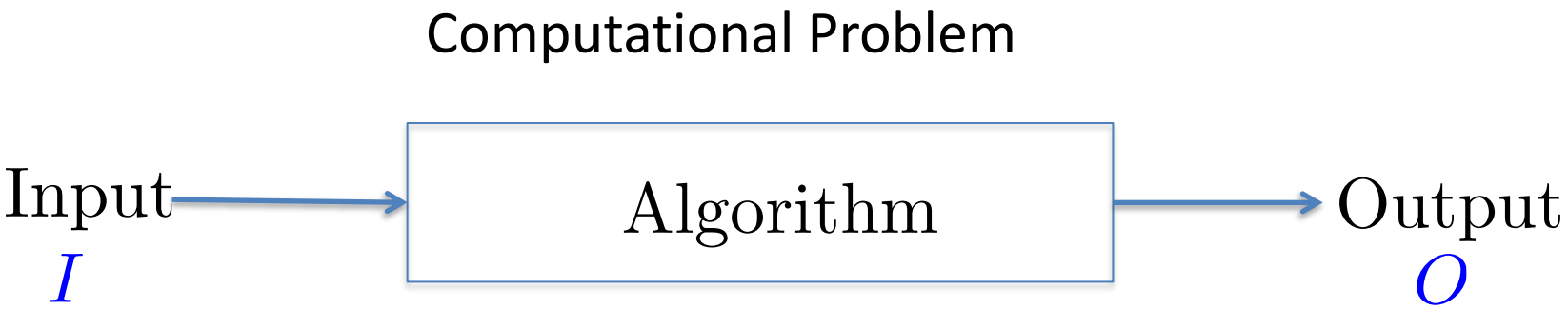
Dynamic Algorithms for Graphs and Clustering

Monika Henzinger

universität wien

Static Algorithm



Performance measures:

1. Running Time
2. Space

What is a Dynamic Algorithm?

Computational Problem

I_0
$I_0 + \Delta I_1$
$I_0 + \Delta I_1 + \Delta I_2$
…

O_0
O_1
O_2
…

Find output without recomputing each time from scratch

What is a Dynamic Algorithm: Data structure

Computational Problem

Operations:

- Initialize(S)
- Update(u)
- Query()
- Query(u)

…

Time per operation?
Total space?

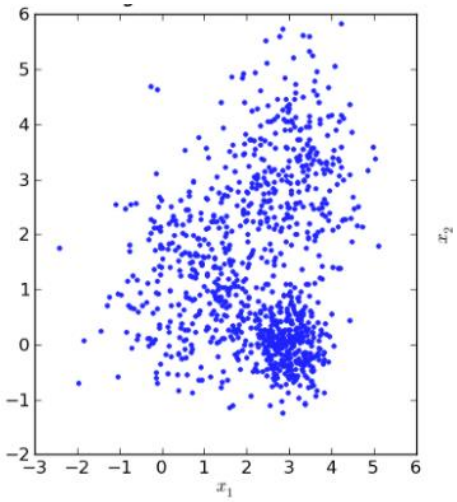
Outline

- Clustering algorithms
- Graph algorithms

Clustering

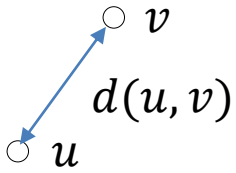
- Define problem(s)
- Algorithms:
  - static
  - dynamic

Metric space



- A metric space (M, d) consists of:
  - finite set of points M
  - Distance function $d: M \times M \to \mathbb{R}_{\geq 0}$, which is a metric, i.e.:
    - $d(u, v) = 0 \Leftrightarrow u = v$
    - $d(u, v) = d(v, u)$
    - Triangle inequality:

$$d(u, v) \leq d(u, x) + d(x, v)$$



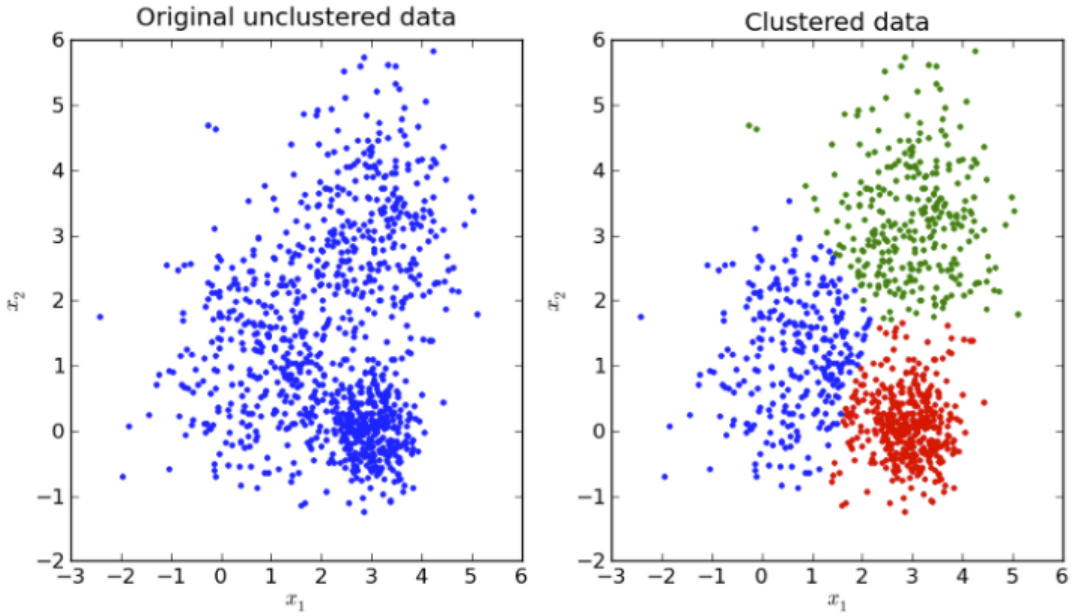
- For simplicity assume:

$$min_{(u,v)\in M\times M} d(u, v) = 1$$

Point set and clustering

- Any **finite** subset $S \subseteq M$ is a point set

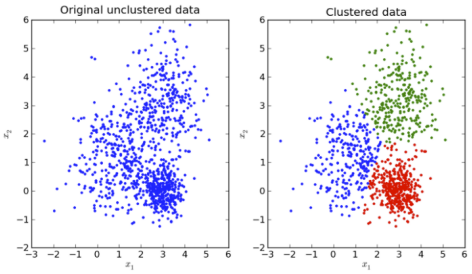
➢ *Clustering is the task of grouping a set of points s.t. points in the same group (cluster) are "closer" to each other than to those in other groups*



Source: Towardsdatascience.com

Point set and clustering

- Any **finite** subset $S \subseteq M$ is a point set
- Clustering is the task of grouping a set of points s.t. points in the same group (cluster) are "closer" to each other than to those in other groups



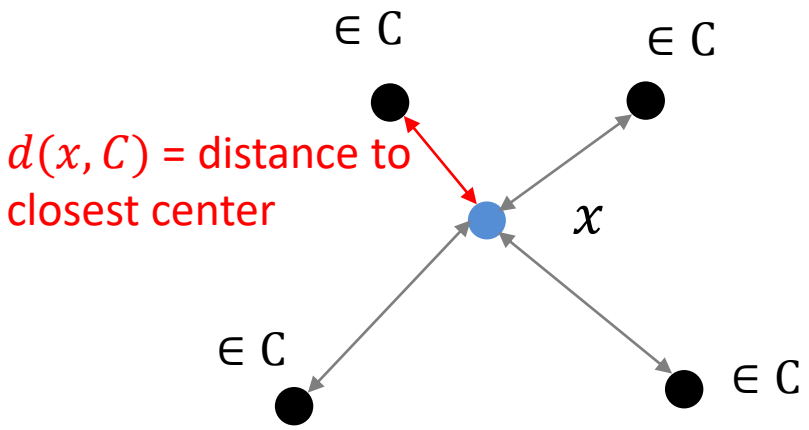
How to measure the quality of a clustering?

➢ *Use a cost function and try to find a clustering that minimizes the cost function*

Which cost function?

One popular approach: Pick set C of k points and call them *centers.*

- *Cost function depends on distance to closest centers, i.e., distance to* C



Which cost function?

