Core-sets for Nano-Drones or:

The Fall and Rise of Computational Geometry

אוניברסיטת חיפה
University of Haifa

Dan Feldman & the Robotics & Big Data Lab

Based on NIPS'19 Papers:
with Y. Marom,
with A. Malouf & I. Jubran (+Oral)

Army develops new drone-killing technology
(6 hours ago)

Drone protests threaten UK's Heathrow Airport with more flight chaos
(10 hours ago)

IDF drone crashes in Gaza; Palestinians claim they shot it down
(15 hours ago)

IDF DENIES HEZBOLLAH SHOT DOWN SURVEILLANCE DRONE
(25 hours ago)

Gas cloud imaging, drones to monitor methane emissions
(30 hours ago)

Syrian army foils drone attack on military base in northwest
(40 hours ago)

But where are the Nano-drones?

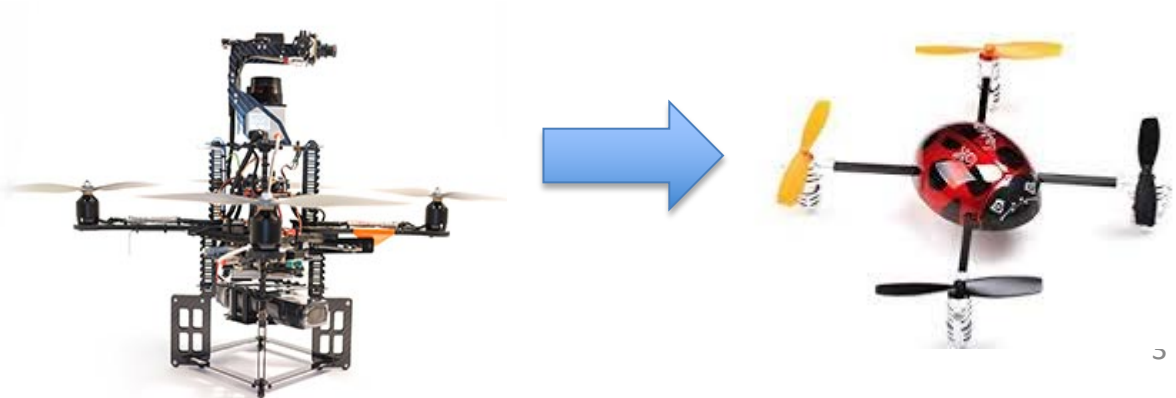
Guiding Drone
(MIT Senseable
City Lab, 2013)

Main Challenge: Autonomous Navigation

- No GPS for indoor navigation
- Law → Low:
  weight→ payload→
  computation power
- Real real-time computations
- Big data with respect to time

Wearable Drone (Intel, 2015)

Strong algorithms for weak hardware



Autonomous Navigation

Amnon Shashua: April 23, 2019, MIT Reviews, by Karen Hao

- Problem: “three-orders-of-magnitude gap”
- Hardware: “create redundancies…using radar & lidar”
- Data: Make “highly detailed map”

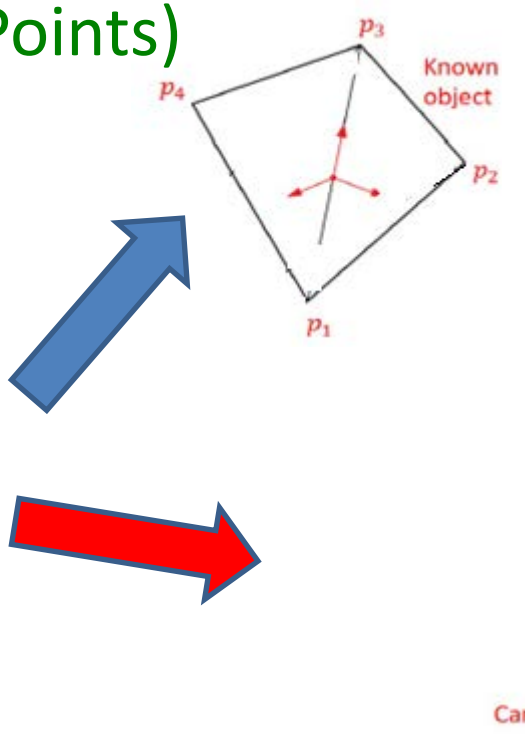
Maybe we just need better software (algorithms)?

- The words “proof” and “theorem” are very rare in Computer Vision top conferences/books/classes
- The less we understand the more it works (e.g. deep learning)

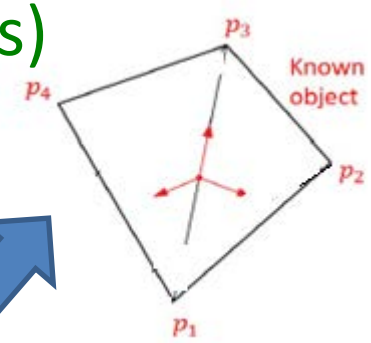
Localization (Perspective-n-Points)

Input:

- A set of points $P = \{p_1, \dots, p_n\}$
- A set of lines $L = \{\ell_1, \dots, \ell_n\}$

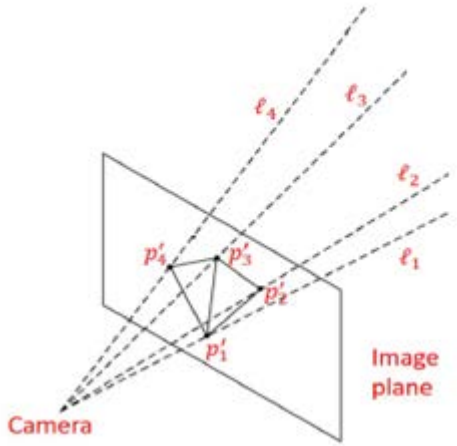


Localization (Perspective-n-Points)



Input:

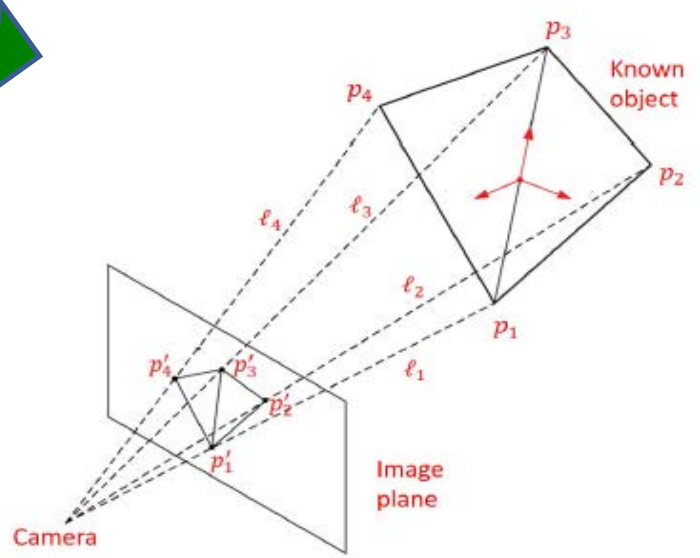
- A set of points $P = \{p_1, \dots, p_n\}$
- A set of lines $L = \{\ell_1, \dots, \ell_n\}$



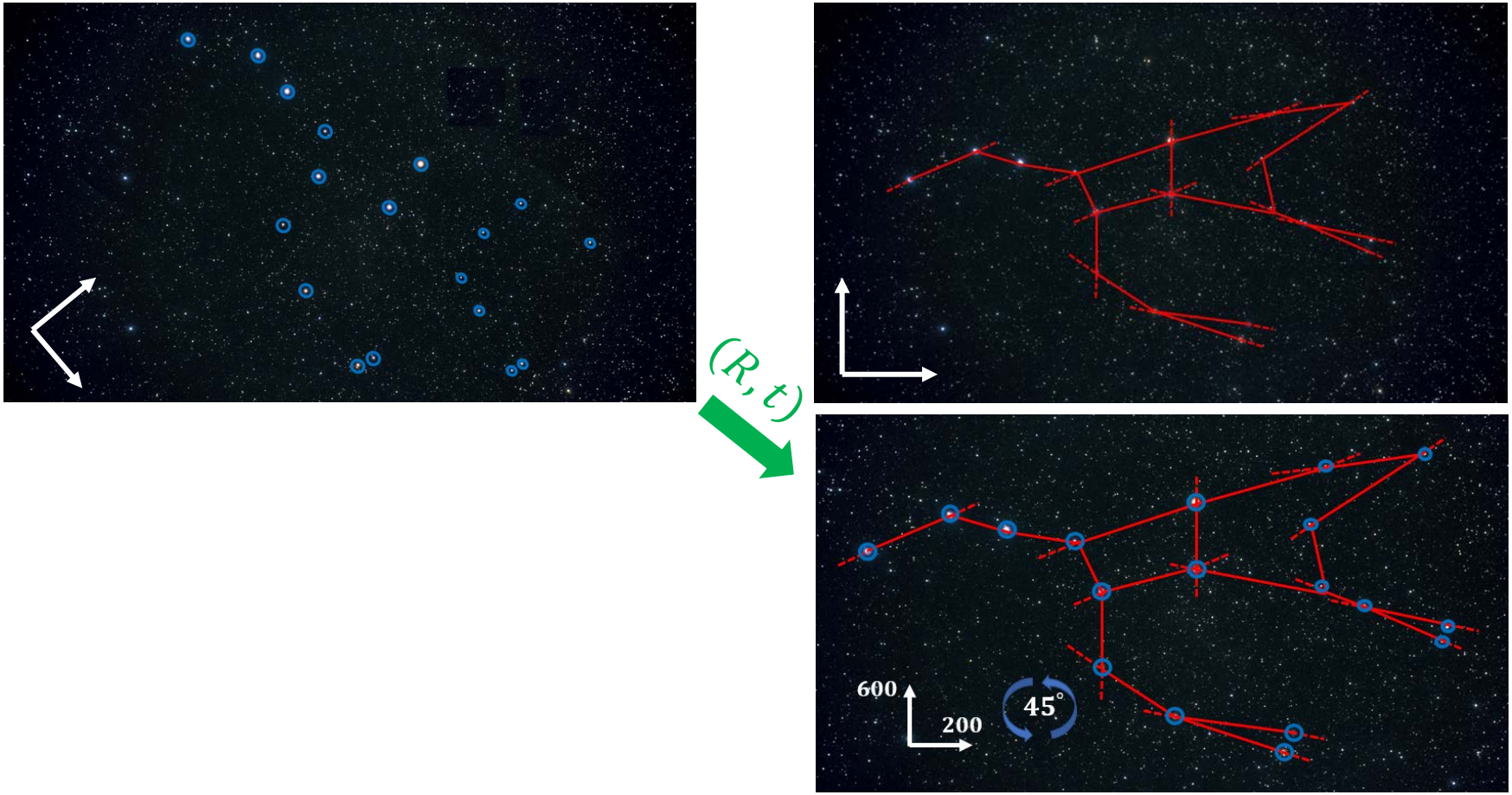
Output:

An alignment rotation+translation (R, t) that minimizes

$$\sum_{i \in [n]} \text{dist}^2(Rp_i - t, \ell_i)$$



2D case: Navigation at night



PhD Thesis, MIT:

July, 1993

On Geometric and Algebraic Aspects of 3D Affine and Projective Structures from Perspective 2D Views

Amnon Shashua

What went wrong?

Good news: PnP can be solved **exactly** (global optimum) in *O(n)* time

Easy reduction in 1 second for $n = 1{,}000{,}000$ and a laptop to:

Minimize $||Ax + By + Cz||$ over mutually orthogonal x, y, z where $A, B, C \in \mathbb{R}^{3\times 3}$

How much time in practice to solve this *O(1)* sized problem?

PhD Thesis, MIT:

July, 1993

On Geometric and Algebraic Aspects of 3D Affine and Projective Structures from Perspective 2D Views

Amnon Shashua

What went wrong?

Good news: PnP can be solved **exactly** (global optimum) in *O(n)* time

- Easy reduction in 1 second for $n = 1{,}000{,}000$ and a laptop to:

Minimize $||Ax + By + Cz||$ over mutually orthogonal x, y, z where $A, B, C \in \mathbb{R}^{3\times3}$

How much time in practice to solve this O(1) sized problem?

- **Bad news:** Via Mathematica & Maple - still running after 1 Month.
- ➔ Exact non-convex optimization is hard
- even for low degree polynomial of 3-variables and quadratic constraints
- NMinimize or other heuristics – a second but x1000 worse single local optimum

- Suggested approach: Coreset + Provable APPROXIMATIONS

Theorem [Jubran] (MSc thesis) :
There is a subset of $O(1)$ points such that solving the PnP on this set would give exact solution to the PnP of the original data.

Theorem [Malouf & Jubran] (NIPS'19, Oral) : This set can be computed in $O(n)$ time.
Point insertion takes $O(1)$ time.

Motivation

“Cormen” usually not support:

- Big Data
- Streaming real-time data
- Distributed data
- Real time computations

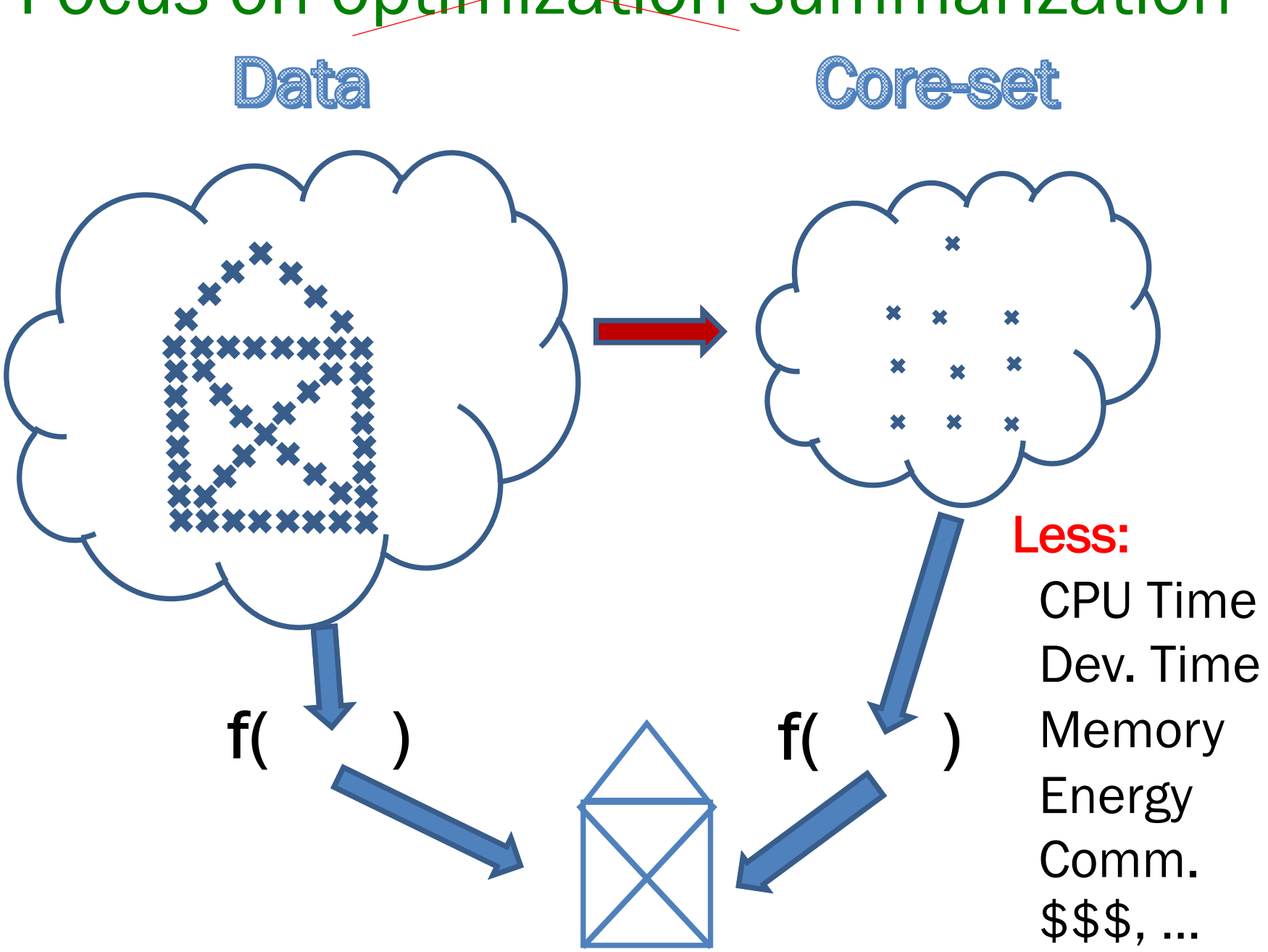
Limited hardware & energy

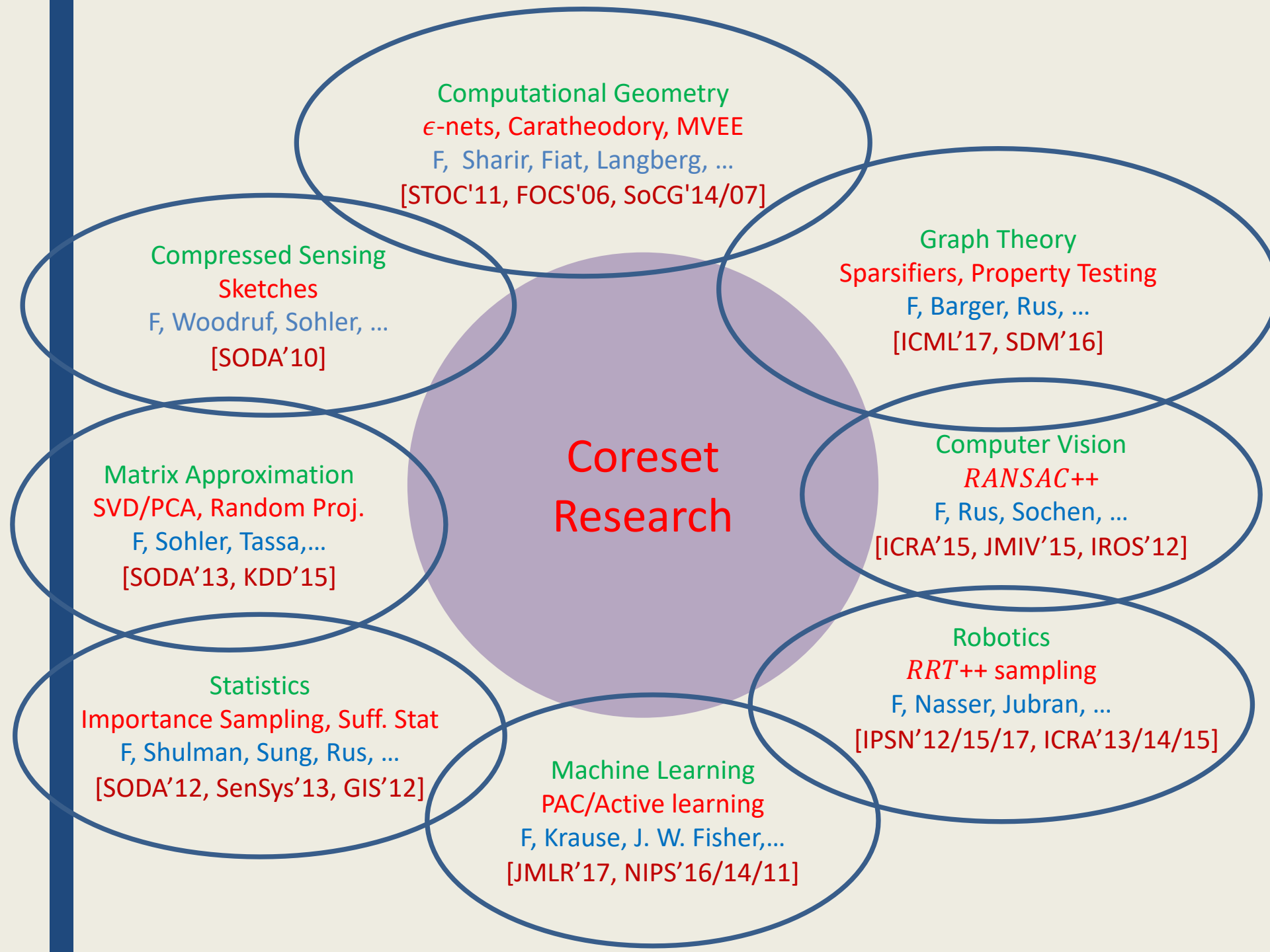
- Smart, IoT, GPU, Cars
- iRobots, drones, cars

Common solution

- New optimization algorithms

Focus on optimization summarization



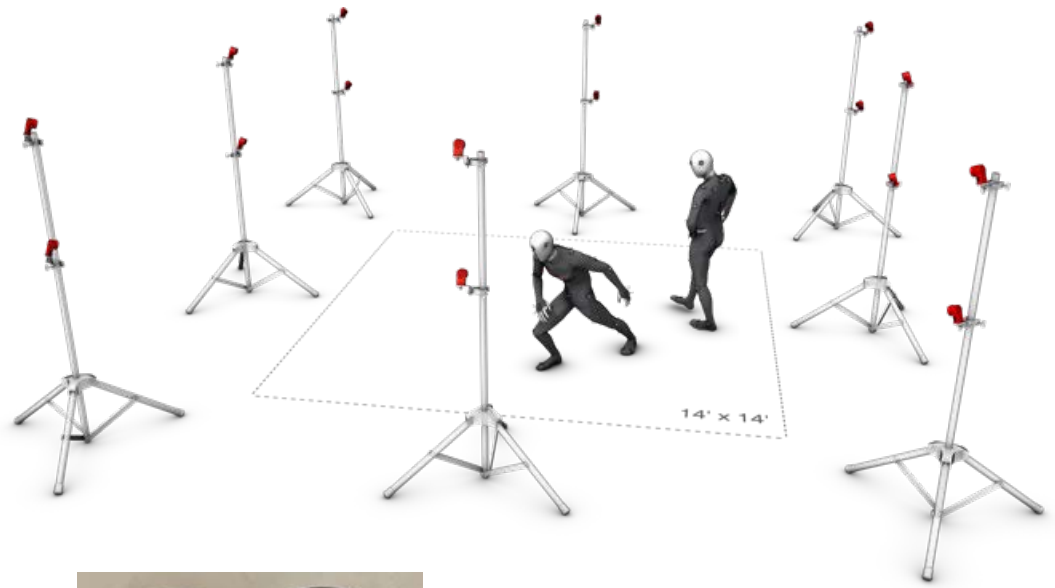

Computational Geometry
ϵ-nets, Caratheodory, MVEE
F, Sharir, Fiat, Langberg, …
[STOC'11, FOCS'06, SoCG'14/07]
Compressed Sensing
Sketches
F, Woodruf, Sohler, …
[SODA'10]
Graph Theory
Sparsifiers, Property Testing
F, Barger, Rus, …
[ICML'17, SDM'16]
Coreset
Research
Matrix Approximation
SVD/PCA, Random Proj.
F, Sohler, Tassa,…
[SODA'13, KDD'15]
Computer Vision
$RANSAC$++
F, Rus, Sochen, …
[ICRA'15, JMIV'15, IROS'12]
Statistics
Importance Sampling, Suff. Stat
F, Shulman, Sung, Rus, …
[SODA'12, SenSys'13, GIS'12]
Robotics
RRT++ sampling
F, Nasser, Jubran, …
[IPSN'12/15/17, ICRA'13/14/15]
Machine Learning
PAC/Active learning
F, Krause, J. W. Fisher,…
[JMLR'17, NIPS'16/14/11]

Example Coresets

- Deep Learning
  - Training [submitted]
  - Model compression [ICLR'19]
- Machine Learning
  - Mixture of Gaussians [JMLR'18]
  - Clustering [SDM'16]
  - Matrix Factorization [KDD15, NIPS16]
  - Segmentation [NIPS'14]
- Real-time Robotics & Computer Vision
  - Swarm of Drones [ICRA'19]
  - Shape fitting [RA-Letters 18]
  - Autonomous cars [IPSN'17]
  - Localization [ICRA'15, IROS'14]

Our RBD Lab's Generations:

- Expensive external tracking cameras + Huge drone + Laptop
- Huge Drone→ Nano-drone
- Expensive → web cameras
- Single drone → swarm
- Web cams→ on-board camera
- Unknown Model
- Current research: all on board
- → Swarm is easy

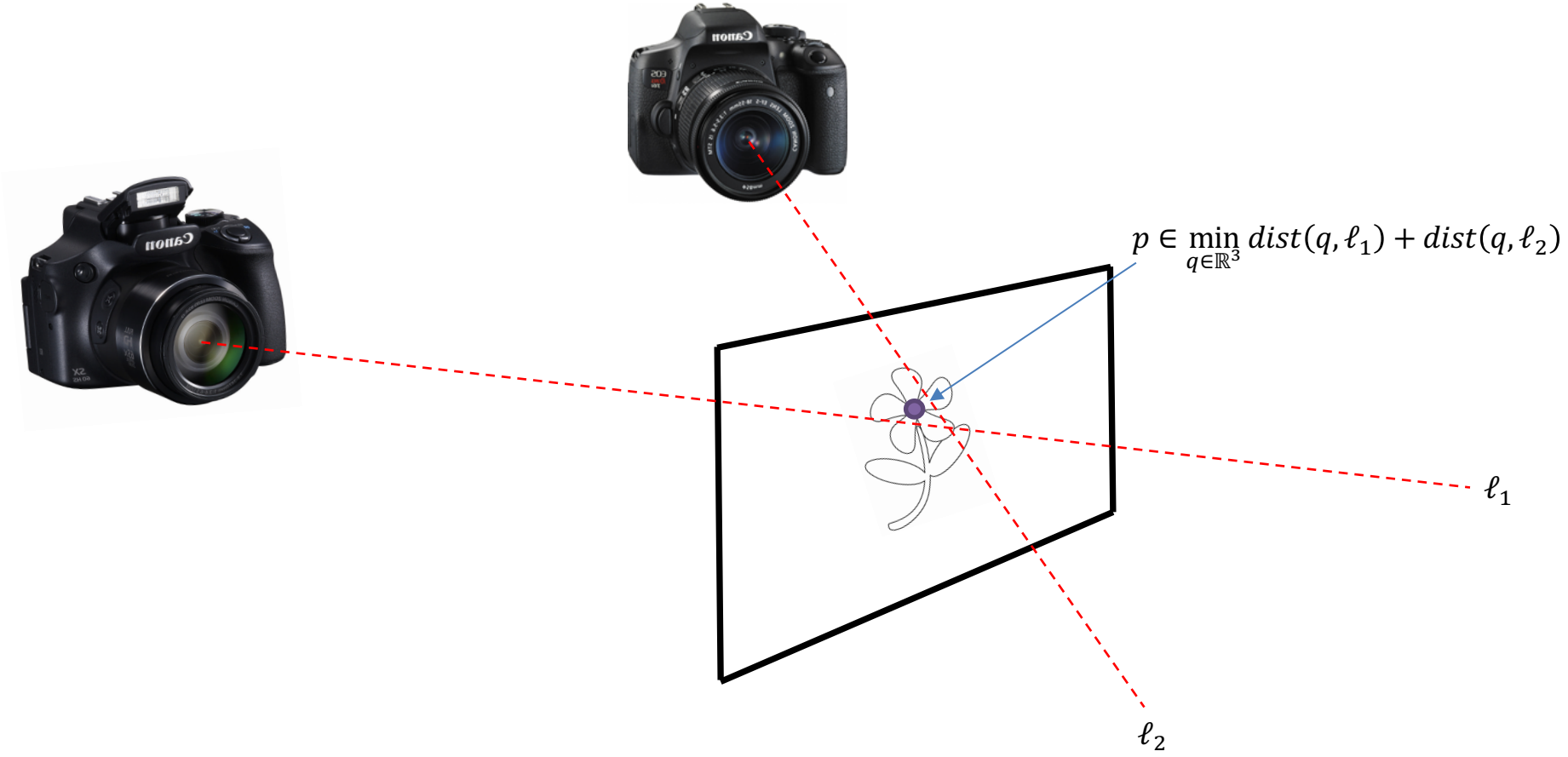


Prime 41 *for* $5,999

OptiTrack's premium motion capture camera. With 4.1 M tracking range, and 51° field of view, the Prime 41 is idea production mocap with impeccable fidelity.

