Local Graph Sparsification for Scalable Clustering

Venu Satuluri, Srinivasan Parthasarathy and Yiye Ruan

Ohaio State Univeristy

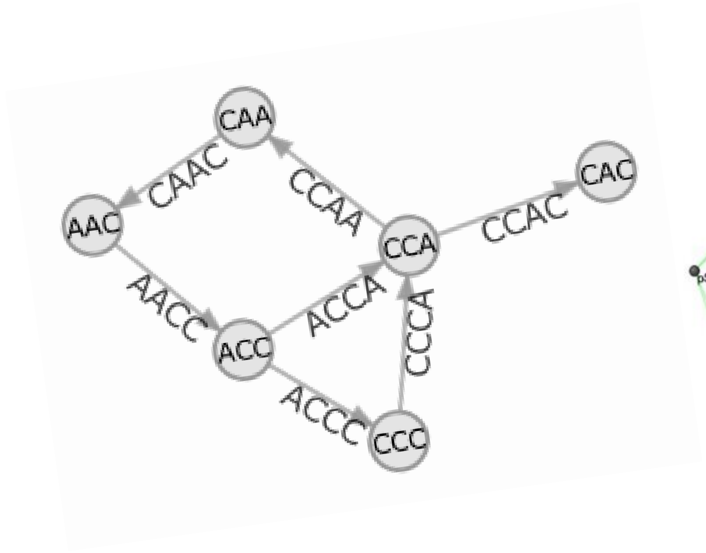
Presented By: Ahmed Elbagoury

Outline

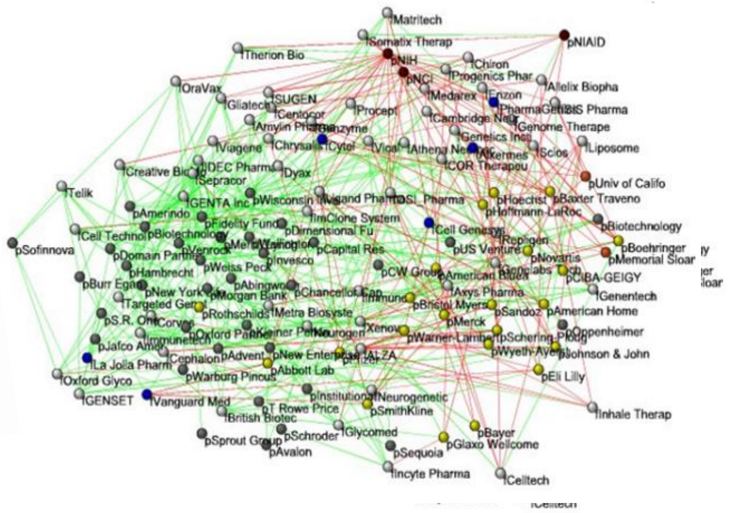
- Motivation
- Related Work
- Graph Sparsification
  - Global Sparsification
  - Local Sparsification
  - Minwise Hashing
- Experimental Evaluation
- Conclusion
- Discussion

Motivation

- Man real problems can be modeled as graphs
- Analysing these graphs using clustering



Genetic Interaction Networks


Economic Networks

Social Network community discovering

Motivation

- Many graph clustering algorithms have been proposed
- Scalability of these algorithm is a challenge
- Large graphs are common in WWW domain
  - Facebook and Twitter

The goal is:

Develop a preprocessing step
- To speedup graph clustering algorithms
- Without loosing quality of the clusters

Not developing a clustering algorithm

Outline

- Motivation
- Related Work
- Graph Sparsification
  - Global Sparsification
  - Local Sparsification
  - Minwise Hashing
- Experimental Evaluation
- Conclusion
- Discussion

Related Work

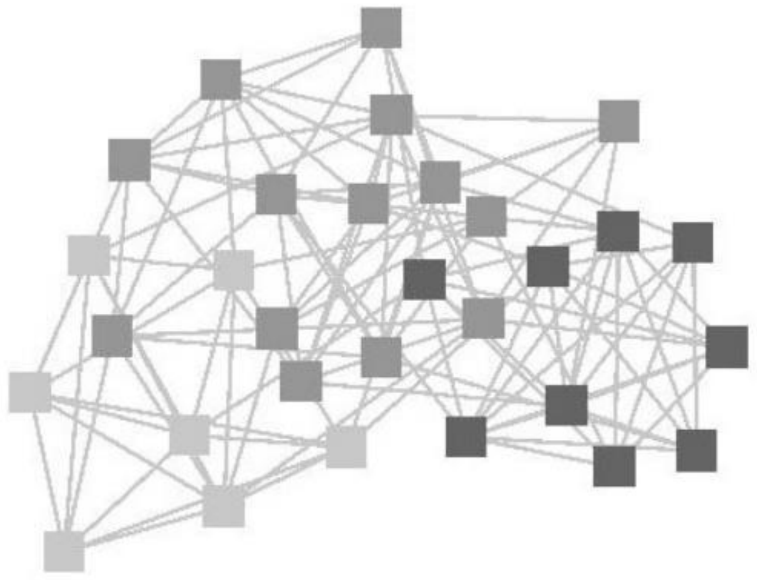
- Preserving edge cut
  - Random Sampling
  - Edge Connectivity
  - Resistance of an edge
- Edge filtering
  - Identifies the most interesting edges without keeping cluster structure
  - Inappropriates for unweight networks
- Graph Sampling
  - Selects subset of edges and nodes
  - How to cluster missing nodes?!

