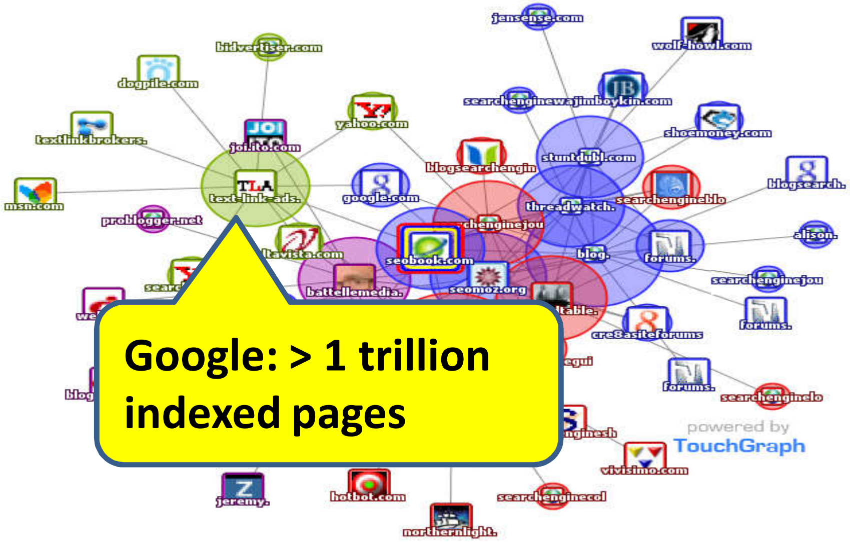
Summarizing Static and Dynamic Big Graphs

Adapted from Tutorial of University, Singapore

Big-Graphs

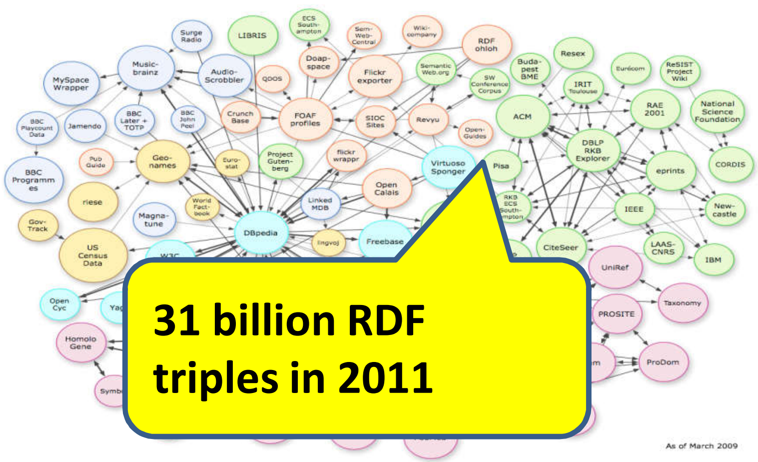


Google: > 1 trillion indexed pages

Web Graph

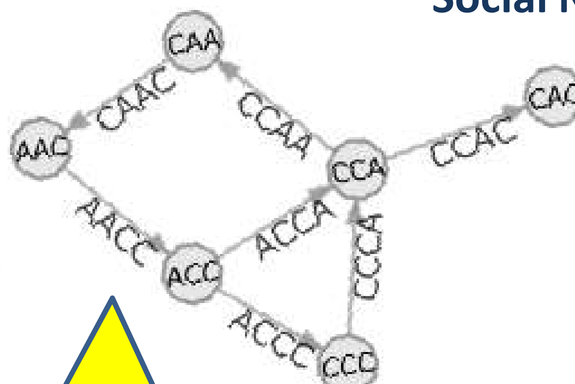
Facebook: > 800 million active users

Social Network



31 billion RDF triples in 2011

Information Network



De Bruijn:
4^k nodes
(k = 20, … , 40)