4.1 MP | 100 FPS | 51° FOV | Ci-E

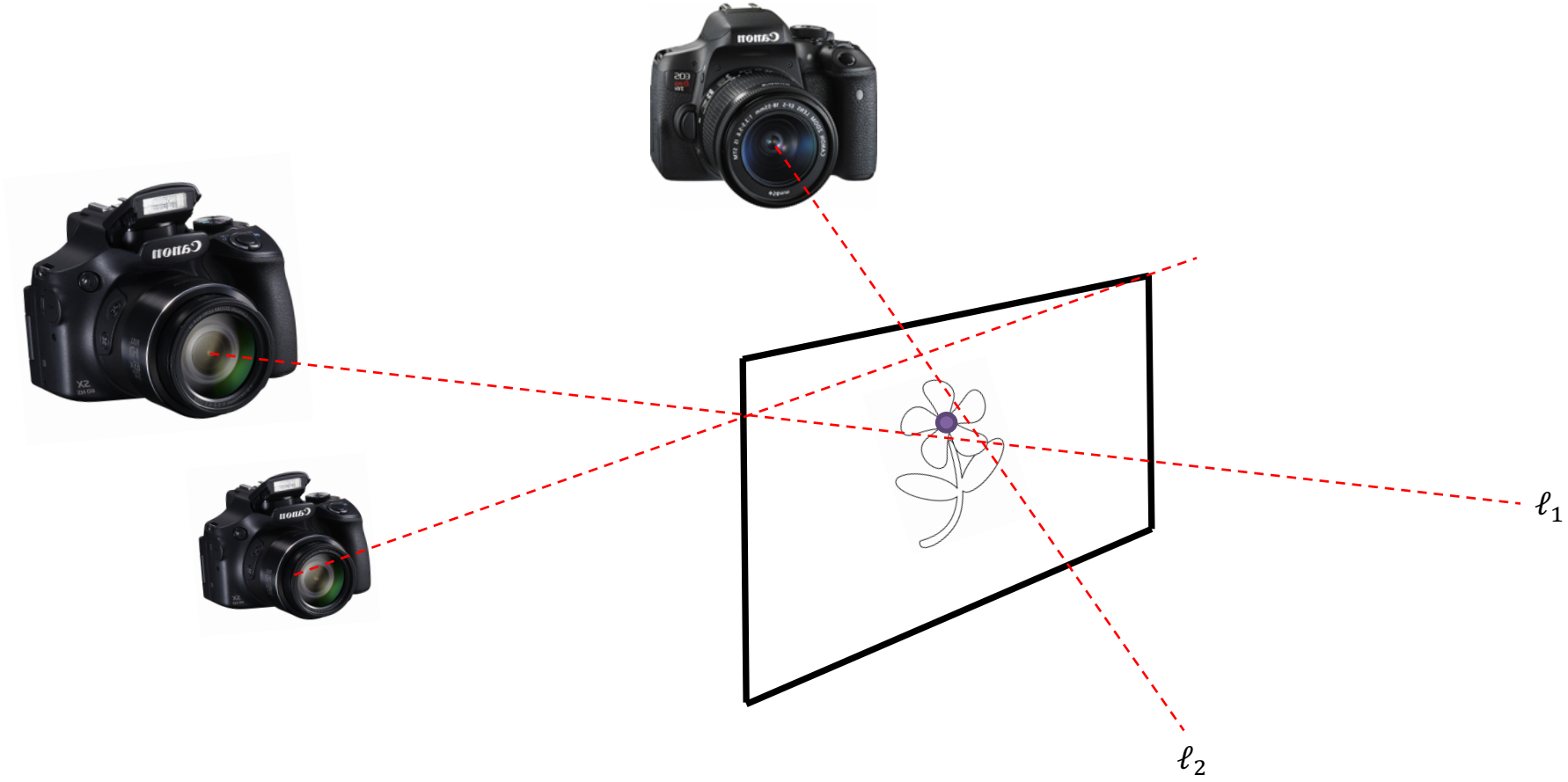
Mapping (Triangulation = 1-mean for 2 lines)



Mapping (1-mean for 3 lines)

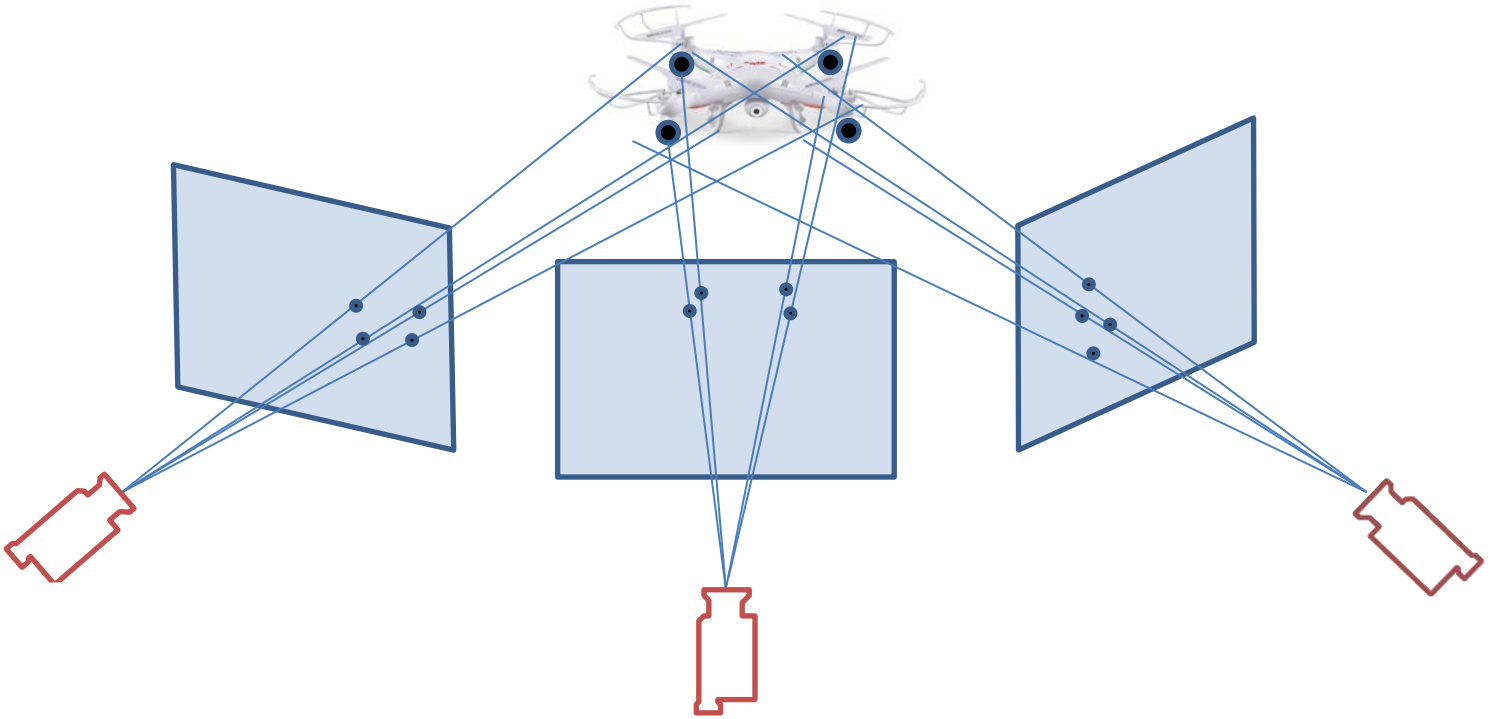
What point minimizes the sum of distances to a given 3 lines?

- Non-convex, Non-linear, no nothing



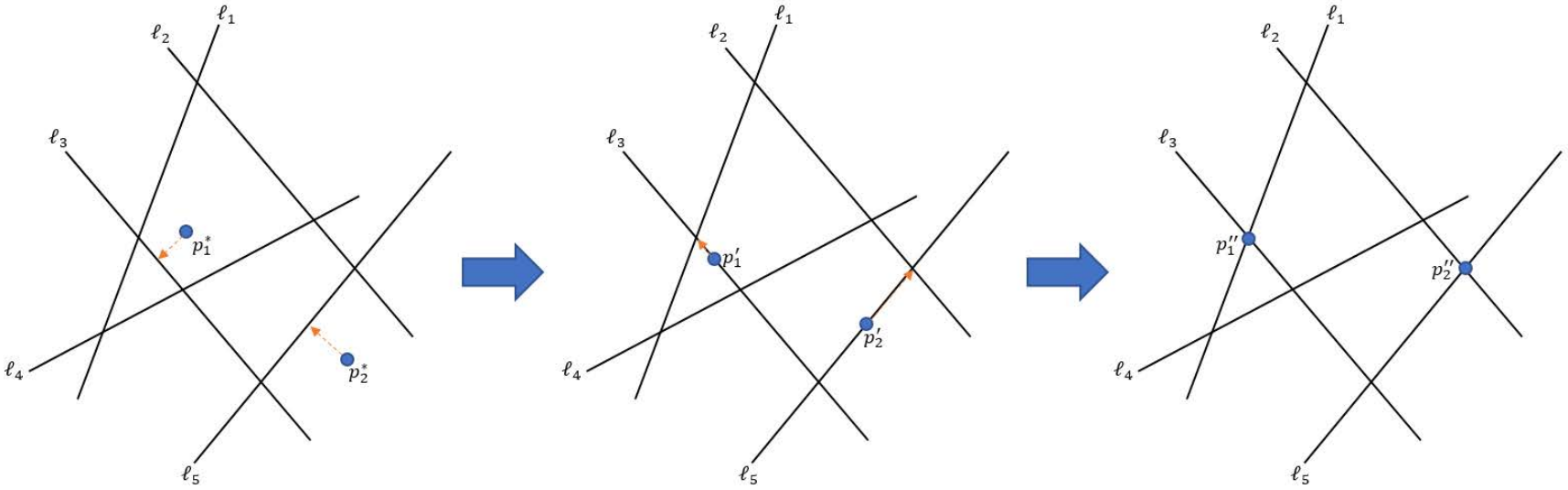
Mapping (k-mean for n lines)

Compute k points (on drone) that minimize the sum of distances to n input lines



2-Factor Approximation

- Compute the closest point to each pair of the n lines
- *k* of them are 2-approx.
- Time too long: $O(n^k)$

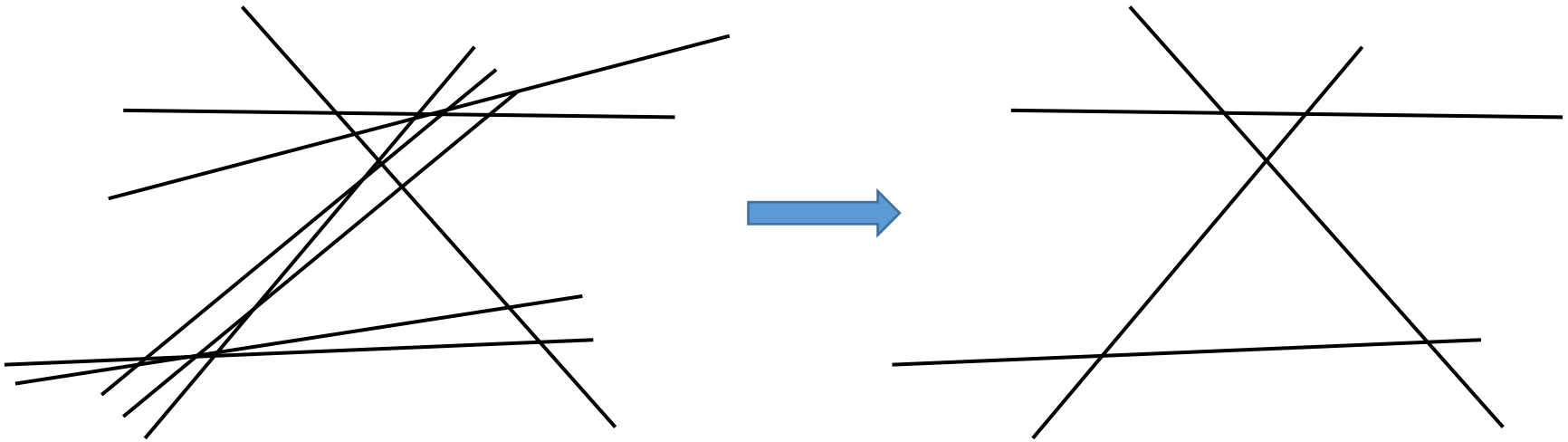


Coreset for k-means of lines

Input: A set L of n lines in $\mathbb{R}^d$,number of means $k \geq 1$ and coreset-size $m \geq 1$.

Output: A set of lines $C = \{c_1, \ldots, c_m\} \subseteq L$ and a set of corresponding weights $u = \{u_1, \ldots, u_m\} \subseteq \mathbb{R}^+$, such that for every set $P \subseteq \mathbb{R}^d$ of k points,

$$\sum_{1 \leq i \leq m} u_i \cdot dist(c_i, P) \in (1 \pm \epsilon) \cdot \sum_{\ell \in L} dist(\ell, P).$$



Theorem [with Y. Marom, NIPS'19]

Every set of n lines has such a coreset of size $\sim \frac{\log(n)}{\varepsilon}$.

- It can be compute in $O(n \log n)$ time
- Point insertion in $\sim \log n$ time/memory (streaming)
- Parallel time reduced by a factor of *M*
  using *M* machines (GPU, cloud, threads)

(α, β)-Approximation for k lines means

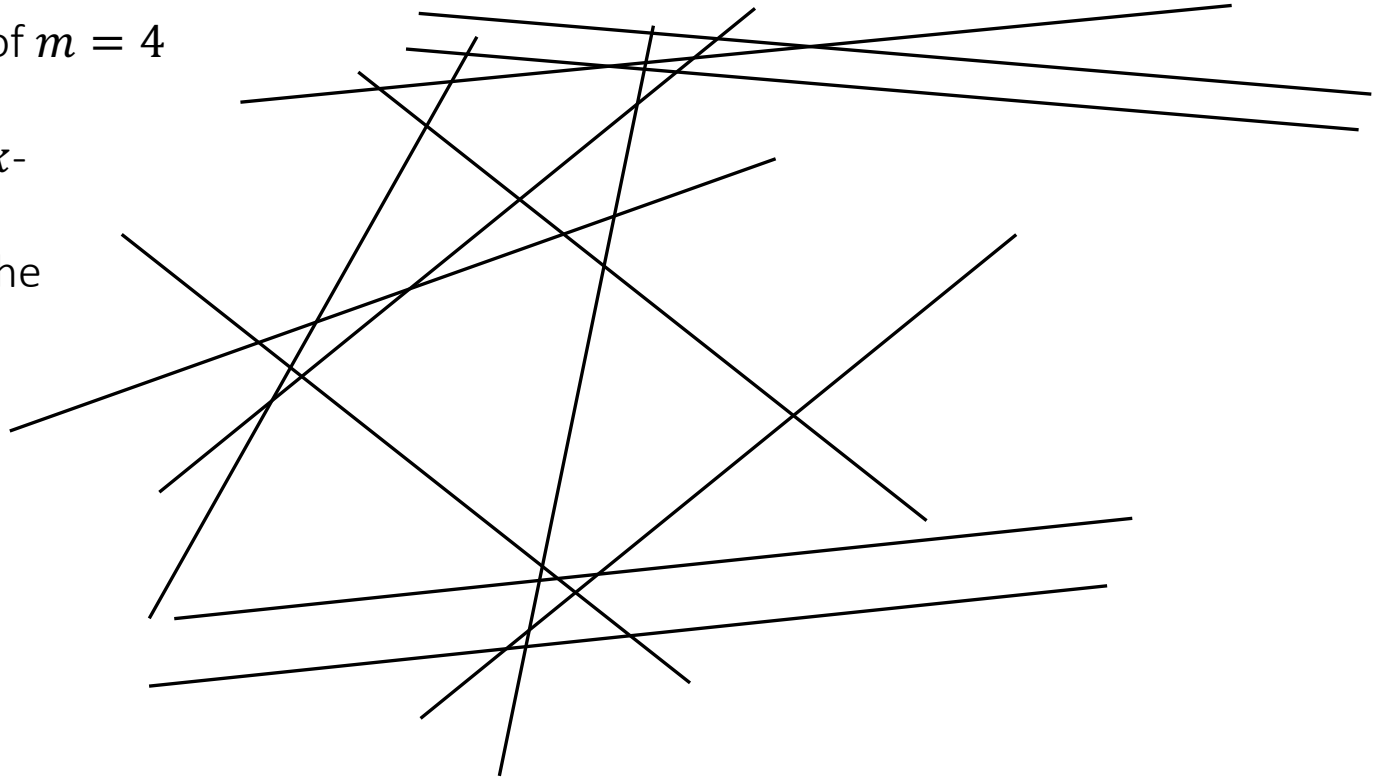
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-meanss of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



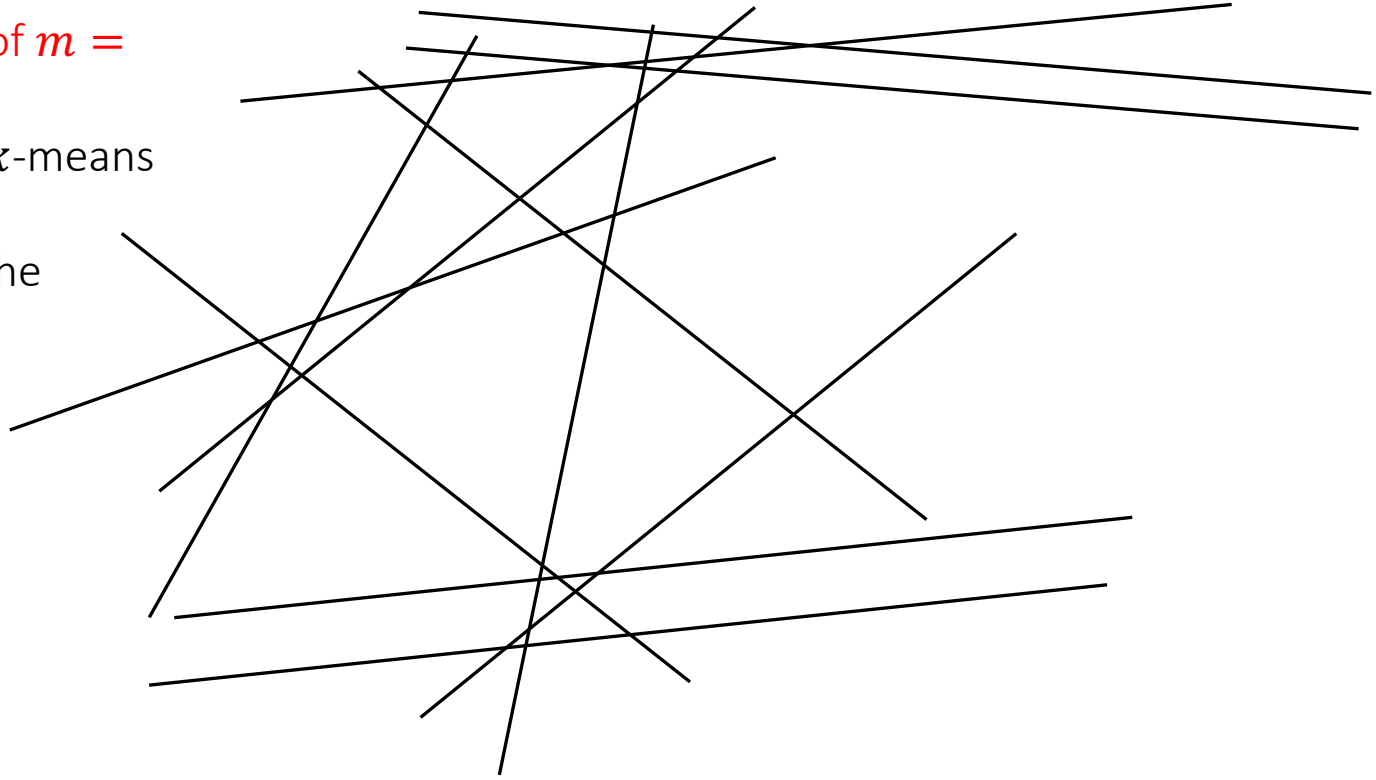
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-means of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



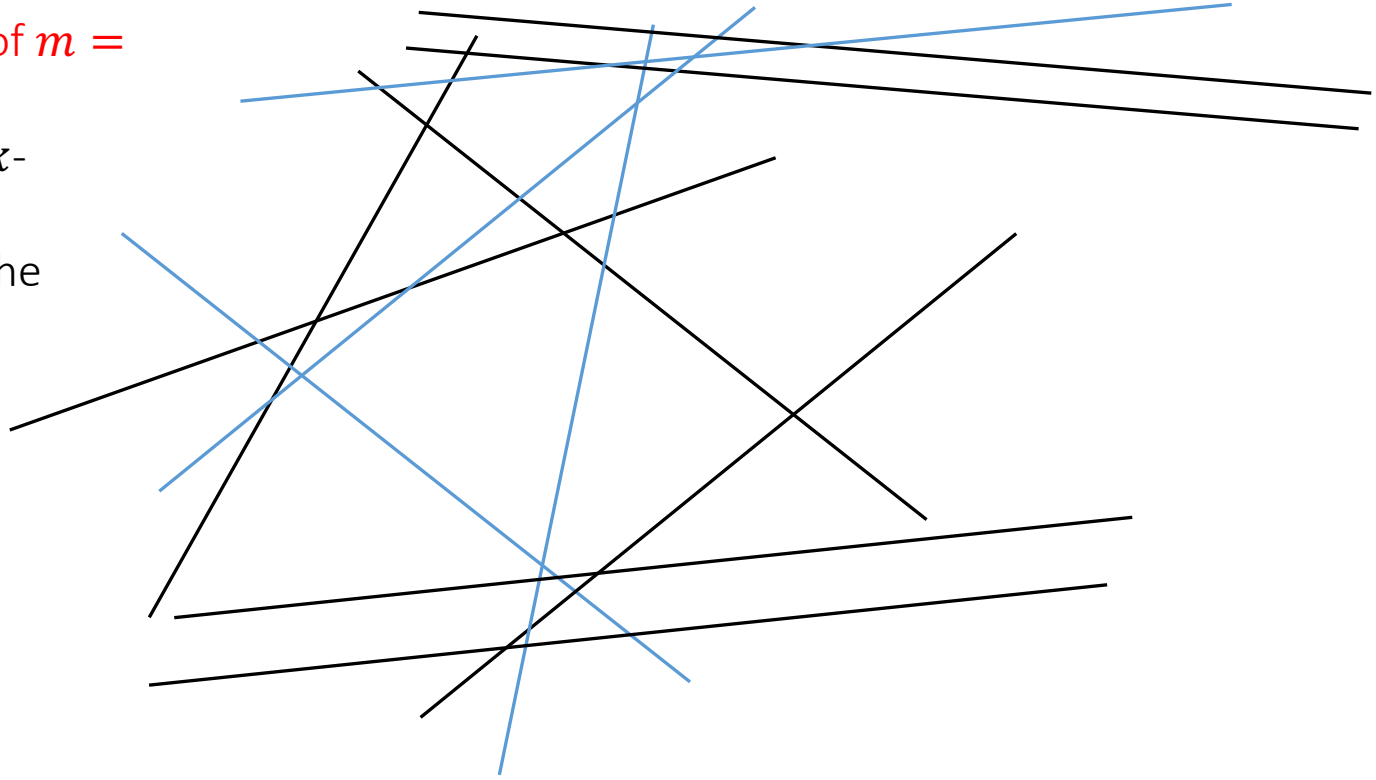
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-meanss of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



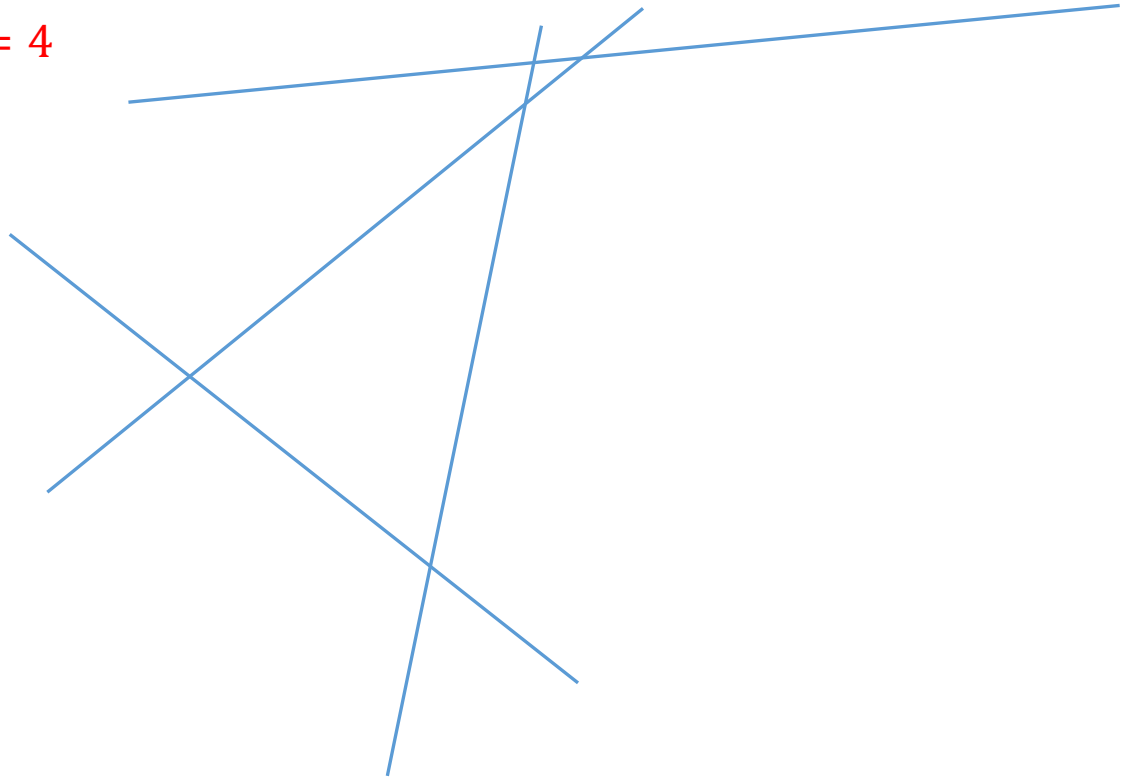
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-meanss of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



