Sampling a Neighbor in High Dimensions

Who is the fairest of them all?

Sepideh Mahabadi
MSR Redmond

Joint work with

Martin Aumüller
IT University of Copenhagen

Sariel Har-Peled
UIUC

Rasmus Pagh
BARC and University of Copenhagen

Francesco Silvestri
University of Padova

Main motivation in the context of Fairness

Goal of fairness: Remove or minimize the harm caused by the algorithms

- Bias in data
- Bias in the data structures that handle it

Main motivation in the context of Fairness

Goal of fairness: Remove or minimize the harm caused by the algorithms

- Bias in data
- Bias in the data structures that handle it

This work:

- Selection bias, not introduce it
- Report uniformly at random an item from acceptable outcomes
- Similarity search (Near Neighbor problem)

Main motivation in the context of Fairness

Goal of fairness: Remove or minimize the harm caused by the algorithms

- Bias in data
- Bias in the data structures that handle it

This work:

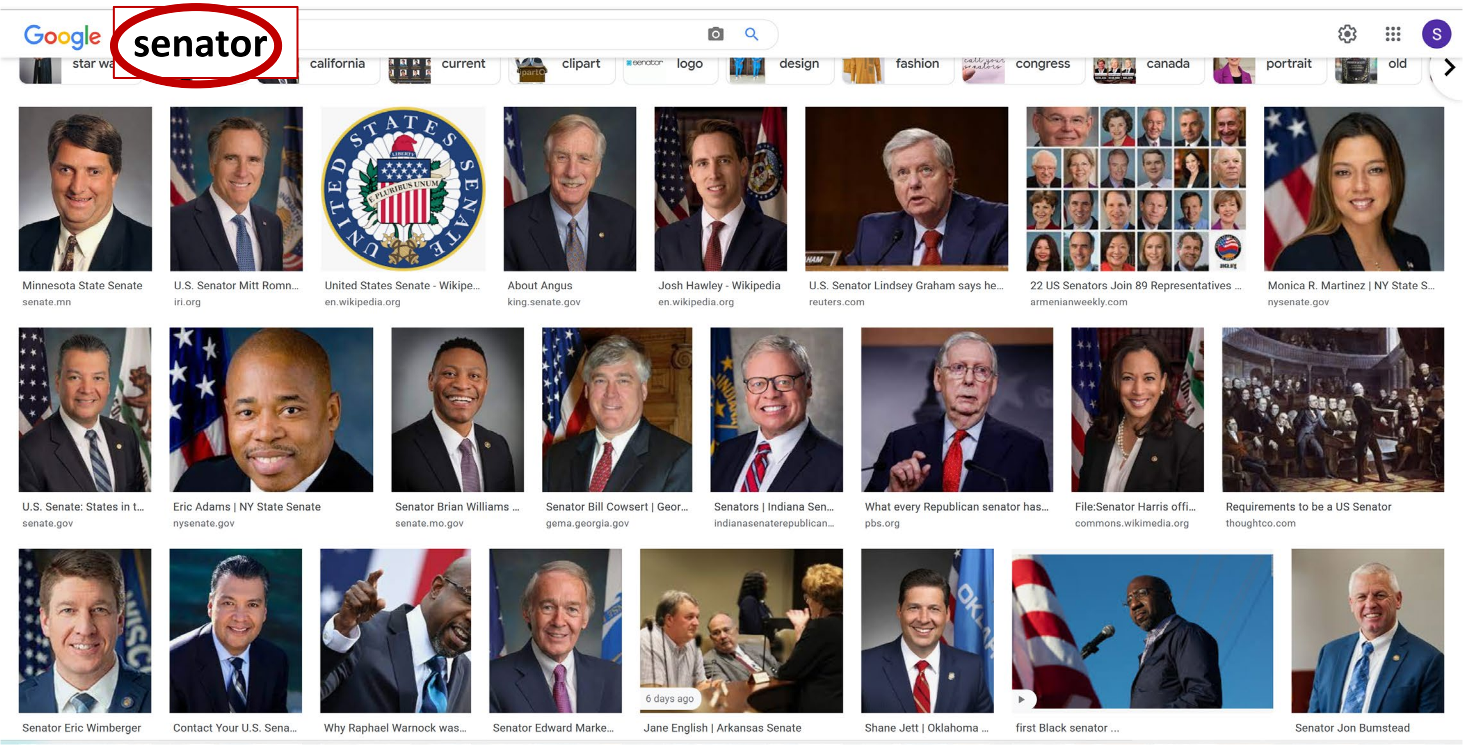
- Selection bias, not introduce it
- Report uniformly at random an item from acceptable outcomes
- Similarity search (Near Neighbor problem)

➢ No unique definition of fairness, e.g.

- Group fairness: demographics of the population are preserved in the outcome
- Individual fairness: treat individuals with similar conditions similarly, equal opportunity

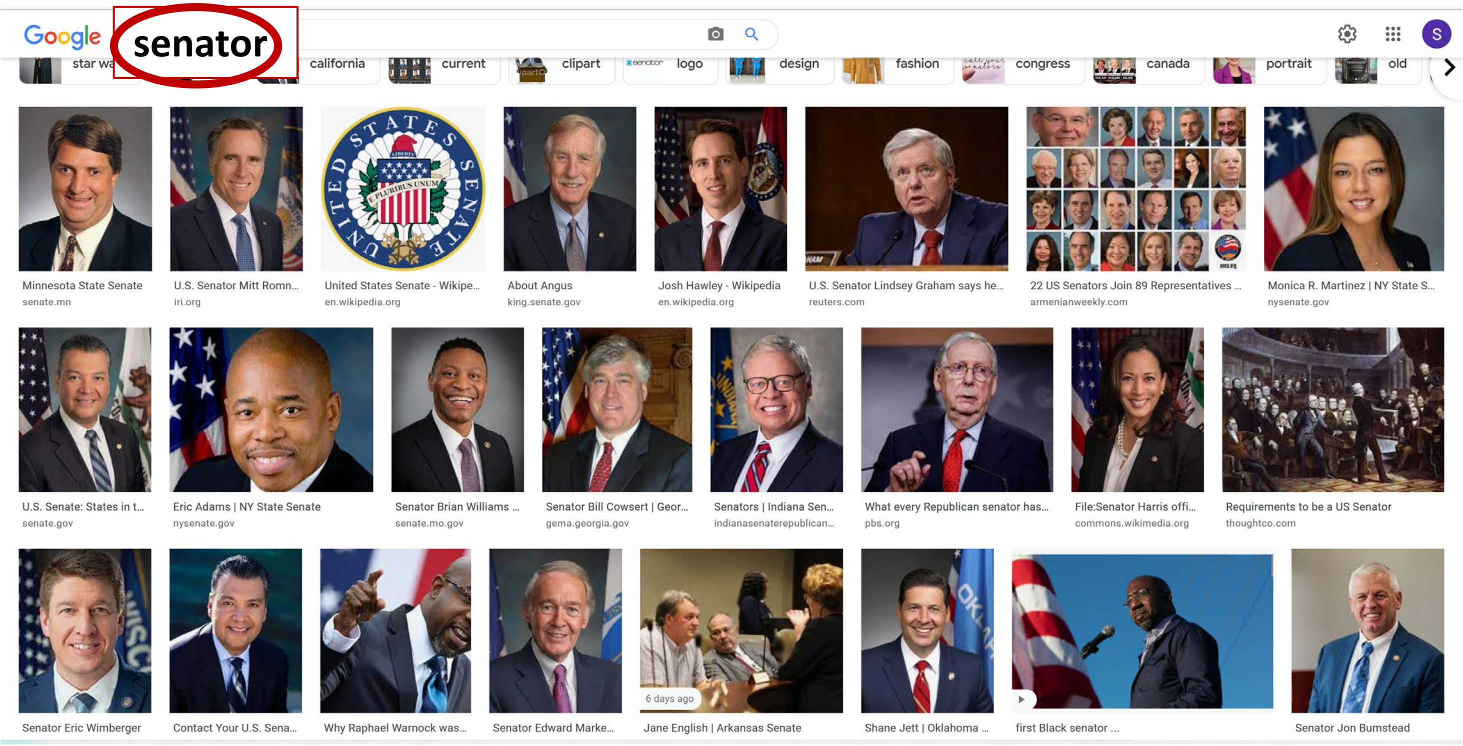
Individual Fairness in Searching

- 27% of senators are women



Individual Fairness in Searching

- 27% of senators are women
- Searching for job applicants (e.g. LinkedIn suggestions)

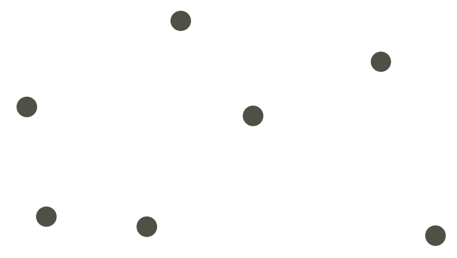


Plan for the talk

- Nearest neighbor
- Sampling version/ fair version
- Applications
- Algorithms
  - Basic Algorithm
  - Improving the dependence on ϵ
  - Handling Outliers
  - Improving the dependence on the neighborhood

Near Neighbor

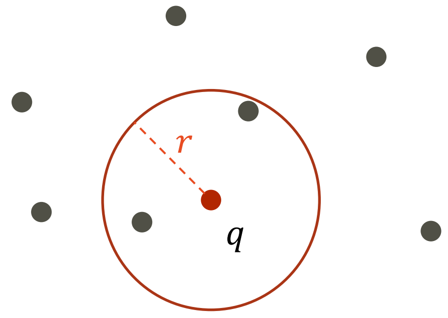
Dataset of n points P in a metric space, e.g. $\mathbb{R}^d$, and a parameter r



Near Neighbor

Dataset of n points P in a metric space, e.g. $\mathbb{R}^d$, and a parameter r

A query point q comes online



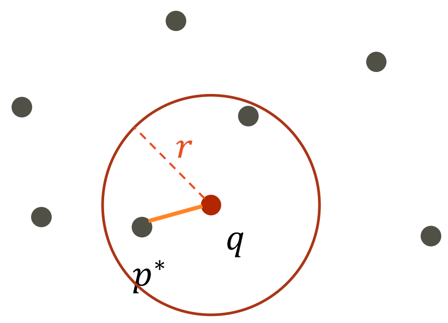
Near Neighbor

Dataset of n points P in a metric space, e.g. $\mathbb{R}^d$, and a parameter r

A query point q comes online

Goal:

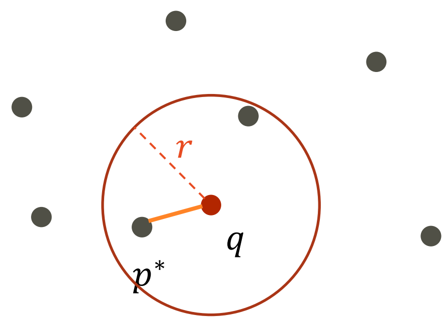
- Find a point p^* in the r-neighborhood