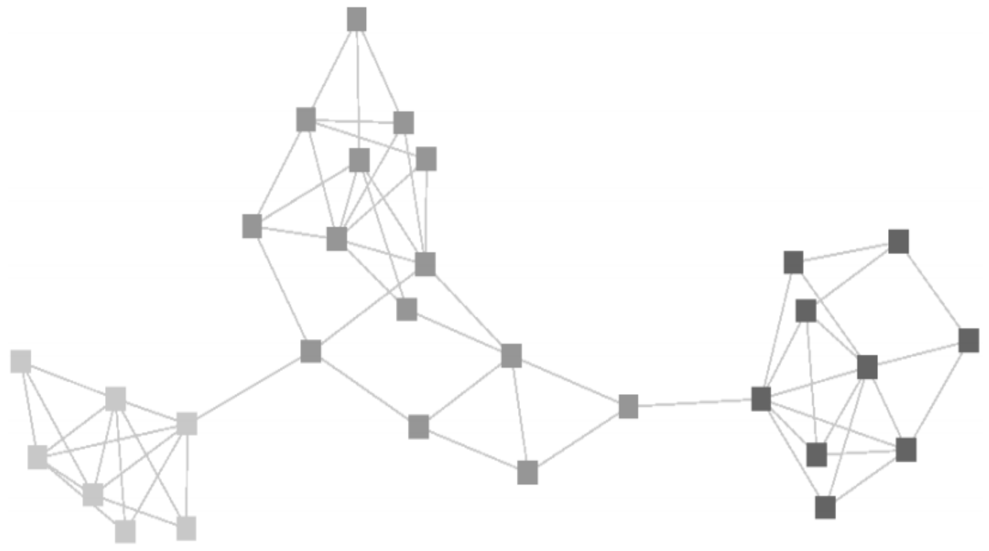
Outline

- Motivation
- Related Work
- Graph Sparsification
  - Global Sparsification
  - Local Sparsification
  - Minwise Hashing
- Experimental Evaluation
- Conclusion
- Discussion

Graph Sparsification

- The objective is scaling up clustering algorithms
- Reduce the size of the graph
- Sparsify the graph: Filter only some edges and retain all the nodes



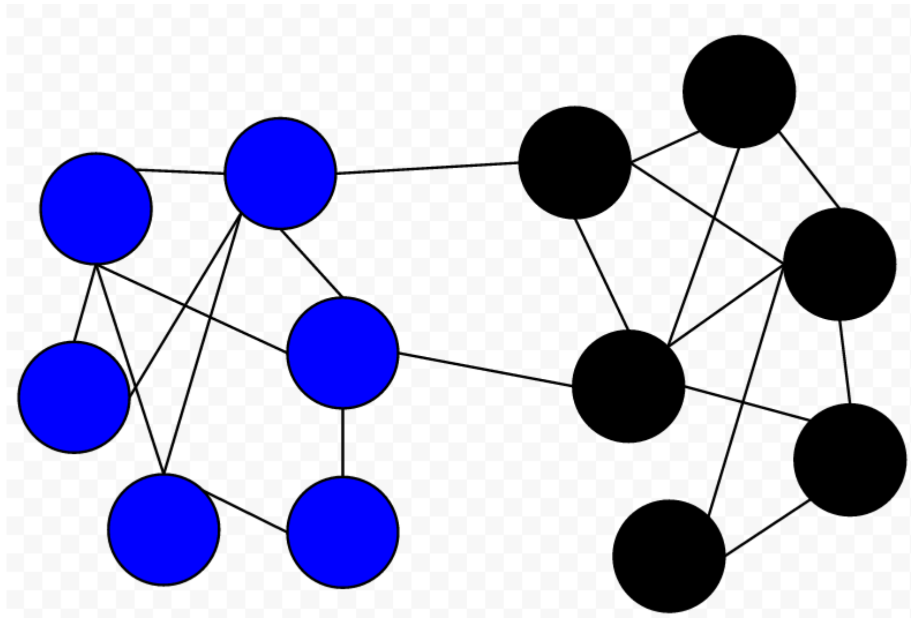


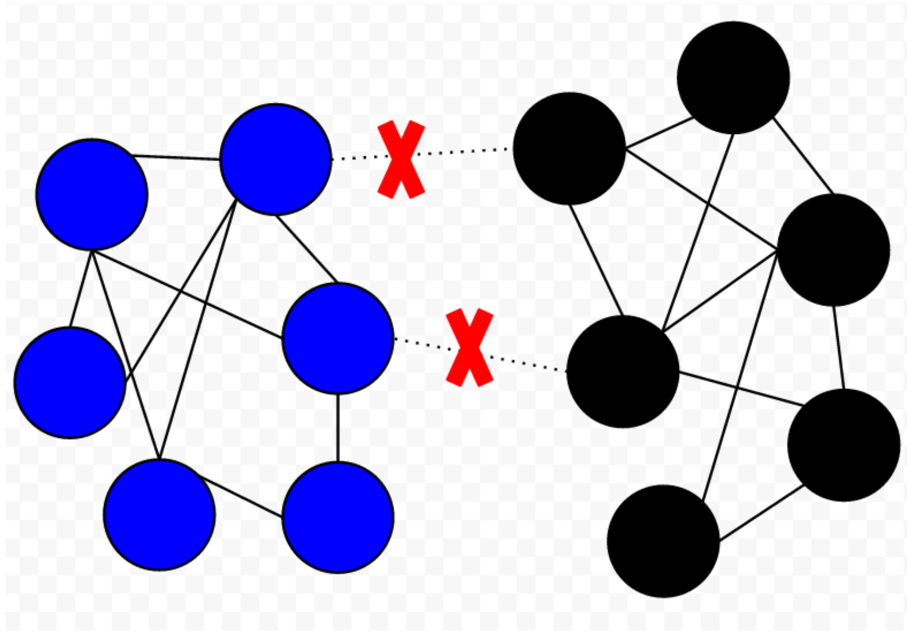
Sparsification Method

- Clustering sparsified graph should be much faster than the original graph
- Retain good clustering quality compared to the clustering over the original graph
- Fast sparsification method