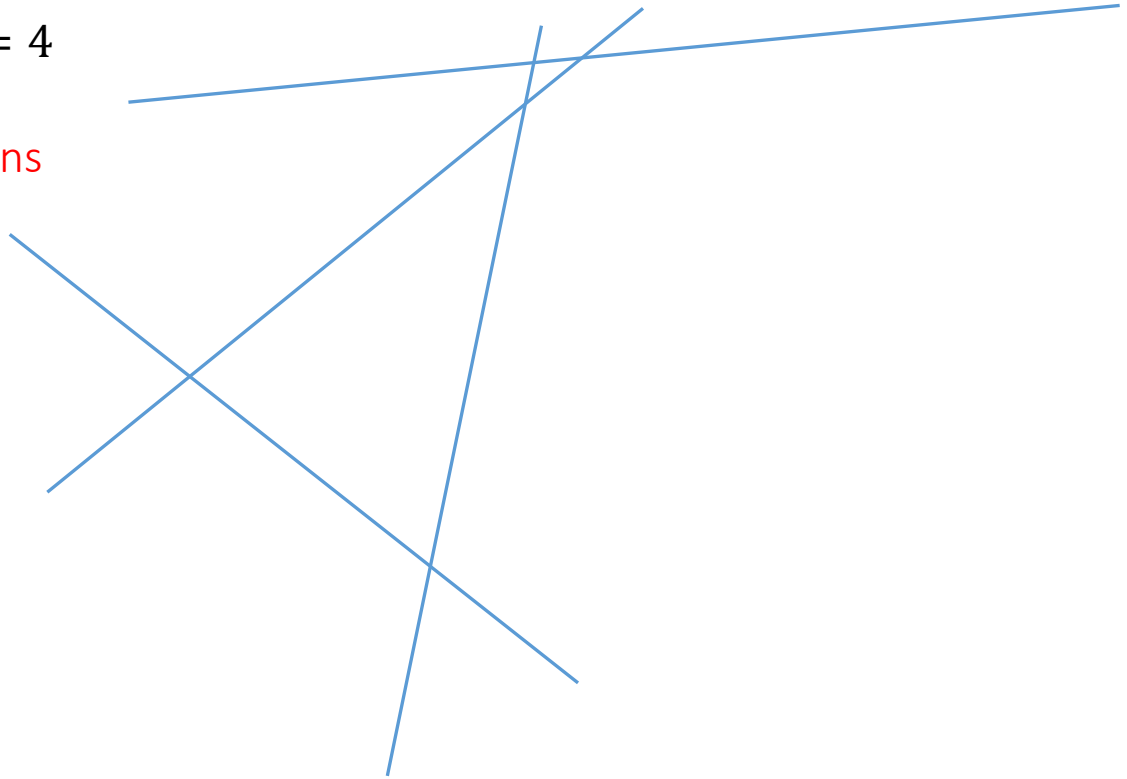
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-means of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



(α, β)-Approximation for k lines means – ***Bicriteria***

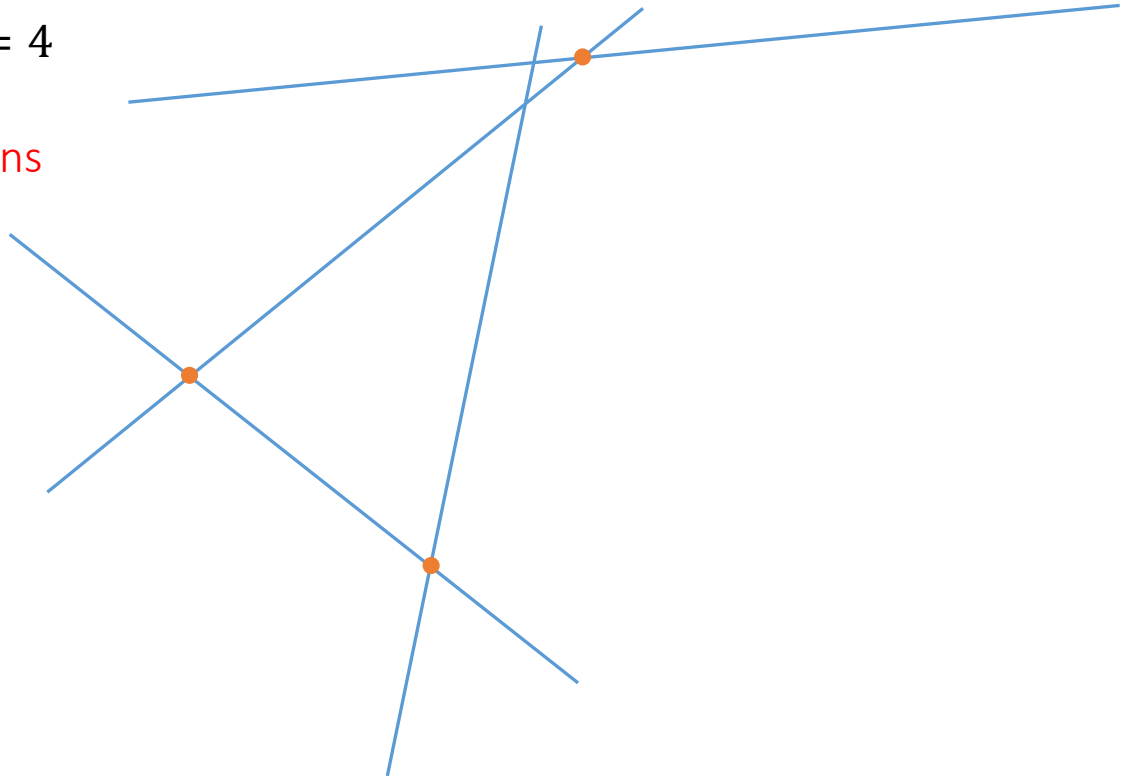
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-means of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



(α, β)-Approximation for k lines means – ***Bicriteria***

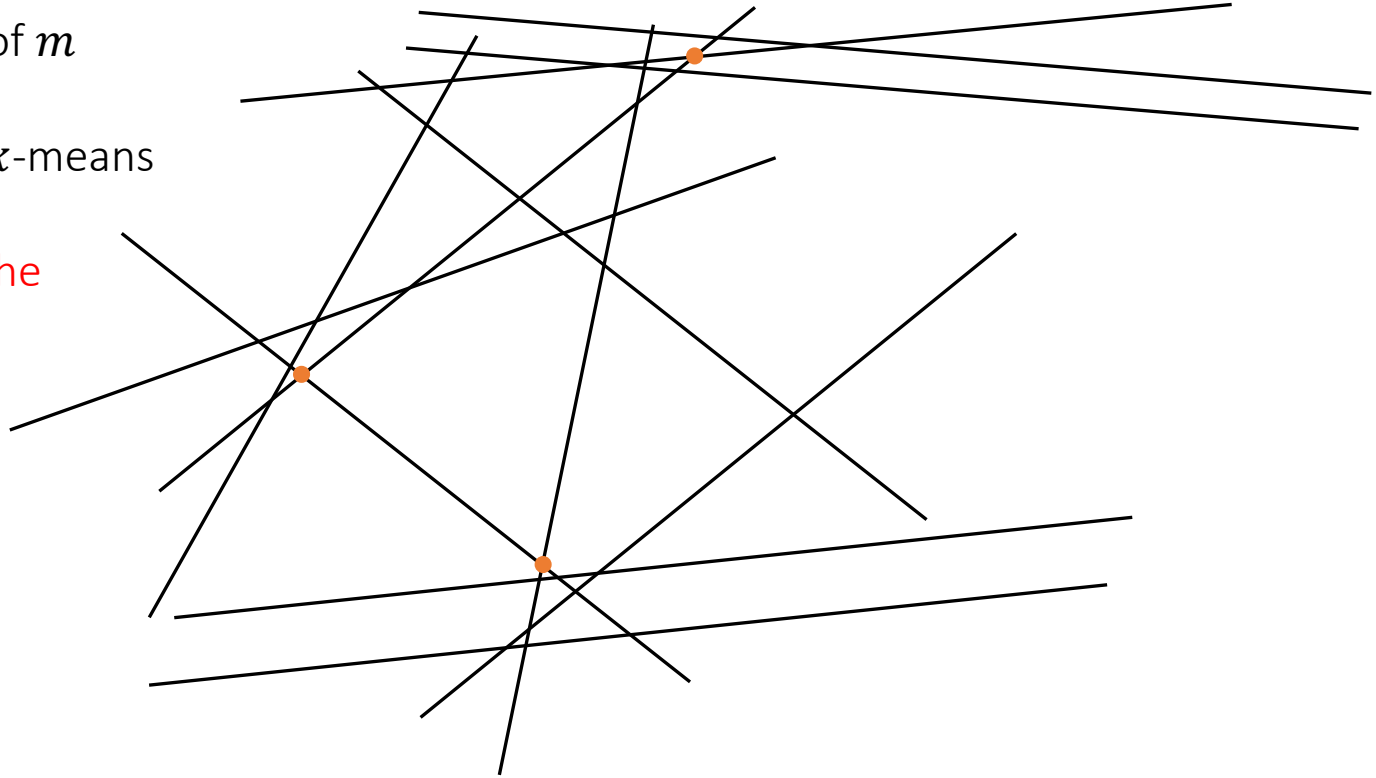
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-means of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



(α, β)-Approximation for k lines means – ***Bicriteria***

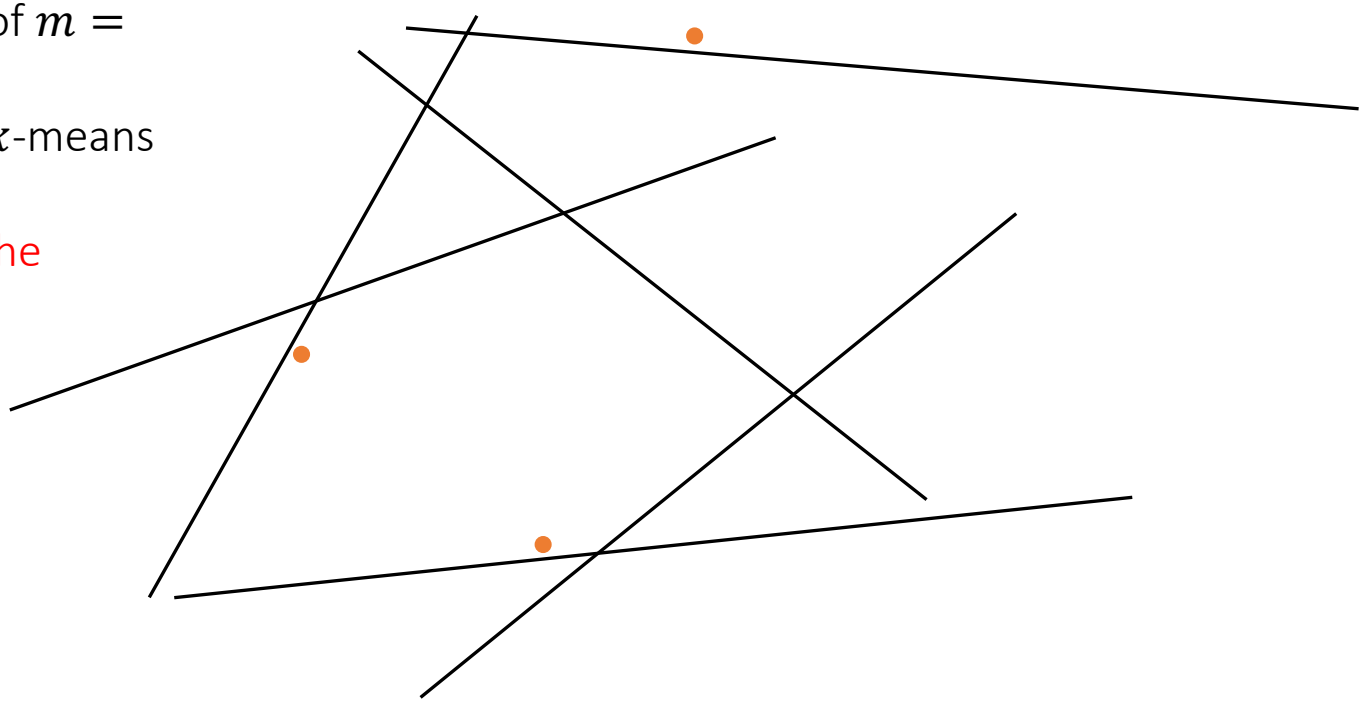
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-means of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



(α, β)-Approximation for k lines means – ***Bicriteria***

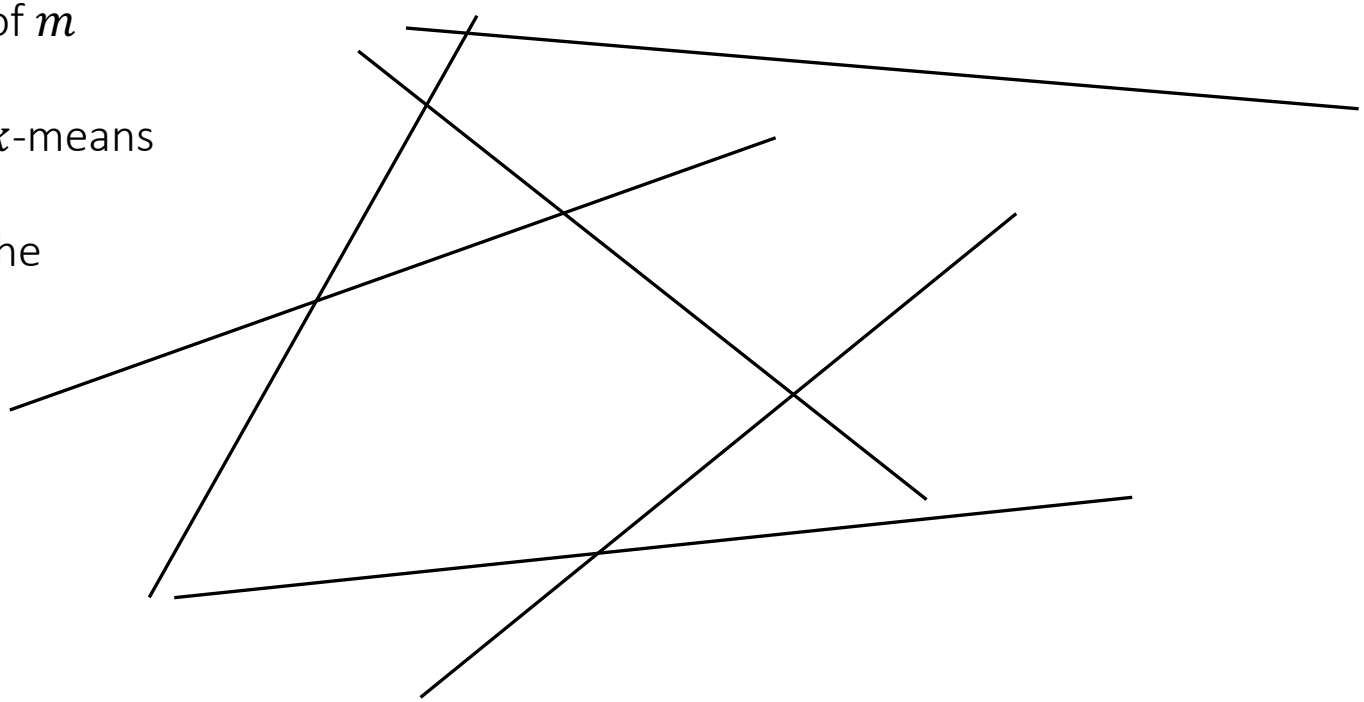
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-means of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



(α, β)-Approximation for k lines means – ***Bicriteria***

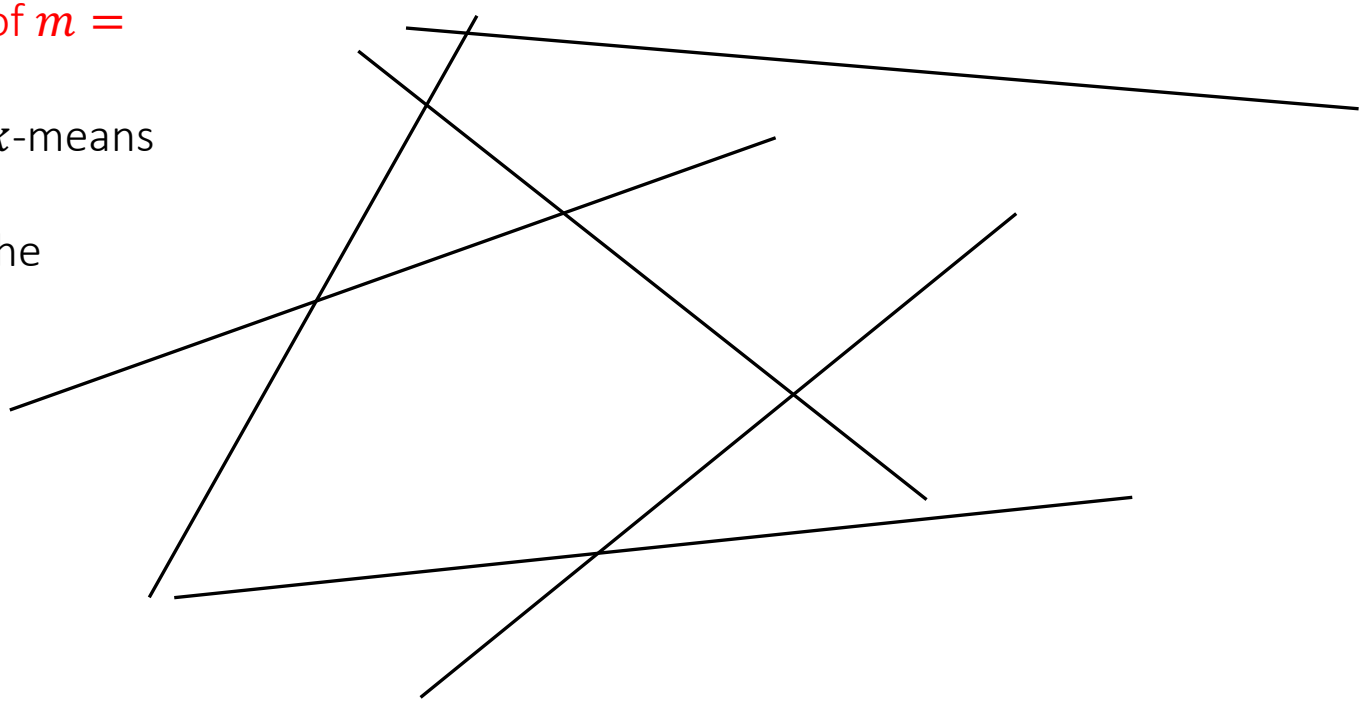
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-means of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



(α, β)-Approximation for k lines means – ***Bicriteria***

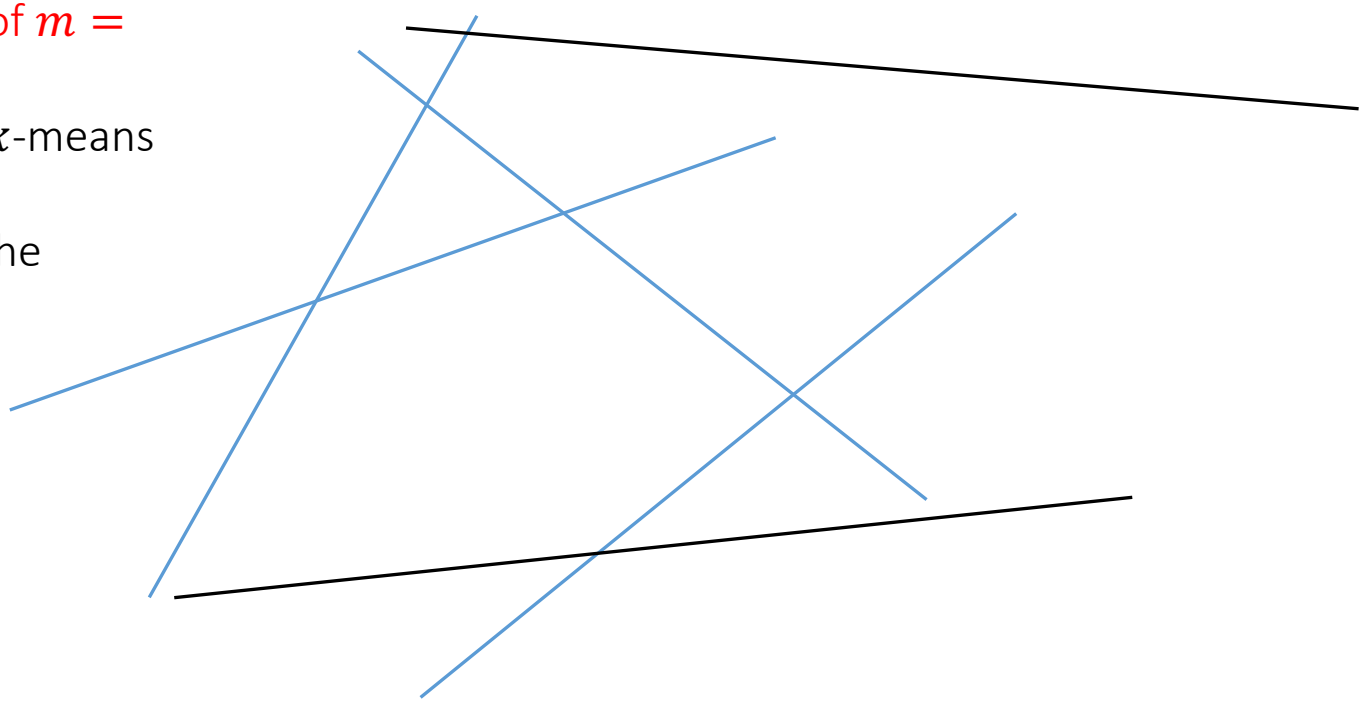
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-means of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



(α, β)-Approximation for k lines means – ***Bicriteria***

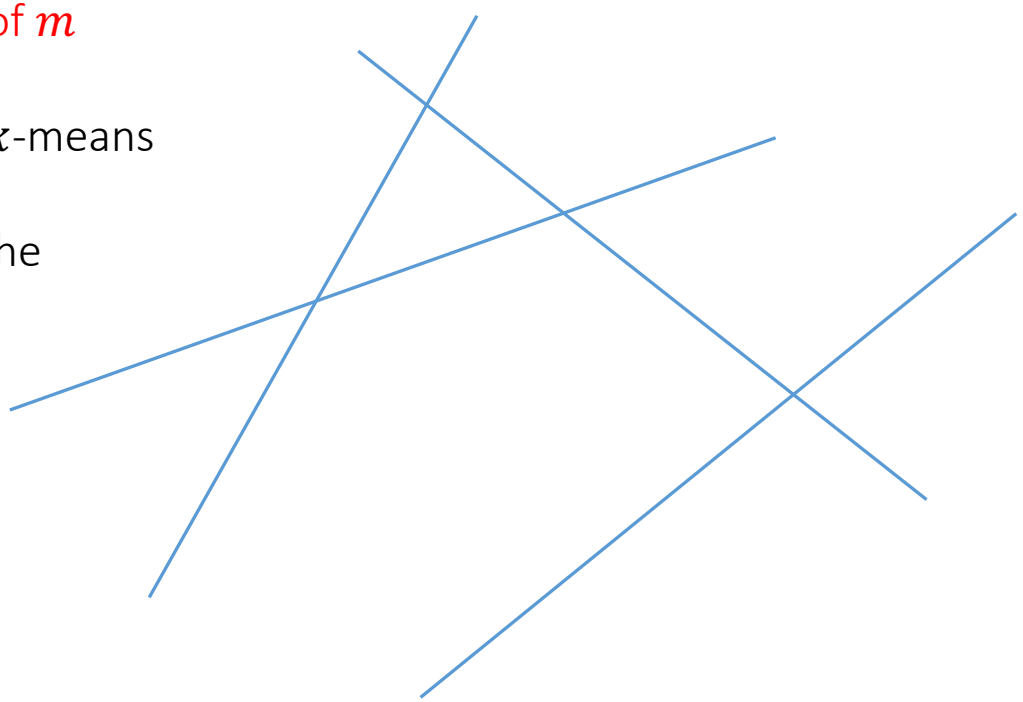
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-means of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



(α, β)-Approximation for k lines means – ***Bicriteria***

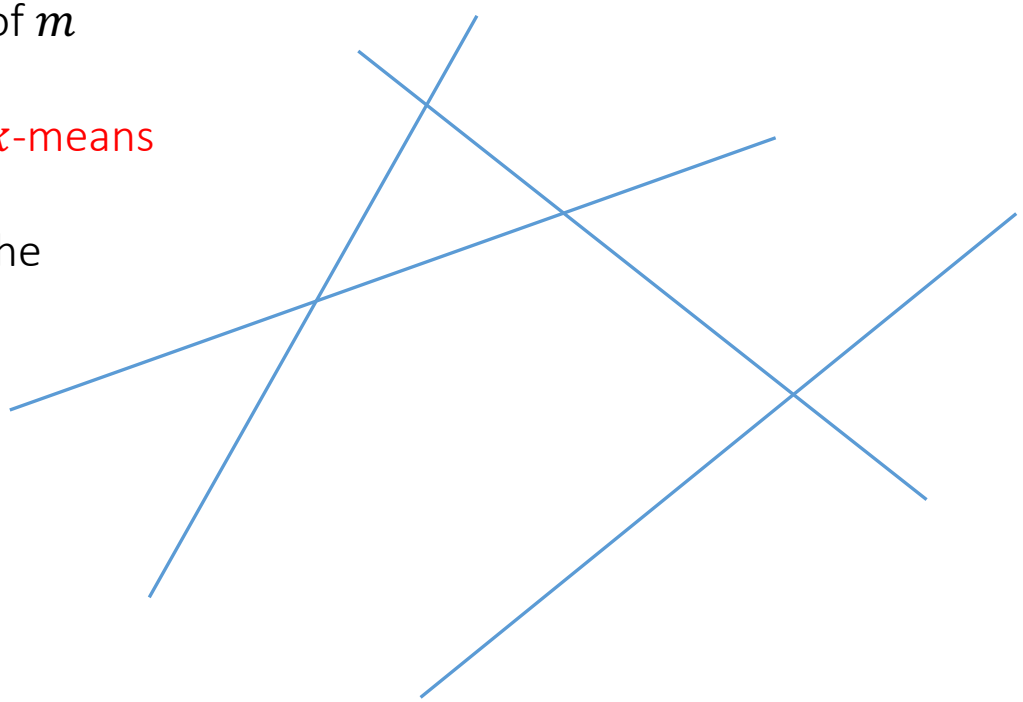
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-means of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