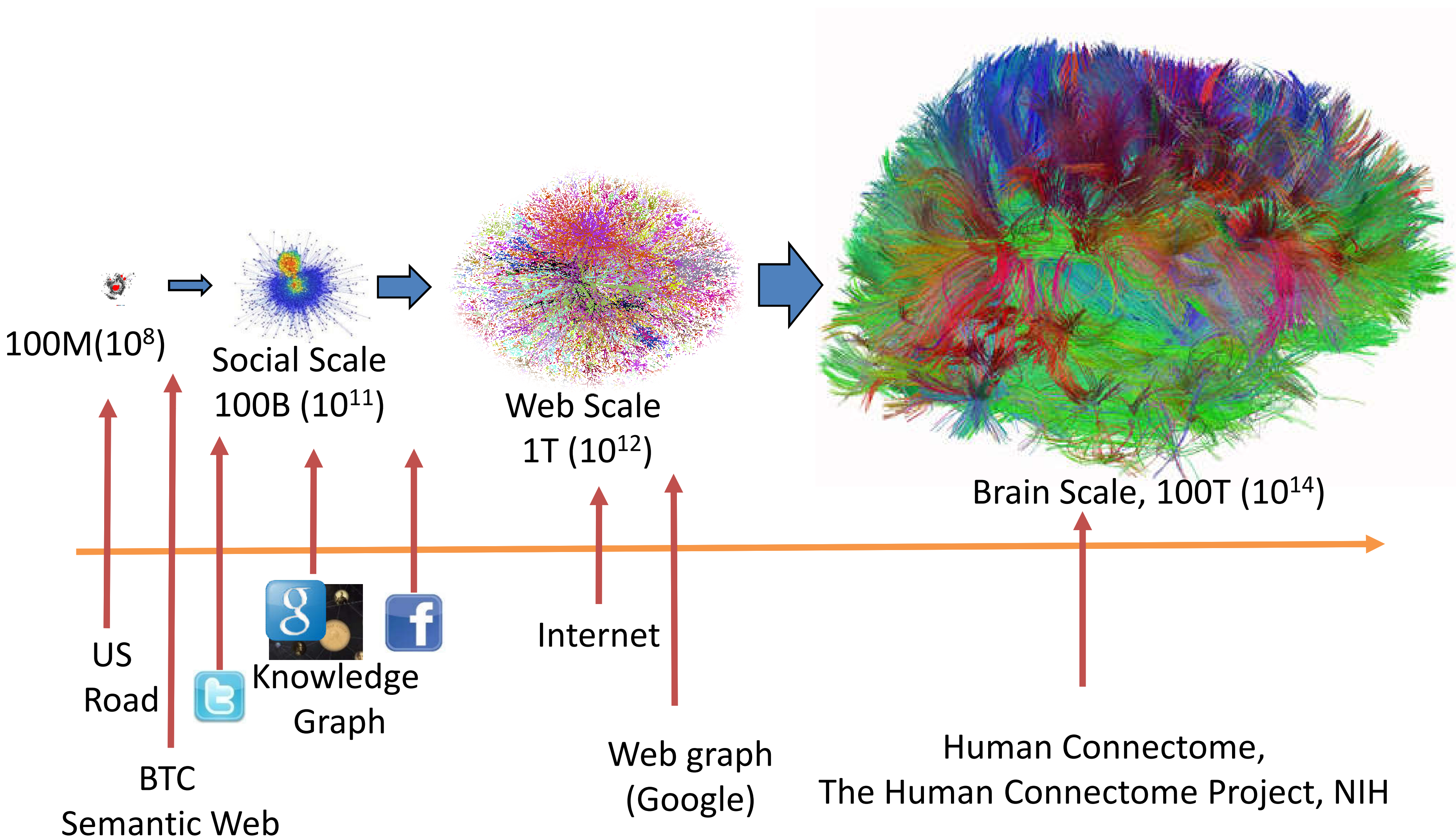
Biological Network

NETFLIX

100M Ratings,
480K Users,
17K Movies

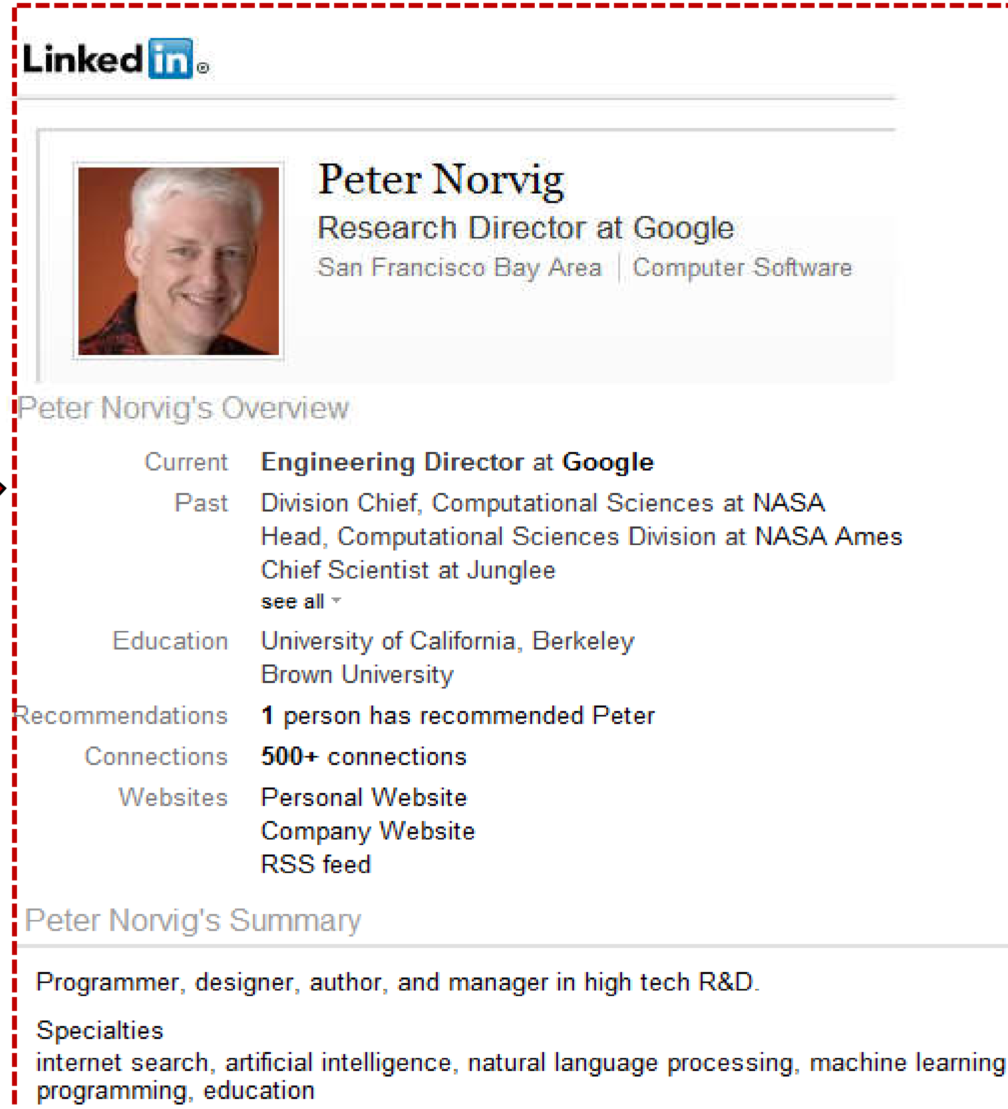
Graphs in Machine Learning

Big-Graph Scales



Complex Graphs: Topology + Attributes

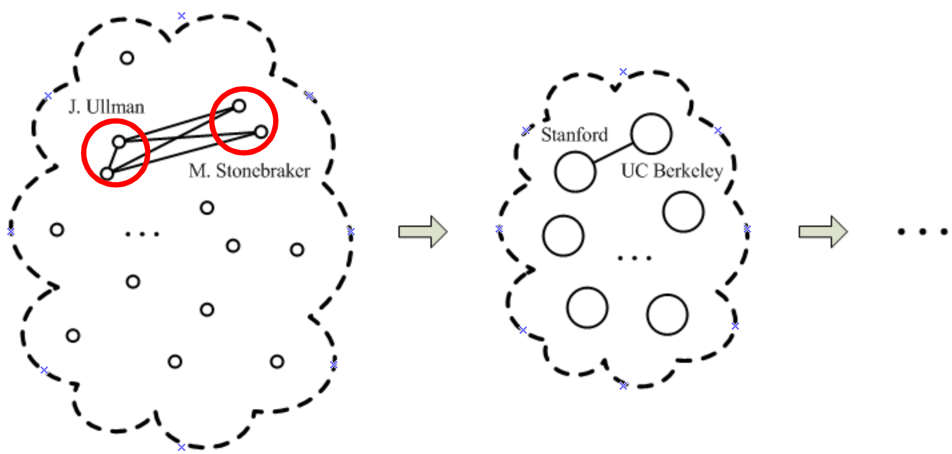
LinkedIn



Why Graph Summarization

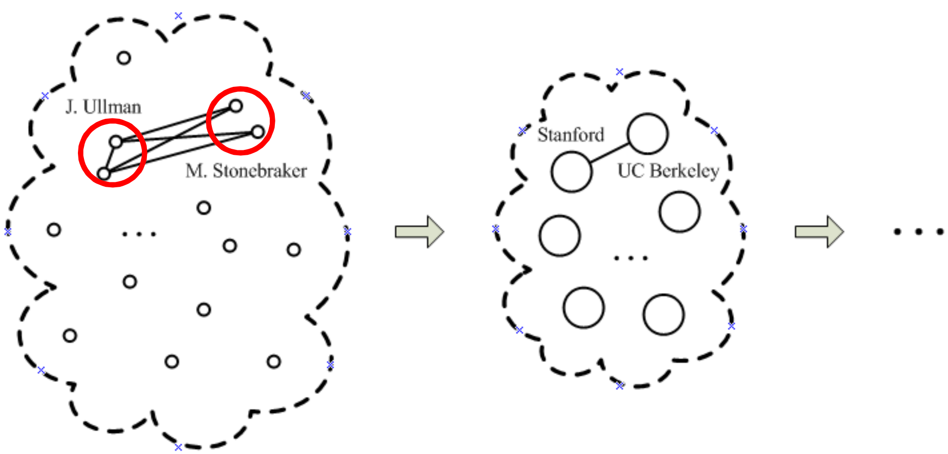
- Large-scale graph data
  - Summarization is critical

- Complex graph data
  - Interactive and exploratory analysis
    - e.g., visualization, pattern mining, anomaly detection



Why Graph Summarization

- Large-scale graph data
  - Summarization is critical
    - Fast/ online query processing
    - Fewer I/O operations
    - Less data transfer over network
- Complex graph data
  - Interactive and exploratory analysis
    - e.g., visualization, pattern mining, anomaly detection



