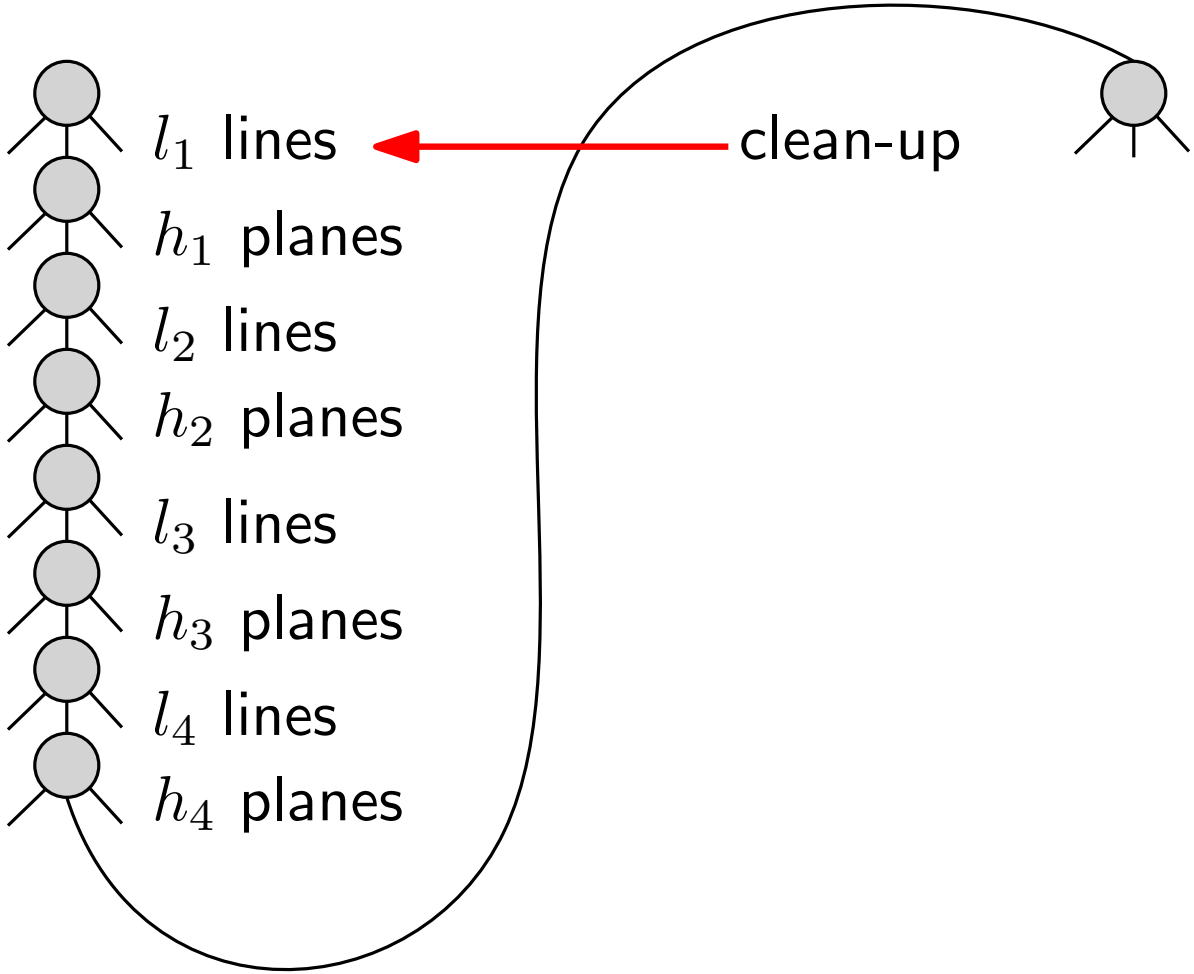
l_3 lines

h_3 planes

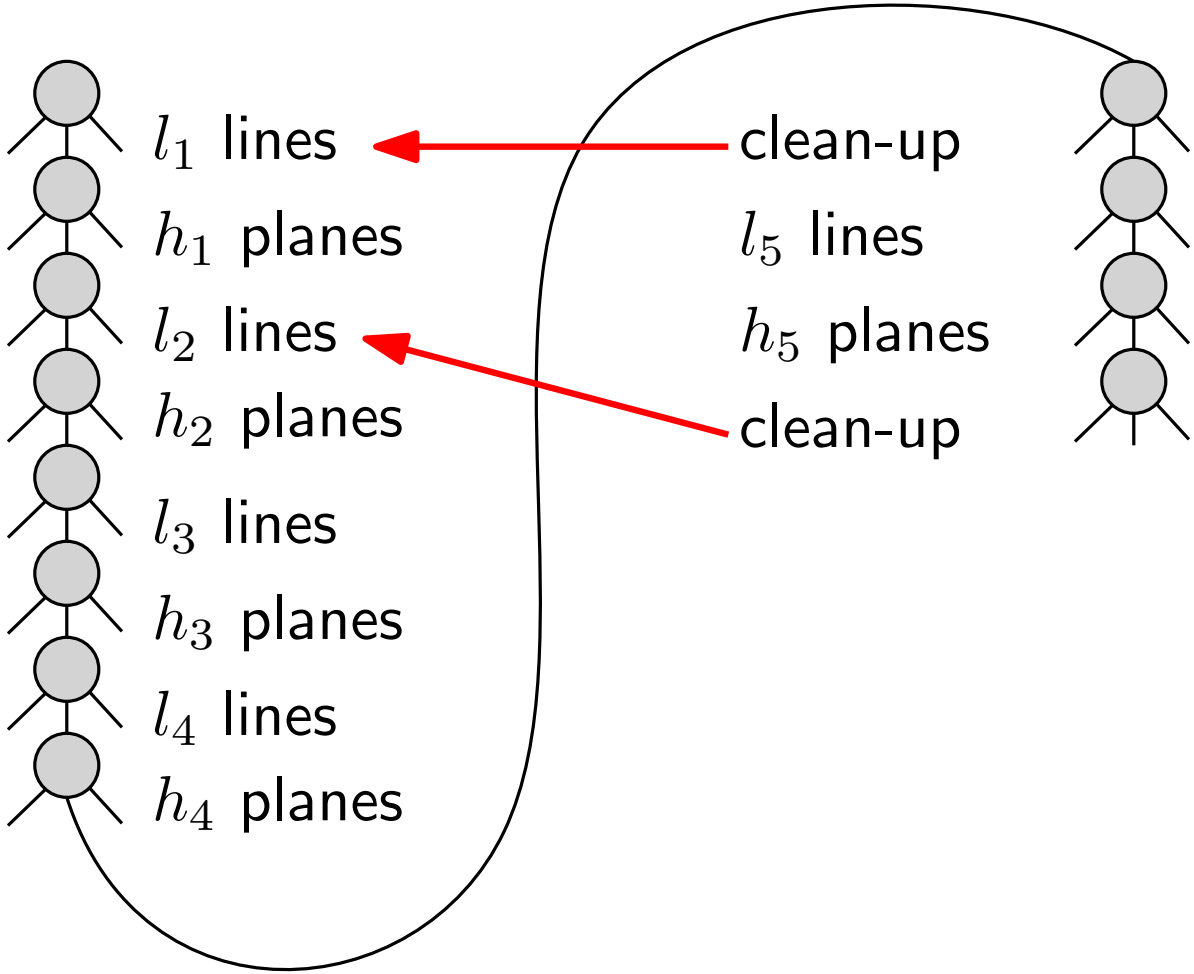
l_4 lines

h_4 planes

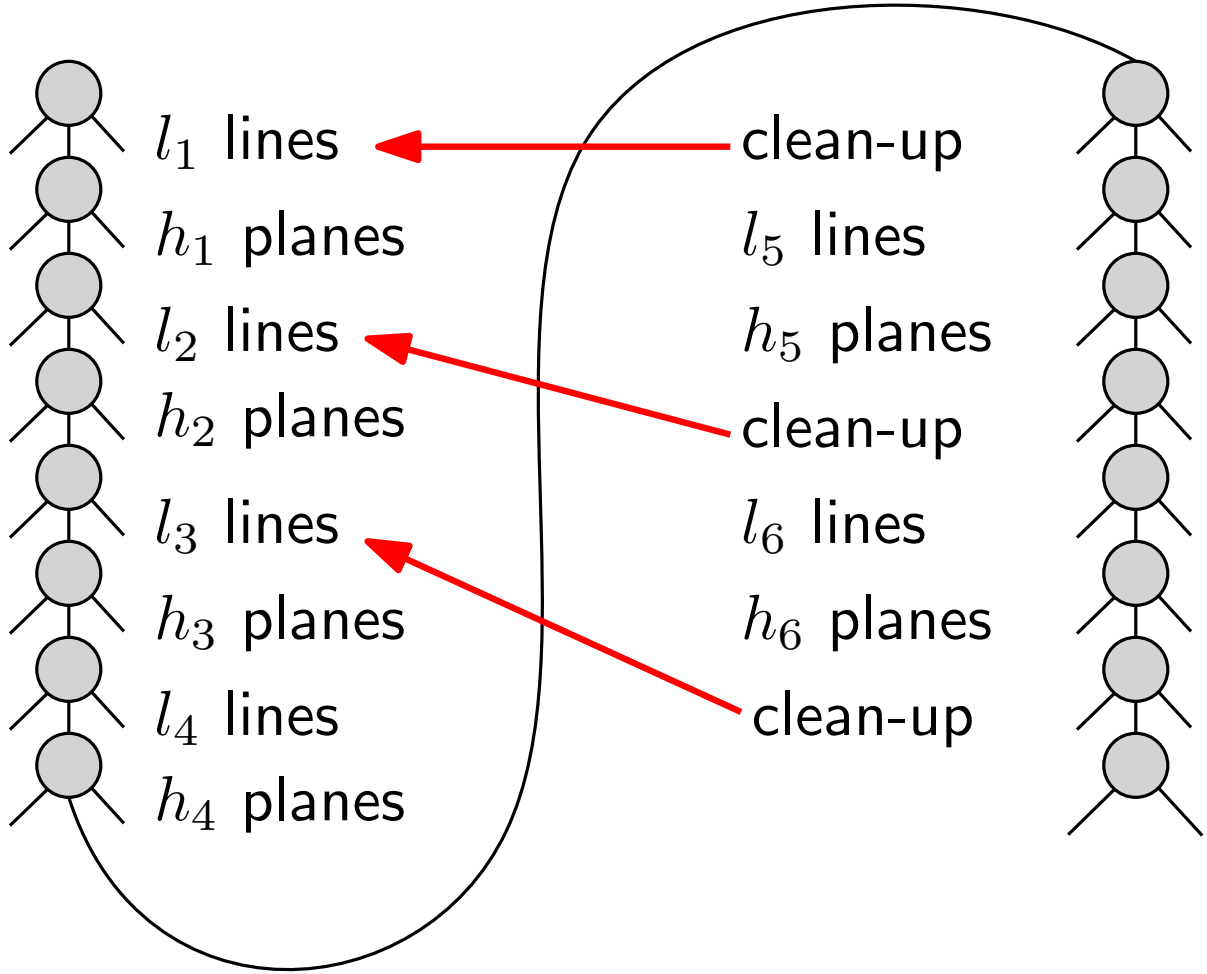
Dealing with too-degenerate plane



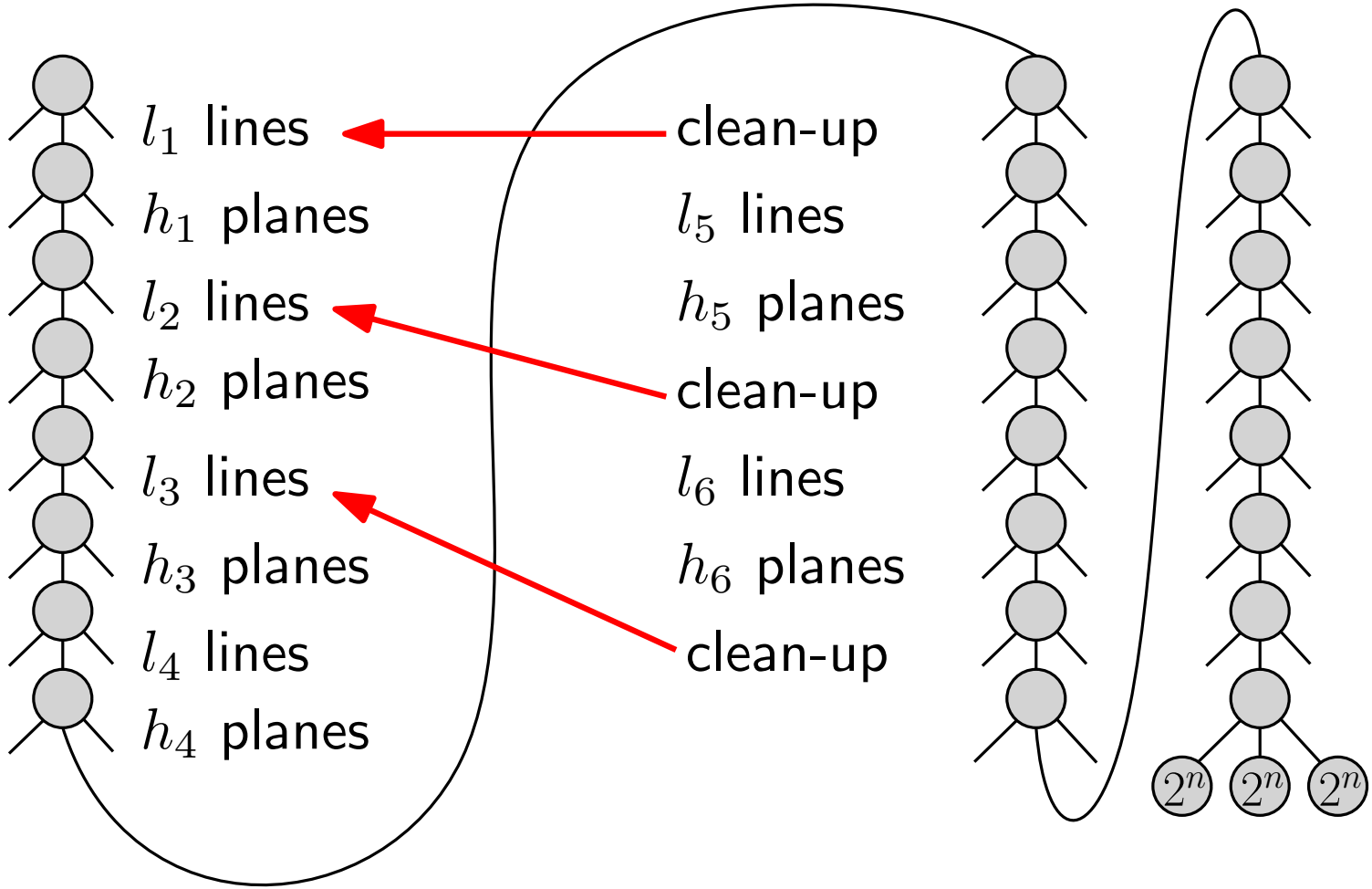
Dealing with too-degenerate plane



Dealing with too-degenerate plane



Dealing with too-degenerate plane



Wrapping

Wrapping

Degeneracy $\delta_i = 1 - \frac{1}{\gamma_i^{1/5}} \Rightarrow \frac{1}{(1-\delta_i)^4} = \frac{1}{\gamma_i^{-4/5}}$.