(α, β)-Approximation for k lines means – ***Bicriteria***

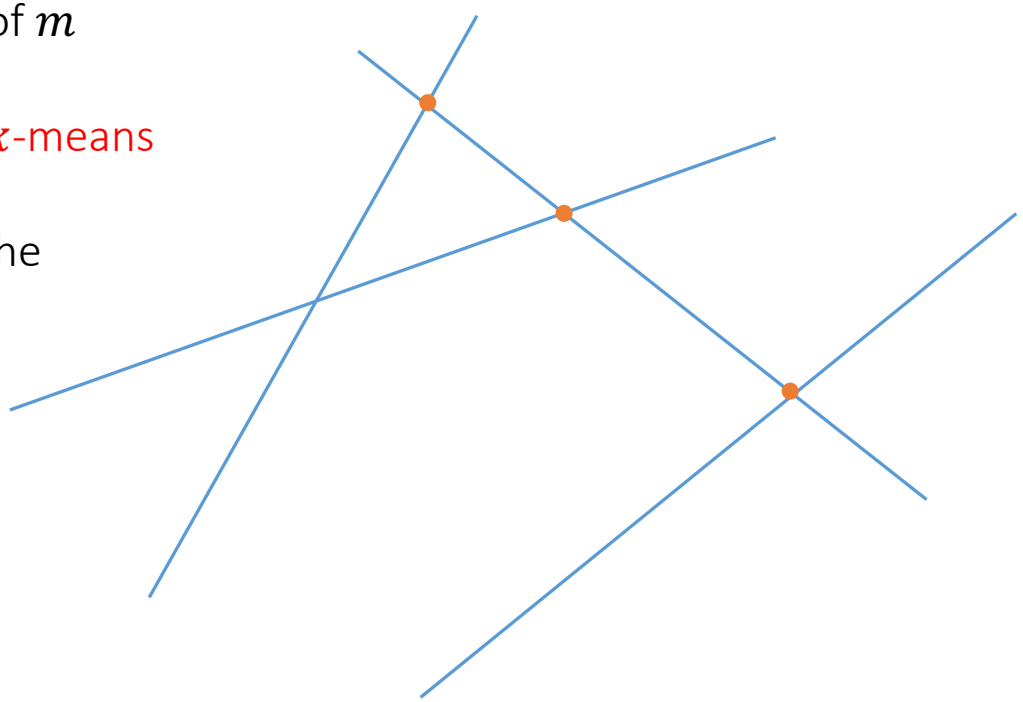
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-means of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



(α, β)-Approximation for k lines means – ***Bicriteria***

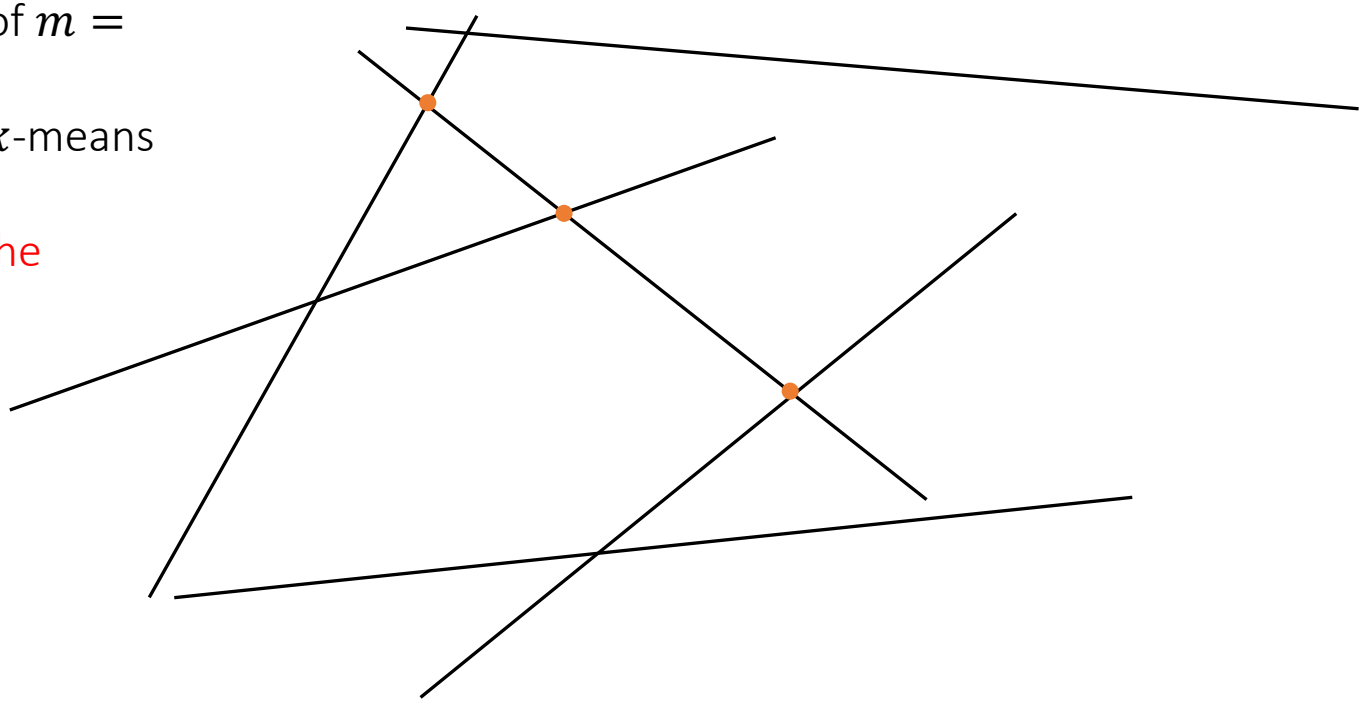
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-means of S.

(3) Remove half of the closest lines to G.

(4) Return to (1)



(α, β)-Approximation for k lines means – ***Bicriteria***

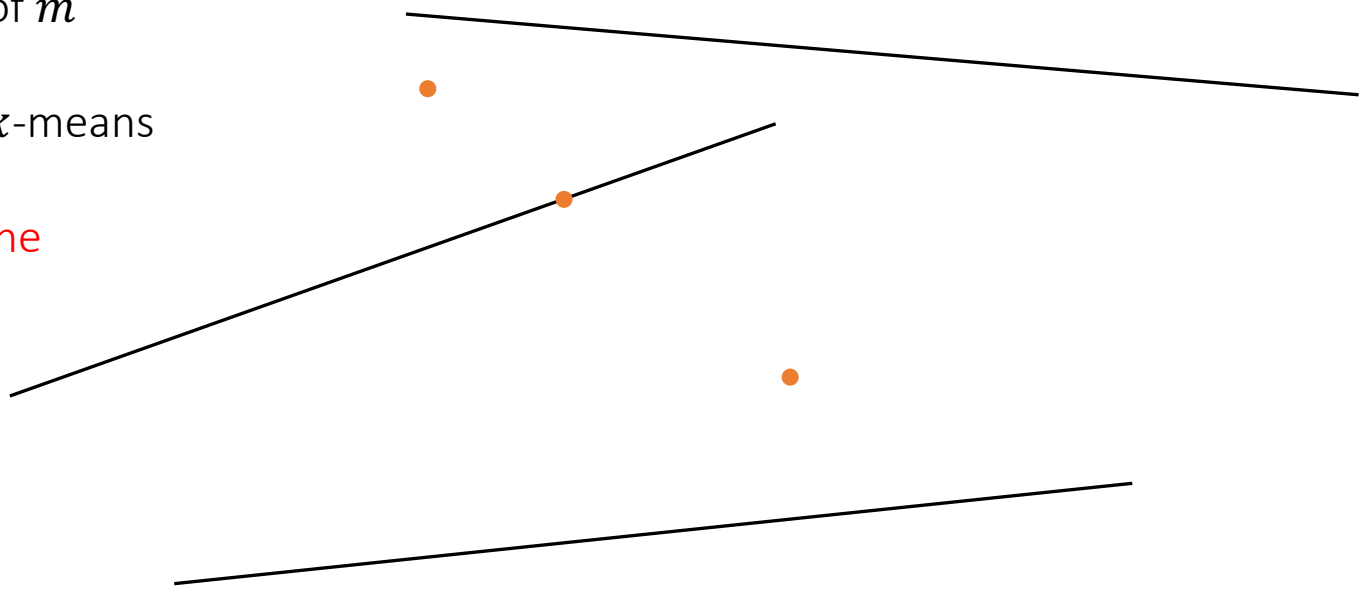
Algorithm:

(1) Pick a sample S of $m = 4$ lines.

(2) Compute G the k-means of S.

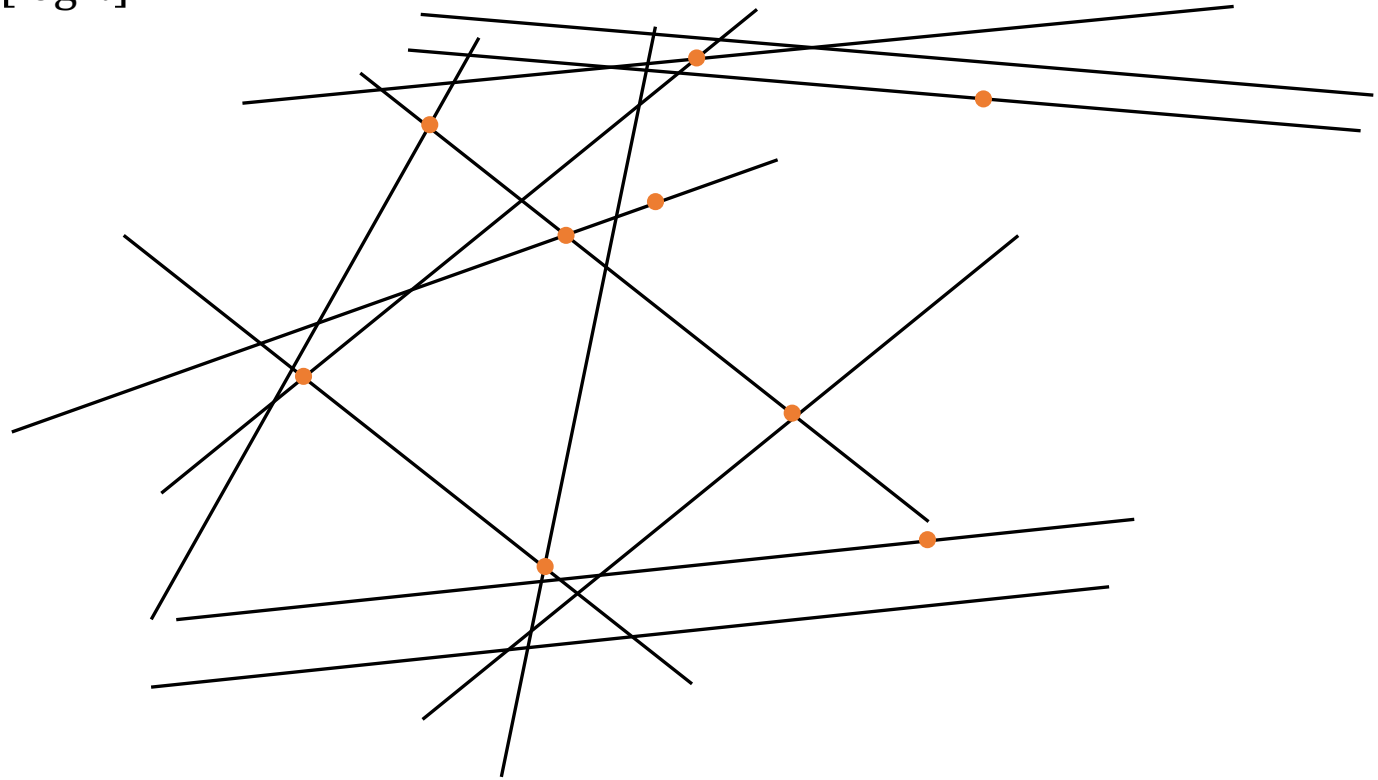
(3) Remove half of the closest lines to G.

(4) Return to (1)



(α, β)-Approximation for k lines means – ***Bicriteria***

Eventually, we get $k\lfloor \log n \rfloor$ $= 9$ points



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$.

(3) For every $\ell \in L$ and its closest point $b \in B$, compute final sensitivity

$$s(\ell) = \frac{dist(\ell, b)}{\sum_{\ell' \in L} dist(\ell', B)} + 2s_b(\ell)$$

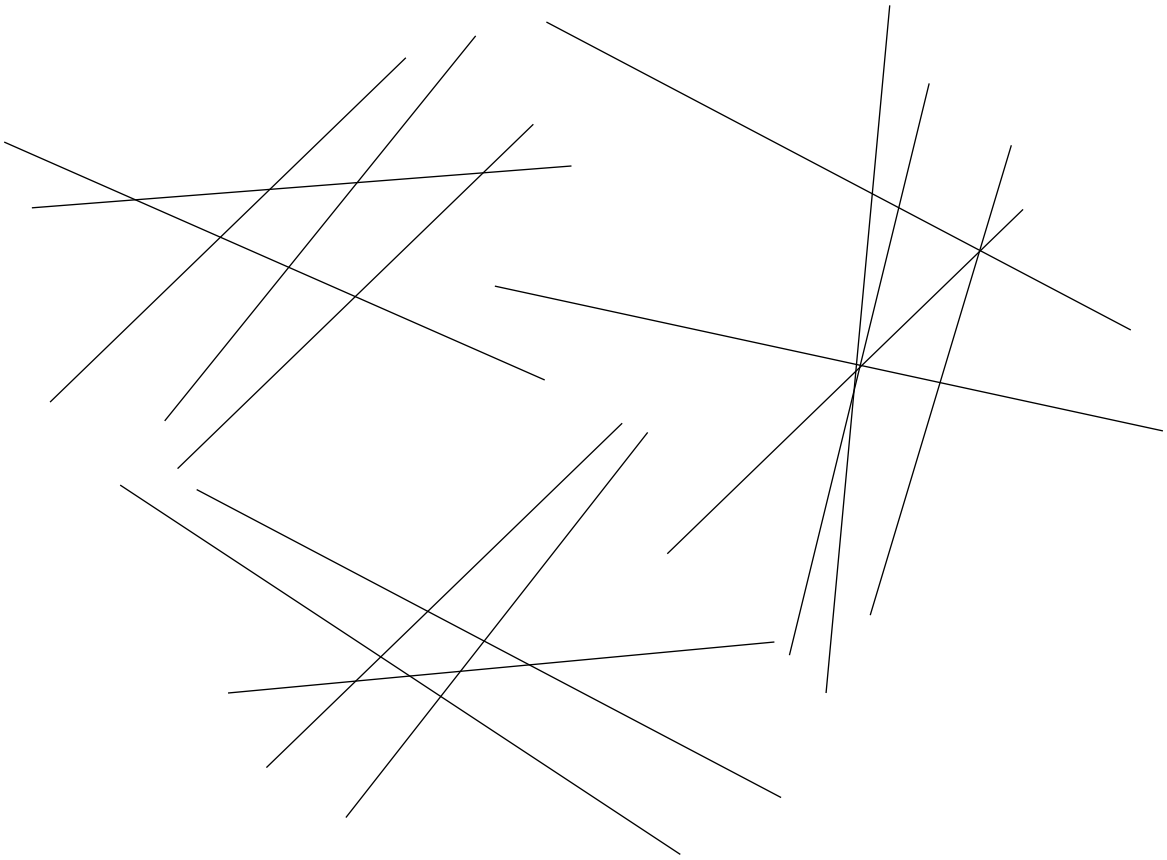
(4) For every line $\ell \in L$, define $prob(\ell) = \frac{s(\ell)}{\sum_{\ell' \in L} s(\ell')}$.

(5) Pick a sample S consist of at least m lines from L with probability $prob(\ell)$.

(6) For every $\ell \in S$ define

$$u(\ell) = \frac{1}{|S| prob(\ell)}.$$

(7) Return (S, u).



Coreset for k lines means

Algorithm:

(1) Compute $B := (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$.

(3) For every $\ell \in L$ and its closest point $b \in B$, compute final sensitivity

$$s(\ell) = \frac{dist(\ell, b)}{\sum_{\ell' \in L} dist(\ell', B)} + 2s_b(\ell)$$

(4) For every line $\ell \in L$, define

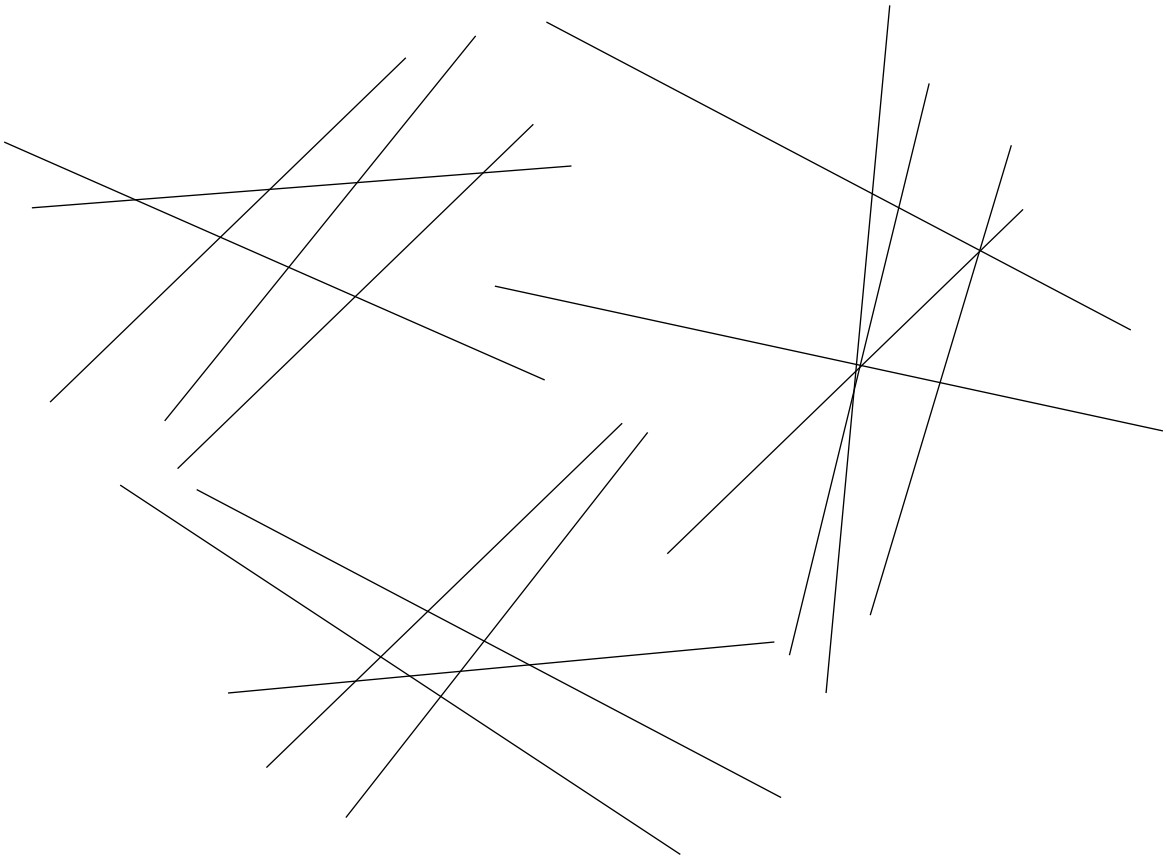
$$prob(\ell) = \frac{s(\ell)}{\sum_{\ell' \in L} s(\ell')}.$$

(5) Pick a sample S consist of at least m lines from L with probability $prob(\ell)$.

(6) For every $\ell \in S$ define

$$u(\ell) = \frac{1}{|S| prob(\ell)}.$$

(7) Return (S, u).



Coreset for k lines means

Algorithm:

(1) Compute $B := (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$.

(3) For every $\ell \in L$ and its closest point $b \in B$, compute final sensitivity

$$s(\ell) = \frac{dist(\ell, b)}{\sum_{\ell' \in L} dist(\ell', B)} + 2s_b(\ell)$$

(4) For every line $\ell \in L$, define

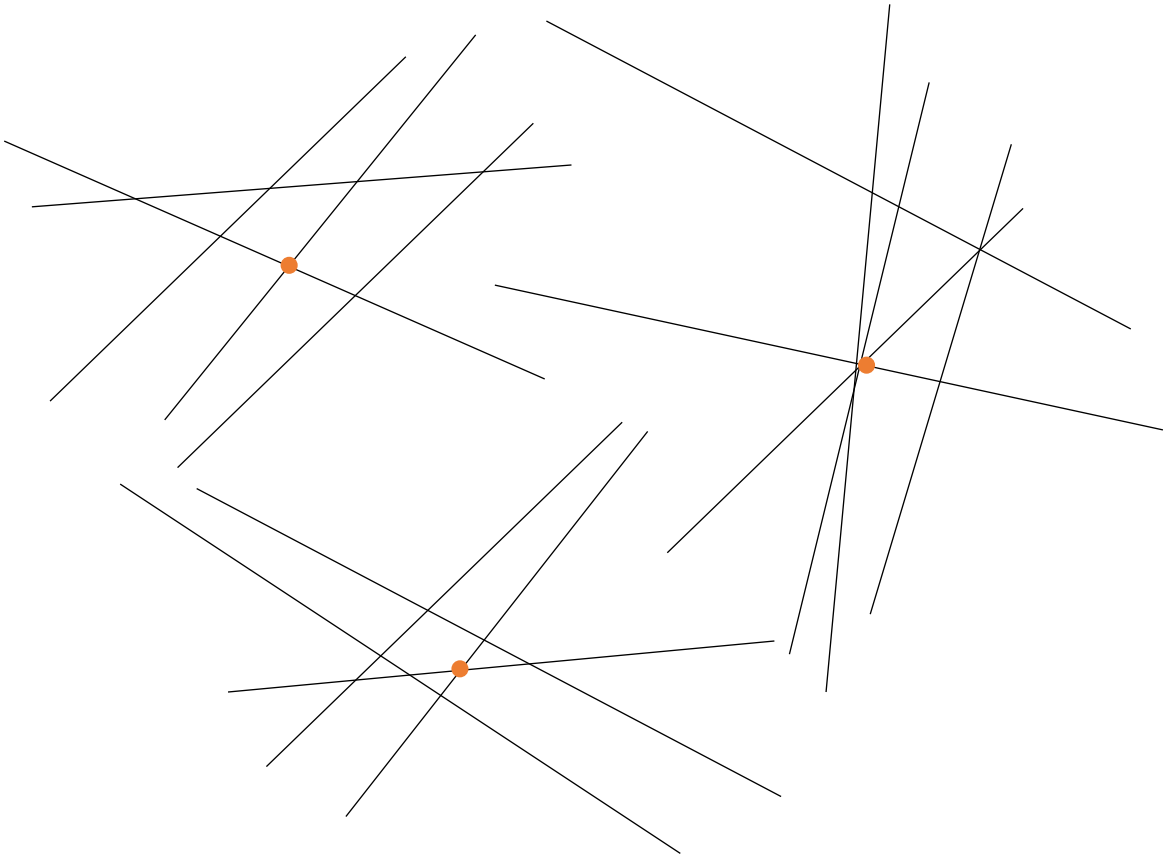
$$prob(\ell) = \frac{s(\ell)}{\sum_{\ell' \in L} s(\ell')}.$$

(5) Pick a sample S consist of at least m lines from L with probability $prob(\ell)$.

(6) For every $\ell \in S$ define

$$u(\ell) = \frac{1}{|S| prob(\ell)}.$$

(7) Return (S, u).



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$.

(3) For every $\ell \in L$ and its closest point $b \in B$, compute final sensitivity

$$s(\ell) = \frac{dist(\ell, b)}{\sum_{\ell' \in L} dist(\ell', B)} + 2s_b(\ell)$$

(4) For every line $\ell \in L$, define

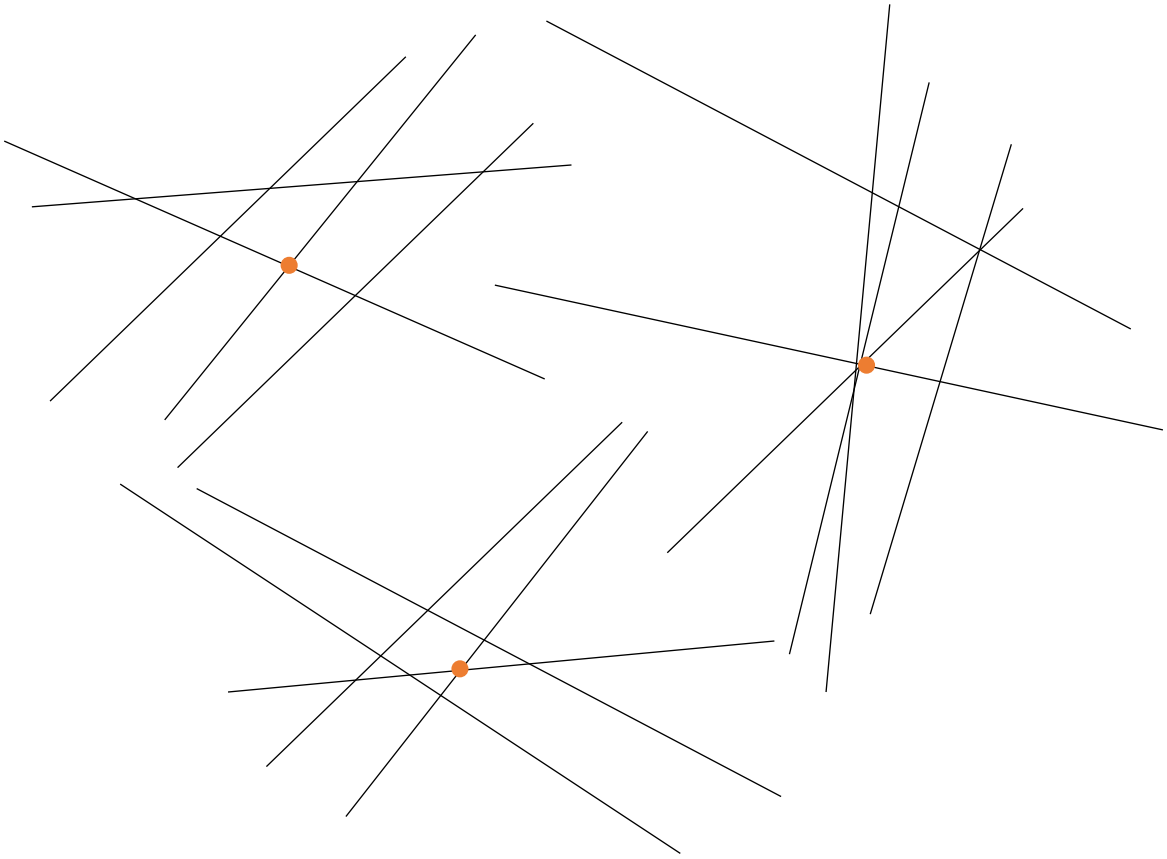
$$prob(\ell) = \frac{s(\ell)}{\sum_{\ell' \in L} s(\ell')}.$$

(5) Pick a sample S consist of at least m lines from L with probability $prob(\ell)$.

(6) For every $\ell \in S$ define

$$u(\ell) = \frac{1}{|S| prob(\ell)}.$$

(7) Return (S, u).



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$.