Near Neighbor

Dataset of n points P in a metric space, e.g. $\mathbb{R}^d$, and a parameter r

A query point q comes online



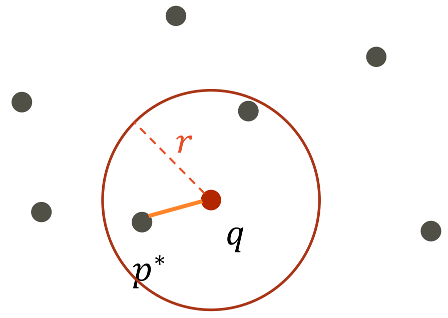
Goal:

- Find a point p^* in the r-neighborhood
- Do it in sub-linear time and small space

Near Neighbor

Dataset of n points P in a metric space, e.g. $\mathbb{R}^d$, and a parameter r

A query point q comes online



Goal:

- Find a point p^* in the r-neighborhood
- Do it in sub-linear time and small space

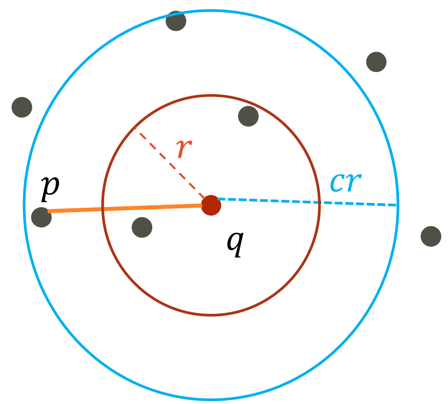
All existing algorithms for this problem

- Either space or query time depending exponentially on d
- Or assume certain properties about the data, e.g., bounded intrinsic dimension

Approximate Near Neighbor

Dataset of n points P in a metric space, e.g. $\mathbb{R}^d$, and a parameter r

A query point q comes online



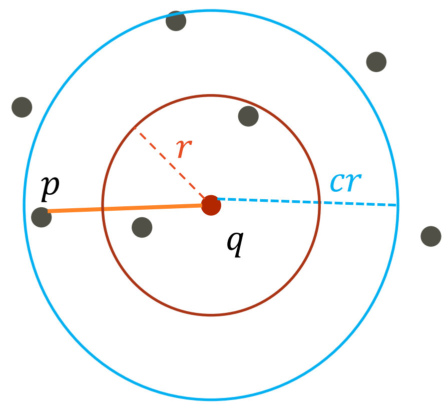
Goal:

- Find a point p^* in the r-neighborhood
- Do it in sub-linear time and small space
- Approximate Near Neighbor
  - Report a point in distance cr for $c > 1$

Approximate Near Neighbor

Dataset of n points P in a metric space, e.g. $\mathbb{R}^d$,
and a parameter r

A query point q comes online



Goal:

- Find a point p^* in the r-neighborhood
- Do it in sub-linear time and small space
- Approximate Near Neighbor
  - Report a point in distance cr for $c > 1$
  - For Hamming (and Manhattan) query time is $n^{O(1/c)}$ [IM98]
  - and for Euclidean it is $n^{O(\frac{1}{c^2})}$ [AI08]

Fair Near Neighbor

Report one of the neighbors uniformly at random

- ❑ Individual fairness: every neighbor has the same chance of being reported.
- ❑ Remove the bias inherent in the NN data structure (also for the downstream tasks)

- ➢ Fair Near Neighbor as a **NN sampling problem**:
  - Sample a point in the neighborhood of the query uniformly at random

Beyond Fairness: When random nearby-by is better than the nearest

- Robustness: input is noisy, and the closest point might be an unrepresentative outlier

  (e.g. why knn is beneficial in reducing the effect of noise)

Beyond Fairness: When random nearby-by is better than the nearest

- ❑ Robustness: input is noisy, and the closest point might be an unrepresentative outlier
  (e.g. why knn is beneficial in reducing the effect of noise)
- ❑ KNN-Classification

Applications beyond Fairness: KNN - Classification

- Data set of points, each has a label
- Given a query: find the closest K neighbors to the query

Applications beyond Fairness: KNN - Classification

- Data set of points, each has a label
- Given a query: find the closest K neighbors to the query
- Compute the majority label ℓ
- Assign the label ℓ to the query

Applications beyond Fairness: KNN - Classification

- Data set of points, each has a label
- Given a query: find the closest K neighbors to the query
- Compute the majority label ℓ
- Assign the label ℓ to the query

- ➢ small values of k, are not robust
- ➢ large values are not time efficient

➢Instead: sample a few points in the neighborhood and assign the label based on the majority of sampled points

Beyond Fairness: When random nearby-by is better than the nearest

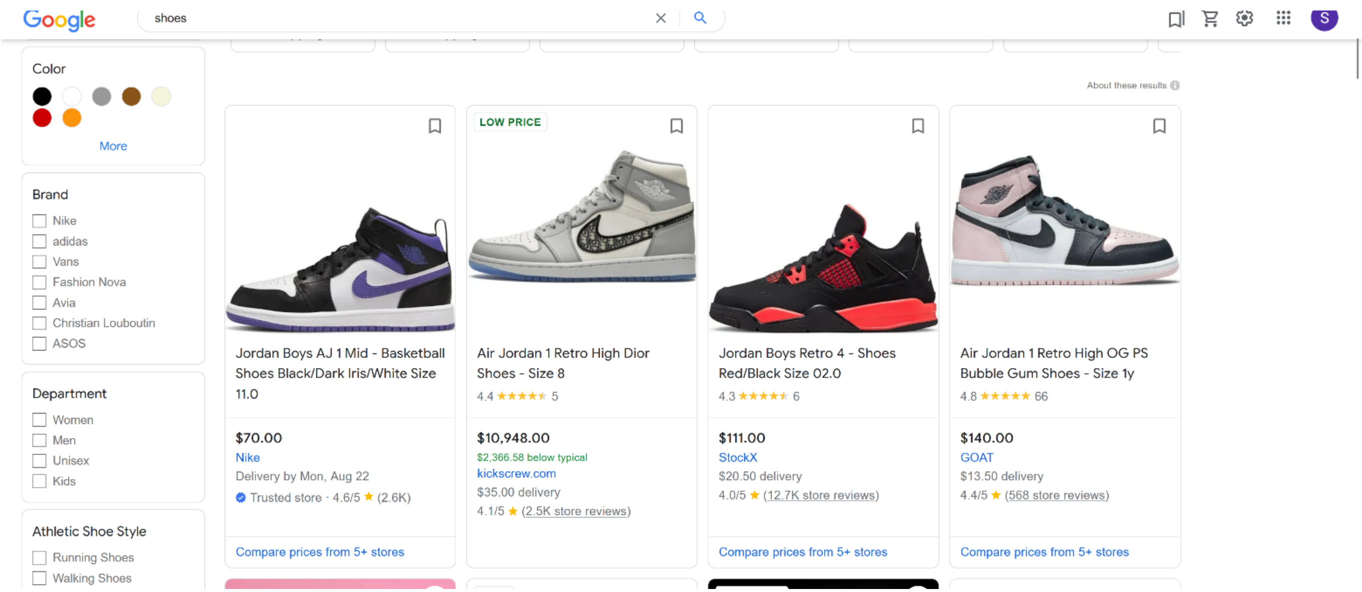
- ❑ Robustness: input is noisy, and the closest point might be an unrepresentative outlier (e.g. why knn is beneficial in reducing the effect of noise)
- ❑ KNN-Classification
- ❑ Statistical Queries: estimate the number of items with a desired property in the neighborhood.

Beyond Fairness: When random nearby-by is better than the nearest

- ❑ Robustness: input is noisy, and the closest point might be an unrepresentative outlier (e.g. why knn is beneficial in reducing the effect of noise)
- ❑ KNN-Classification
- ❑ Statistical Queries: estimate the number of items with a desired property in the neighborhood.
- ❑ Filtered Searching

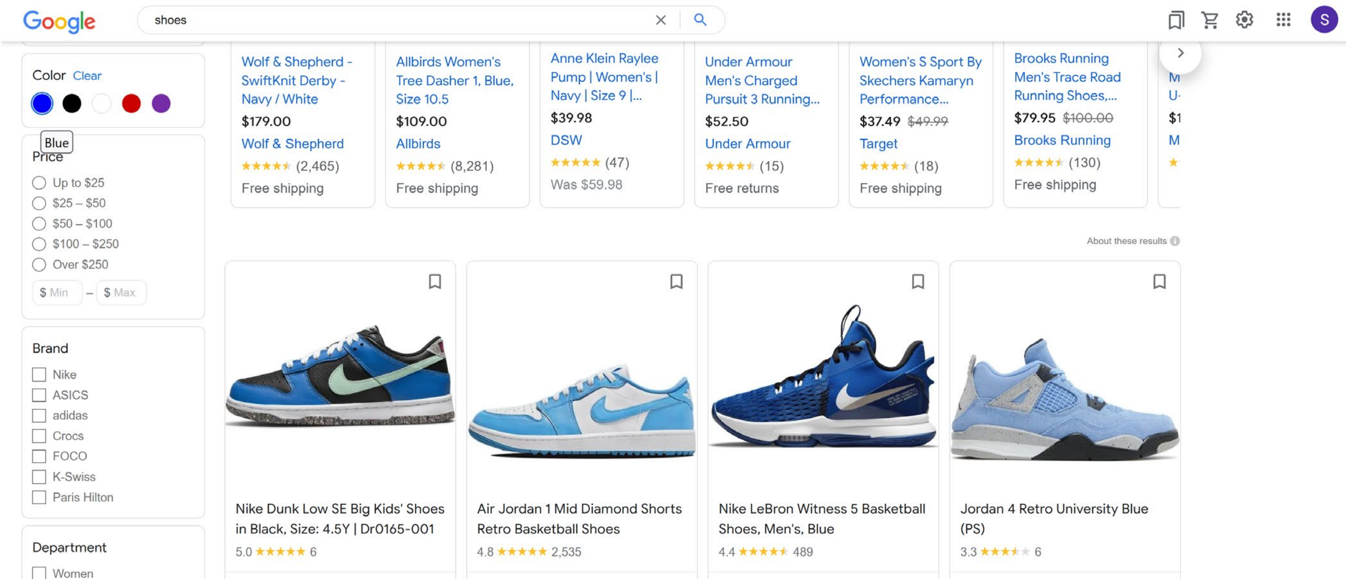
Applications beyond Fairness: Filtered Searching

- Apply filters on top of our search.
- E.g. in a shopping scenario, person looking for “blue” shoes
  - Searches for “shoes”
  - Adds a filter of color being “blue”



Applications beyond Fairness: Filtered Searching

- Apply filters on top of our search.
- E.g. in a shopping scenario, person looking for “blue” shoes
  - Searches for “shoes”
  - Adds a filter of color being “blue”
- If the desired property is common in the neighborhood:
  - Retrieve random shoes until blue shoes are found.
  - Can be combined with a different procedure for rare filters



Beyond Fairness: When random nearby-by is better than the nearest

- ❑ Robustness: input is noisy, and the closest point might be an unrepresentative outlier (e.g. why knn is beneficial in reducing the effect of noise)
- ❑ KNN-Classification
- ❑ Statistical Queries: estimate the number of items with a desired property in the neighborhood.
- ❑ Filtered Searching
- ❑ Anonymizing the data