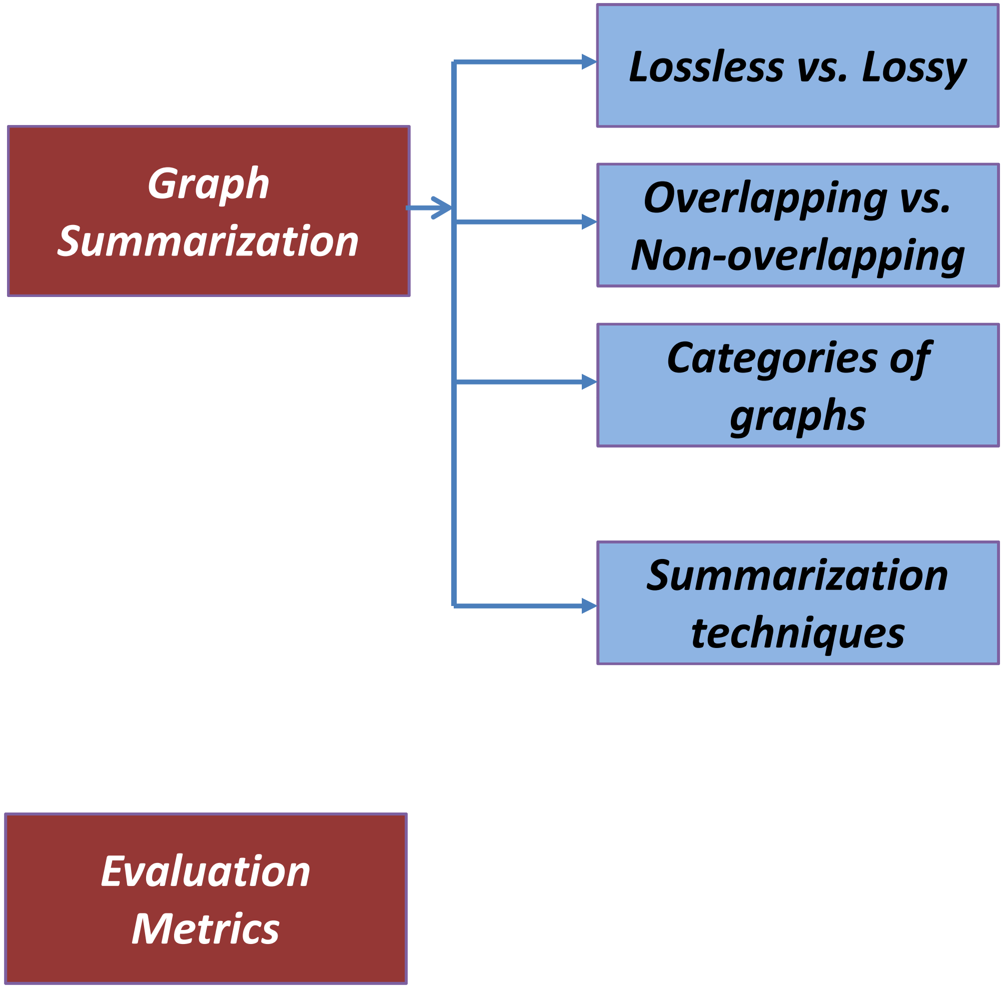
Why Graph Summarization

- Interactive and exploratory analysis
- Approximate query processing
- Visualization and visual query interface
- Distributed graph processing systems