(3) For every $\ell \in L$ and its closest point $b \in B$, compute final sensitivity

$$s(\ell) = \frac{dist(\ell, b)}{\sum_{\ell' \in L} dist(\ell', B)} + 2s_b(\ell)$$

(4) For every line $\ell \in L$, define

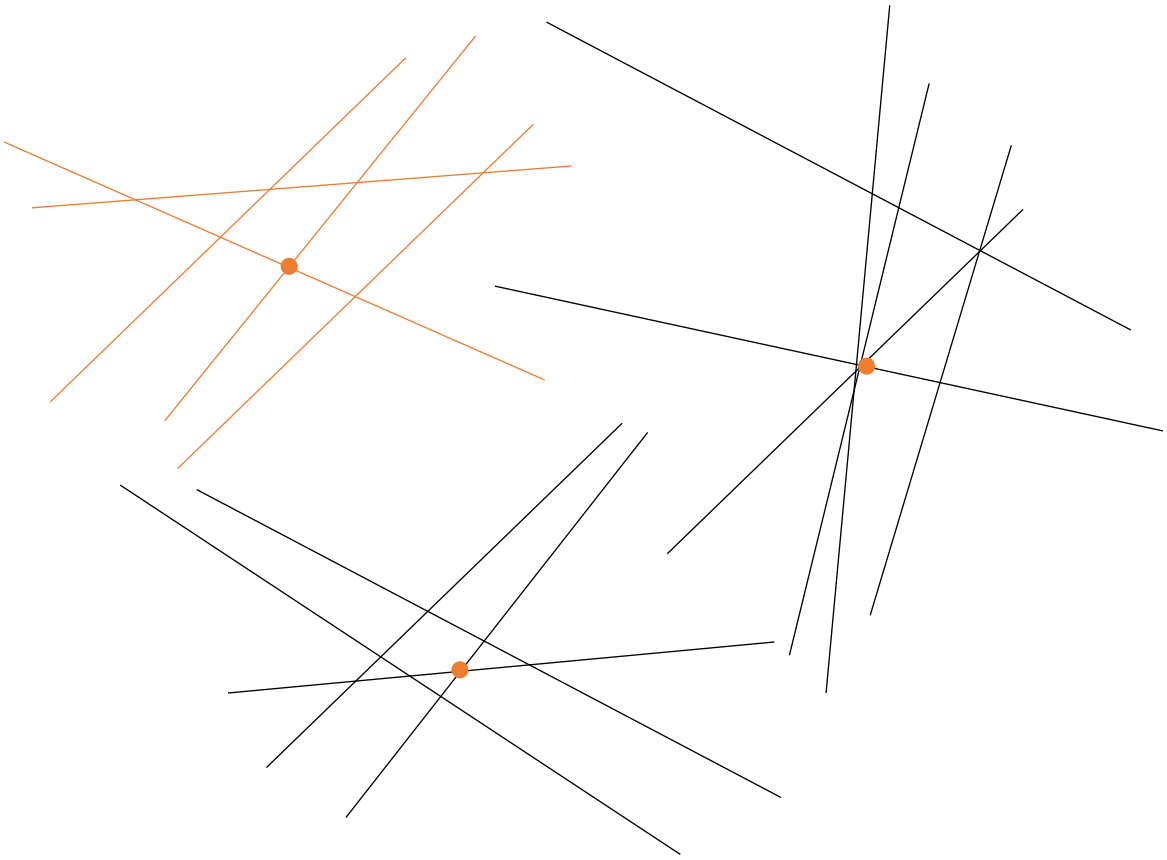
$$prob(\ell) = \frac{s(\ell)}{\sum_{\ell' \in L} s(\ell')}.$$

(5) Pick a sample S consist of at least m lines from L with probability $prob(\ell)$.

(6) For every $\ell \in S$ define

$$u(\ell) = \frac{1}{|S| prob(\ell)}.$$

(7) Return (S, u).



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$.

(3) For every $\ell \in L$ and its closest point $b \in B$, compute final sensitivity

$$s(\ell) = \frac{dist(\ell, b)}{\sum_{\ell' \in L} dist(\ell', B)} + 2s_b(\ell)$$

(4) For every line $\ell \in L$, define

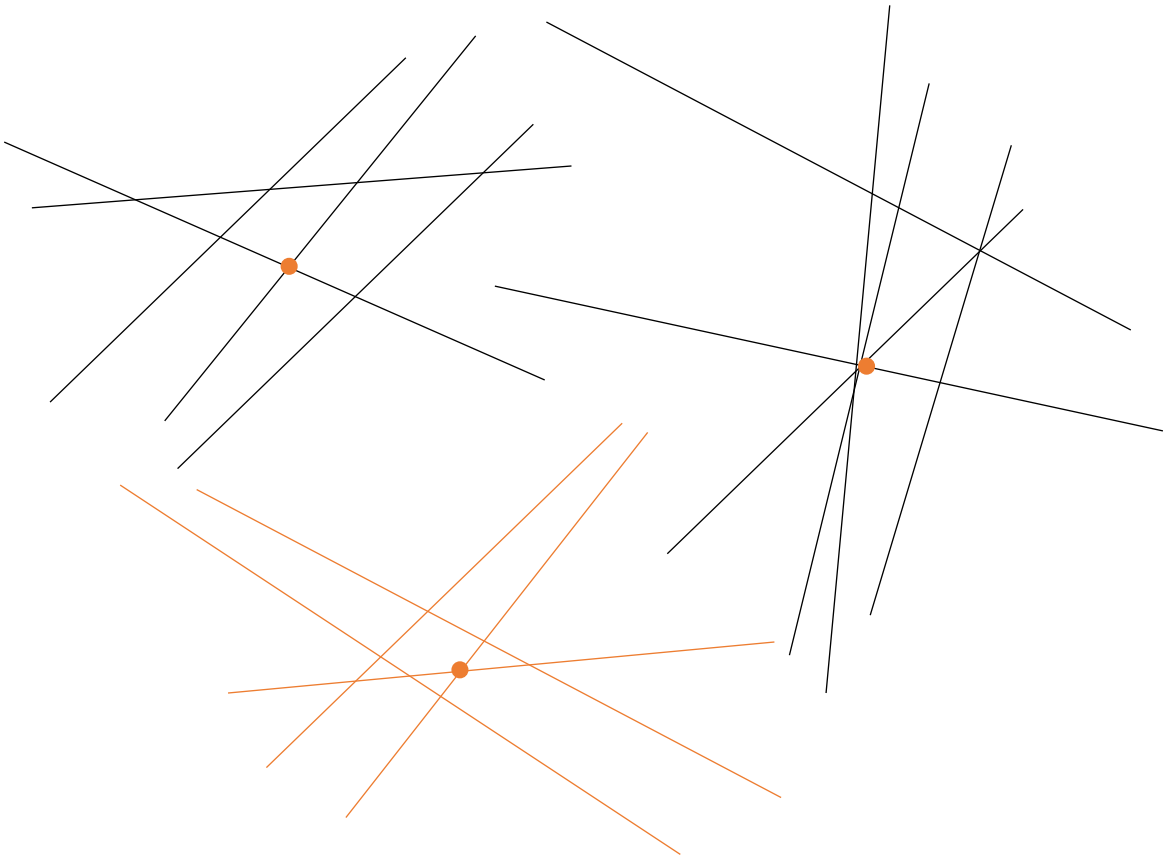
$$prob(\ell) = \frac{s(\ell)}{\sum_{\ell' \in L} s(\ell')}.$$

(5) Pick a sample S consist of at least m lines from L with probability $prob(\ell)$.

(6) For every $\ell \in S$ define

$$u(\ell) = \frac{1}{|S| prob(\ell)}.$$

(7) Return (S, u).



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$.

(3) For every $\ell \in L$ and its closest point $b \in B$, compute final sensitivity

$$s(\ell) = \frac{dist(\ell, b)}{\sum_{\ell' \in L} dist(\ell', B)} + 2s_b(\ell)$$

(4) For every line $\ell \in L$, define

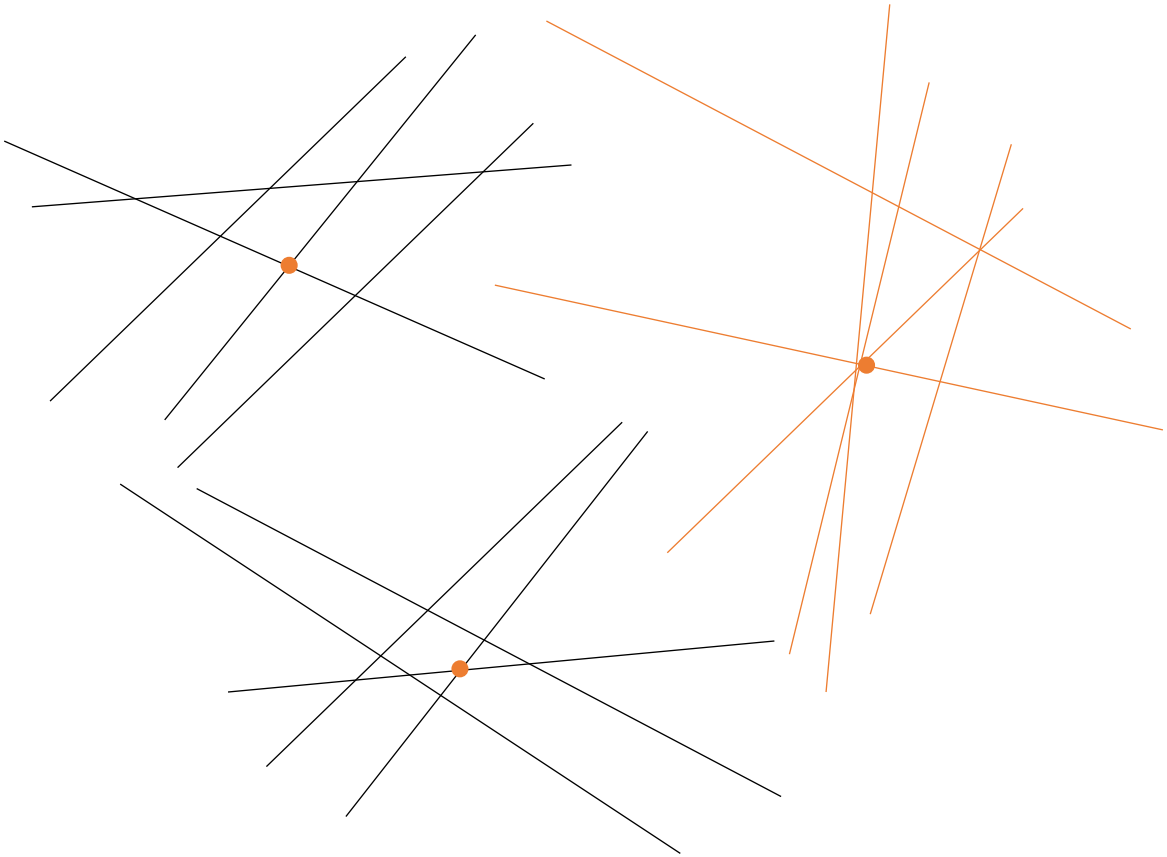
$$prob(\ell) = \frac{s(\ell)}{\sum_{\ell' \in L} s(\ell')}.$$

(5) Pick a sample S consist of at least m lines from L with probability $prob(\ell)$.

(6) For every $\ell \in S$ define

$$u(\ell) = \frac{1}{|S| prob(\ell)}.$$

(7) Return (S, u).



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$:

   i. Let ℓ' be the parallel line to ℓ that intersect b (the projection of ℓ onto b).

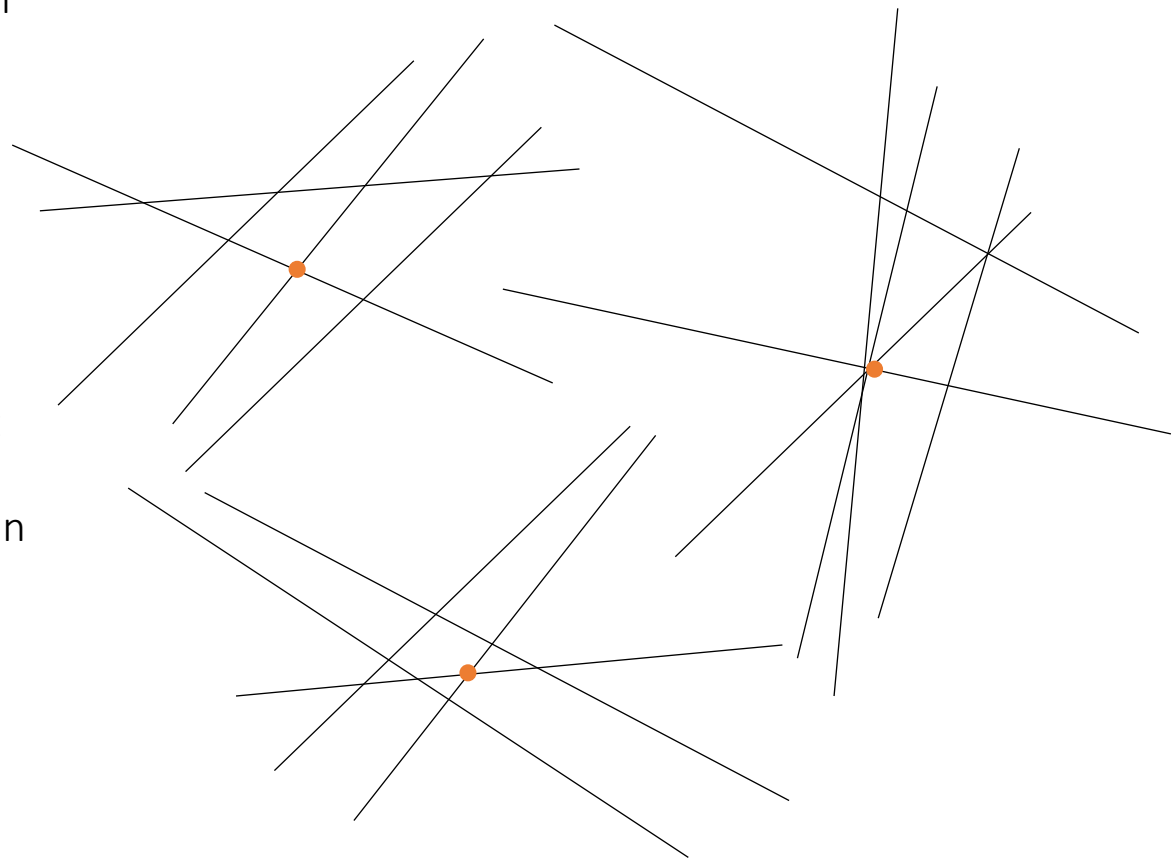
   ii. Set S to be the unit sphere that is centered at b.

   iii. Set $p(\ell') \coloneqq$ an arbitrary point in the pair $\ell' \cap S$.

   iv. Set $Q \coloneqq \{p(\ell') \mid \ell \in L\}$.

   v. Set $u \coloneqq \mathrm{Weighted} - \mathrm{Centers} - \mathrm{Sensitivity}(\mathrm{Q}, 2\mathrm{k})$

   vi. Set $s_b(\ell) = u\big(p(\ell')\big)$



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$:

i. Let ℓ' be the parallel line to ℓ that intersect b (the projection of ℓ onto b).

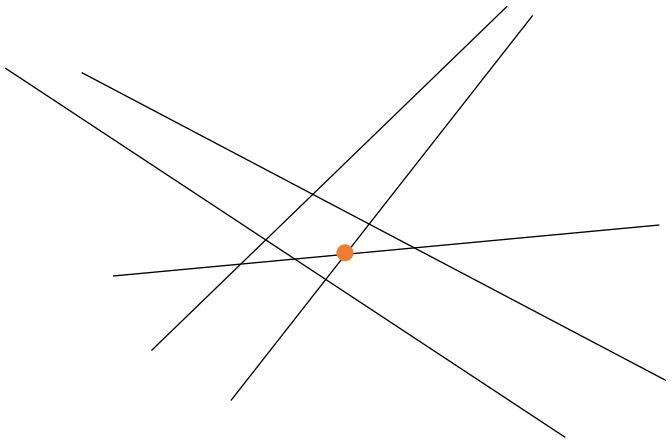
ii. Set S to be the unit sphere that is centered at b.

iii. Set $p(\ell') \coloneqq$ an arbitrary point in the pair $\ell' \cap S$.

iv. Set $Q \coloneqq \{p(\ell') \mid \ell \in L\}$.

v. Set $u \coloneqq \mathrm{Weighted} - \mathrm{Centers} - \mathrm{Sensitivity}(\mathrm{Q}, 2\mathrm{k})$

vi. Set $s_b(\ell) = u\big(p(\ell')\big)$



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$:

   i. Let ℓ' be the parallel line to ℓ that intersect b (the projection of ℓ onto b).

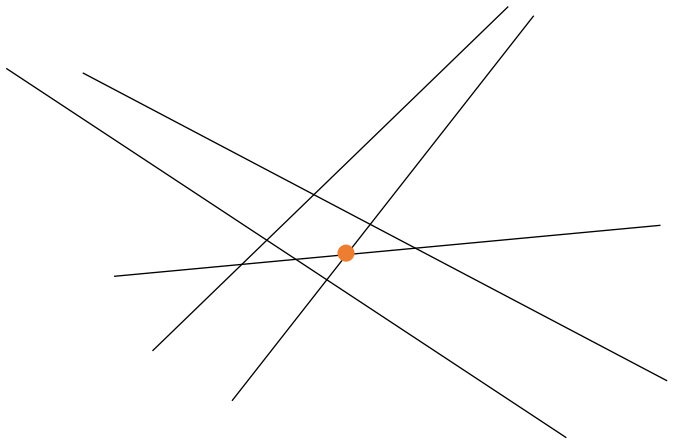
   ii. Set S to be the unit sphere that is centered at b.

   iii. Set $p(\ell') \coloneqq$ an arbitrary point in the pair $\ell' \cap S$.

   iv. Set $Q \coloneqq \{p(\ell') \mid \ell \in L\}$.

   v. Set $u \coloneqq \mathrm{Weighted} - \mathrm{Centers} - \mathrm{Sensitivity}(\mathrm{Q}, 2\mathrm{k})$

   vi. Set $s_b(\ell) = u\big(p(\ell')\big)$



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$:

   i. Let ℓ' be the parallel line to ℓ that intersect b (the projection of ℓ onto b).

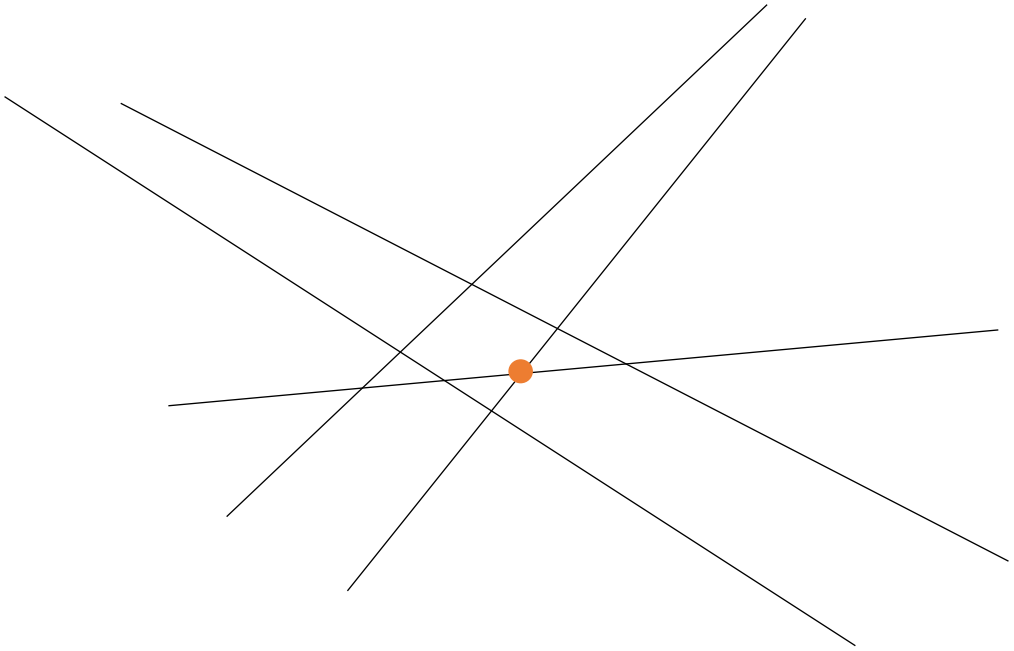
   ii. Set S to be the unit sphere that is centered at b.

   iii. Set $p(\ell') \coloneqq$ an arbitrary point in the pair $\ell' \cap S$.

   iv. Set $Q \coloneqq \{p(\ell') \mid \ell \in L\}$.

   v. Set $u \coloneqq \mathrm{Weighted} - \mathrm{Centers} - \mathrm{Sensitivity}(\mathrm{Q}, 2\mathrm{k})$

   vi. Set $s_b(\ell) = u\big(p(\ell')\big)$



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$:

i. Let ℓ' be the parallel line to ℓ that intersect b (the projection of ℓ onto b).

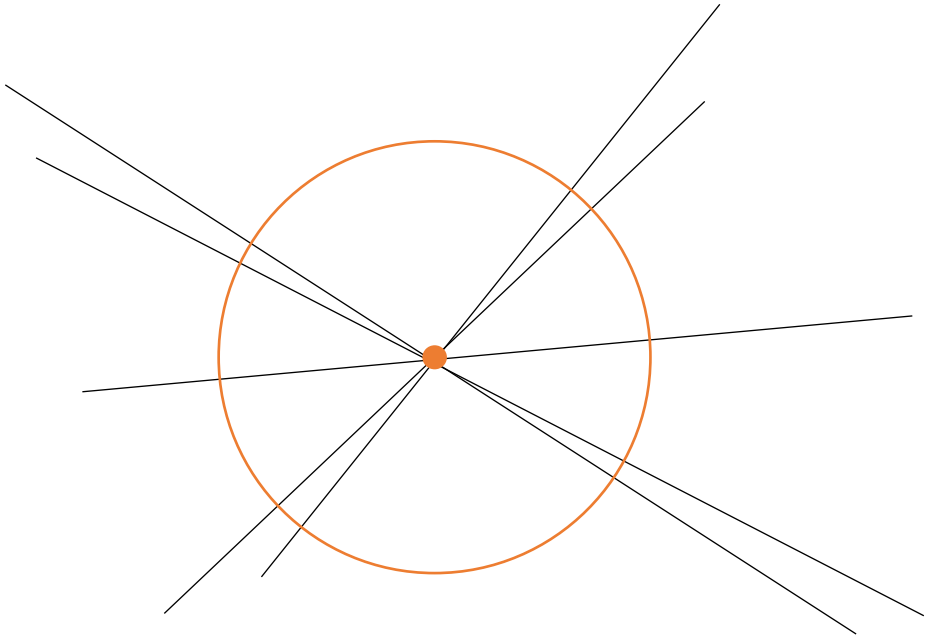
ii. Set S to be the unit sphere that is centered at b.

iii. Set $p(\ell') \coloneqq$ an arbitrary point in the pair $\ell' \cap S$.

iv. Set $Q \coloneqq \{p(\ell') \mid \ell \in L\}$.

v. Set $u \coloneqq \mathrm{Weighted} - \mathrm{Centers} - \mathrm{Sensitivity}(\mathrm{Q}, 2\mathrm{k})$

vi. Set $s_b(\ell) = u\big(p(\ell')\big)$



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$:

   i. Let ℓ' be the parallel line to ℓ that intersect b (the projection of ℓ onto b).

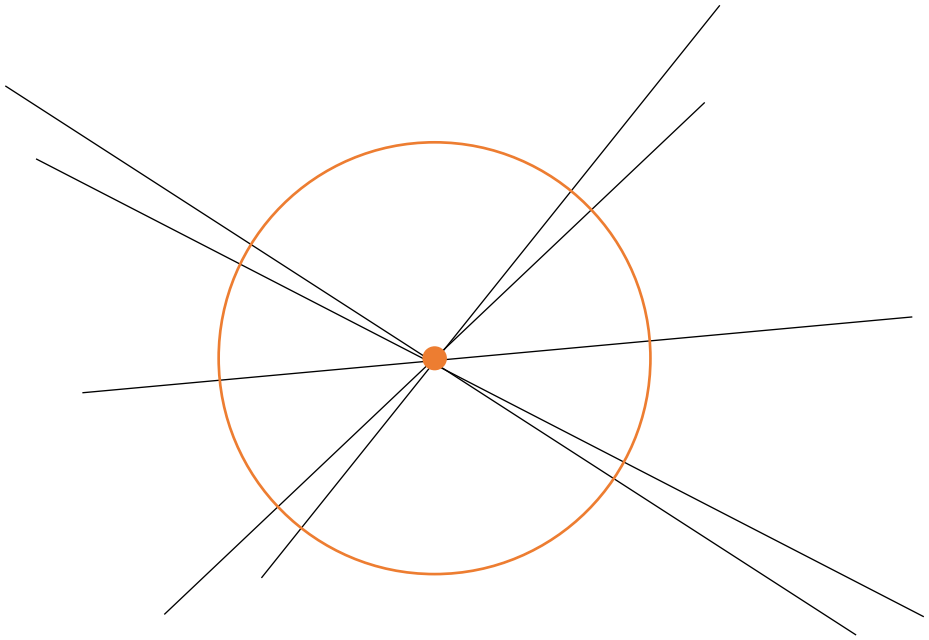
   ii. Set S to be the unit sphere that is centered at b.

   iii. Set $p(\ell') \coloneqq$ an arbitrary point in the pair $\ell' \cap S$.

   iv. Set $Q \coloneqq \{p(\ell') \mid \ell \in L\}$.

   v. Set $u \coloneqq \mathrm{Weighted} - \mathrm{Centers} - \mathrm{Sensitivity}(\mathrm{Q}, 2\mathrm{k})$

   vi. Set $s_b(\ell) = u\big(p(\ell')\big)$



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$:

   i. Let ℓ' be the parallel line to ℓ that intersect b (the projection of ℓ onto b).