Beyond Fairness: When random nearby-by is better than the nearest

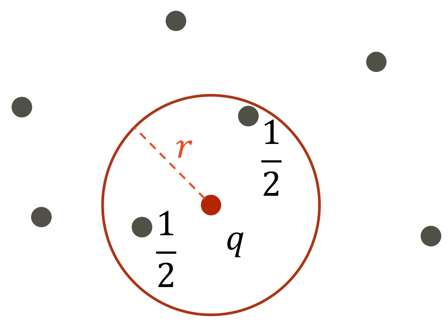
- ❑ Robustness: input is noisy, and the closest point might be an unrepresentative outlier (e.g. why knn is beneficial in reducing the effect of noise)
- ❑ KNN-Classification
- ❑ Statistical Queries: estimate the number of items with a desired property in the neighborhood.
- ❑ Filtered Searching
- ❑ Anonymizing the data
- ❑ Diversifying the output (e.g. in a recommendation system)

Problem formulation and our results

Fair Near Neighbor

Dataset of n points P in a metric space, e.g. $\mathbb{R}^d$, and a parameter r

A query point q comes online



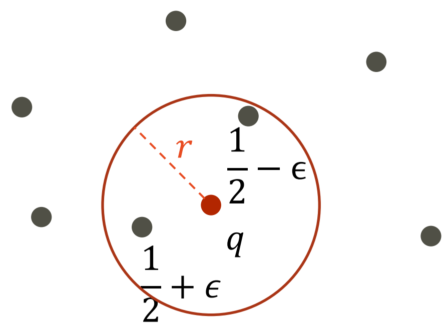
Goal:

- Return each point p in the neighborhood of q with uniform probability
- Do it in sub-linear time and small space

Approximately Fair Near Neighbor

Dataset of n points P in a metric space, e.g. $\mathbb{R}^d$, and a parameter r

A query point q comes online



Goal of Approximately Fair NN

- Any point p in $N(q,r)$ is reported with "almost uniform" probability, i.e., $\lambda_q(p)$ where

$$\frac{1}{(1+\epsilon)|N(q,r)|} \leq \lambda_q(p) \leq \frac{(1+\epsilon)}{|N(q,r)|}$$

Further notes

Need Independence

- Need a Fresh Sample each time, i.e., require independence between queries:

$$\mathbf{Pr}\left[\boldsymbol{out}_{i,q_i} = \boldsymbol{p} \middle| \boldsymbol{out}_{i-1,q_{i-1}} = \boldsymbol{p}_{i-1}, \ldots, \boldsymbol{out}_{1,q_1} = \boldsymbol{p}_1\right] \approx \frac{\mathbf{1}}{|\boldsymbol{N}(\boldsymbol{q}, \boldsymbol{r})|}$$

Further notes

Need Independence

- Need a Fresh Sample each time, i.e., require independence between queries:

$$\mathbf{Pr}\left[\boldsymbol{out}_{i,q_i} = \boldsymbol{p} \middle| \boldsymbol{out}_{i-1,q_{i-1}} = \boldsymbol{p}_{i-1}, \ldots, \boldsymbol{out}_{1,q_1} = \boldsymbol{p}_1\right] \approx \frac{\mathbf{1}}{|\boldsymbol{N}(\boldsymbol{q}, \boldsymbol{r})|}$$

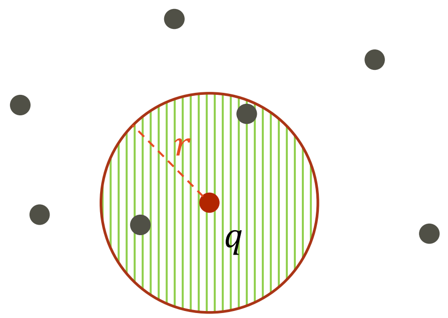
Pior Work

- In low dimensions, "Independent Range Sampling" [Xiaocheng Hu, Miao Qiao, and Yufei Tao.]
  - Exponential dependence on dim runtime

Results on $(1+\epsilon)$-Approximate Fair NN

Domain	Space	Query
Exact Neighborhood $N(q,r)$	$O(S_{ANN})$	$\tilde{O}(T_{ANN} + \frac{\lvert N(q,cr)\rvert}{\lvert N(q,r)\rvert})$

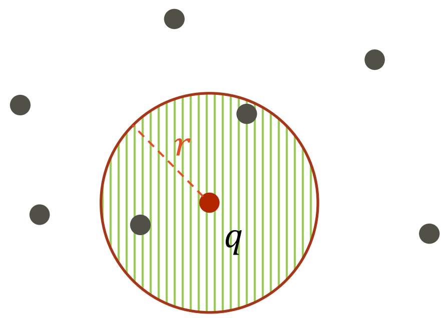
- S_{ANN} and T_{ANN} are the space and query time of standard ANN

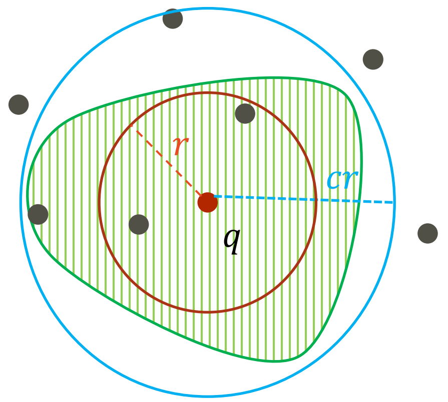


Results on $(1+\epsilon)$-Approximate Fair NN

Domain	Space	Query
Exact Neighborhood $N(q,r)$	$O(S_{ANN})$	$\tilde{O}(T_{ANN} + \frac{\lvert N(q,cr)\rvert}{\lvert N(q,r)\rvert})$
Approximate Neighborhood $N(q,r) \subseteq S \subseteq N(q,cr)$	$\tilde{O}(S_{ANN})$	$\tilde{O}(T_{ANN})$

- S_{ANN} and T_{ANN} are the space and query time of standard ANN
- Approximate neighborhood: a set S such that $N(q,r) \subseteq S \subseteq N(q,cr)$





Results on $(1+\epsilon)$-Approximate Fair NN

Domain	Space	Query
Exact Neighborhood $N(q,r)$	$O(S_{ANN})$	$\tilde{O}(T_{ANN} + \frac{\|N(q,cr)\|}{\|N(q,r)\|})$
Approximate Neighborhood $N(q,r) \subseteq S \subseteq N(q,cr)$	$\tilde{O}(S_{ANN})$	$\tilde{O}(T_{ANN})$

- S_{ANN} and T_{ANN} are the space and query time of standard ANN
- Approximate neighborhood: a set S such that $N(q,r) \subseteq S \subseteq N(q,cr)$
- Dependence on ϵ is $O(\log(\frac{1}{\epsilon}))$

Results on $(1+\epsilon)$-Approximate Fair NN

Domain	Space	Query
Exact Neighborhood $N(q,r)$	$O(S_{ANN})$	$\tilde{O}(T_{ANN} + \frac{\lvert N(q,cr)\rvert}{\lvert N(q,r)\rvert})$
Approximate Neighborhood $N(q,r) \subseteq S \subseteq N(q,cr)$	$\tilde{O}(S_{ANN})$	$\tilde{O}(T_{ANN})$

- S_{ANN} and T_{ANN} are the space and query time of standard ANN
- Approximate neighborhood: a set S such that $N(q,r) \subseteq S \subseteq N(q,cr)$
- Dependence on ϵ is $O(\log(\frac{1}{\epsilon}))$
- Black-box reduction

Results on $(1+\epsilon)$-Approximate Fair NN

Domain	Space	Query
Exact Neighborhood $N(q,r)$	$O(S_{ANN})$	$\tilde{O}(T_{ANN} + \frac{\vert N(q,cr)\vert}{\vert N(q,r)\vert})$
Approximate Neighborhood $N(q,r) \subseteq S \subseteq N(q,cr)$	$\tilde{O}(S_{ANN})$	$\tilde{O}(T_{ANN})$

- S_{ANN} and T_{ANN} are the space and query time of standard ANN
- Approximate neighborhood: a set S such that $N(q,r) \subseteq S \subseteq N(q,cr)$
- Dependence on ϵ is $O(\log(\frac{1}{\epsilon}))$
- Black-box reduction
- Our approach solves a more general problem

Results on $(1+\epsilon)$-Approximate Fair NN

Domain	Space	Query
Exact Neighborhood $N(q,r)$	$O(S_{ANN})$	$\tilde{O}(T_{ANN} + \frac{\|N(q,cr)\|}{\|N(q,r)\|})$
Approximate Neighborhood $N(q,r) \subseteq S \subseteq N(q,cr)$	$\tilde{O}(S_{ANN})$	$\tilde{O}(T_{ANN})$

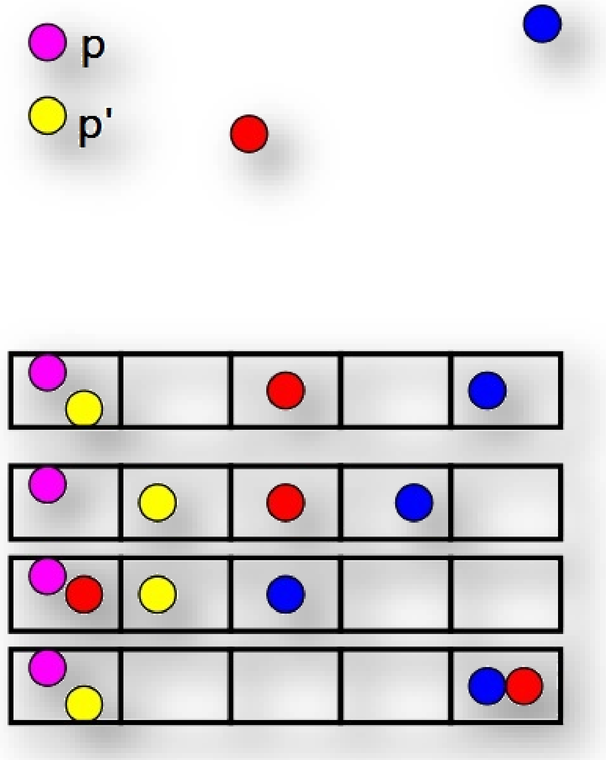
- S_{ANN} and T_{ANN} are the space and query time of standard ANN
- Approximate neighborhood: a set S such that $N(q,r) \subseteq S \subseteq N(q,cr)$
- Dependence on ϵ is $O(\log(\frac{1}{\epsilon}))$
- Black-box reduction
- Our approach solves a more general problem
- Experiments (Naïve randomization of ANN is not fair)

Locality Sensitive Hashing (LSH) [Indyk, Motwani'98]

One of the main approaches to solve the Nearest Neighbor problems

Locality Sensitive Hashing (LSH)

Hashing scheme s.t. close points have higher probability of collision than far points