One popular approach: Pick k points and call them *centers.*

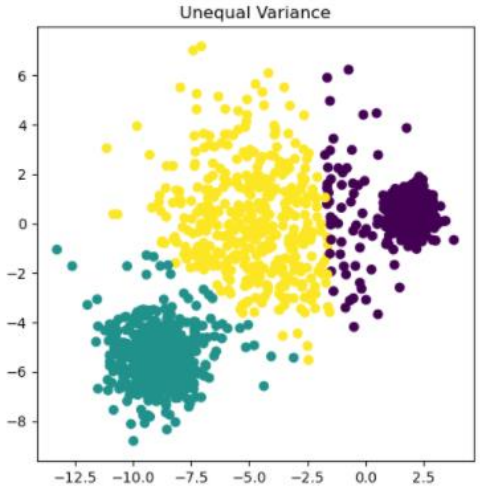
- *Cost function depends on distance to these centers*

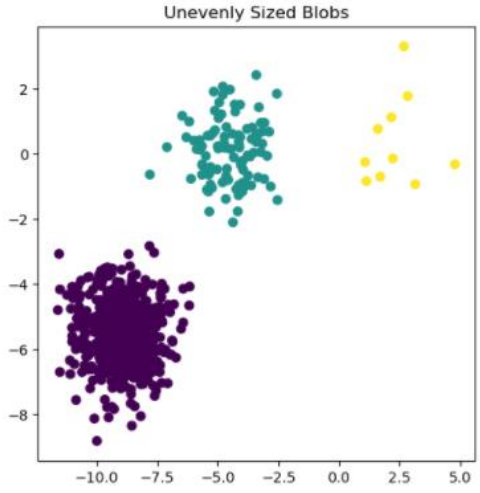
For each $x \in M$: $d(x, C)$= distance to closest center.

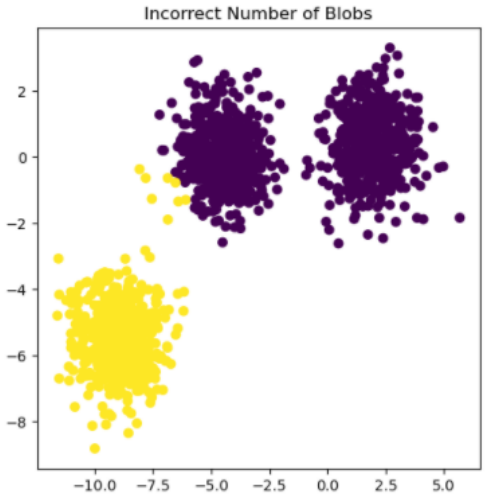
- k-means: $\sum_{x \in S} d(x, C)^2$
- k-median: $\sum_{x \in S} d(x, C)$
- k-center: $max_{x \in S}\, d(x, C)$
- …

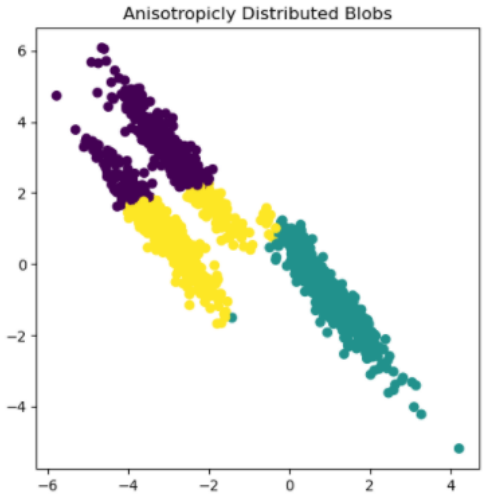
- *Computing k centers that minimize these cost function is NP-hard*