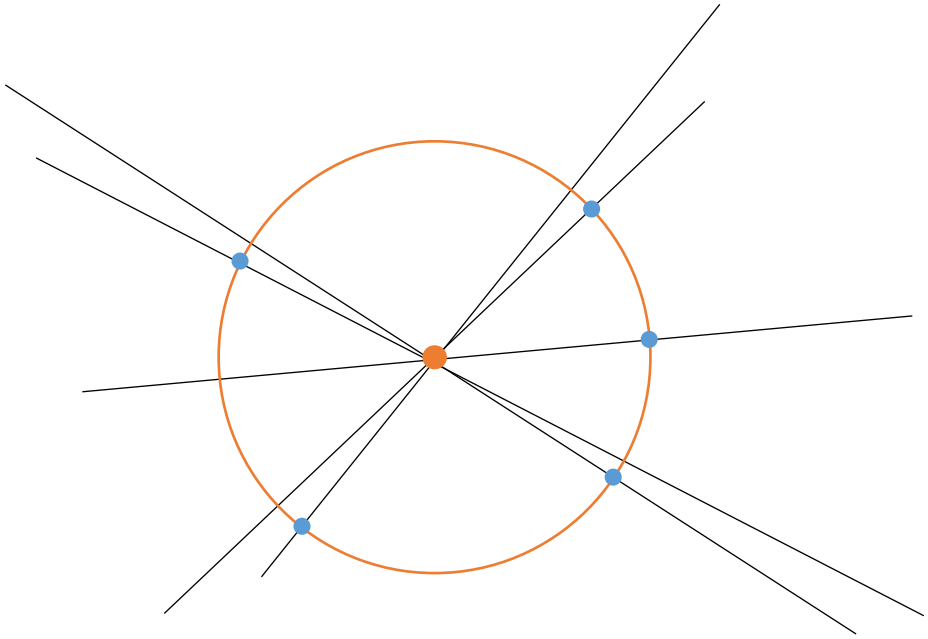
   ii. Set S to be the unit sphere that is centered at b.

   iii. Set $p(\ell') \coloneqq$ an arbitrary point in the pair $\ell' \cap S$.

   iv. Set $Q \coloneqq \{p(\ell') \mid \ell \in L\}$.

   v. Set $u \coloneqq \mathbf{Weighted - Centers - Sensitivity(Q, 2k)}$

   vi. Set $s_b(\ell) = u\big(p(\ell')\big)$



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$:

i. Let ℓ' be the parallel line to ℓ that intersect b (the projection of ℓ onto b).

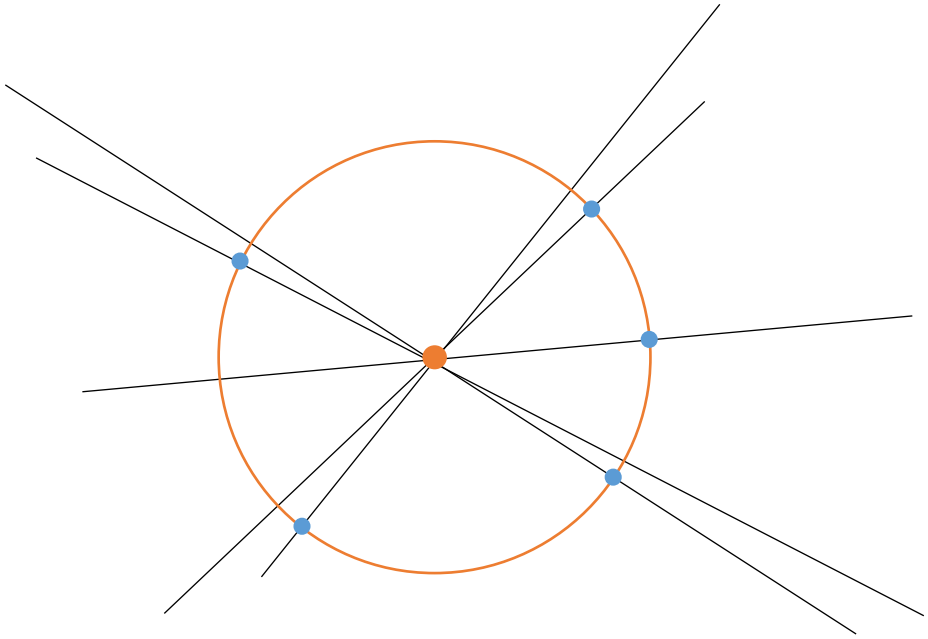
ii. Set S to be the unit sphere that is centered at b.

iii. Set $p(\ell') \coloneqq$ an arbitrary point in the pair $\ell' \cap S$.

iv. Set $Q \coloneqq \{p(\ell') \mid \ell \in L\}$.

v. Set $u \coloneqq \mathrm{Weighted} - \mathrm{Centers} - \mathrm{Sensitivity}(\mathrm{Q}, 2\mathrm{k})$

vi. Set $s_b(\ell) = u\big(p(\ell')\big)$



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$:

i. Let ℓ' be the parallel line to ℓ that intersect b (the projection of ℓ onto b).

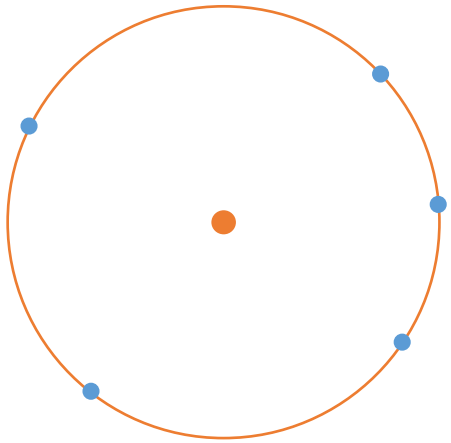
ii. Set S to be the unit sphere that is centered at b.

iii. Set $p(\ell') \coloneqq$ an arbitrary point in the pair $\ell' \cap S$.

iv. Set $Q \coloneqq \{p(\ell') \mid \ell \in L\}$.

v. Set $u \coloneqq \mathrm{Weighted} - \mathrm{Centers} - \mathrm{Sensitivity}(\mathrm{Q}, 2\mathrm{k})$

vi. Set $s_b(\ell) = u\big(p(\ell')\big)$



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$:

   i. Let ℓ' be the parallel line to ℓ that intersect b (the projection of ℓ onto b).

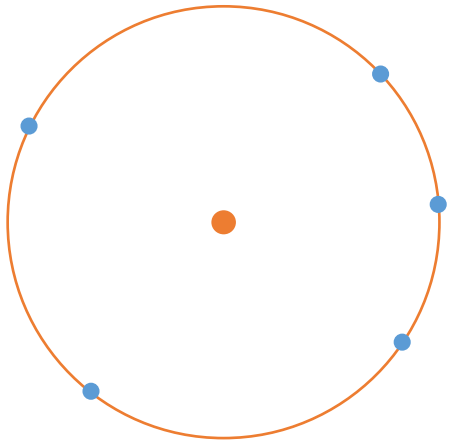
   ii. Set S to be the unit sphere that is centered at b.

   iii. Set $p(\ell') \coloneqq$ an arbitrary point in the pair $\ell' \cap S$.

   iv. Set $Q \coloneqq \{p(\ell') \mid \ell \in L\}$.

   v. Set $u \coloneqq \text{Weighted} - \text{Centers} - \text{Sensitivity}(\text{Q}, 2\text{k})$

   vi. Set $s_b(\ell) = u\big(p(\ell')\big)$



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$:

   i. Let ℓ' be the parallel line to ℓ that intersect b (the projection of ℓ onto b).

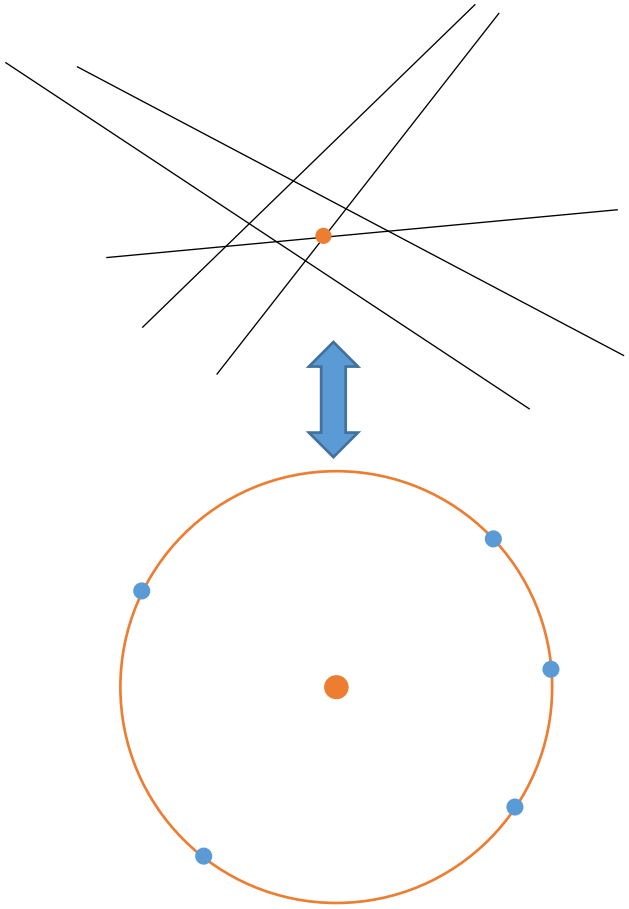
   ii. Set S to be the unit sphere that is centered at b.

   iii. Set $p(\ell') \coloneqq$ an arbitrary point in the pair $\ell' \cap S$.

   iv. Set $Q \coloneqq \{p(\ell') \mid \ell \in L\}$.

   v. Set $u \coloneqq \text{Weighted} - \text{Centers} - \text{Sensitivity}(Q, 2k)$

   vi. Set $s_b(\ell) = u(p(\ell'))$



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$.

(3) For every $\ell \in L$ and its closest point $b \in B$, compute final sensitivity

$$s(\ell) = \frac{dist(\ell, b)}{\sum_{\ell' \in L} dist(\ell', b)} + 2s_b(\ell)$$

(4) For every line $\ell \in L$, define

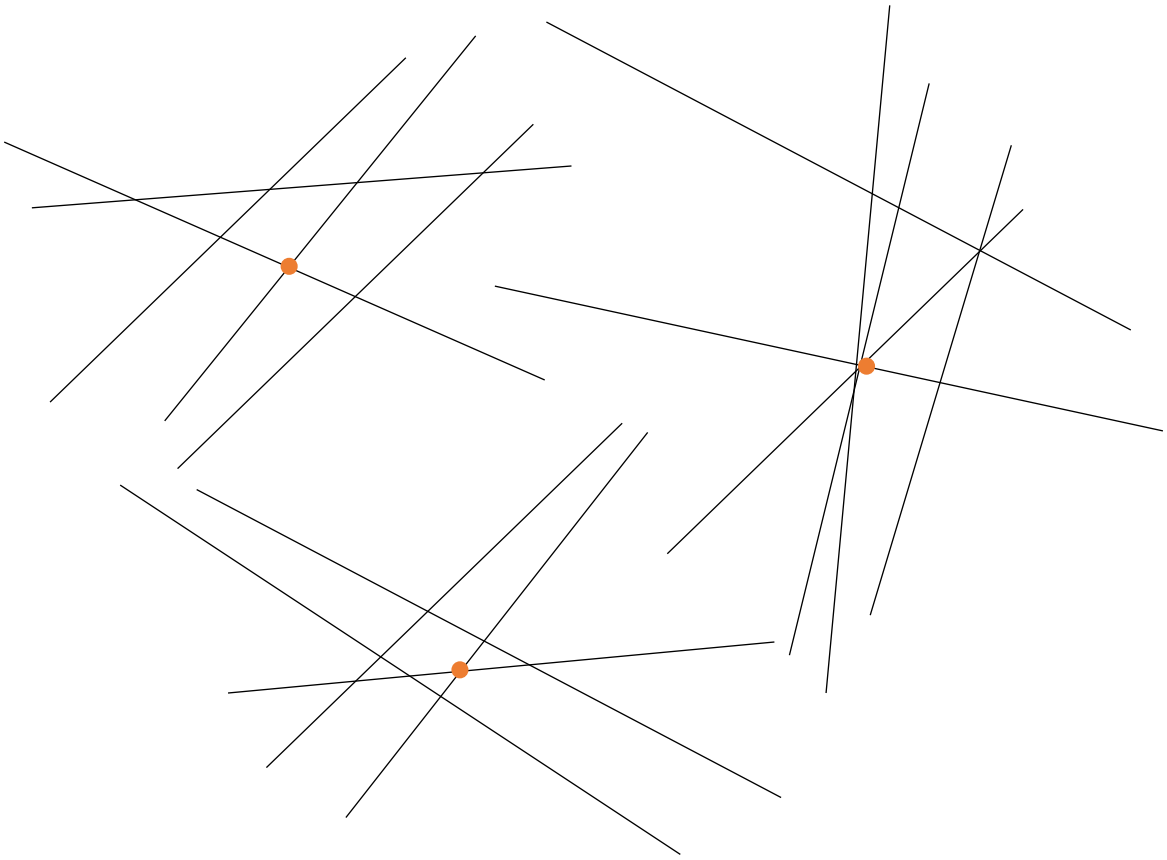
$$prob(\ell) = \frac{s(\ell)}{\sum_{\ell' \in L} s(\ell')}.$$

(5) Pick a sample S consist of at least m lines from L with probability $prob(\ell)$.

(6) For every $\ell \in S$ define

$$u(\ell) = \frac{1}{|S| prob(\ell)}.$$

(7) Return (S, u).



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$.

(3) For every $\ell \in L$ and its closest point $b \in B$, compute final sensitivity

$$s(\ell) = \frac{dist(\ell, b)}{\sum_{\ell' \in L} dist(\ell', B)} + 2s_b(\ell)$$

(4) For every line $\ell \in L$, define

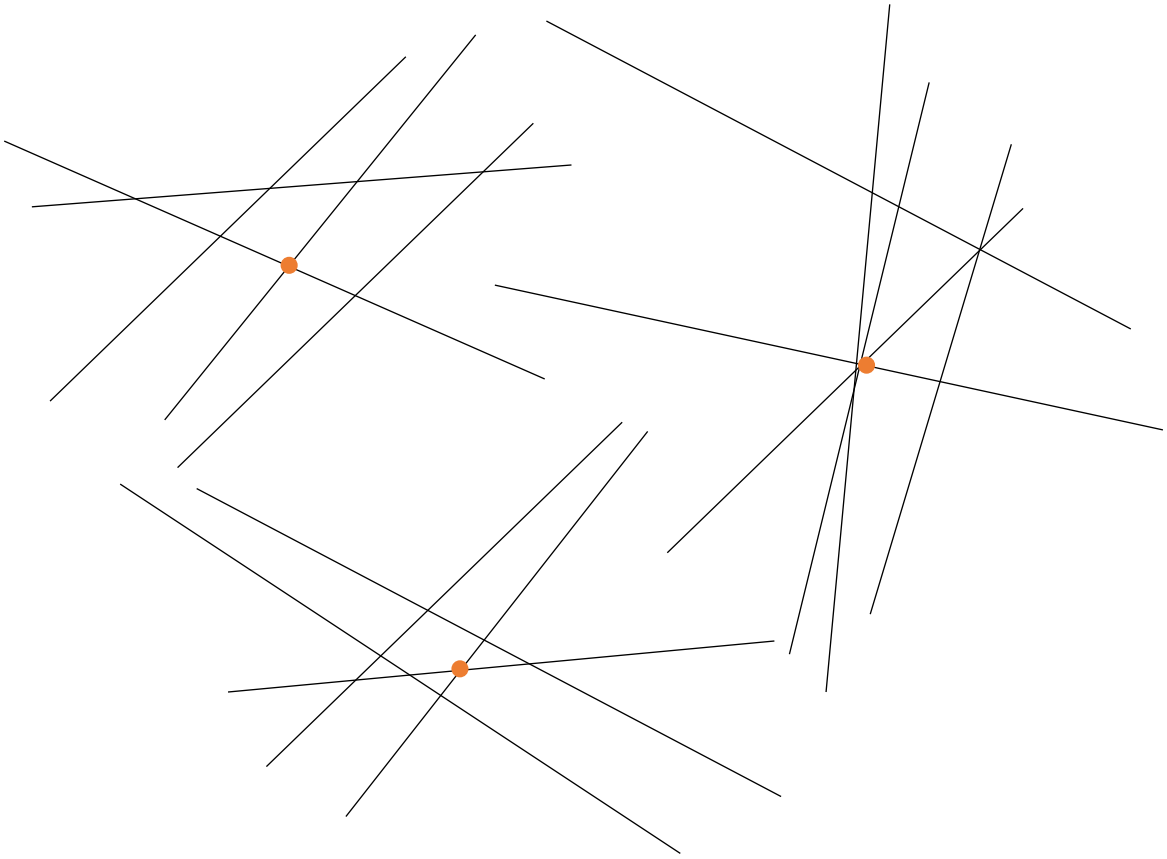
$$prob(\ell) = \frac{s(\ell)}{\sum_{\ell' \in L} s(\ell')}.$$

(5) Pick a sample S consist of at least m lines from L with probability $prob(\ell)$.

(6) For every $\ell \in S$ define

$$u(\ell) = \frac{1}{|S| prob(\ell)}.$$

(7) Return (S, u).



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$.

(3) For every $\ell \in L$ and its closest point $b \in B$, compute final sensitivity

$$s(\ell) = \frac{dist(\ell, b)}{\sum_{\ell' \in L} dist(\ell', B)} + 2s_b(\ell)$$

(4) For every line $\ell \in L$, define

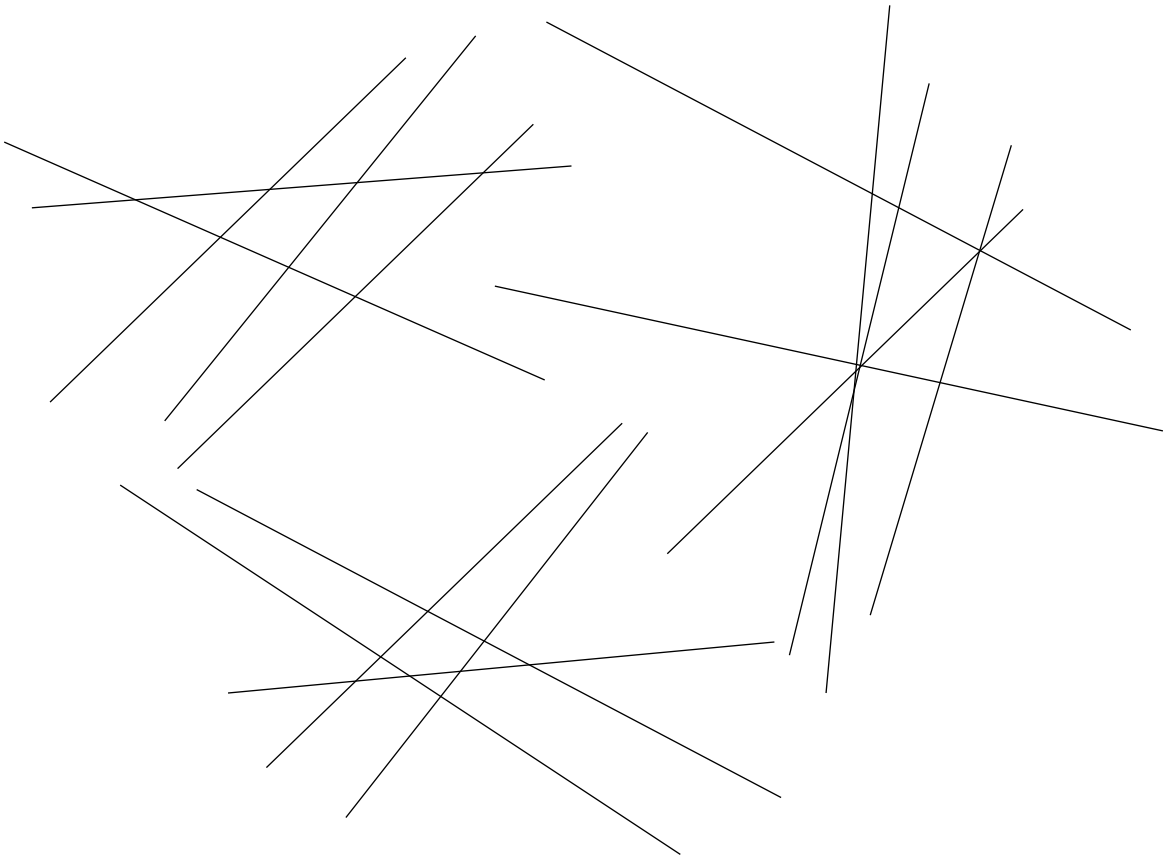
$$prob(\ell) = \frac{s(\ell)}{\sum_{\ell' \in L} s(\ell')}.$$

(5) Pick a sample S consist of at least m lines from L with probability $prob(\ell)$.

(6) For every $\ell \in S$ define

$$u(\ell) = \frac{1}{|S| prob(\ell)}.$$

(7) Return (S, u).



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$.

(3) For every $\ell \in L$ and its closest point $b \in B$, compute final sensitivity

$$s(\ell) = \frac{dist(\ell, b)}{\sum_{\ell' \in L} dist(\ell', B)} + 2s_b(\ell)$$

(4) For every line $\ell \in L$, define

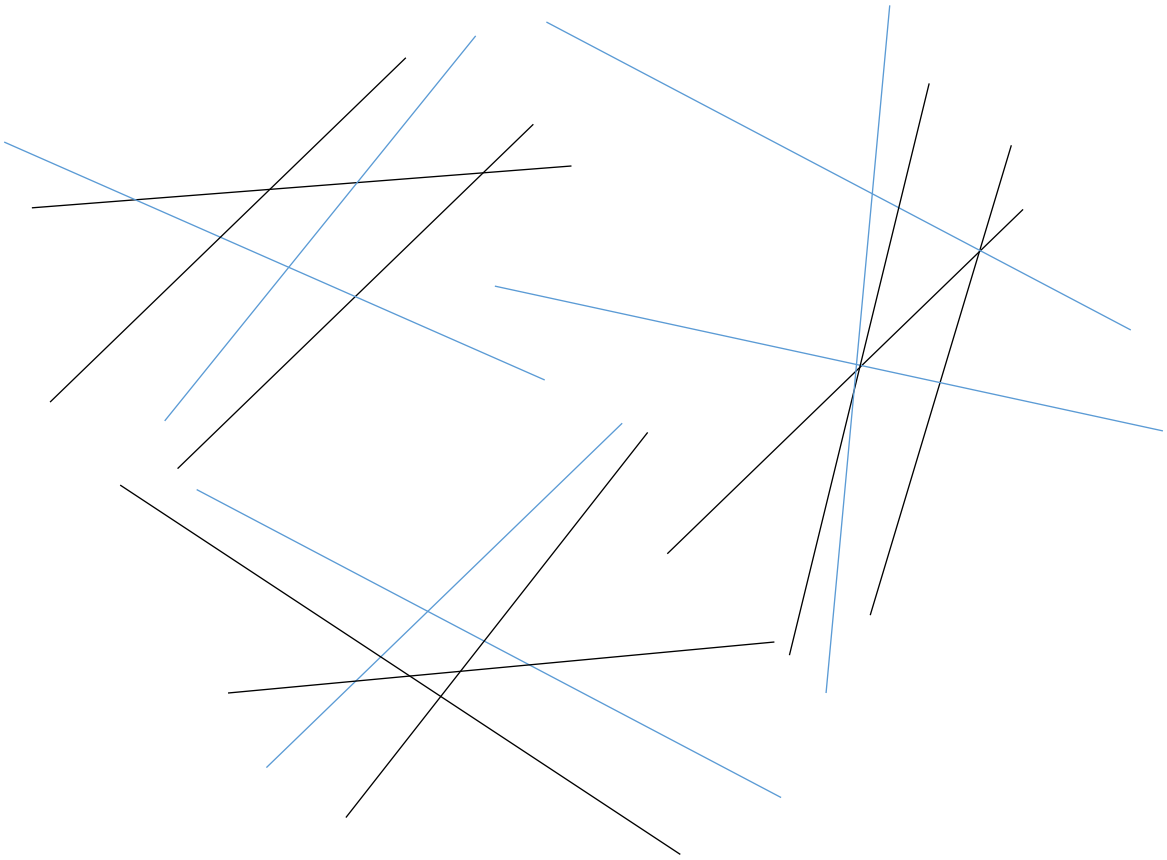
$$prob(\ell) = \frac{s(\ell)}{\sum_{\ell' \in L} s(\ell')}.$$

(5) Pick a sample S consist of at least m lines from L with probability $prob(\ell)$.

(6) For every $\ell \in S$ define

$$u(\ell) = \frac{1}{|S| prob(\ell)}.$$

(7) Return (S, u).



Coreset for k lines means

Algorithm:

(1) Compute $B \coloneqq (\alpha, \beta)$-approximation for the k-means of L.

(2) Cluster each $\ell \in L$ to its closest $b \in B$ and compute its partial sensitivity $s_b(\ell)$.

(3) For every $\ell \in L$ and its closest point $b \in B$, compute final sensitivity

$$s(\ell) = \frac{dist(\ell, b)}{\sum_{\ell' \in L} dist(\ell', B)} + 2s_b(\ell)$$

(4) For every line $\ell \in L$, define

$$prob(\ell) = \frac{s(\ell)}{\sum_{\ell' \in L} s(\ell')}.$$

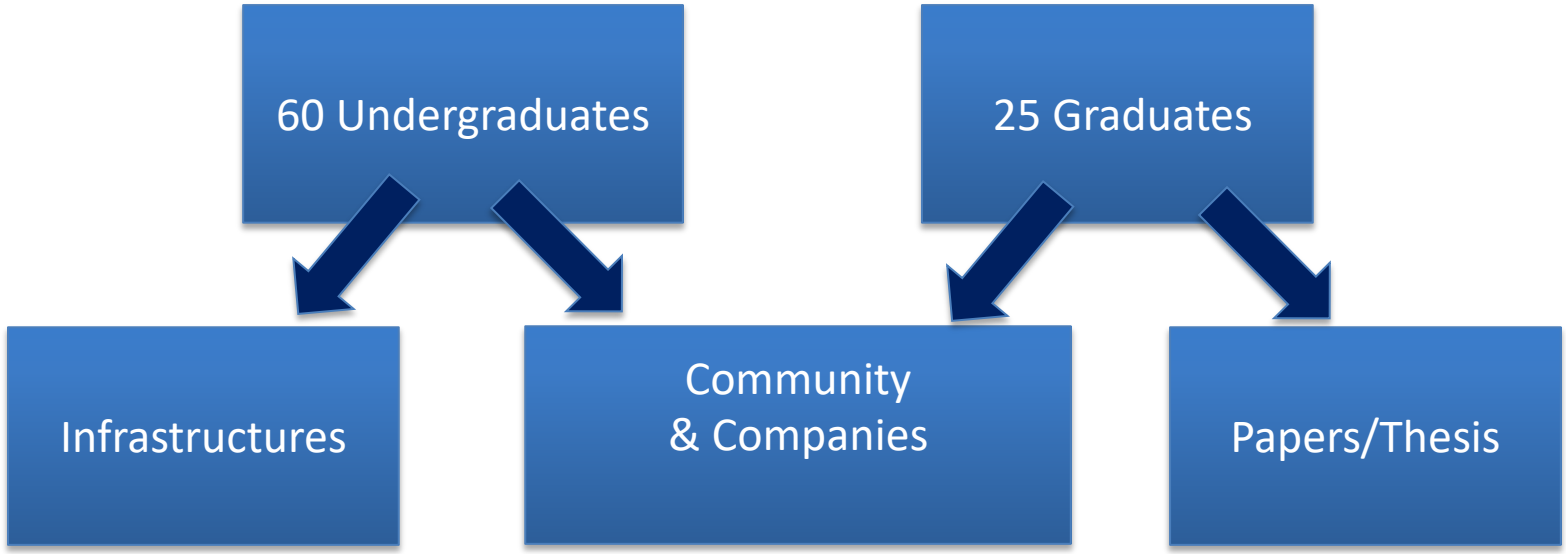
(5) Pick a sample S consist of at least m lines from L with probability $prob(\ell)$.