Sparsification Method

Retain the intra-cluster edges compared to inter-cluster edges

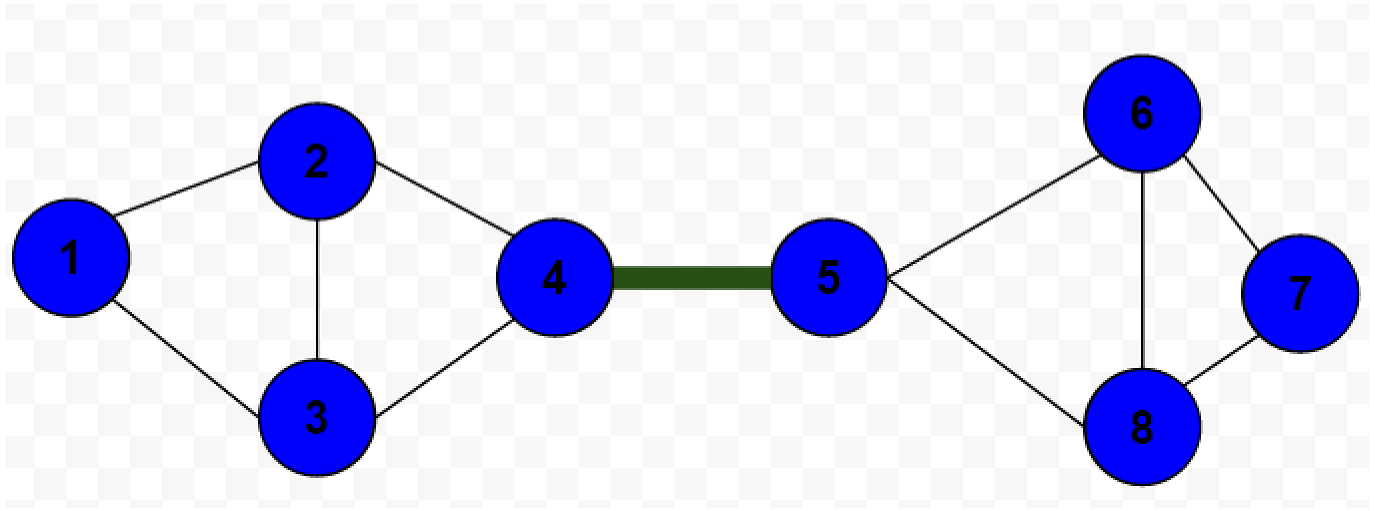




Intra and inter edges

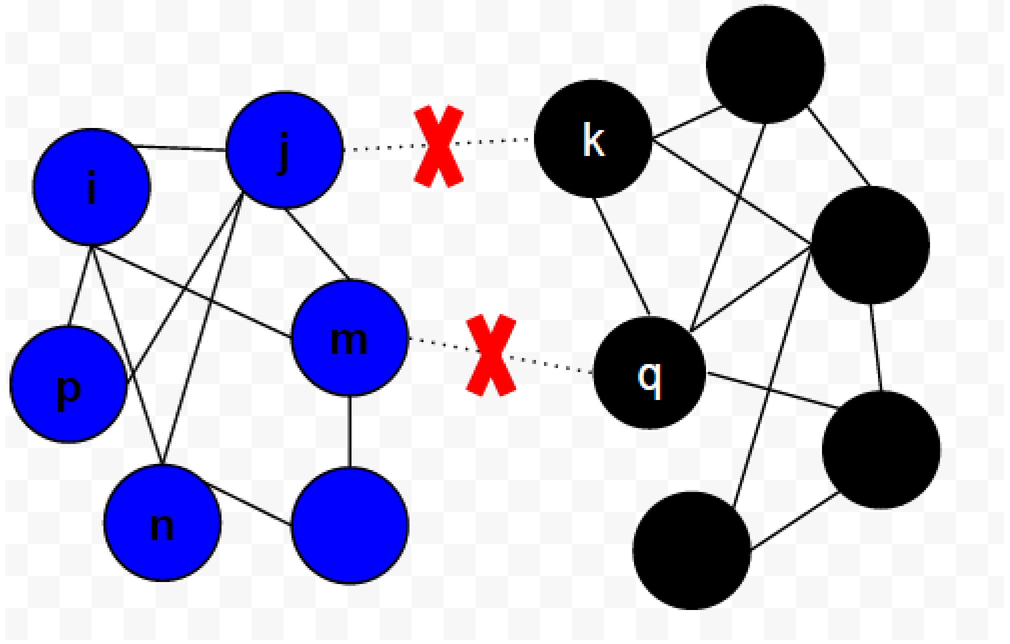
One way to differentiate between intra and inter edges is

- Edge betweenness centrality
  - Number of shortest paths between any two vertices that pass through the edge (i, j)
  - The higher the betweenness the higher the edge is an inter-cluster edge
  - Expensive to compute the betweenness



Similarity-based Sparsification Heuristic

- An edge (i, j) is likely to lie within a cluster
  - If the vertices i and j have adjacency list with high overlap

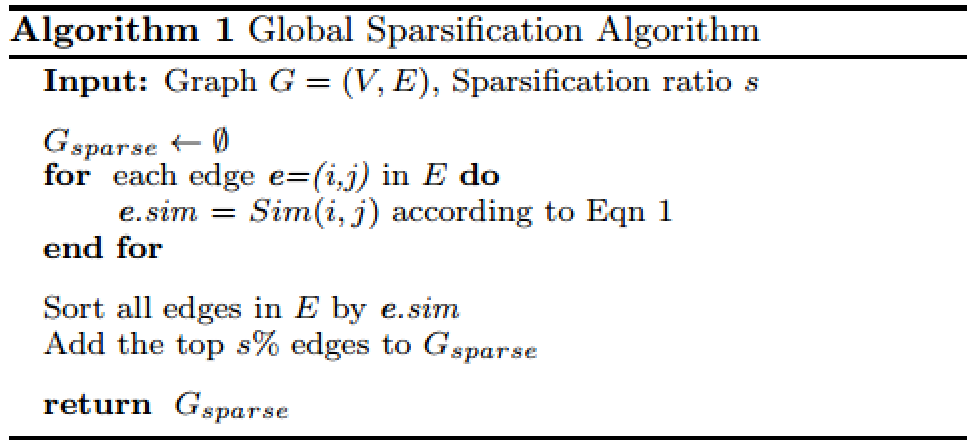


- Jaccard measure quantifies the overlap between adjacency lists

$$Sim(i, j) = \frac{|Adj(i) \cap Adj(j)|}{|Adj(i) \cup Adj(j)|}$$

Global Sparsification (G-Spar)

- Each edge (i, j), will be assigned a similarity $sim(i, j)$
- Return the graph with the top $s\%$ of all the edges in the graph
  - s is a sparsification parameter