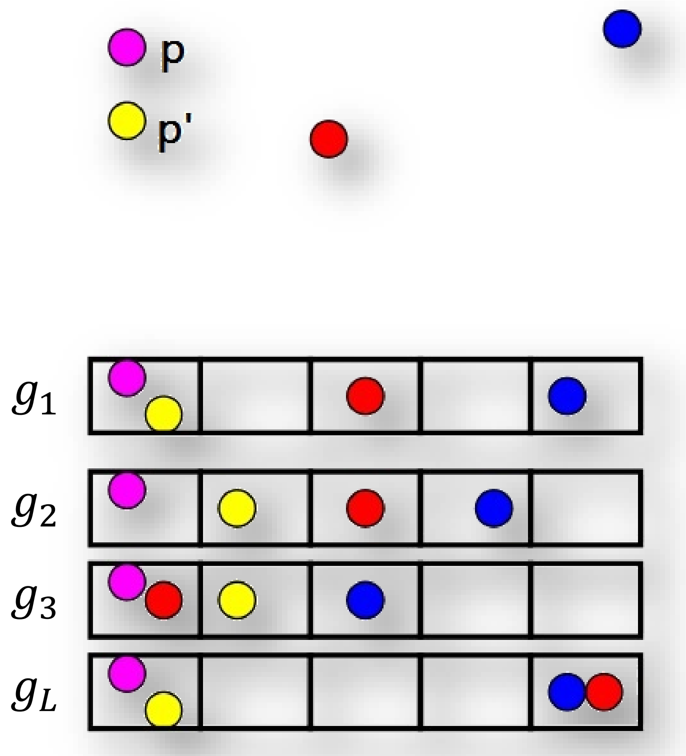


Locality Sensitive Hashing (LSH)

Hashing scheme s.t. close points have higher probability of collision than far points

Hash functions: $g_1, \ldots, g_L$

- g_i is an independently chosen hash function



Locality Sensitive Hashing (LSH)

Hashing scheme s.t. close points have higher probability of collision than far points

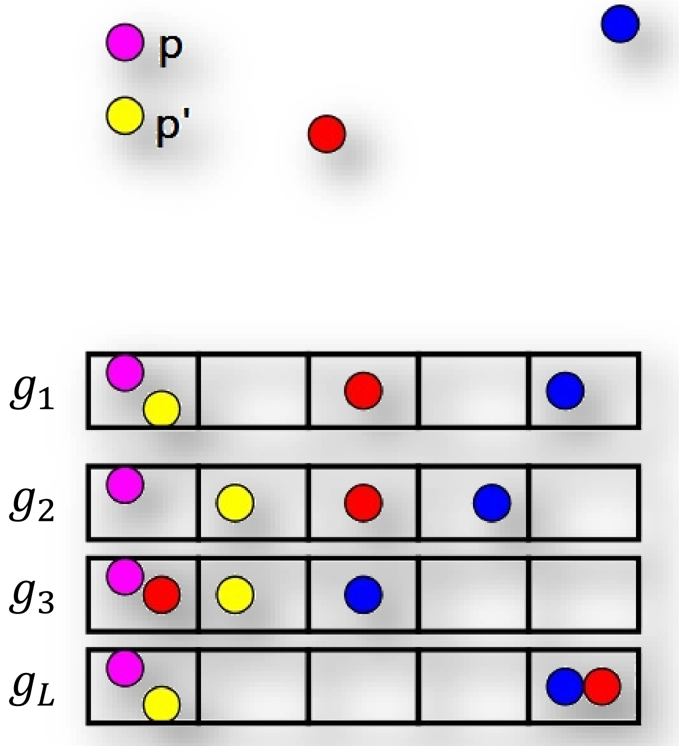
Hash functions: $g_1 , \ldots , g_L$

- g_i is an independently chosen hash function

If $||p - p'|| \leq r$, they collide w.p. $\geq P_{high}$

If $||p - p'|| \geq cr$, they collide w.p. $\leq P_{low}$

For $P_{high} \geq P_{low}$

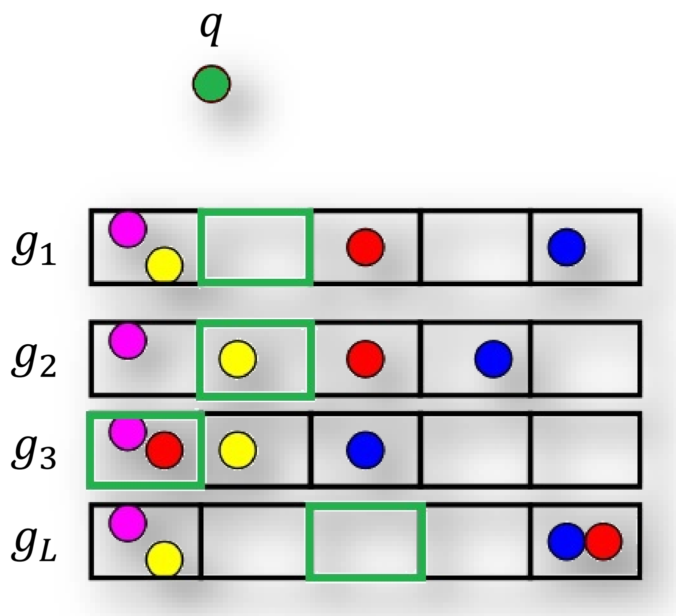


Locality Sensitive Hashing (LSH)

Retrieval: [Indyk, Motwani'98]

- The union of the query buckets is roughly the neighborhood of q

- $\bigcup_i B_i(g_{i(q)})$ is roughly the neighborhood
  - Contains all points within distance r
  - Contains at most L outlier points (farther than cr)



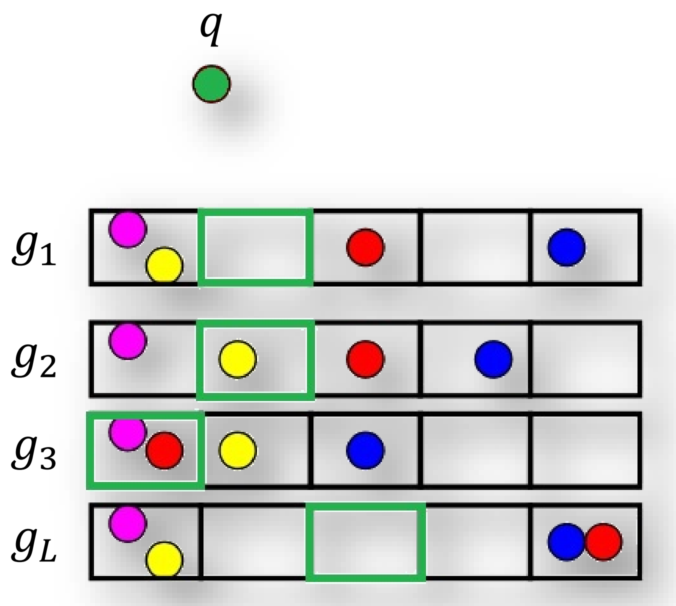
Locality Sensitive Hashing (LSH)

Retrieval: [Indyk, Motwani'98]

- The union of the query buckets is roughly the neighborhood of q

- $\bigcup_i B_i(g_{i(q)})$ is roughly the neighborhood
  - Contains all points within distance r
  - Contains at most L outlier points (farther than cr)

- How to report a uniformly random neighbor from union of these buckets?



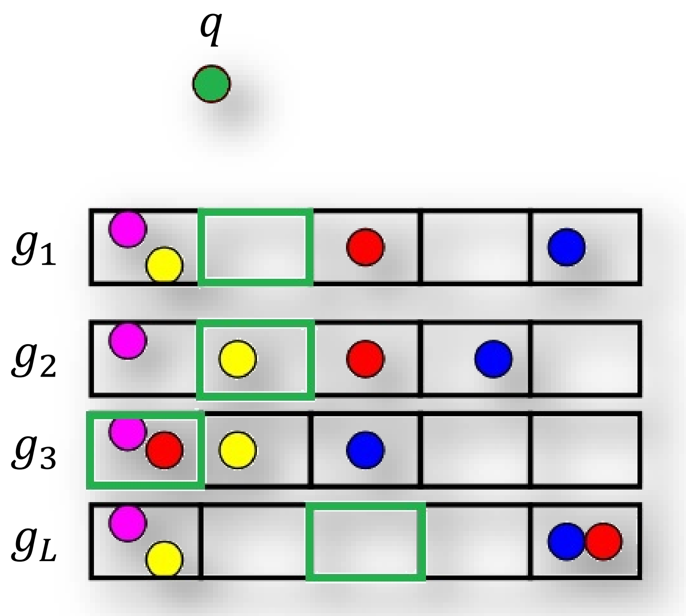
Locality Sensitive Hashing (LSH)

Retrieval: [Indyk, Motwani'98]

- The union of the query buckets is roughly the neighborhood of q

- $\bigcup_i B_i\left(g_{i(q)}\right)$ is roughly the neighborhood
  - Contains all points within distance r
  - Contains at most L outlier points (farther than cr)

- How to report a uniformly random neighbor from union of these buckets?
  - Collecting all points might take $O(n)$ time



A more general problem

Sampling from a sub-collection of Sets

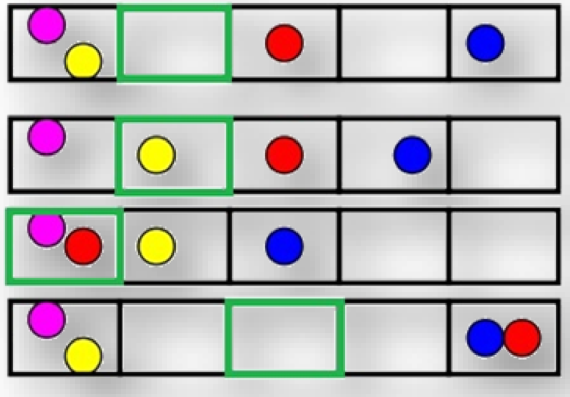
Sampling from a sub-collection of sets

Preprocess: a collection $\mathcal{F}$ of subsets of a universe U

Sampling from a sub-collection of sets

Preprocess: a collection $\mathcal{F}$ of subsets of a universe U

- *E.g. in LSH:* all buckets in all hash tables



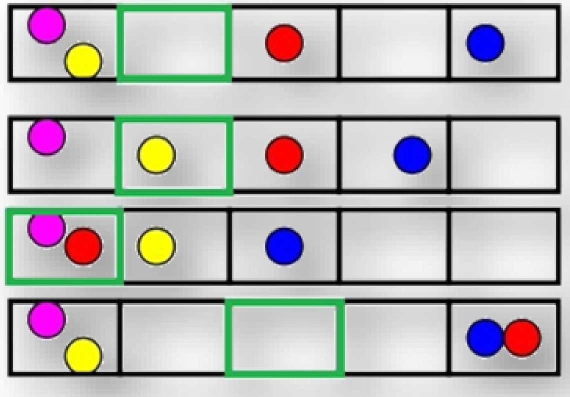
Sampling from a sub-collection of sets

Preprocess: a collection $\mathcal{F}$ of subsets of a universe U

- *E.g. in LSH:* all buckets in all hash tables

Query: a sub-collection $\mathcal{G} \subseteq \mathcal{F}$

- *E.g. in LSH:* buckets corresponding to the query



Sampling from a sub-collection of sets

Preprocess: a collection $\mathcal{F}$ of subsets of a universe U

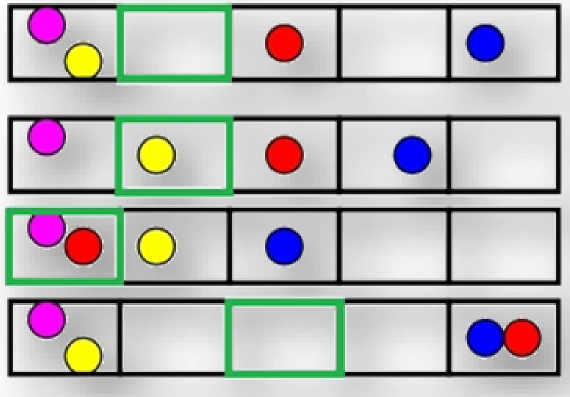
- *E.g. in LSH:* all buckets in all hash tables