(6) For every $\ell \in S$ define

$$u(\ell) = \frac{1}{|S| prob(\ell)}.$$

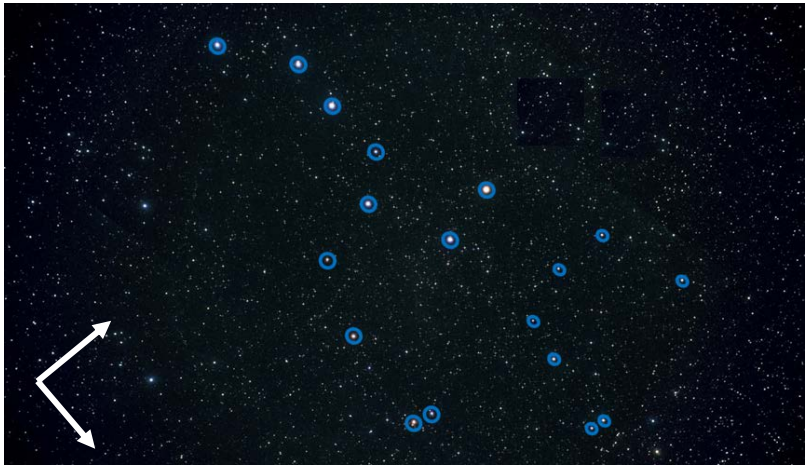
(7) Return (S, u).

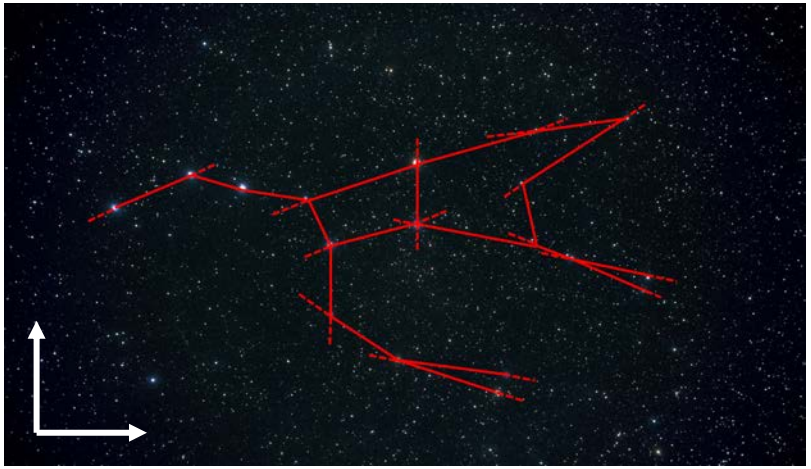
Thank you and the RBD students!



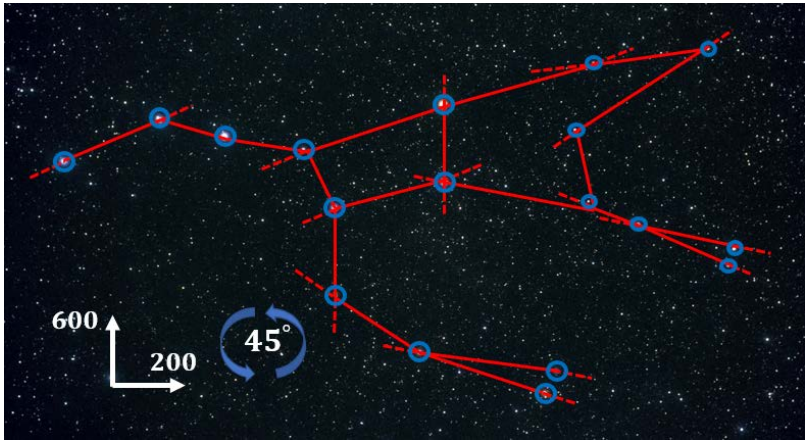
אוניברסיטת חיפה
University of Haifa

Localization Using Sky Patterns





(R, t)



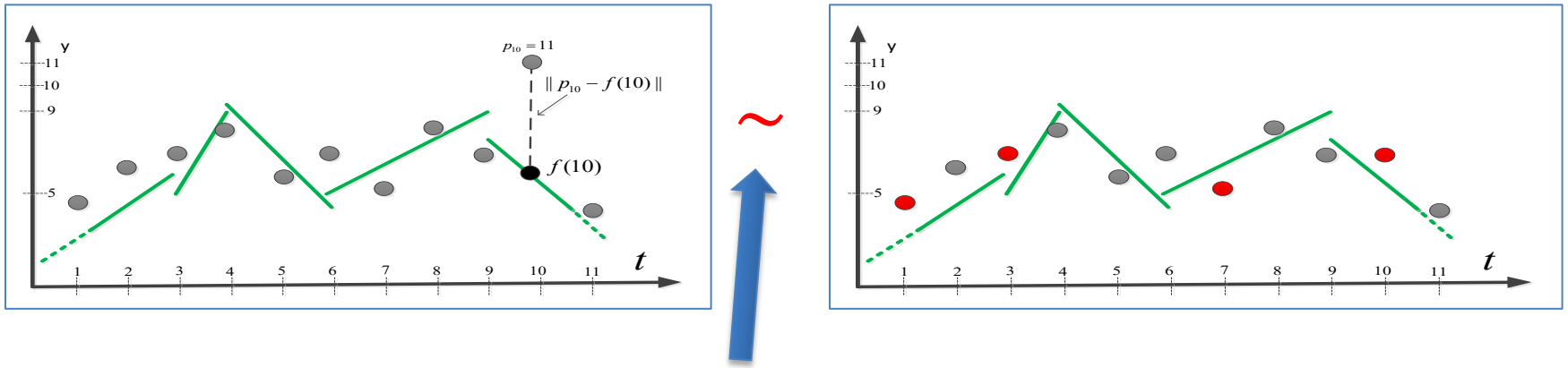
Input: *(Left) A set* $P = \{p_1, \dots, p_n\}$ *of observed stars of Ursa Major. (Top right) Approximation of the known model of Ursa Major by a set* $L = \{\ell_1, \dots, \ell_n\}$ *of lines.*

Output: *(Bottom right) Rotation and Translation* (R, t) *that minimize* $\sum_{i \in [n]} dist(Rp_i - t, \ell_i)$.

From Big Data to Small Data

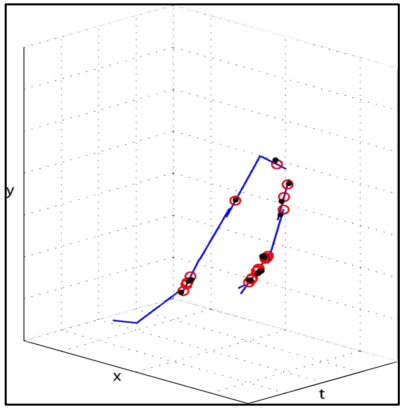
Suppose that we can compute such a corset C of size $\frac{1}{\epsilon}$ for every set P of n points

- in time n^5,
- off-line, non-parallel, non-streaming algorithm



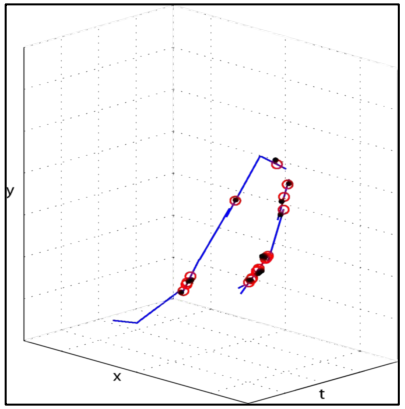
Read the first $\frac{2}{\epsilon}$ streaming points and reduce them into $\frac{1}{\epsilon}$ weighted points in time $\left(\frac{2}{\epsilon}\right)^5$

$1+\epsilon$ corset for P_1

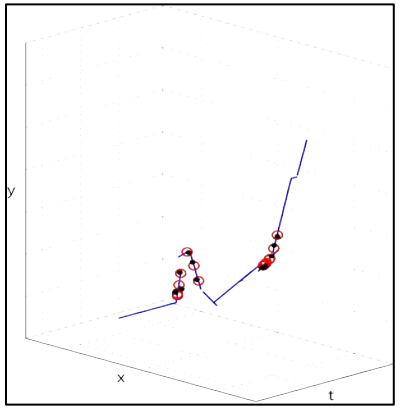


Read the next $\frac{2}{\epsilon}$ streaming point and reduce them into $\frac{1}{\epsilon}$ weighted points in time $\left(\frac{2}{\epsilon}\right)^5$

$1+\epsilon$ corset for P_1

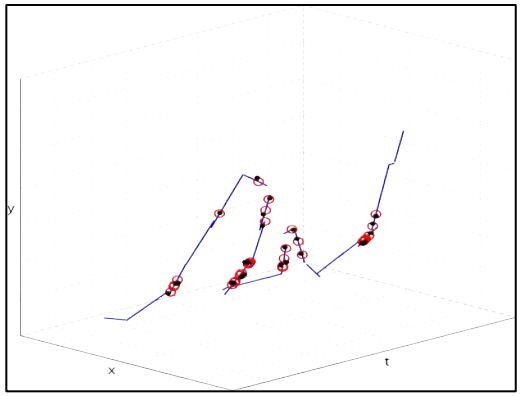


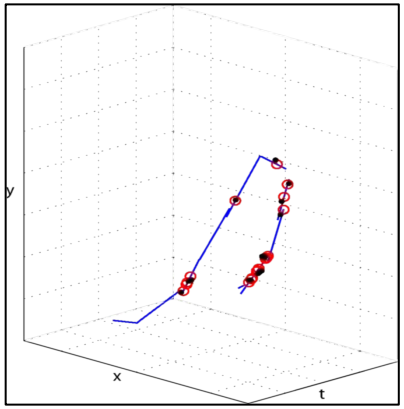
$1+\epsilon$ corset for P_2

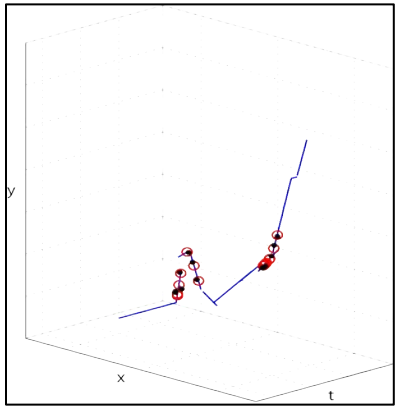


Merge the pair of ϵ-coresets into an ϵ-corset of $\frac{2}{\epsilon}$ weighted points

$1+\epsilon$-corset for $P_1 \cup P_2$

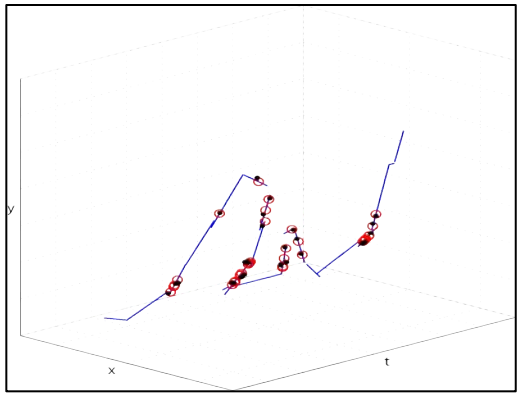


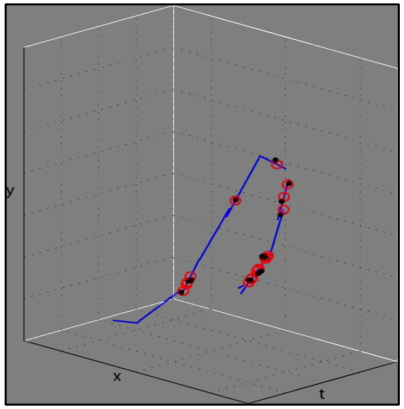


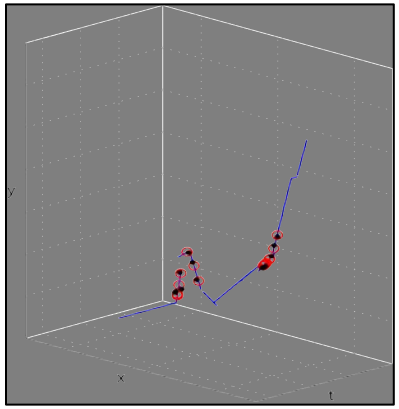


Delete the pair of original coresets from memory

$1+\epsilon$-corset for $P_1 \cup P_2$



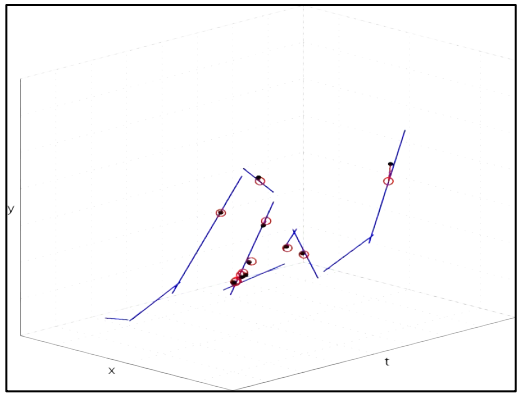


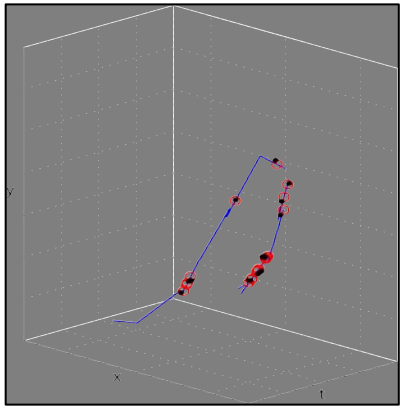


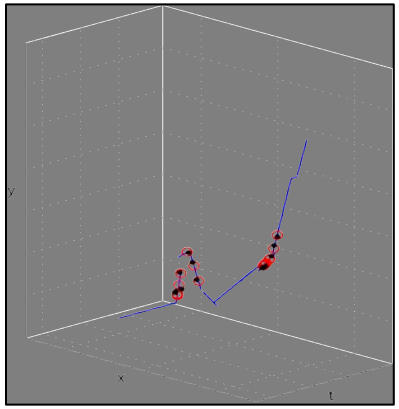
Reduce the $\frac{2}{\epsilon}$ weighted points into $\frac{1}{\epsilon}$ weighted points by constructing their coreset

$1+\epsilon$-corset for

$1+\epsilon$-corset for $P_1 \cup P_2$



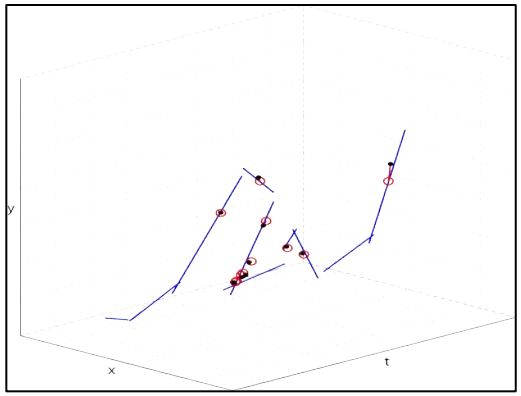




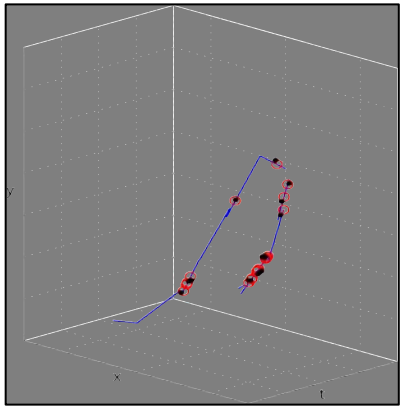
Reduce the $\frac{2}{\epsilon}$ weighted points into $\frac{1}{\epsilon}$ weighted points by constructing their coreset

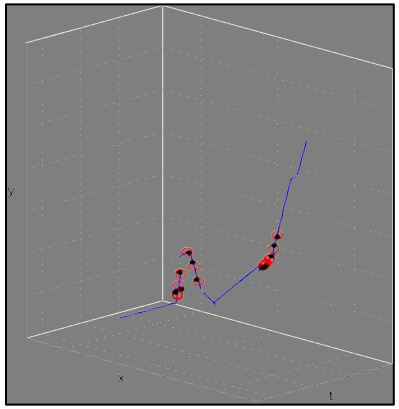
$1+\epsilon$-corset for

$1+\epsilon$-corset for $P_1 \cup P_2$

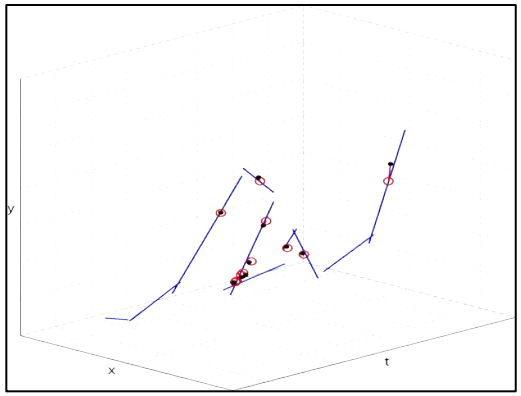


$=(1+\epsilon)^2$-corset for $P_1 \cup P_2$

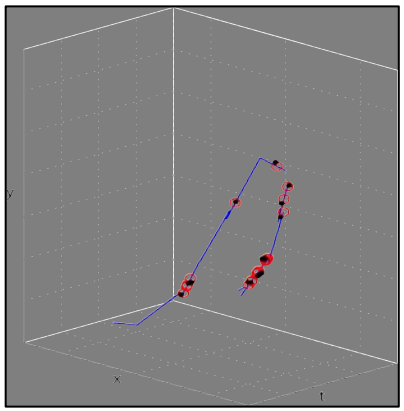


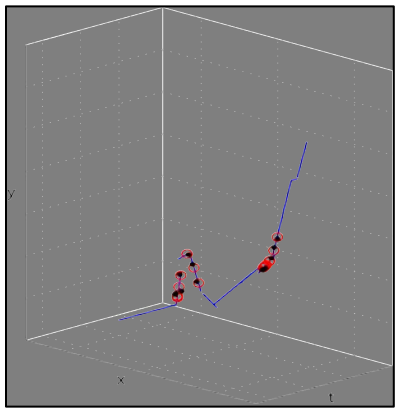


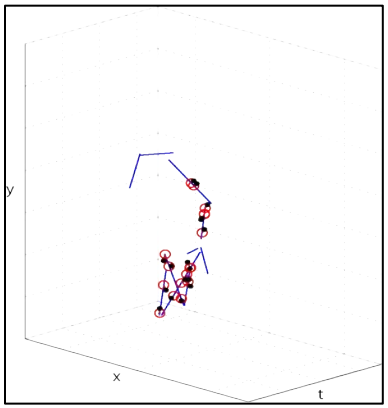
$(1+\epsilon)^2$-corset for $P_1 \cup P_2$



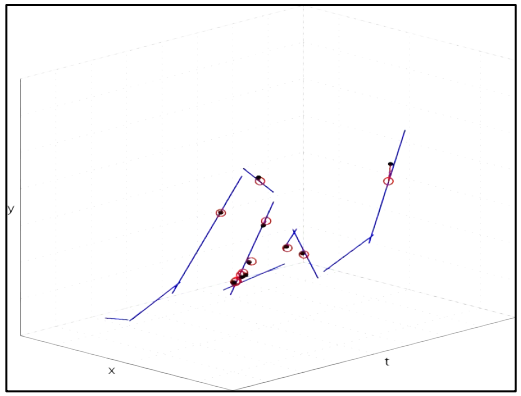
$(1+\epsilon)$-corset for P_3



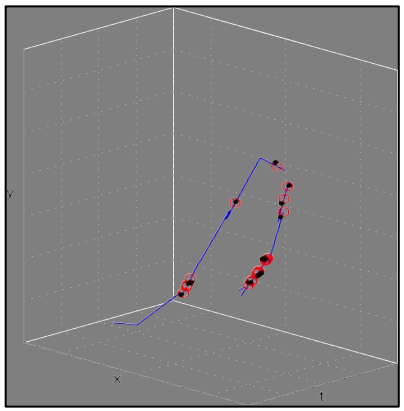


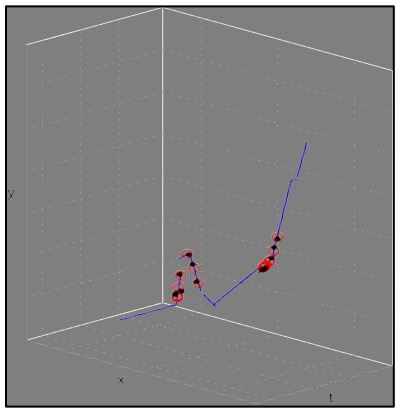


$(1+\epsilon)^2$-corset for $P_1 \cup P_2$

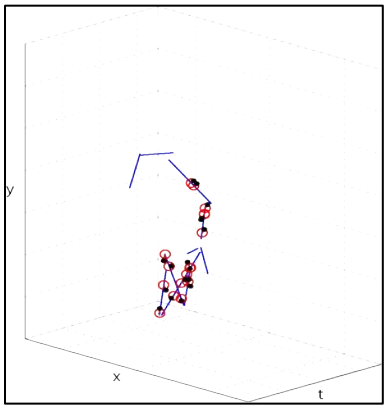


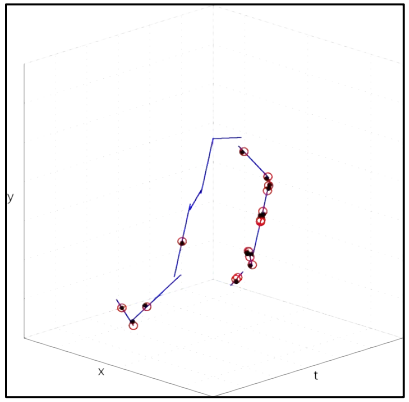
$(1+\epsilon)$-corset for P_3



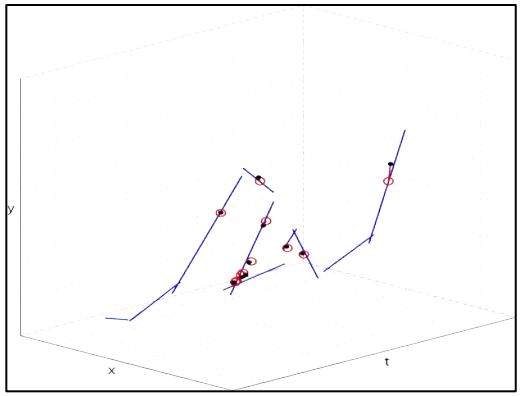


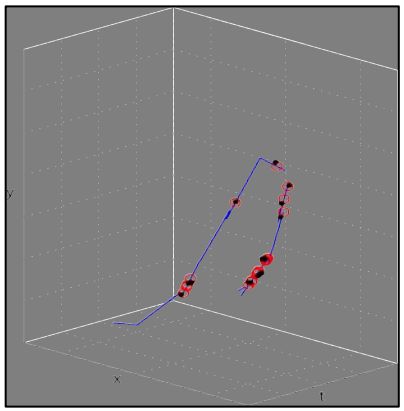
$(1+\epsilon)$-corset for P_4

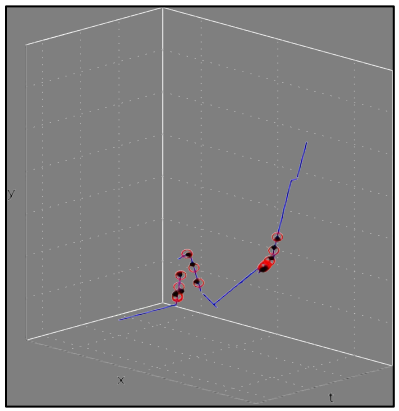




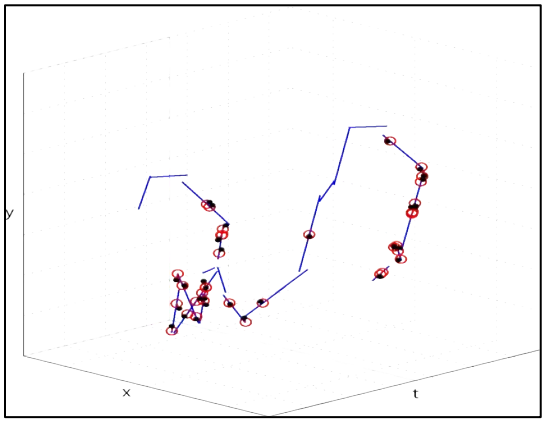
$(1+\epsilon)^2$-corset for $P_1 \cup P_2$

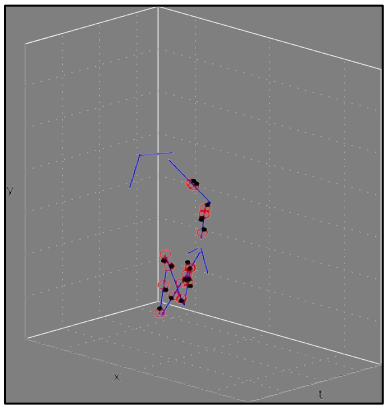


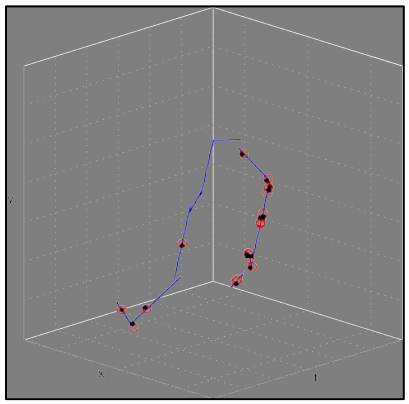




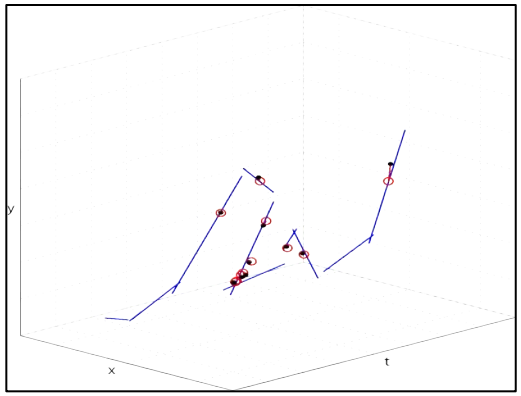
$(1+\epsilon)$-corset for $P_3 \cup P_4$