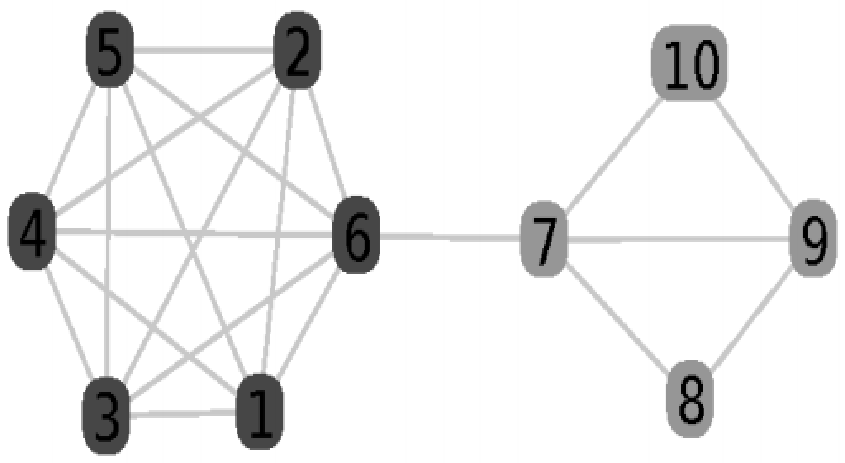


Algorithm 1 Global Sparsification Algorithm

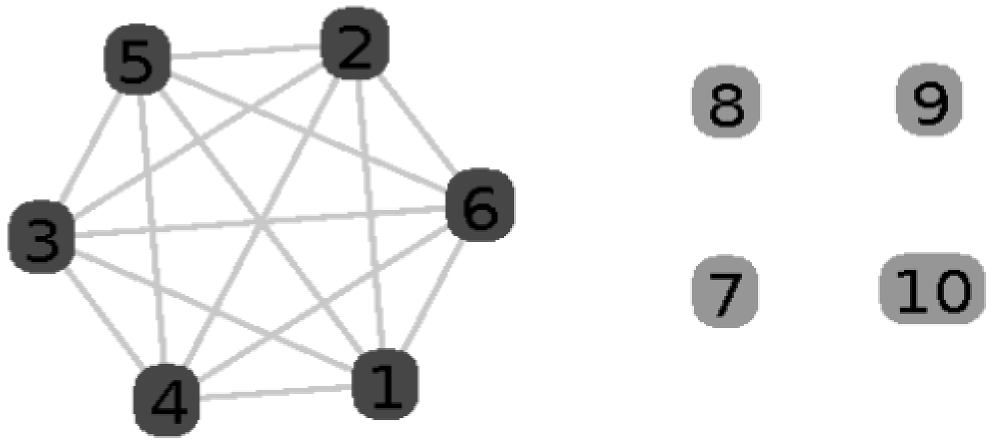
```
Input: Graph $G = (V, E)$, Sparsification ratio $s$

$G_{sparse} \leftarrow \emptyset$
for each edge $e$=$(i,j)$ in $E$ do
    $e.sim = Sim(i, j)$ according to Eqn 1
end for

Sort all edges in $E$ by $e.sim$
Add the top $s\%$ edges to $G_{sparse}$

return $G_{sparse}$
```

Global Sparsification(G-Spar)

Using one global threshold is not suitable for multiple density clusters



Original graph

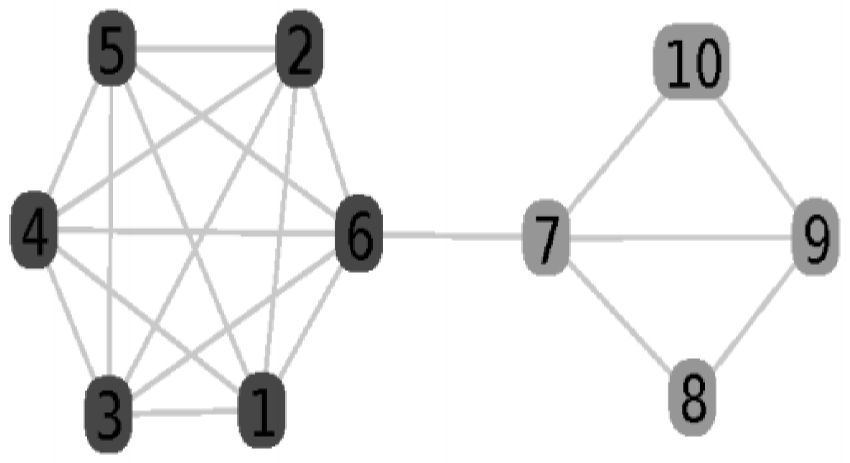


Globally sparsified graph

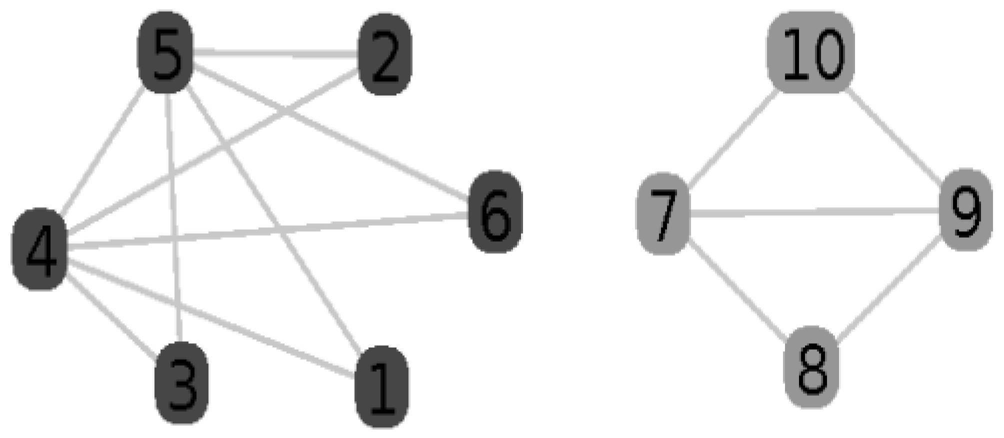
Local Sparsification (L-Spar)

- Avoid using a global threshold
- For each node i (with degree d_i), compute the similarity of each edge
- Pick the top $f(d_i) = d_i^e$
  - Ensure that for each node, at least one of its incident edges is picked
- Adapt to different densities

Local Sparsification (L-Spar)