Query: a sub-collection $\mathcal{G} \subseteq \mathcal{F}$

- *E.g. in LSH:* buckets corresponding to the query

Goal: report a point uniformly at random from $\cup\mathcal{G} = \cup_{F\in\mathcal{G}} F$

- Runtime of $|\mathcal{G}|$, (*e.g. in LSH:* the number of hash functions L)



Sampling from a sub-collection of sets

Preprocess: a collection $\mathcal{F}$ of subsets of a universe U

- *E.g. in LSH:* all buckets in all hash tables

Query: a sub-collection $\mathcal{G} \subseteq \mathcal{F}$

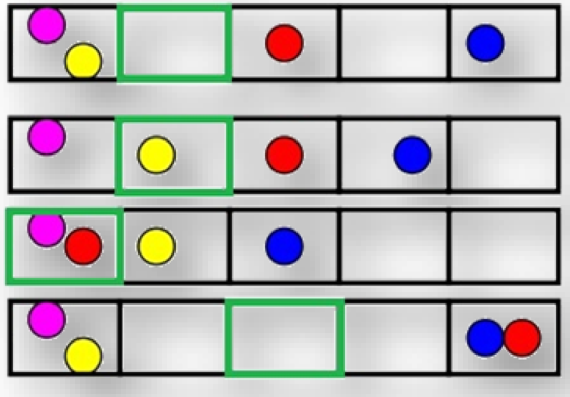
- *E.g. in LSH:* buckets corresponding to the query

Goal: report a point uniformly at random from $\cup\mathcal{G} = \cup_{F\in\mathcal{G}} F$

- Runtime of $|\mathcal{G}|$, (*e.g. in LSH:* the number of hash functions L)

Other applications:

- Sampling from neighbors of a subset of vertices in a graph
- Uniform sampling for range searching



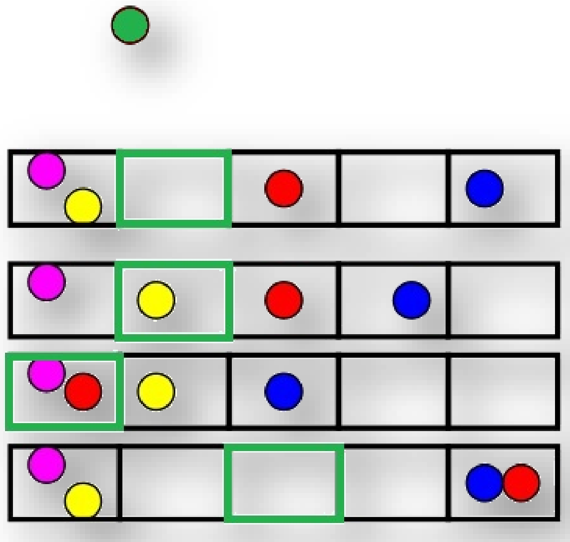
- Nearest neighbor
- Sampling version/ fair version
- Applications
- Algorithms
  - Basic Algorithm
  - Improving the dependence on ϵ
  - Handling Outliers
  - Improving the dependence on the neighborhood

Basic Algorithm

Algorithm

How to output a random neighbor from $\bigcup \mathcal{G} = \bigcup_{F \in \mathcal{G}} F$

1. Choose a set $F \in \mathcal{G}$ w.p. $\propto |F|$
2. Choose a uniformly random point in F

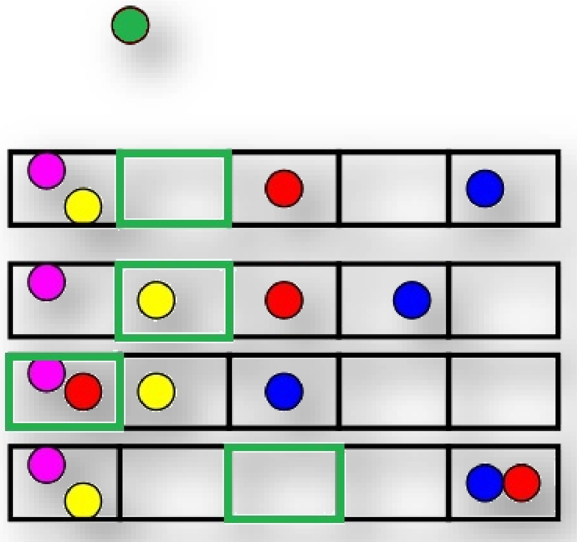


Algorithm

How to output a random neighbor from $\bigcup \mathcal{G} = \bigcup_{F \in \mathcal{G}} F$

1. Choose a set $F \in \mathcal{G}$ w.p. $\propto |F|$
2. Choose a uniformly random point in F
   - Each point is picked w.p. proportional to its degree d_p

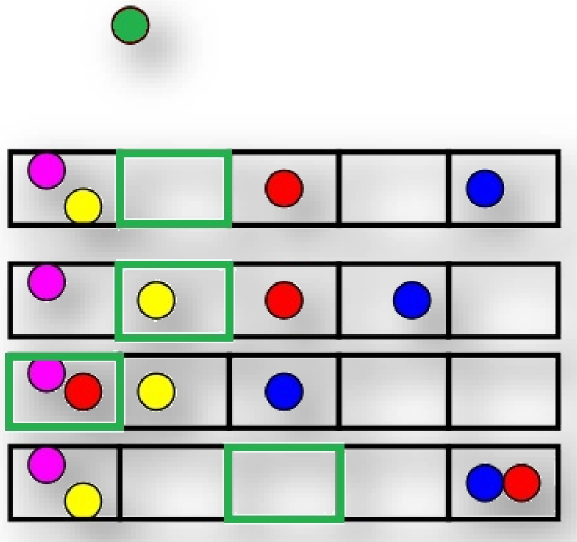
Number of sets in $\mathcal{G}$ that p appears in



Algorithm

How to output a random neighbor from $\bigcup \mathcal{G} = \bigcup_{F \in \mathcal{G}} F$

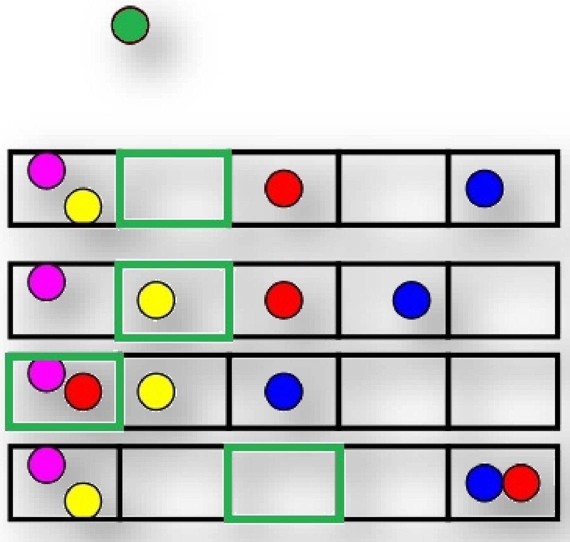
1. Choose a set $F \in \mathcal{G}$ w.p. $\propto |F|$
2. Choose a uniformly random point in F
   - Each point is picked w.p. proportional to its degree d_p
3. Keep p with probability $\frac{1}{d_p}$, o.w. repeat



Algorithm

How to output a random neighbor from $\cup\mathcal{G} = \cup_{F\in\mathcal{G}} F$

1. Choose a set $F \in \mathcal{G}$ w.p. $\propto |F|$
2. Choose a uniformly random point in F
   - ➢Each point is picked w.p. proportional to its degree d_p
3. Keep p with probability $\frac{1}{d_p}$, o.w. repeat
   - ➢ Uniform probability

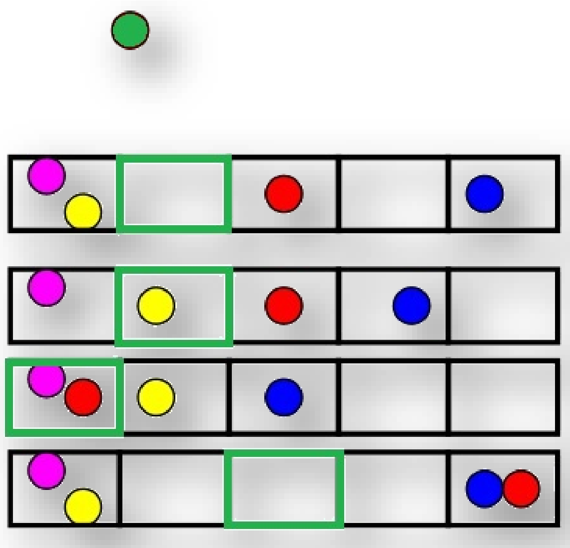


Algorithm

How to output a random neighbor from $\cup\mathcal{G} = \cup_{F\in\mathcal{G}} F$

$\boldsymbol{L} = |\mathcal{G}|$

1. Choose a set $F \in \mathcal{G}$ w.p. $\propto |F|$
2. Choose a uniformly random point in F
   - Each point is picked w.p. proportional to its degree d_p
3. Keep p with probability $\frac{1}{d_p}$, o.w. repeat
   - Uniform probability
   - Need to spend $O(L)$ to find the degree

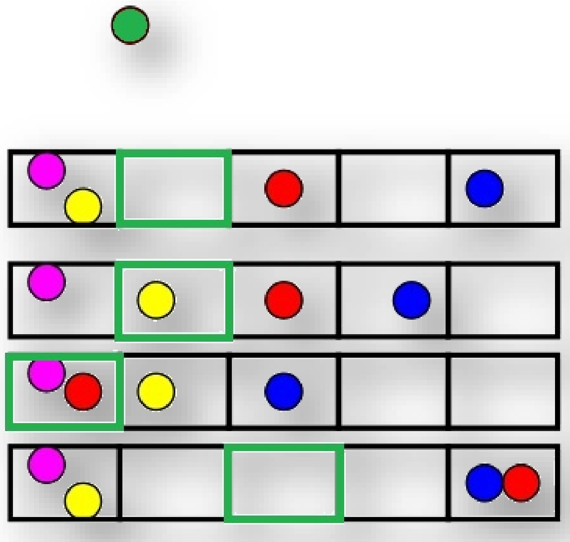


Algorithm

How to output a random neighbor from $\cup\mathcal{G} = \cup_{F\in\mathcal{G}} F$