- Processing in modern hardware

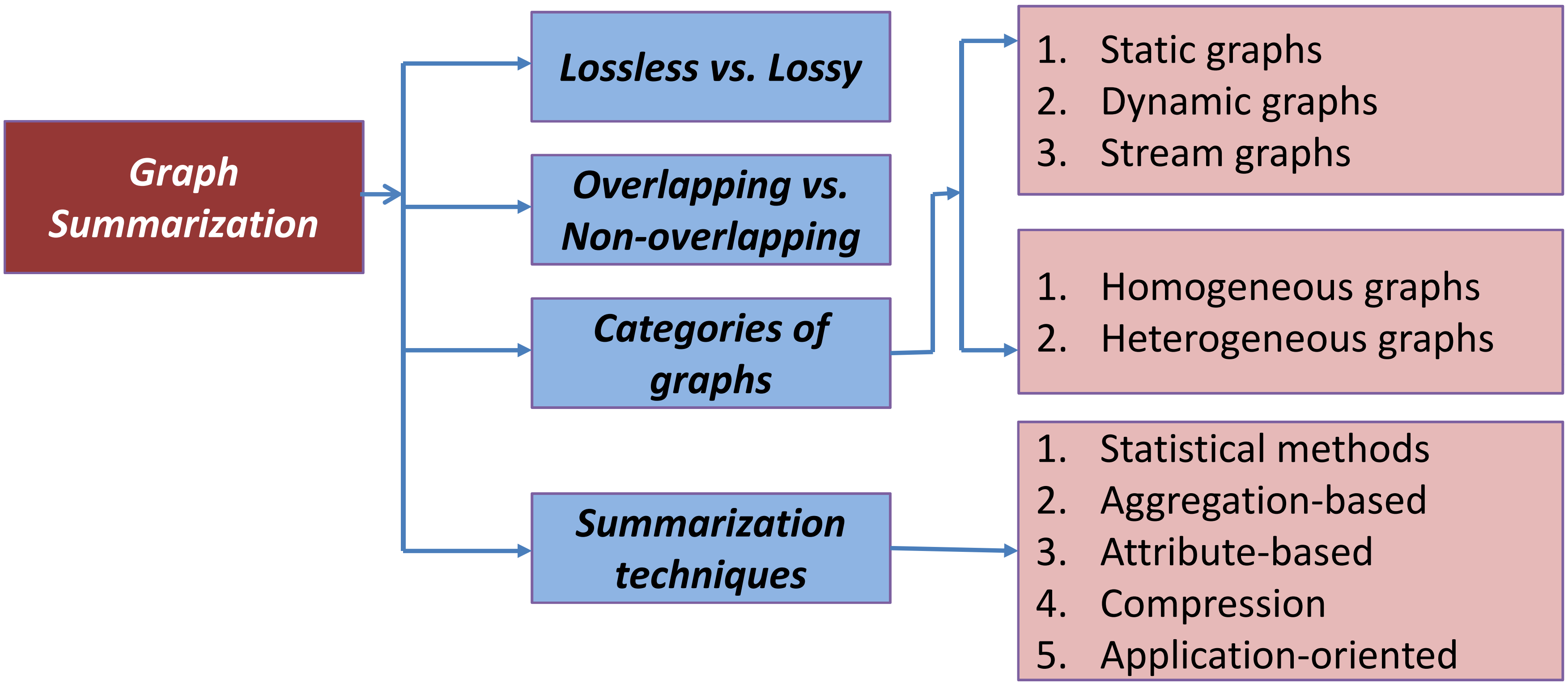
Roadmap

- Introduction
- Summarizing Static Graphs
- Summarizing Dynamic Graphs
- Summarizing Heterogeneous Graphs
- Future Work and Conclusion