Original graph



Locally sparsified graph

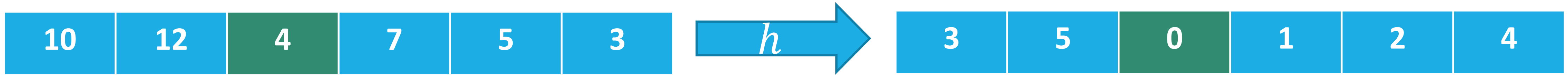
Time Complexity

- For node i (with degree d_i) and node j (with degree d_j)
  - Number of operations to intersect their adjacency lists is proportional to $d_i + d_j$
- A node of degree d_i requires d_i intersections
  - The total number of operations is proportional to $\sum_i d_i^2$
- The total number of operations is proportional to $n \cdot d_{avg}^2$
  - d_{avg} is the average degree of the graph

Prohibitively large for most graphs

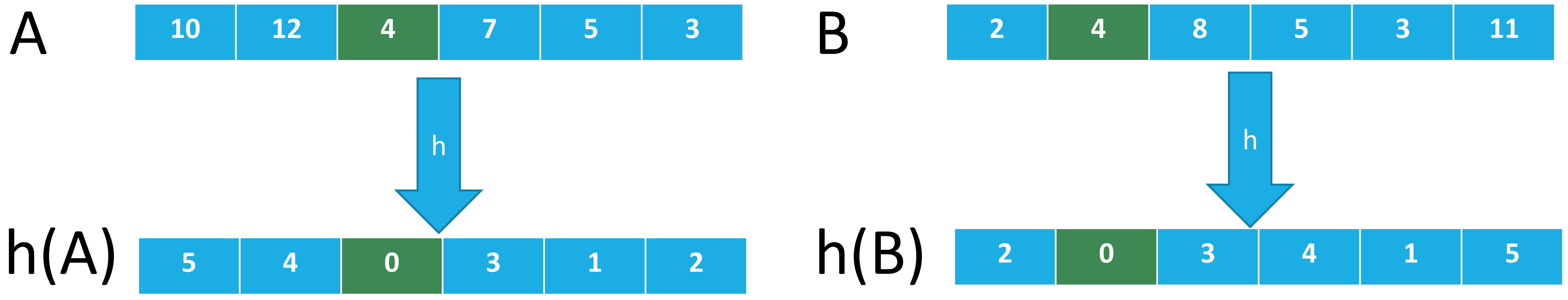
Minwise Hashing

- Min-wise hashing is as a technique for estimating the jaccard coefficient between sets A and B
- Let h be a hash function that maps the members of a set S of size n to the range $\{0, \ldots., n-1\}$



- Define $h_{min}(S)$ to be $\min_{x \in S} h(x)$
  - The member x of S with the minimum value of $h(x)$
  - $h_{min}(S)$ = 4

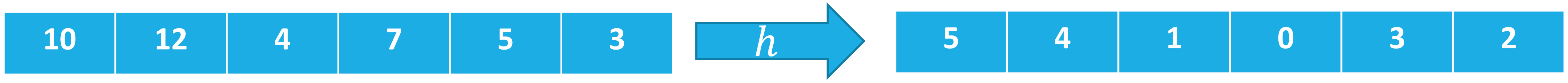
Minwise Hashing



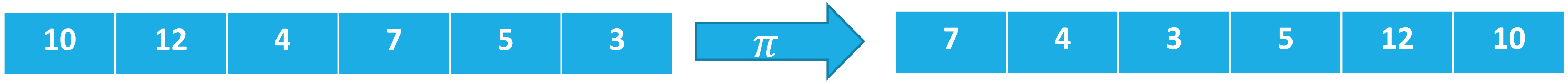
- $h_{min}(A) = h_{min}(B)$ when $h_{min}(A)$ is in B
- Which means $h_{min}(A \cup B)$ lies in the intersection $A \cap B$
- This happens with probability $\frac{|Adj(i) \cap Adj(j)|}{|Adj(i) \cup Adj(j)|}$ Which is equal to $sim(i, j)$

Minwise Hashing

- A simple estimator for $sim(A, B)$ is $I[h_{min}(A) = h_{min}(B)]$
  - $I[x] = 1$ if x is true and 0 otherwise.
- Hashing can be considered as a random permutation π on n numbers rather than a hash function



$$h_{min}(A) = 7$$

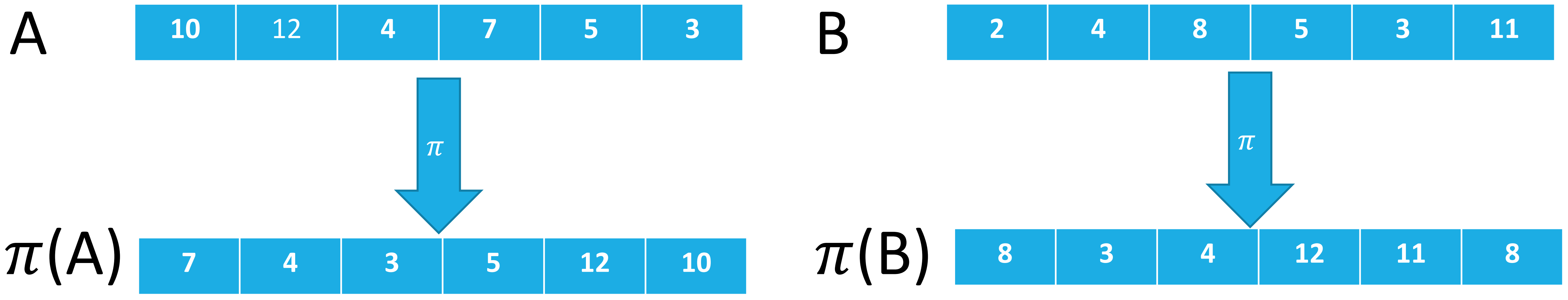


$$\text{first}(\pi(A)) = 7$$

which is the first element in $\pi(A)$

Minwise Hashing

This can be considered as a random permutation π on n numbers rather than a hash function



$\text{first}(\pi(A)) = 7$

$\text{first}(\pi(B)) = 8$

Minwise Hashing

- A simple estimator for $sim(A, B)$ is $I\left[first\left(\pi(A)\right) = first\left(\pi(B)\right)\right]$
- This estimate has a high variance
  - It is always zero or one

Minwise Hashing

To reduce variance, use k independent permutations π_i , $i = 1{:}\,k$

A	10	12	4	7	5	3	
π_1(A)	7	4	3	5	12	10	$first(\pi_1(A)) = 7$
π_2(A)	5	7	4	10	3	12	$first(\pi_2(A)) = 5$
π_3(A)	4	10	7	3	12	5	$first(\pi_3(A)) = 4$