$\boldsymbol{L} = |\mathcal{G}|$

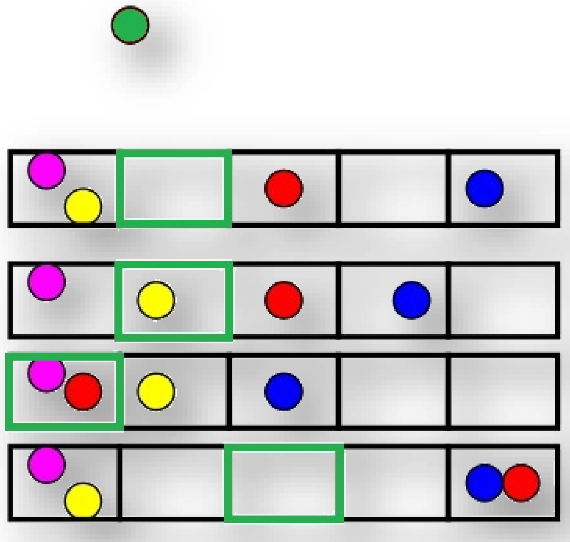
1. Choose a set $F \in \mathcal{G}$ w.p. $\propto |F|$
2. Choose a uniformly random point in F
   - Each point is picked w.p. proportional to its degree d_p
3. Keep p with probability $\frac{1}{d_p}$, o.w. repeat
   - Uniform probability
   - Need to spend $O(L)$ to find the degree
   - Might need $O(d_{max}) = O(L)$ samples
   - Total time is $O(L^2)$



Approximate the degree d_p

Sample $O(\frac{L}{d_p \cdot \epsilon^2})$ sets out of L sets in $\mathcal{G}$ to $(1+\epsilon)$-approximate the degree.

$$\boldsymbol{L} = |\mathcal{G}|$$

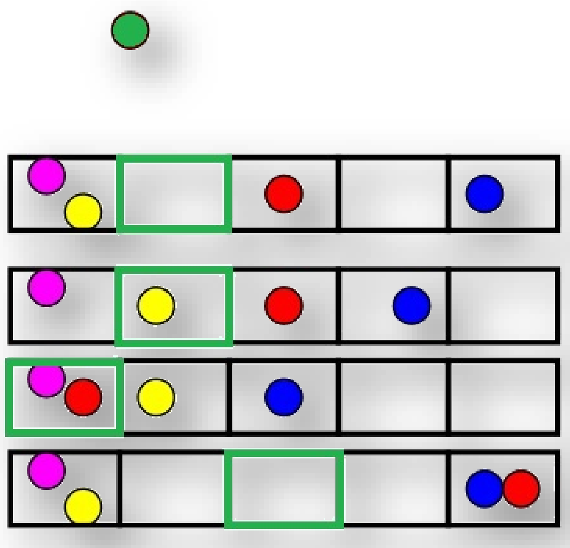


Approximate the degree d_p

$L = |\mathcal{G}|$

Sample $O(\frac{L}{d_p \cdot \epsilon^2})$ sets out of L sets in $\mathcal{G}$ to $(1+\epsilon)$-approximate the degree.

- Still if the degree is low this takes $O(L)$ samples.



Approximate the degree d_p

Sample $O(\frac{L}{d_p \cdot \epsilon^2})$ sets out of L sets in $\mathcal{G}$ to $(1+\epsilon)$-approximate the degree.

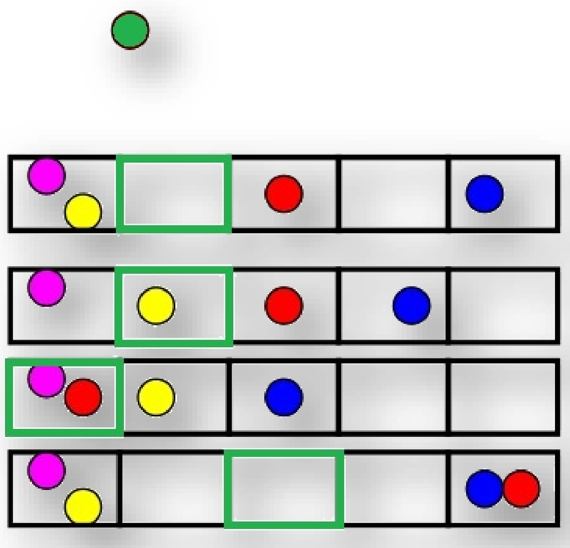
➢ Still if the degree is low this takes $O(L)$ samples.

$L = |\mathcal{G}|$

Keep p with probability $\frac{1}{d_p}$

Case 1: Small degree d_p:

- More samples are required to estimate
- Reject with lower probability -> Fewer queries of this type



Approximate the degree d_p

Sample $O(\frac{L}{d_p \cdot \epsilon^2})$ sets out of L sets in $\mathcal{G}$ to $(1+\epsilon)$-approximate the degree.

- Still if the degree is low this takes $O(L)$ samples.

$\boldsymbol{L} = |\mathcal{G}|$

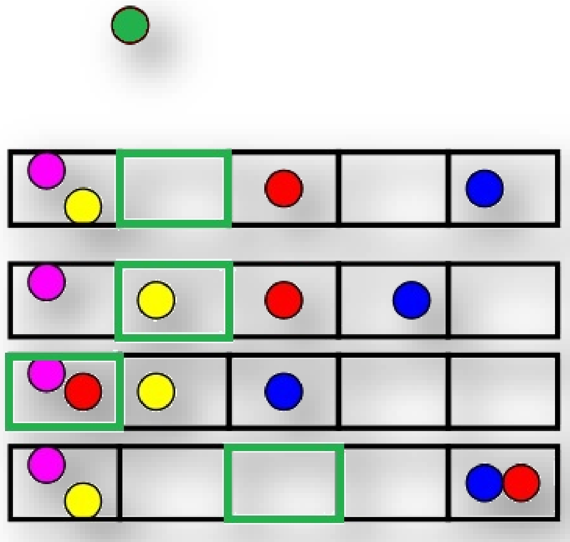
Keep p with probability $\frac{1}{d_p}$

Case 1: Small degree $\boldsymbol{d_p}$:

- More samples are required to estimate
- Reject with lower probability -> Fewer queries of this type

Case 2: Large degree $\boldsymbol{d_p}$:

- Fewer samples are required to estimate
- Reject with higher probability -> More queries of this type



Approximate the degree d_p

Sample $O(\frac{L}{d_p \cdot \epsilon^2})$ sets out of L sets in $\mathcal{G}$ to $(1+\epsilon)$-approximate the degree.

- Still if the degree is low this takes $O(L)$ samples.

$L = |\mathcal{G}|$

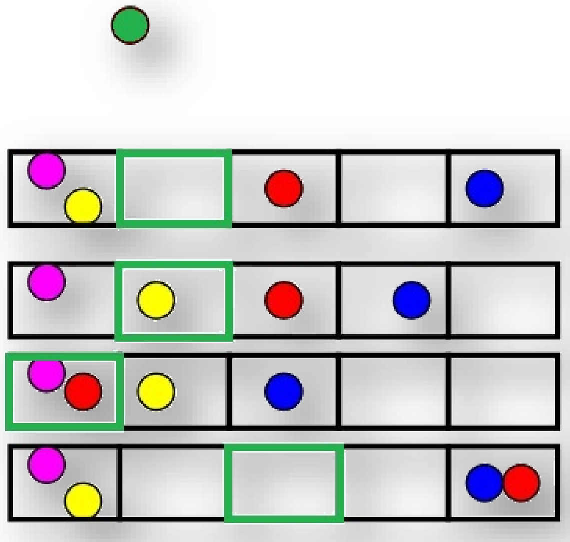
Keep p with probability $\frac{1}{d_p}$

Case 1: Small degree d_p:

- More samples are required to estimate
- Reject with lower probability -> Fewer queries of this type

Case 2: Large degree d_p:

- Fewer samples are required to estimate
- Reject with higher probability -> More queries of this type



- This decreases $O(L^2)$ runtime to $\tilde{O}(L)$

Approximate the degree d_p

$L = |\mathcal{G}|$

Keep p with probability $\frac{1}{d_p}$

Sample $O(\frac{L}{d_p \cdot \epsilon^2})$ sets out of L sets in $\mathcal{G}$ to $(1+\epsilon)$-approximate the degree.

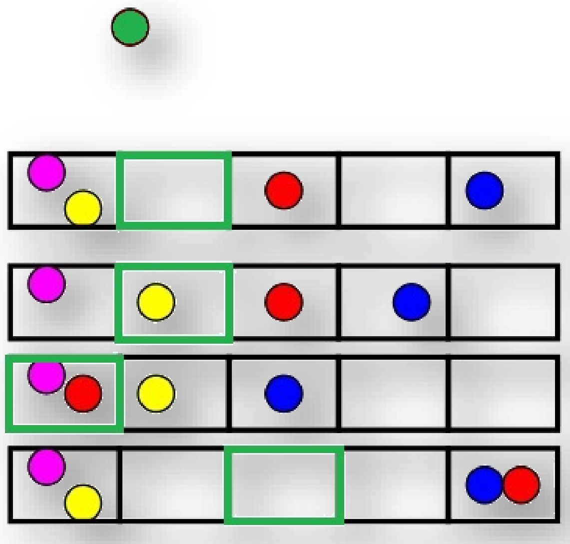
- Still if the degree is low this takes $O(L)$ samples.