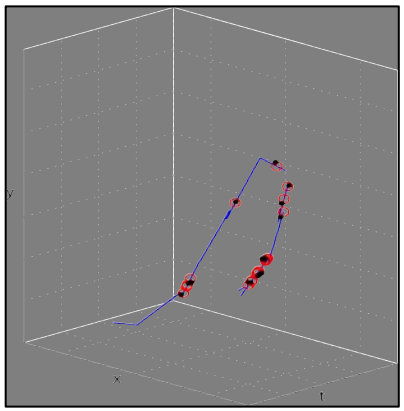


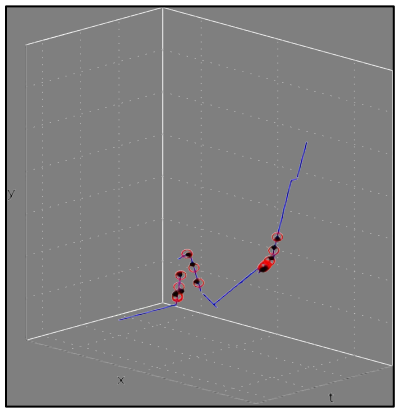




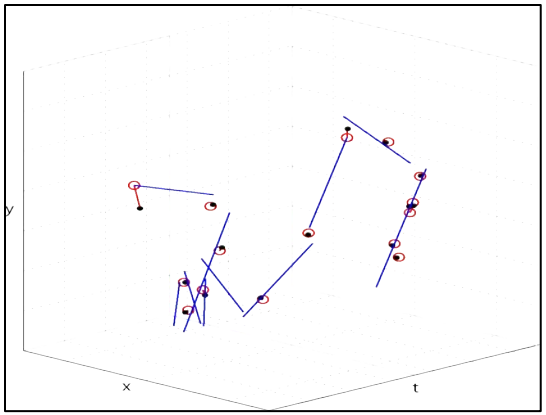
$(1+\epsilon)^2$-corset for $P_1 \cup P_2$

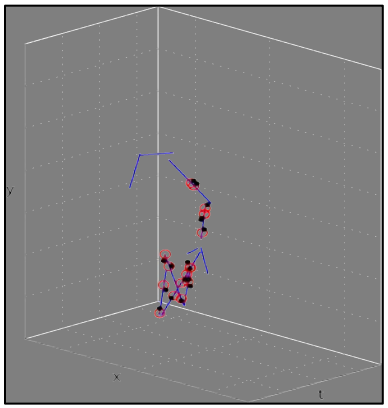


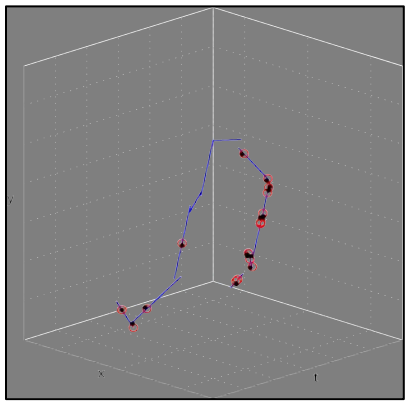


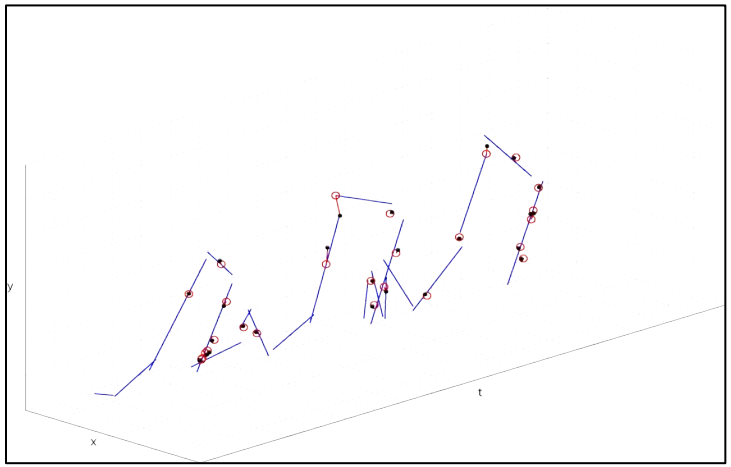


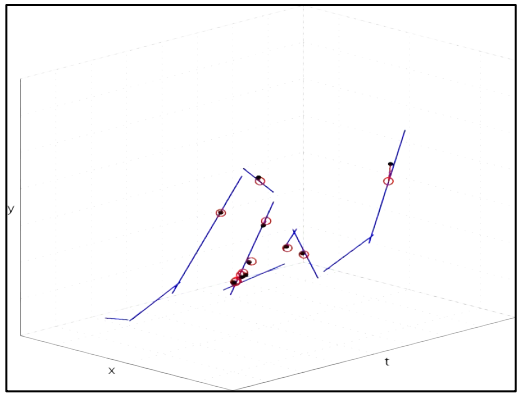
$(1+\epsilon)^2$-corset for $P_3 \cup P_4$

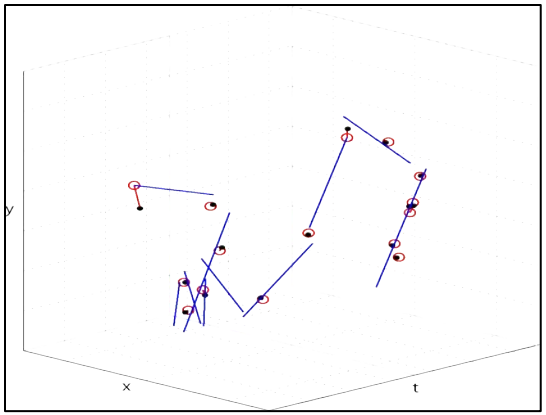


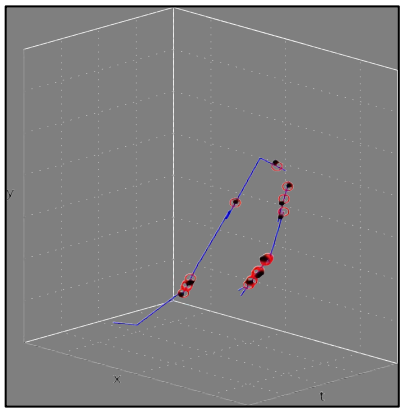


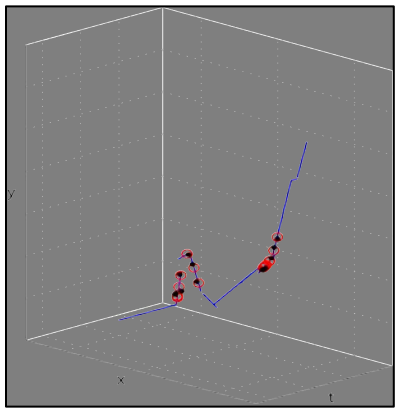


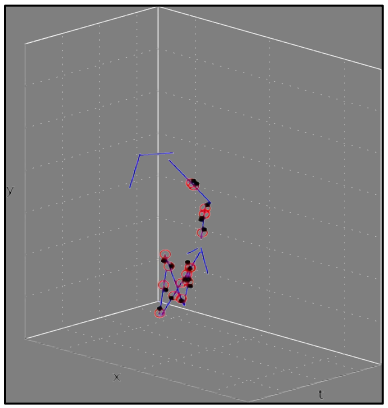

y
x
t

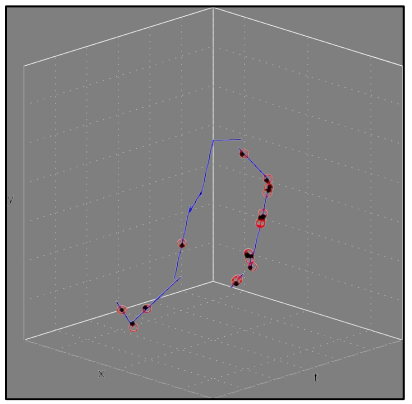

y
x
t


y
x
t


y
x
t

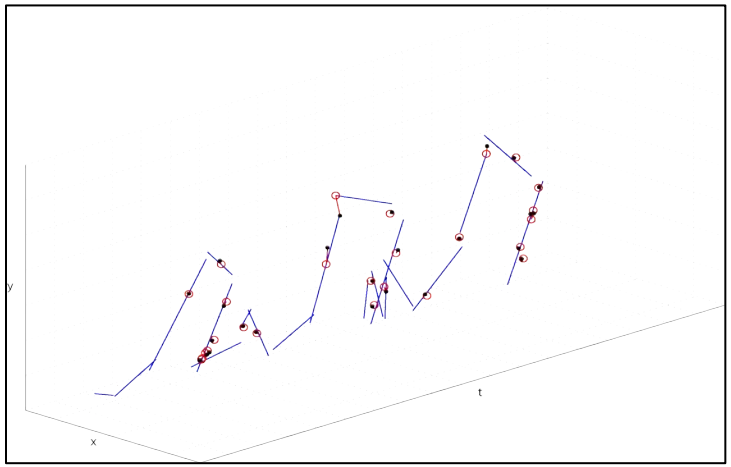

y
x
t

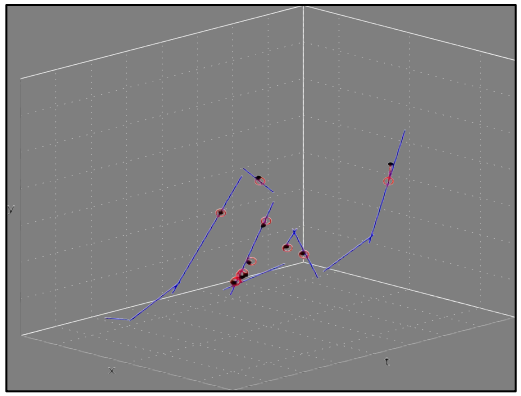

y
x
t

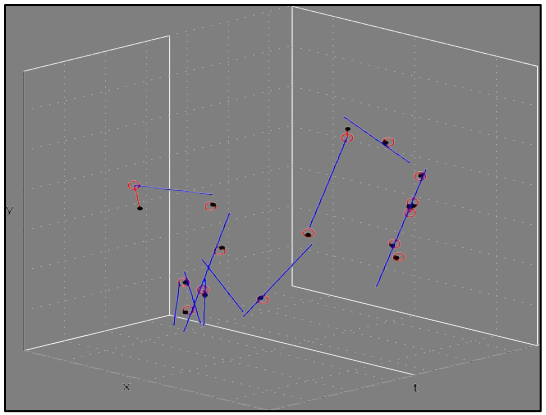

y
x
t

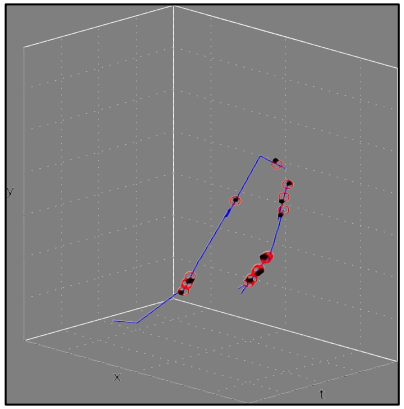
$(1+\epsilon)^2$-coreset for

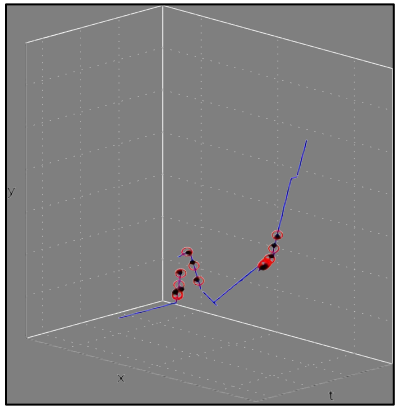
$P_1 \cup P_2 \cup P_3 \cup P_4$

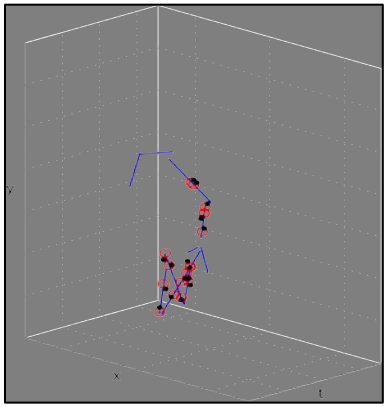


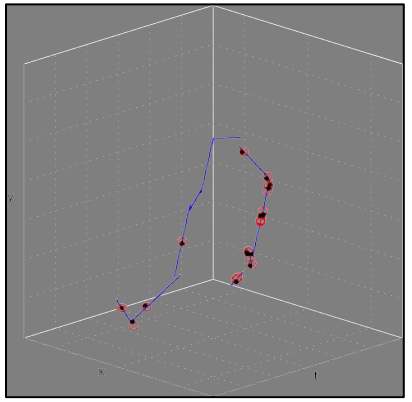






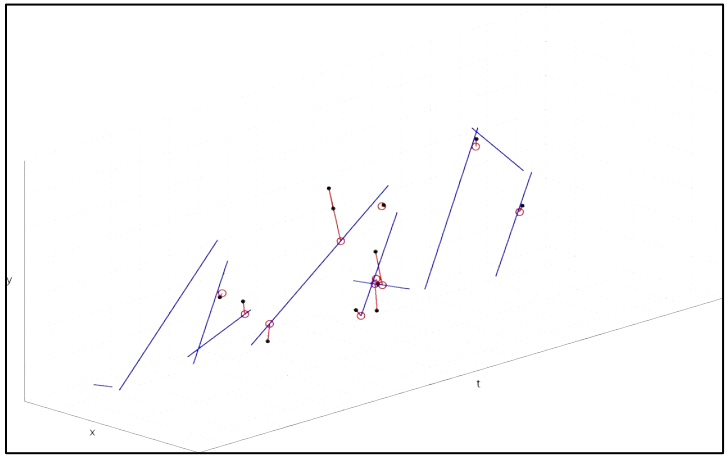


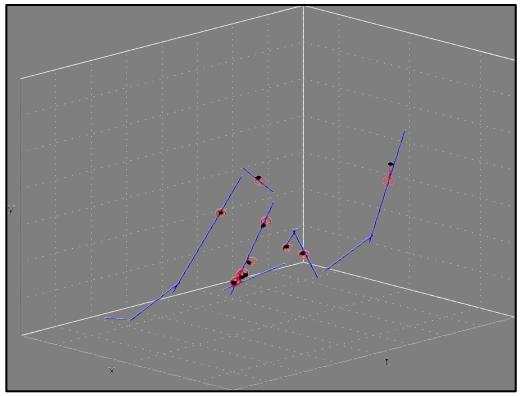


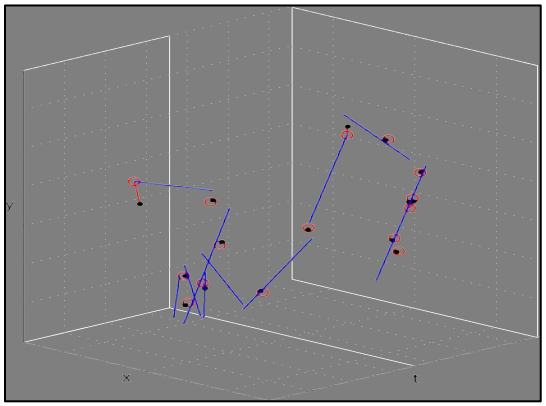


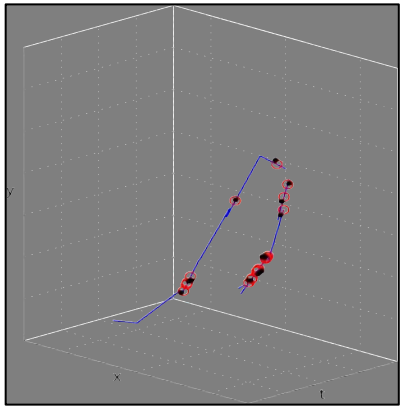
$(1+\epsilon)^3$-coreset for

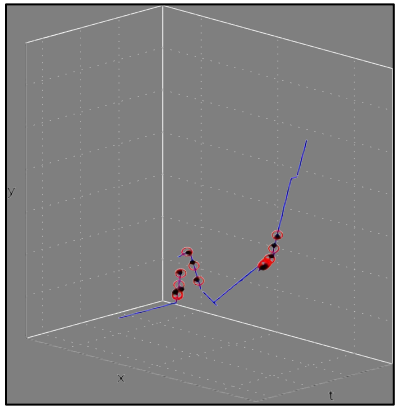
$P_1 \cup P_2 \cup P_3 \cup P_4$

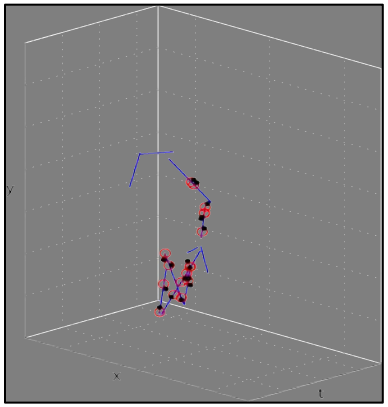


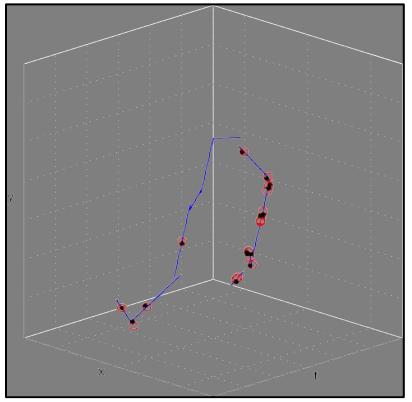


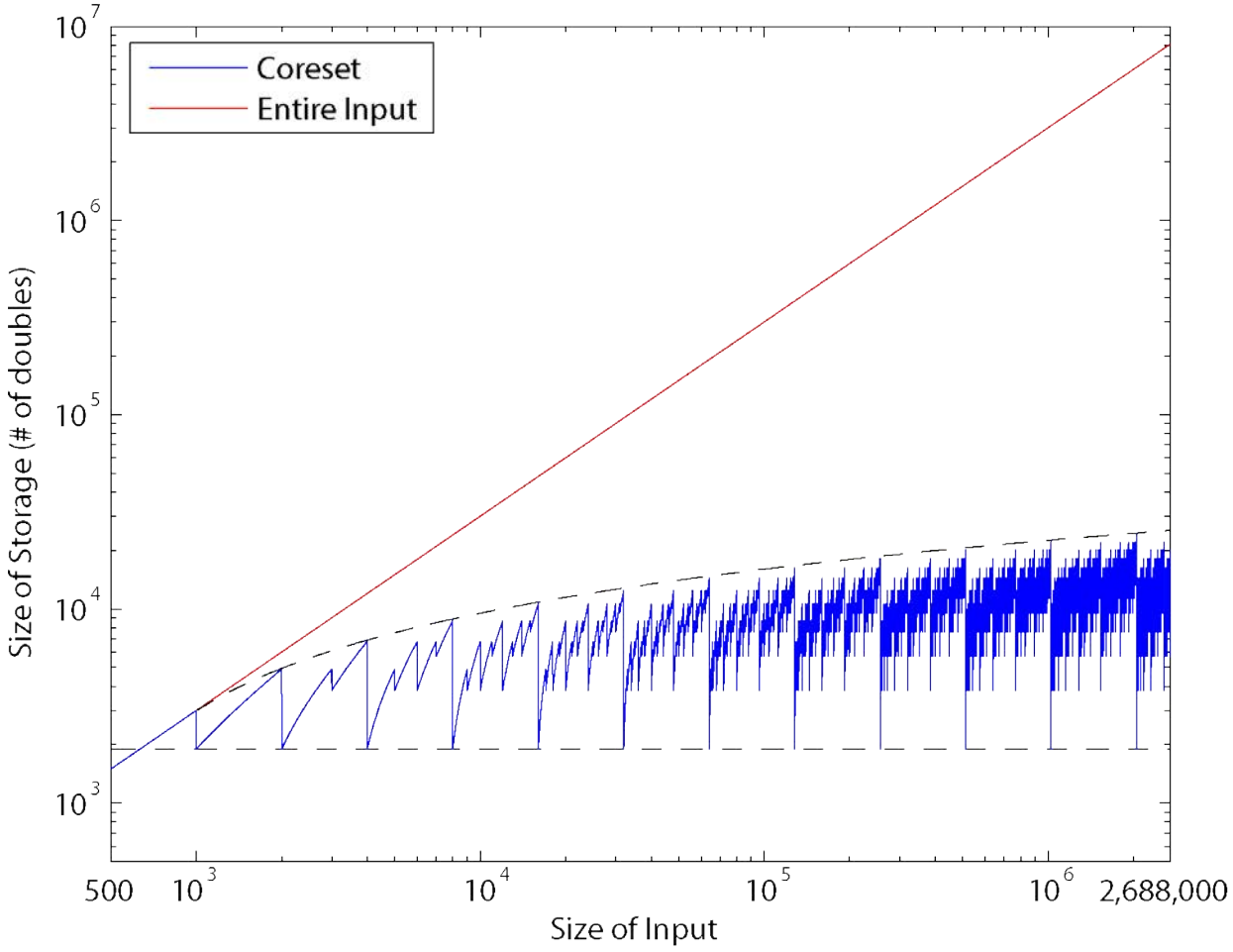




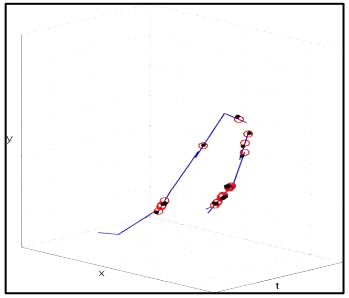


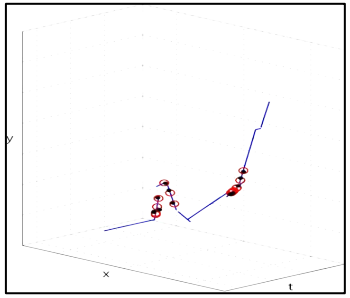


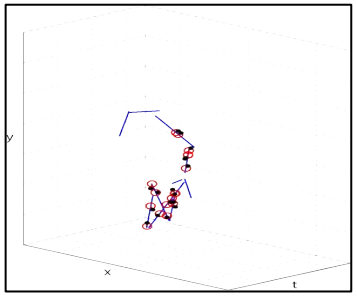


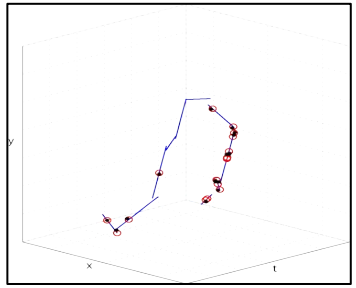

Coreset
Entire Input
Size of Storage (# of doubles)
10^7
10^6
10^5
10^4
10^3
500
10^3
10^4
10^5
10^6
2,688,000
Size of Input

Parallel Computation

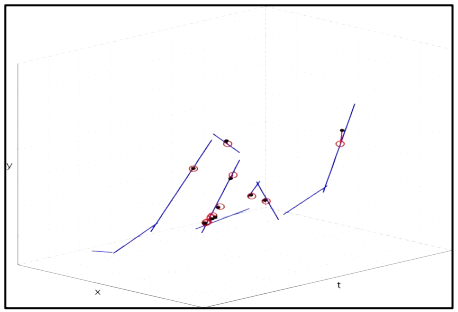


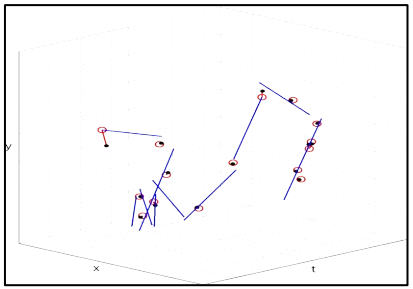


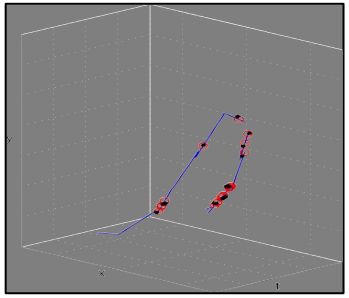


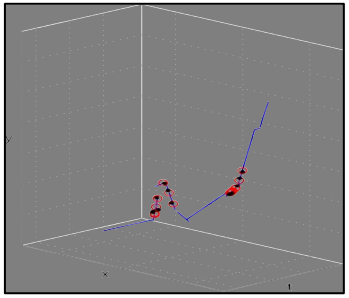


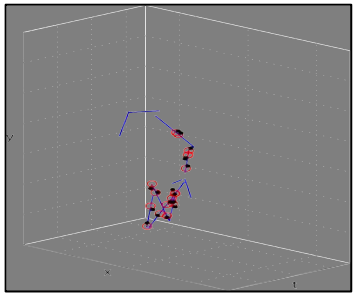
Parallel Computation

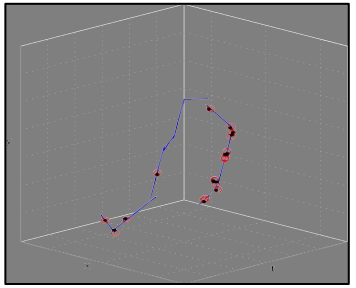






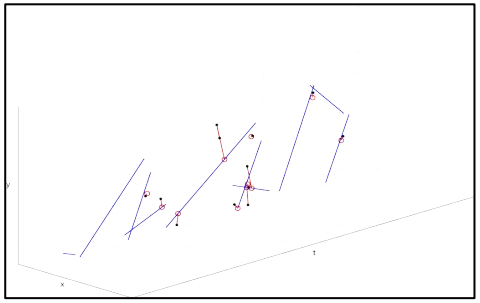


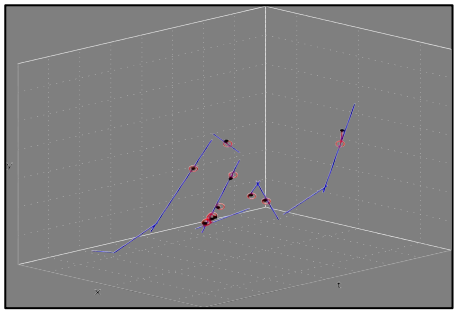


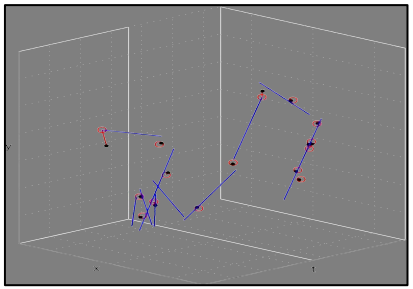


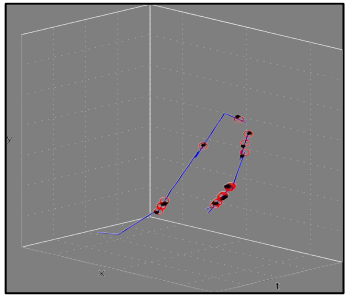
Parallel Computation

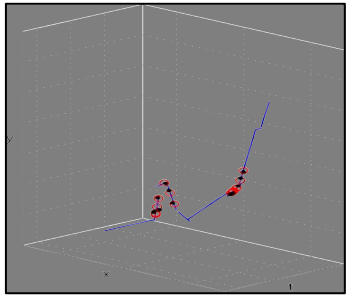
Run off-line algorithm on corset using single computer

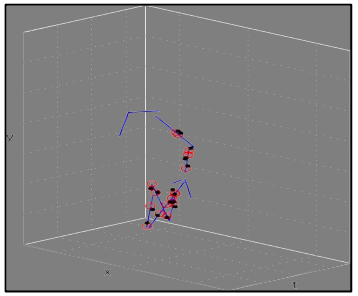


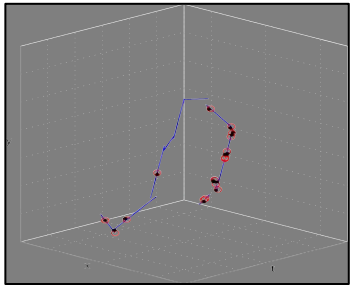












Parallel+ Streaming Computation