Minwise Hashing

- To reduce variance, use k independent permutations π_i , $i = 1{:}\,k$

$$sim(A,B) = \frac{1}{k}\sum_{i=1}^{k} I[first(\pi_i(A)) = first(\pi_i(B))]$$

Generating Permutations

- Linear permutations

$$\pi(i) = (a * i + b) \% P$$

- P is large prime number
- a is an integer drawn uniformly at random from $[1, P - 1]$
- b is an integer drawn uniformly at random from $[0, P - 1]$
- Multiple linear permutations can be generated by generating random pairs of (a, b, P)

Minwise Hashing

- Generate k linear permutations, by generating k triplets (a, b, P)
- Compute a length k signature for each node by minhashing its adjacency list k times
- Fill up a hashtable of size $n * k$
- $\forall (i, j) \in G$
  - Compare the signatures of i and j
  - Count the number of matching minhashes
- Sort the edges incident to a node i by the number of matches of each edge
- Include the top d_i^e inclusion in the sparsified graph

Outline

- Motivation
- Related Work
- Graph Sparsification
  - Global Sparsification
  - Local Sparsification
  - Minwise Hashing
- Experimental Evaluation
- Conclusion
- Discussion

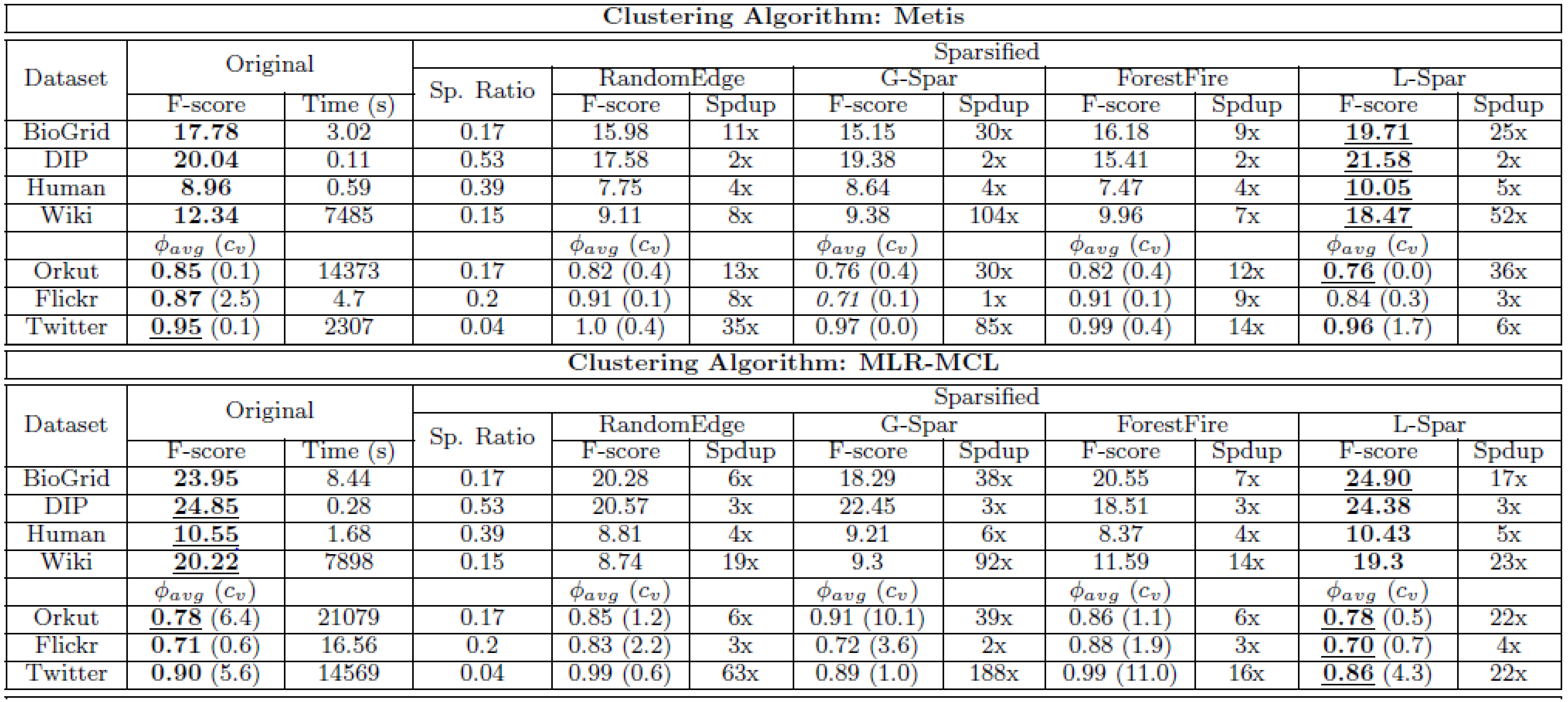
Experimental Results

- Seven real-world datasets
  - Three biological datasets: Yeast-BioGrid (BioGrid), Yeast-DIP (DIP), Human-PPI (Human)
  - Wikipedia: an undirected version of the graph of hyperlinks between Wikipedia articles
  - Three social networks datasets: Orkut, Twitter, Flickr tags
- Baselines
  - L-Sparse
  - G-Sparse
  - Random Edge Sampling
  - ForestFire
- All the experiments were performed on a PC with 16 GB RAM

Experimental Results

- Average F-score (higher is better) when ground truth is available
- Average Conductance (lower is better) when ground truth is not available
  - $\phi(S) = \frac{|(i,j)\ st\ i \in S, j \in \bar{S}|}{\min(|(i,j)\ st\ i \in S, j \in V|,\ |(i,j)\ st\ i \in \bar{S},\ j \in V|\ )}$
- Four state-of- the-art algorithms for clustering
  - Metis
  - Metis+MQI
  - MLR-MCL
  - Graclus