Case 1: Small degree d_p:

- More samples are required to estimate
- Reject with lower probability -> Fewer queries of this type

Case 2: Large degree d_p:

- Fewer samples are required to estimate
- Reject with higher probability -> More queries of this type



- This decreases $O(L^2)$ runtime to $\tilde{O}(L)$
- Large dependency on ϵ of the form $O(\frac{1}{\epsilon^2})$

- Nearest neighbor
- Sampling version/ fair version
- Applications
- Algorithms
  - Basic Algorithm
  - Improving the dependence on ϵ
  - Handling Outliers
  - Improving the dependence on the neighborhood

Improving the dependence on ϵ

From $1/\epsilon^2$ to $\log(1/\epsilon)$

Goal: A procedure that given a sample p out of the L sets in $\mathcal{G}$

- Keeps a sample p with probability $\frac{1}{d_p}$
- In time $\tilde{O}(\frac{L}{d_p})$

$L = |\mathcal{G}|$ sets

Goal: A procedure that given a sample p out of the L sets in $\mathcal{G}$

- Keeps a sample p with probability $\frac{1}{d_p}$
- In time $\tilde{O}(\frac{L}{d_p})$

$L = |\mathcal{G}|$ sets

Need to repeat $\approx d_p$ times

Goal: A procedure that given a sample p out of the L sets in $\mathcal{G}$

- Keeps a sample p with probability $\frac{1}{d_p}$
- In time $\tilde{O}(\frac{L}{d_p})$

$L = |\mathcal{G}|$ sets

Need to repeat $\approx d_p$ times

Total runtime would be $\approx d_p \cdot \tilde{O}\left(\frac{L}{d_p}\right) = \tilde{O}(L)$

Goal: A procedure that given a sample p out of the L sets in $\mathcal{G}$

- Keeps a sample p with probability $\frac{1}{d_p}$
- In time $\tilde{O}(\frac{L}{d_p})$

$L = |\mathcal{G}|$ sets

- Sample sets from $\mathcal{G}$ until you find a set F such that $p \in F$

Assuming one can check if $p \in F$ in constant time

Goal: A procedure that given a sample p out of the L sets in $\mathcal{G}$

- Keeps a sample p with probability $\frac{1}{d_p}$
- In time $\tilde{O}(\frac{L}{d_p})$

$L = |\mathcal{G}|$ sets

- Sample sets from $\mathcal{G}$ until you find a set F such that $p \in F$
- Assume it happens at iteration i

$E[i] = \frac{L}{d_p}$

Goal: A procedure that given a sample p out of the L sets in $\mathcal{G}$

- Keeps a sample p with probability $\frac{1}{d_p}$
- In time $\tilde{O}(\frac{L}{d_p})$

$L = |\mathcal{G}|$ sets

- Sample sets from $\mathcal{G}$ until you find a set F such that $p \in F$
- Assume it happens at iteration i
- Keep the sample p with probability $\frac{i}{L} \approx \left(\frac{L}{d_p}\right) \cdot \frac{1}{L} = 1/d_p$

$E[i] = \frac{L}{d_p}$

Goal: A procedure that given a sample p out of the L sets in $\mathcal{G}$

- Keeps a sample p with probability $\frac{1}{d_p}$
- In time $\tilde{O}(\frac{L}{d_p})$

$L = |\mathcal{G}|$ sets

- Sample sets from $\mathcal{G}$ until you find a set F such that $p \in F$
- Assume it happens at iteration i
- Keep the sample p with probability $\frac{i}{L} \approx \left(\frac{L}{d_p}\right) \cdot \frac{1}{L} = 1/d_p$
  - Correct except that i/L could be larger than 1

$E[i] = \frac{L}{d_p}$

Goal: A procedure that given a sample p out of the L sets in $\mathcal{G}$

- Keeps a sample p with probability $\frac{1}{d_p}$
- In time $\tilde{O}(\frac{L}{d_p})$

$L = |\mathcal{G}|$ sets

- Sample sets from $\mathcal{G}$ until you find a set F such that $p \in F$
- Assume it happens at iteration i
- Keep the sample p with probability $\frac{i}{L} \approx \left(\frac{L}{d_p}\right) \cdot \frac{1}{L} = 1/d_p$
  - Correct except that i/L could be larger than 1
  - Keep the sample with probability $\frac{i}{\Delta \cdot L} \approx \frac{1}{\Delta \cdot d_p}$
    - Still uniform
    - Probability that $i > (\Delta L)$ is exponentially small in Δ
    - Sufficient to set $\Delta = \log\frac{1}{\epsilon}$

$E[i] = \frac{L}{d_p}$

The number of iterations increases by a factor of Δ

So far

- Get a sample uniformly at random from the union of the buckets

- $\bigcup_i B_i\left(g_{i(q)}\right)$ is roughly the neighborhood
  - Contains all points within distance r
  - Contains at most L outlier points (farther than cr)

- What about the outliers?

- Nearest neighbor
- Sampling version/ fair version
- Applications
- Algorithms
  - Basic Algorithm
  - Improving the dependence on ϵ
  - Handling Outliers
  - Improving the dependence on the neighborhood

Handling Outliers

Sampling from a sub-collection of sets with outliers

Preprocess: a collection $\mathcal{F}$ of subsets of a universe U

Sampling from a sub-collection of sets with outliers

Preprocess: a collection $\mathcal{F}$ of subsets of a universe U

Query: a sub-collection $\mathcal{G} \subseteq \mathcal{F}$, and a set of outliers $O \subseteq U$, s.t. $\sum_{o \in O} d_o(\mathcal{G}) \leq m_O$

Sampling from a sub-collection of sets with outliers

Preprocess: a collection $\mathcal{F}$ of subsets of a universe U

Query: a sub-collection $\mathcal{G} \subseteq \mathcal{F}$, and a set of outliers $O \subseteq U$, s.t.
$\sum_{o \in O} d_o(\mathcal{G}) \leq m_O$

Goal: report a point uniformly at random from $\cup\mathcal{G} \setminus O = \cup_{F \in \mathcal{G}} F \setminus O$

Sampling from a sub-collection of sets with outliers

Preprocess: a collection $\mathcal{F}$ of subsets of a universe U

Query: a sub-collection $\mathcal{G} \subseteq \mathcal{F}$, and a set of outliers $O \subseteq U$, s.t. $\sum_{o \in O} d_o(\mathcal{G}) \leq m_O$

Goal: report a point uniformly at random from $\cup\mathcal{G} \setminus O = \cup_{F \in \mathcal{G}} F \setminus O$

Trivial solution:

- Whenever you see an outlier sample, ignore it and repeat.
- Runtime in the worst case: $|\mathcal{G}| \cdot m_O$

Sampling from a sub-collection of sets with outliers

Preprocess: a collection $\mathcal{F}$ of subsets of a universe U

Query: a sub-collection $\mathcal{G} \subseteq \mathcal{F}$, and a set of outliers $O \subseteq U$, s.t. $\sum_{o \in O} d_o(\mathcal{G}) \leq m_O$

Goal: report a point uniformly at random from $\cup\mathcal{G} \setminus O = \cup_{F \in \mathcal{G}} F \setminus O$

- **Runtime of** $|\mathcal{G}| + \boldsymbol{m_o}$

Trivial solution:

- Whenever you see an outlier sample, ignore it and repeat.
- Runtime in the worst case: $|\mathcal{G}| \cdot m_O$

Goal: Runtime of $|\mathcal{G}| + m_o$

- Implement each bucket (each set in $\mathcal{F}$) as an array

Cnt=5

↓

2, 4, 6, 9, 3

Goal: Runtime of $|\mathcal{G}| + m_o$

- Implement each bucket (each set in $\mathcal{F}$) as an array
  - Once we encounter an outlier, swap it with the last element of the array.
  - Update the count of that bucket/set

Cnt=5

2, 4, 6, 9, 3

Goal: Runtime of $|\mathcal{G}| + m_o$

- Implement each bucket (each set in $\mathcal{F}$) as an array
  - Once we encounter an outlier, swap it with the last element of the array.
  - Update the count of that bucket/set

Cnt=4

2, 3, 6, 9, 4

Goal: Runtime of $|\mathcal{G}| + m_o$

- Implement each bucket (each set in $\mathcal{F}$) as an array
  - Once we encounter an outlier, swap it with the last element of the array.
  - Update the count of that bucket/set

Need to **(dynamically)** sample a set with probability proportional to its **active size**

Goal: Runtime of $|\mathcal{G}| + m_o$

- Implement each bucket (each set in $\mathcal{F}$) as an array
  - Once we encounter an outlier, swap it with the last element of the array.
  - Update the count of that bucket/set

Need to **(dynamically)** sample a set with probability proportional to its **active size**

At the query time upon receiving $\mathcal{G}$,

- Build a tree on with $L = |\mathcal{G}|$ leaves containing the count of the sets in $\mathcal{G}$

Goal: Runtime of $|\mathcal{G}| + m_o$

- Implement each bucket (each set in $\mathcal{F}$) as an array
  - Once we encounter an outlier, swap it with the last element of the array.
  - Update the count of that bucket/set

Need to **(dynamically)** sample a set with probability proportional to its **active size**

At the query time upon receiving $\mathcal{G}$,

- Build a tree on with $L = |\mathcal{G}|$ leaves containing the count of the sets in $\mathcal{G}$
- Each node keeps the sum of the counts of the leaves in its subtree

Goal: Runtime of $|\mathcal{G}| + m_o$

- Implement each bucket (each set in $\mathcal{F}$) as an array
  - Once we encounter an outlier, swap it with the last element of the array.
  - Update the count of that bucket/set

Need to **(dynamically)** sample a set with probability proportional to its **active size**

At the query time upon receiving $\mathcal{G}$,

- Build a tree on with $L = |\mathcal{G}|$ leaves containing the count of the sets in $\mathcal{G}$
- Each node keeps the sum of the counts of the leaves in its subtree
- Taking a sample from sets can be done by moving down the tree from the root

Goal: Runtime of $|\mathcal{G}| + m_o$

- Implement each bucket (each set in $\mathcal{F}$) as an array
  - Once we encounter an outlier, swap it with the last element of the array.
  - Update the count of that bucket/set

Need to **(dynamically)** sample a set with probability proportional to its **active size**

At the query time upon receiving $\mathcal{G}$,

- Build a tree on with $L = |\mathcal{G}|$ leaves containing the count of the sets in $\mathcal{G}$
- Each node keeps the sum of the counts of the leaves in its subtree
- Taking a sample from sets can be done by moving down the tree from the root
- Update the counts in time $O(\log L)$

Goal: Runtime of $|\mathcal{G}| + m_o$

- Implement each bucket (each set in $\mathcal{F}$) as an array
  - Once we encounter an outlier, swap it with the last element of the array.
  - Update the count of that bucket/set

At the query time upon receiving $\mathcal{G}$,

- Build a tree on with $L = |\mathcal{G}|$ leaves containing the count of the sets in $\mathcal{G}$
- Each node keeps the sum of the counts of the leaves in its subtree
- Taking a sample from sets
- Update the counts in time

> - We see each outlier $o \in O$ at most d_o times
> - Total number of times we encounter an outlier is m_o

So far

- Get a sample uniformly at random from the union of the buckets

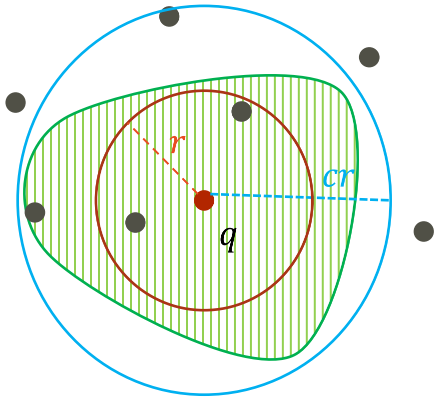
- $\cup_i B_i\left(g_{i(q)}\right)$ is roughly the neighborhood
  - Contains all points within distance r
  - Contains at most L outlier points (farther than cr)

- What about the outliers?
  - Total degree of outliers is $O(L)$
  - Get a sample in time $\tilde{O}(|\mathcal{G}| + m_o) = \tilde{O}(L + L) = \tilde{O}(L)$

Results on $(1+\epsilon)$-Approximate Fair NN

Domain	Space	Query
Exact Neighborhood $N(q,r)$	$O(S_{ANN})$	$\tilde{O}(T_{ANN} + \frac{\lvert N(q,cr)\rvert}{\lvert N(q,r)\rvert})$
Approximate Neighborhood $N(q,r) \subseteq S \subseteq N(q,cr)$	$\tilde{O}(S_{ANN})$	$\tilde{O}(T_{ANN})$