k-means clustering: $\sum_{x \in S} d(x, C)^2$

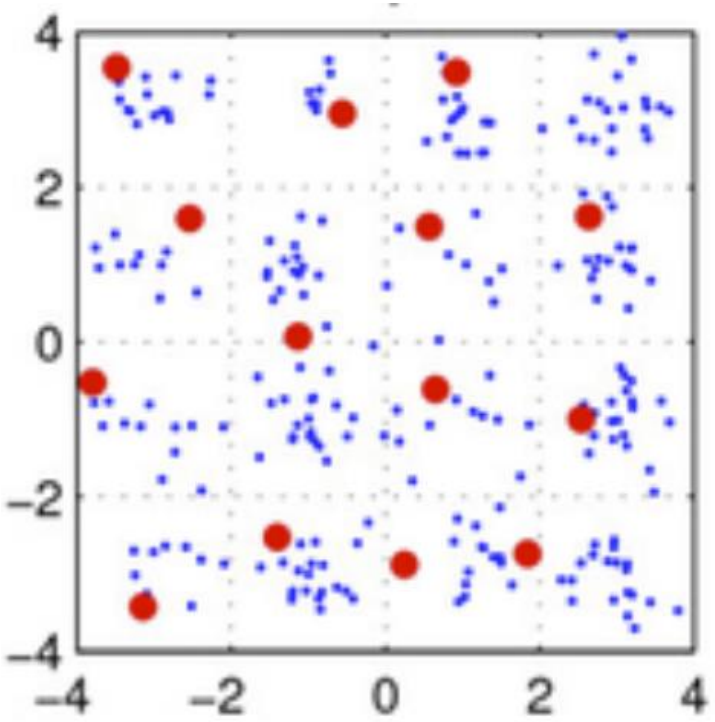








k-center clustering: $max_{x \in S}\, d(x, C)$



Approximation Algorithm

Let $\rho > 1$. An algorithm is a ρ-approximation algorithm for a minimization problem if on any input (M, d) it outputs a value $\text{cost}(M, d)$ such that

$$OPT \leq cost(M, d) \leq \rho \cdot OPT$$

Example: 2-approximate k-center algorithm computes subset $C \subseteq S$ such that

$$OPT \leq max_{x \in S} d(x, C) \leq 2 \cdot OPT$$

Static 2-approx. k-center algorithm

Greedy (furthest-first) approach [Gonzalez '85]:

- $C = \emptyset$
- Arbitrary point of S is added to C
- While $|C| < k$:
  - Add point u of S which maximizes $d(u, C)$ to C

Lemma: Greedy gives a 2-approximation to the optimal k-center clustering.

Running time: $O(kn)$

i.e. $\leq c \cdot k \cdot n$ for some constant c

Static 2-approximate k-center algorithm

Computational Problem:

Compute 2-approximate k-center clustering

Performance measures:

1. Running Time: $O(kn)$ [Gonzalez '85]
2. Space: $O(n)$

Note: $(2-\epsilon)$-approximate k-center is NP-hard

Static Approx. k-Center Results

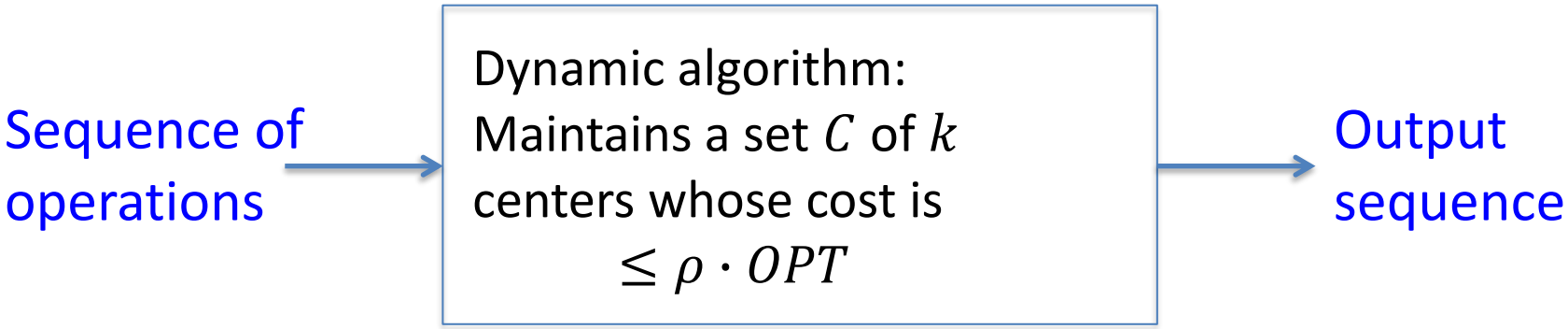
Algorithms:

Approx. ratio	Running Time	Metric	Authors
2	$O(kn)$	general	Gonzalez ’85

Hardness:

Approx. ratio	Metric	Authors
$2-\epsilon$	general	Gonzalez ’85

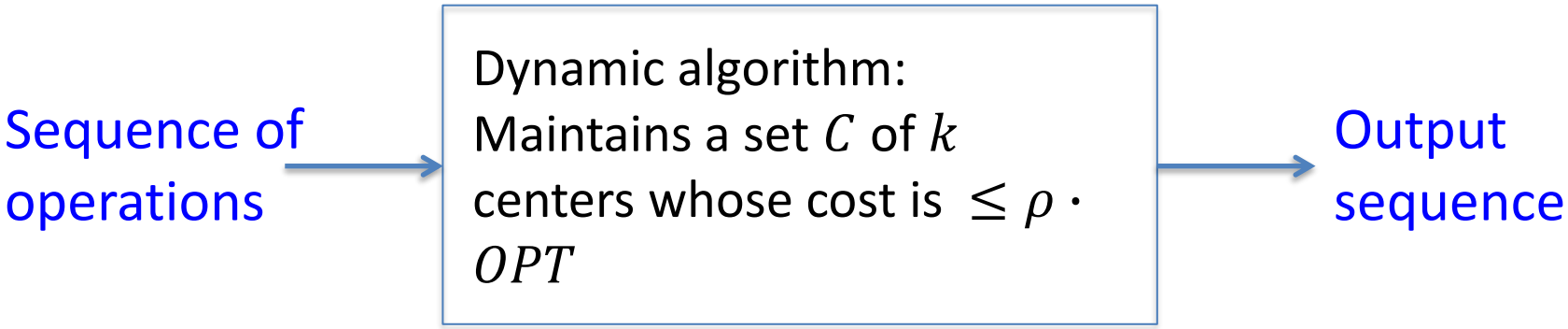
Dynamic ρ-Approx. k-Center Algorithm



Operations:

- Initialize(S)
- InsertPoint(u)
- DeletePoint(u)
- CentersQuery(): Output the k centers
- ValueQuery(): Output a ρ-approximation of $max_{x \in S} d(x, C)$
- PointQuery(u): Return center closest to u

Dynamic ρ-Approx. k-Center Algorithm



Operations:

- Initialize()
- InsertPoint(u)
- DeletePoint(u)
- CentersQuery(): Output the k centers
- ValueQuery(): Output a ρ-approximation of $max_{x \in S} d(x, C)$
- PointQuery(u): Return center closest to u

Time per operation?

Dynamic Approx. k-Center Algorithm

Approx. ratio	Update types	Time per operation	Randomized	Amortized	
$2+\epsilon$	insert-only	$O(\frac{k \log k}{\epsilon \log \epsilon})$	no	yes	McCutchen, Khuller '08
$2+\epsilon$	fully dynamic	$O(\frac{k^2 \log \Delta}{\epsilon})$	yes	yes	Chan, Guerquin, Sozio '18

Special case: Euclidean space $\mathbb{R}^d$?

Static Approx. k-Center Results

Algorithms:

Approx. ratio	Time per operation	Metric	Authors
2	$O(kn)$	general	Gonzalez '85
2	$O(n \log n)$	$\mathbb{R}^d$, constant d	Har-Peled&Mendel '04

Hardness:

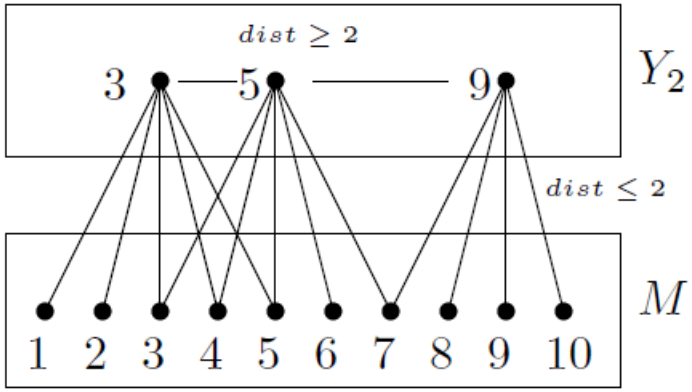
Approx. ratio	Metric	Authors
$2-\epsilon$	general	Gonzalez '85
1.932	$\mathbb{R}^d$, constant d	Mentzer '88