Wrapping

Degeneracy $\delta_i = 1 - \frac{1}{\gamma_i^{1/5}} \Rightarrow \frac{1}{(1-\delta_i)^4} = \frac{1}{\gamma_i^{-4/5}}$.

$\mathcal{O}\left(\frac{1}{(1-\delta_i)^4}\frac{n^3}{\gamma_i^4}\right) = \mathcal{O}\left(\frac{n^3}{\gamma_i^{4-4/5}}\right) = \mathcal{O}\left(\frac{n^3}{\gamma_i^{3+1/5}}\right)$ candidates.

Wrapping

Degeneracy $\delta_i = 1 - \frac{1}{\gamma_i^{1/5}} \Rightarrow \frac{1}{(1-\delta_i)^4} = \frac{1}{\gamma_i^{-4/5}}$.

$\mathcal{O}\left(\frac{1}{(1-\delta_i)^4}\frac{n^3}{\gamma_i^4}\right) = \mathcal{O}\left(\frac{n^3}{\gamma_i^{4-4/5}}\right) = \mathcal{O}\left(\frac{n^3}{\gamma_i^{3+1/5}}\right)$ candidates.

$\Rightarrow \mathcal{O}\left(\left(\frac{ck^2}{\log^{1/5} k}\right)^k\right)$ running time.

Wrapping

Degeneracy $\delta_i = 1 - \frac{1}{\gamma_i^{1/5}} \Rightarrow \frac{1}{(1-\delta_i)^4} = \frac{1}{\gamma_i^{-4/5}}$.

$\mathcal{O}\left(\frac{1}{(1-\delta_i)^4}\frac{n^3}{\gamma_i^4}\right) = \mathcal{O}\left(\frac{n^3}{\gamma_i^{4-4/5}}\right) = \mathcal{O}\left(\frac{n^3}{\gamma_i^{3+1/5}}\right)$ candidates.

$\Rightarrow \mathcal{O}\left(\left(\frac{ck^2}{\log^{1/5} k}\right)^k\right)$ running time.

Beats $\mathcal{O}\left(\left(\frac{k^{(d-1)}}{1.3}\right)^k\right)$ by Wang et al. '10, when $d = 3$ ($\mathbb{R}^3$).

Special incidence bound does not apply for $d > 3$.