Experimental Results



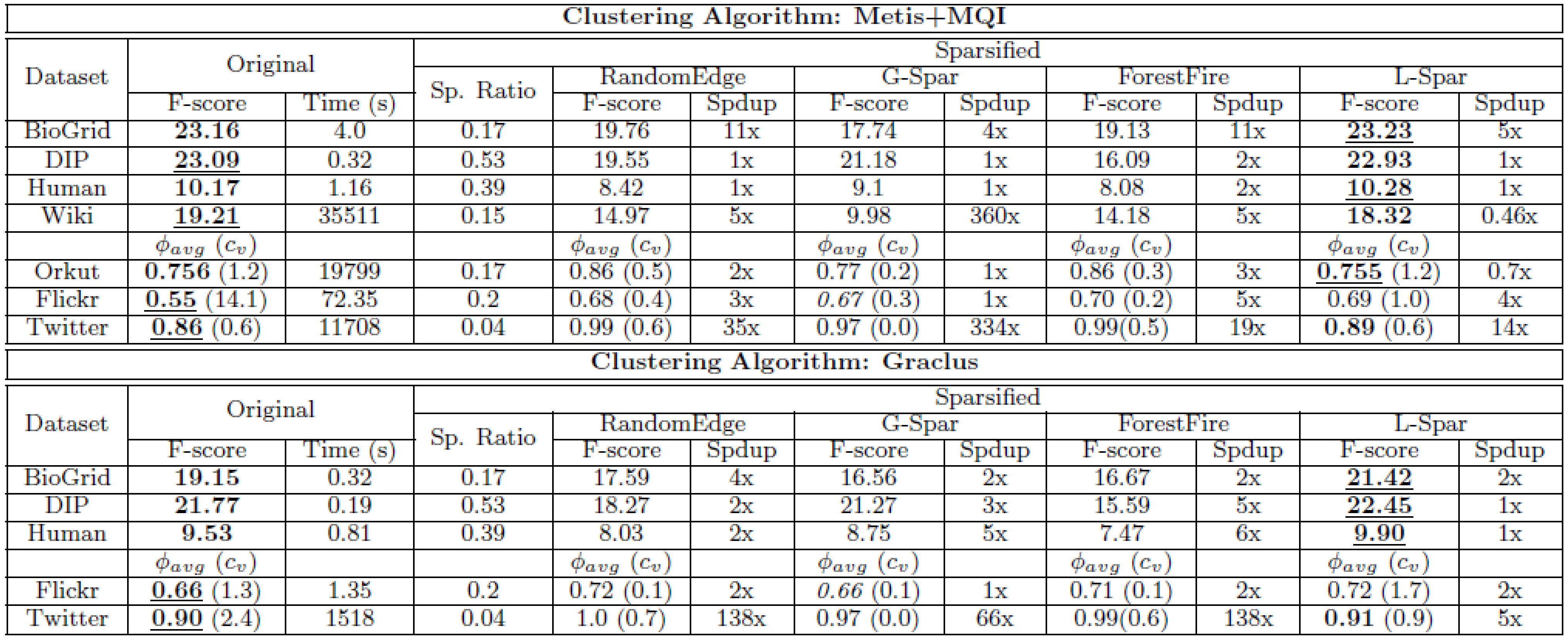
Clustering Algorithm: Metis

Dataset	Original		Sparsified								
			Sp. Ratio	RandomEdge		G-Spar		ForestFire		L-Spar	
	F-score	Time (s)		F-score	Spdup	F-score	Spdup	F-score	Spdup	F-score	Spdup
BioGrid	**17.78**	3.02	0.17	15.98	11x	15.15	30x	16.18	9x	**19.71**	25x
DIP	**20.04**	0.11	0.53	17.58	2x	19.38	2x	15.41	2x	**21.58**	2x
Human	**8.96**	0.59	0.39	7.75	4x	8.64	4x	7.47	4x	**10.05**	5x
Wiki	**12.34**	7485	0.15	9.11	8x	9.38	104x	9.96	7x	**18.47**	52x
	ϕ_{avg} (c_v)			ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)	
Orkut	**0.85** (0.1)	14373	0.17	0.82 (0.4)	13x	0.76 (0.4)	30x	0.82 (0.4)	12x	**0.76** (0.0)	36x
Flickr	**0.87** (2.5)	4.7	0.2	0.91 (0.1)	8x	*0.71* (0.1)	1x	0.91 (0.1)	9x	0.84 (0.3)	3x
Twitter	**0.95** (0.1)	2307	0.04	1.0 (0.4)	35x	0.97 (0.0)	85x	0.99 (0.4)	14x	**0.96** (1.7)	6x

Clustering Algorithm: MLR-MCL

Dataset	Original		Sparsified								
			Sp. Ratio	RandomEdge		G-Spar		ForestFire		L-Spar	
	F-score	Time (s)		F-score	Spdup	F-score	Spdup	F-score	Spdup	F-score	Spdup
BioGrid	**23.95**	8.44	0.17	20.28	6x	18.29	38x	20.55	7x	**24.90**	17x
DIP	**24.85**	0.28	0.53	20.57	3x	22.45	3x	18.51	3x	**24.38**	3x
Human	**10.55**	1.68	0.39	8.81	4x	9.21	6x	8.37	4x	**10.43**	5x
Wiki	**20.22**	7898	0.15	8.74	19x	9.3	92x	11.59	14x	**19.3**	23x
	ϕ_{avg} (c_v)			ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)	
Orkut	**0.78** (6.4)	21079	0.17	0.85 (1.2)	6x	0.91 (10.1)	39x	0.86 (1.1)	6x	**0.78** (0.5)	22x
Flickr	**0.71** (0.6)	16.56	0.2	0.83 (2.2)	3x	0.72 (3.6)	2x	0.88 (1.9)	3x	**0.70** (0.7)	4x
Twitter	**0.90** (5.6)	14569	0.04	0.99 (0.6)	63x	0.89 (1.0)	188x	0.99 (11.0)	16x	**0.86** (4.3)	22x