Dynamic Approx. k-Center Algorithm

Approx. ratio	Update types	Time per operation	Randomized	Metric	
$2+\epsilon$	insert-only	$O(\frac{k \log k}{\epsilon \log \epsilon})$	no	general	McCutchen, Khuller '08
$2+\epsilon$	fully dynamic	$O(\frac{k^2 \log \Delta}{\epsilon})$	yes	general	Chan et al. '18
$2+\epsilon$	fully dynamic	$2^{O(d)} \frac{\log \Delta \log \log \Delta}{\epsilon}$	no	$\mathbb{R}^d$, constant d	Goranci, Henzinger, Leniowski, Svozil '19

r-net

Example: 2-net

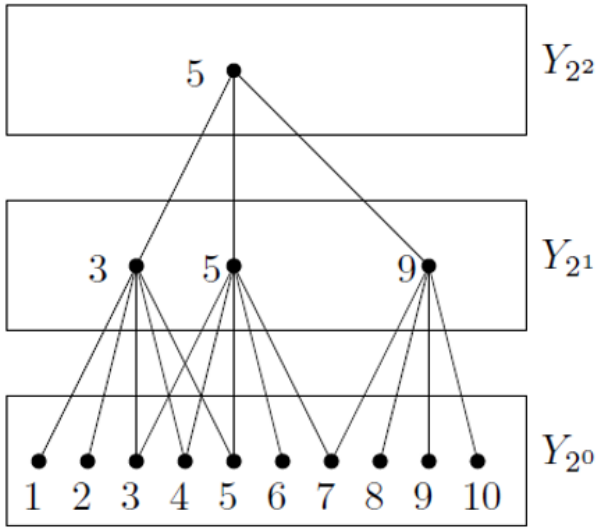


Definition (r-net)

Given a metric space (M, d) and integer $r \geq 0$, r-net $Y_r \subseteq M$ is a set of points (centers), with

- (covering) $M \subseteq \bigcup_{y \in Y_r} B(y, r)$,
- (separating) distinct points $y, y' \in Y_r$ have $d(y, y') \geq r$

Navigating net



Can be maintained
under points updates
in time $O(\log \Delta \log\log \Delta)$
(Krauthgamer Lee '04)

A navigating net is the following hierarchy of 2-nets:

- $Y_{2^0} = S$
- For i = 1 to log n:

  Y_{2^i} = 2^i-net of $Y_{2^{i-1}}$

8-approximation algorithm

- **Step 1:** Construct/update navigating net
- **Step 2:** Output deepest level Y_r with $\leq k$ centers

Lemma: $max_{x \in M} d(x, C) \leq 8 \cdot OPT$

Note:

- Can be improved to $(2 + \varepsilon)$ for any small $\varepsilon > 0$

Experimental Results

$\mathcal{A}_{Cov}$: our algorithm, $2+\epsilon$ approx., $\frac{\log \Delta \log\log \Delta}{\epsilon}$ time per operation

$\mathcal{A}_{CGS}$: Chan et al.' 18, $2+\epsilon$ approx., $\frac{k^2 \log \Delta}{\epsilon}$ time per operation

- Instances:
  - Twitter (geotagged tweets),
  - Flickr (metadata from pictures) and
  - Random (sample centers then sample points for each centers)
- All instances use euclidean distance
- Dynamic Instances:
  - Random insert/delete (30% deletions, 10% deletions, 5% deletions)
  - Sliding Window: insert point at time t and remove at time $t+W$

Experimental Results

Speedup of $\mathcal{A}_{Cov}$ over $\mathcal{A}_{CGS}$

$\epsilon \rightarrow$ $k \downarrow$	0.1	0.5	1.0	4.0
20	0.02	0.14	0.32	0.72
50	0.10	0.59	**1.34**	**3.05**
100	0.33	**2.01**	**4.45**	**10.32**
200	**1.15**	**7.66**	**17.74**	**39.60**

Solution improvement of $\mathcal{A}_{Cov}$ over $\mathcal{A}_{CGS}$

20	0.97	**1.12**	**1.27**	**1.07**
50	0.97	**1.10**	**1.36**	**1.46**
100	0.96	**1.12**	**1.12**	**2.14**
200	0.96	**1.12**	**1.19**	**1.28**

- Larger k and $\epsilon \rightarrow$ larger speedups of $\mathcal{A}_{\mathrm{Cov}}$ over $\mathcal{A}_{\mathrm{CGS}}$
- For $k = 200$, $\mathcal{A}_{\mathrm{Cov}}$ is faster than $\mathcal{A}_{\mathrm{CGS}}$ for all values of ϵ.
- Solution quality: For $\epsilon \geq 0.5 \rightarrow \mathcal{A}_{\mathrm{Cov}}$ $10 - 12\%$ better solutions

Dynamic graph algorithms

- Define problem
- State of the art

What is a *(Fully) Dynamic Graph Algorithm*?