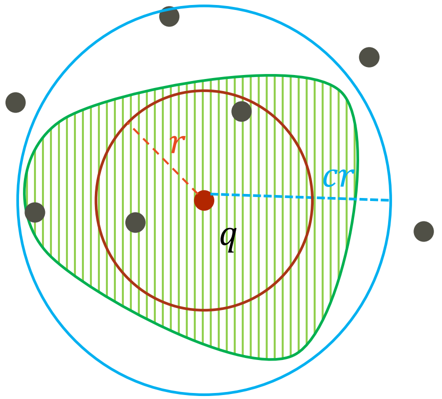
- Get a sample from the union of the buckets
- Approximate neighborhood: a set S such that $N(q,r) \subseteq S \subseteq N(q,cr)$
- Dependence on ϵ is $O(\log(\frac{1}{\epsilon}))$
- Black-box reduction



Results on $(1+\epsilon)$-Approximate Fair NN

Domain	Space	Query
Exact Neighborhood $N(q,r)$	$O(S_{ANN})$	$\tilde{O}(T_{ANN} + \frac{\|N(q,cr)\|}{\|N(q,r)\|})$
Approximate Neighborhood $N(q,r) \subseteq S \subseteq N(q,cr)$	$\tilde{O}(S_{ANN})$	$\tilde{O}(T_{ANN})$

- Get a sample from the union of the buckets
- Approximate neighborhood: a set S such that $N(q,r) \subseteq S \subseteq N(q,cr)$
- Dependence on ϵ is $O(\log(\frac{1}{\epsilon}))$
- Black-box reduction



Exact Neighborhood?

- Treat the points within distance r and cr also as outliers.
- Unlucky event: we hit all the $n(q, cr)$ outliers first
- Total runtime: $\tilde{O}(|\mathcal{G}| + m_o) = \tilde{O}(L + |N(q, cr)| - |N(q, r)|) = \tilde{O}(L + |N(q, cr)|)$

Results on $(1+\epsilon)$-Approximate Fair NN

Domain	Space	Query
Exact Neighborhood $N(q,r)$	$O(S_{ANN})$	$\tilde{O}(T_{ANN} + \lvert N(q,cr)\rvert)$
Approximate Neighborhood $N(q,r) \subseteq S \subseteq N(q,cr)$	$\tilde{O}(S_{ANN})$	$\tilde{O}(T_{ANN})$

- S_{ANN} and T_{ANN} are the space and query time of standard ANN
- Approximate neighborhood: a set S such that $N(q,r) \subseteq S \subseteq N(q,cr)$
- Dependence on ϵ is $O(\log(\frac{1}{\epsilon}))$
- Black-box reduction

Results on $(1+\epsilon)$-Approximate Fair NN

Domain	Space	Query
Exact Neighborhood $N(q,r)$	$O(S_{ANN})$	$\tilde{O}(T_{ANN} + \lvert N(q,cr)\rvert)$
Approximate Neighborhood $N(q,r) \subseteq S \subseteq N(q,cr)$	$\tilde{O}(S_{ANN})$	$\tilde{O}(T_{ANN})$

Improve to

$$T_{ANN} + \frac{|N(q,cr)|}{|N(q,r)|}$$

- S_{ANN} and T_{ANN} are the space and query time of standard ANN
- Approximate neighborhood: a set S such that $N(q,r) \subseteq S \subseteq N(q,cr)$
- Dependence on ϵ is $O(\log(\frac{1}{\epsilon}))$
- Black-box reduction

Improving the dependence on the density of the neighborhood

From $T_{ANN} + |N(q, cr)|$ to $T_{ANN} + \frac{|N(q,cr)|}{|N(q,r)|}$

High Level Idea:

- Partition the elements $\cup\mathcal{G}$ **randomly** into k bins s.t.
  - Each bin gets $O(1)$ good elements, i.e., from $\cup\mathcal{G} \setminus O$
  - Each bin gets $O(\frac{|O|}{|\cup\mathcal{G}\setminus O|})$ points from the outliers
- Time will improve to $\tilde{O}(|\mathcal{G}| + m_o) = (L + \frac{|\boldsymbol{N(q,cr)}|}{|\boldsymbol{N(q,r)}|})$

More Precisely,

Preprocess:

- To partition all elements in U among k bins
  - Give each of the elements in U a random unique rank from 1 to $N = |U|$, (i.e, pick a random permutation)
  - Each set in $\mathcal{F}$ stores its elements in sorted order

More Precisely,

Preprocess:

- To partition all elements in U among k bins
  - Give each of the elements in U a random unique rank from 1 to $N = |U|$, (i.e, pick a random permutation)
  - Each set in $\mathcal{F}$ stores its elements in sorted order

Query Time:

- Consider k bins based on the ranks, i.e.,
  Bin i $= [\left(\frac{N}{k}\right) i \,, \left(\frac{N}{k}\right) (i + 1)]$

More Precisely,

Preprocess:

- To partition all elements in U among k bins
  - Give each of the elements in U a random unique rank from 1 to $N = |U|$, (i.e, pick a random permutation)
  - Each set in $\mathcal{F}$ stores its elements in sorted order

Query Time:

- Consider k bins based on the ranks, i.e.,
  Bin i $= [\left(\frac{N}{k}\right) i \,, \left(\frac{N}{k}\right)(i+1)]$
- Select one bin (almost) uniformly at random
- Get a sample from the sampled bin

More Precisely,

Preprocess:

- To partition all ele
  - Give each of t
    from 1 to $N =$
  - Each set in $\mathcal{F}$

How to choose k

- $\boldsymbol{k}$ large: many bins get no element from $\cup\mathcal{G}$
- $\boldsymbol{k}$ small: finding an element in $\cup\mathcal{G}$ that is in a particular bin takes a long time

➢ Set $\boldsymbol{k}$ roughly equal to $|\cup\mathcal{G}|$. Then each bin has roughly $O(1)$ elements from $\cup\mathcal{G}$

➢ Don't know $|\cup\mathcal{G}|$ in advance

  ➢ Count the number of distinct elements using a sketch for Distinct Elements Problem

Query Time:

- Consider k bins based on the ranks, i.e.,
  Bin i $= [\left(\frac{N}{k}\right) i, \left(\frac{N}{k}\right)(i+1)]$
- Select one bin (almost) uniformly at random
- Get a sample from the sampled bin

More Precisely,

Preprocess:

- To partition all elements in U among k bins
  - Give each of the elements in U a random unique rank from 1 to $N = |U|$, (i.e, pick a random permutation)
  - Each set in $\mathcal{F}$ stores its elements in sorted order
  - **Keep a sketch for distinct elements**

Query Time:

- Consider k bins based on the ranks, i.e.,
  Bin i $= [\left(\frac{N}{k}\right) i \,, \left(\frac{N}{k}\right)(i+1)]$
- Select one bin (almost) uniformly at random
- Get a sample from the sampled bin

More Precisely,

Preprocess:

- To partition all ele
  - Give each of t
    from 1 to N =
  - Each set in $\mathcal{F}$
  - **Keep a sketch**

Query Time:

- Consider k bi
  Bin i $= [\left(\frac{N}{k}\right) i$
- Select one bin
- Get a sample from the sampled bin

How to choose $\boldsymbol{k}$

- $\boldsymbol{k}$ large: many bins get no element from $\cup\mathcal{G}$
- $\boldsymbol{k}$ small: finding an element in $\cup\mathcal{G}$ that is in a particular bin takes a long time

➢ Set $\boldsymbol{k}$ roughly equal to $|\cup\mathcal{G}|$. Then each bin has roughly $O(1)$ elements from $\cup\mathcal{G}$

➢ Don't know $|\cup\mathcal{G}|$ in advance

  ➢ Count the number of distinct elements using a sketch for Distinct Elements Problem

❑ Set $\boldsymbol{k} = \boldsymbol{n}(\boldsymbol{q}, \boldsymbol{r})$
❑ Number of outliers in a bin is at most $\boldsymbol{n}(\boldsymbol{q}, \boldsymbol{cr})/\boldsymbol{n}(\boldsymbol{q}, \boldsymbol{r})$

More Precisely,

Preprocess:

- To partition all elements in U among k bins
  - Give each of the elements in U a random unique rank from 1 to $N = |U|$, (i.e, pick a random permutation)
  - Each set in $\mathcal{F}$ stores its elements in sorted order
  - Keep a sketch for distinct elements

Query Time:

- Consider k bins based on the ranks, i.e., Bin i $= [\left(\frac{N}{k}\right) i \, , \left(\frac{N}{k}\right)(i+1)]$
- Select one bin (almost) uniformly at random
- **Get a sample from the sampled bin**

How to sample from $\cup\mathcal{G} \cap bin_i$?

- One can iterate over $F \cap Bin_i$ in time $O(\log n + |F \cap Bin_i|)$
  - Because the elements are kept sorted in F
  - And the Bin is continuous

➢ Compute $|F \cap Bin_i|$ for each $F \in \mathcal{G}$

➢ Build a BST on these counts, sample from them

Results on $(1+\epsilon)$-Approximate Fair NN

Domain	Space	Query
Exact Neighborhood $N(q,r)$	$O(S_{ANN})$	$\tilde{O}(T_{ANN} + \frac{\lvert N(q,cr)\rvert}{\lvert N(q,r)\rvert})$
Approximate Neighborhood $N(q,r) \subseteq S \subseteq N(q,cr)$	$\tilde{O}(S_{ANN})$	$\tilde{O}(T_{ANN})$

- S_{ANN} and T_{ANN} are the space and query time of standard ANN
- Approximate neighborhood: a set S such that $N(q,r) \subseteq S \subseteq N(q,cr)$
- Dependence on ϵ is $O(\log(\frac{1}{\epsilon}))$
- Black-box reduction
- Our approach solves a more general problem
- Experiments

Summary

- Defined NN problem with respect to fairness, i.e., the sampling variant
  - Applications of sampling NN
- How to sample from a sub-collection of sets
- Improve dependency on ϵ
- How to handle outliers
- Improve dependency on the density parameter of the neighborhood

Summary

Thanks
Questions?

Domain	Space	Query
Exact Neighborhood $N(q,r)$	$O(S_{ANN})$	$\tilde{O}(T_{ANN} + \frac{\lvert N(q,cr)\rvert}{\lvert N(q,r)\rvert})$
Approximate Neighborhood $N(q,r) \subseteq S \subseteq N(q,cr)$	$\tilde{O}(S_{ANN})$	$\tilde{O}(T_{ANN})$

- S_{ANN} and T_{ANN} are the space and query time of standard ANN
- Approximate neighborhood: a set S such that $N(q,r) \subseteq S \subseteq N(q,cr)$
- Dependence on ϵ is $\mathrm{O}(\log(\frac{1}{\epsilon}))$
- Black-box reduction
- Our approach solves a more general problem
- Experiments

Open Problem:

- Finding the optimal dependency on the density parameter