Experimental Results



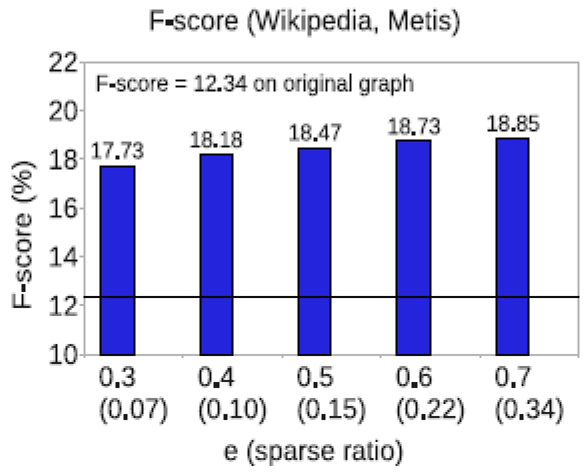
Clustering Algorithm: Metis+MQI

Dataset	Original			Sparsified							
			Sp. Ratio	RandomEdge		G-Spar		ForestFire		L-Spar	
	F-score	Time (s)		F-score	Spdup	F-score	Spdup	F-score	Spdup	F-score	Spdup
BioGrid	**23.16**	4.0	0.17	19.76	11x	17.74	4x	19.13	11x	**23.23**	5x
DIP	**23.09**	0.32	0.53	19.55	1x	21.18	1x	16.09	2x	**22.93**	1x
Human	**10.17**	1.16	0.39	8.42	1x	9.1	1x	8.08	2x	**10.28**	1x
Wiki	**19.21**	35511	0.15	14.97	5x	9.98	360x	14.18	5x	**18.32**	0.46x
	ϕ_{avg} (c_v)			ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)	
Orkut	**0.756** (1.2)	19799	0.17	0.86 (0.5)	2x	0.77 (0.2)	1x	0.86 (0.3)	3x	**0.755** (1.2)	0.7x
Flickr	**0.55** (14.1)	72.35	0.2	0.68 (0.4)	3x	*0.67* (0.3)	1x	0.70 (0.2)	5x	0.69 (1.0)	4x
Twitter	**0.86** (0.6)	11708	0.04	0.99 (0.6)	35x	0.97 (0.0)	334x	0.99(0.5)	19x	**0.89** (0.6)	14x

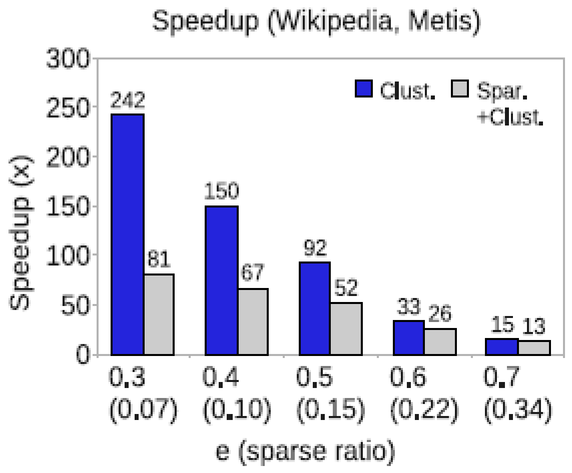
Clustering Algorithm: Graclus

Dataset	Original			Sparsified							
			Sp. Ratio	RandomEdge		G-Spar		ForestFire		L-Spar	
	F-score	Time (s)		F-score	Spdup	F-score	Spdup	F-score	Spdup	F-score	Spdup
BioGrid	**19.15**	0.32	0.17	17.59	4x	16.56	2x	16.67	2x	**21.42**	2x
DIP	**21.77**	0.19	0.53	18.27	2x	21.27	3x	15.59	5x	**22.45**	1x
Human	**9.53**	0.81	0.39	8.03	2x	8.75	5x	7.47	6x	**9.90**	1x
	ϕ_{avg} (c_v)			ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)		ϕ_{avg} (c_v)	
Flickr	**0.66** (1.3)	1.35	0.2	0.72 (0.1)	2x	*0.66* (0.1)	1x	0.71 (0.1)	2x	0.72 (1.7)	2x
Twitter	**0.90** (2.4)	1518	0.04	1.0 (0.7)	138x	0.97 (0.0)	66x	0.99(0.6)	138x	**0.91** (0.9)	5x

Speedup and F-score as *e* changes

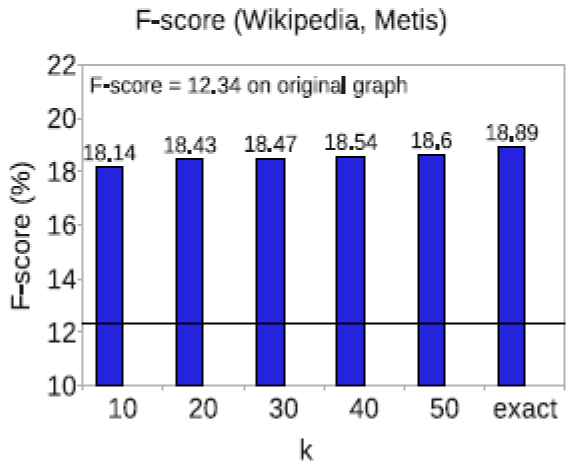


(a) F-score as e changes; Metis

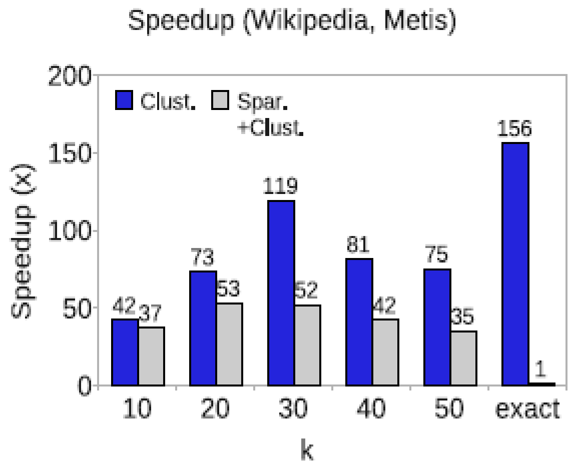


(b) Speedup as e changes; Metis

Speedup and F-score as k changes



(e) F-score as k changes; Metis



(f) Speedup as k changes; Metis

Conclusion

- Scaling clustering algorithm is an important task
- Introduced an efficient and effective localized method to sparsify a graph
  - Retain only a fraction of the original number of edges
  - Handless multiple density clusters
- The proposed sparsification speeds up graph clustering algorithms without sacrificing quality

Thanks

Questions?

Discussion

Implementing the sparsification method on multiple machines

- How to partition the data?
- Some framework doesn’t support Graph mutations: GraphLab and PowerGraph

How to handle streams (incremental clustering)?

- Will we need to re-cluster
- Will the density change, do we need to change e

Can we use other measures that takes in account edge weight and edge direction?

How to handle outliers more efficiently?