Computational graph problem

Operations:

- Initialize(G)
- InsertEdge(u, v, $weight$)
- DeleteEdge(u,v)
- (InsertNode(u))
- (DeleteNode(u))
- Query(G) or Query(u) or Query(u, v)

Motivation

1. **Real-world applications**
    1. 62% graphs in survey of Sahu et al. ’17 are dynamic
2. **Fundamental computational question**
3. **Subroutines in static graph algorithms**

Update time for "classic" problems

with polylog query time

- **Undirected:** For any small constant $\epsilon > 0$
  - **Connectivity:** $\Omega(\log n)$ [Patrascu-Demaine '04], $\tilde{O}(1)$ [H-King '95, Holm-deLichtenberg-Thorup '98]
  - **MST:** $\Omega(\log n)$ [Patrascu-Demaine '04], $\tilde{O}(1)$ [Holm-deLichtenberg-Thorup '98]
  - **Single-source shortest paths**:
    - Exact: $\Omega(m^{1-\epsilon})$,$\tilde{O}(m)$
  - **All-pairs shortest paths**: Exact: $\Omega(m^{1-\epsilon})$, $\tilde{O}(n^2)$ [Demetrescu-Italiano '03]
  - **Minimum Cut**:
    - $\Omega(n^{1-\epsilon})$ weighted exact, $\tilde{O}(m)$
  - **Maximum cardinality matching**:
    - $\Omega\left(m^{1/2-\epsilon}\right)$ exact, $O(m)$
    - 2-approximate: $O(1)$ [Salomon '16, Bhattacharya-H-Italiano'15, Bhattacharya-Kulnikarni '18]

Update time for ``classic'' problems

with **polylog query time**

- **Directed:** For any small constant $\epsilon > 0$
  - **Reachability/SSSP:**
    - $\Omega(m^{1-\epsilon})$, $O(m)$
  - **SCC:** Is the graph strongly connected?
    - $\Omega(m^{1/2-\epsilon})$, $O(\min(m, n^{1.406}))$ [van den Brand, Nanongkai, Saranurak '19]
  - **Transitive closure:**
    - $\Omega(m^{1-\epsilon})$, $\tilde{O}(n^2)$
  - **All-pairs shortest paths:**
    - $\Omega(m^{1-\epsilon})$, $\tilde{O}(n^2)$ update, $\tilde{O}(n)$ path reporting query Demetrescu-Italiano '03]

Update time for approximation algorithms

with polylog query time

- $(\Delta + 1) -$ **vertex coloring:**
  - $O(1)$ [H-Peng '19, Bhattacharya, Grandoni, Kulkarni, Liu '19]
- $(1 + \varepsilon)$**- approx min spanning forest value:**
  - $O(1)$ if max weight is $O(m^{1/3})$ [H-Peng'19]
- **Edge orientation with low outdegree:**
  - $O(1)$ for $O\left((\log n)^2\right)$ −approximation [Berglin-Brodal '17]
- $(1 + \varepsilon)$**- approx densest subgraph and** $(2 + \varepsilon)$**- approx degeneracy :**
  - $\tilde{O}(1)$ [Sawlani-Wang '19]
- $(4 + \varepsilon)$**- approx k-core decomposition:**
  - $\tilde{O}(1)$ [Sun-Chan-Sozio '20]
- $(1 + \varepsilon)$**- electrical flow:**
  - $O(\min(m^{3/4}, n^{5/6}))$ [Durfee-Gao-Goranci-Peng '19]

Summary

- Dynamic k-center clustering:
  - Efficient algorithms exist
- Dynamic k-means clustering:
  - More research needed
- Dynamic graph algorithms:
  - Efficient algorithms for many problems exist – or are not possible
  - More research needed:
    - Dynamic graph partitioning into roughly equally-sized subgraphs
    - Dynamic graph clustering