Categories of Graph Summarization

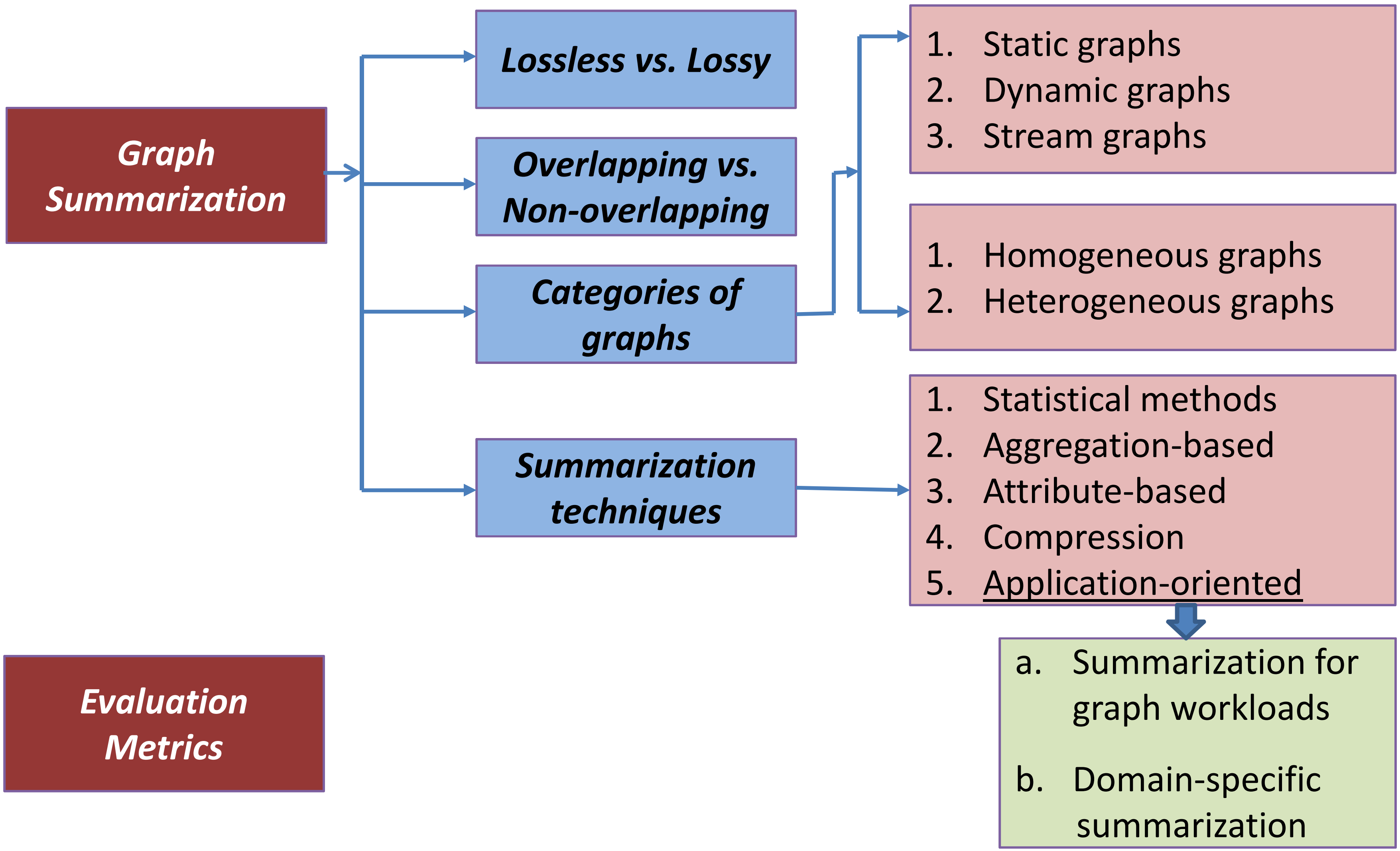


Categories of Graph Summarization

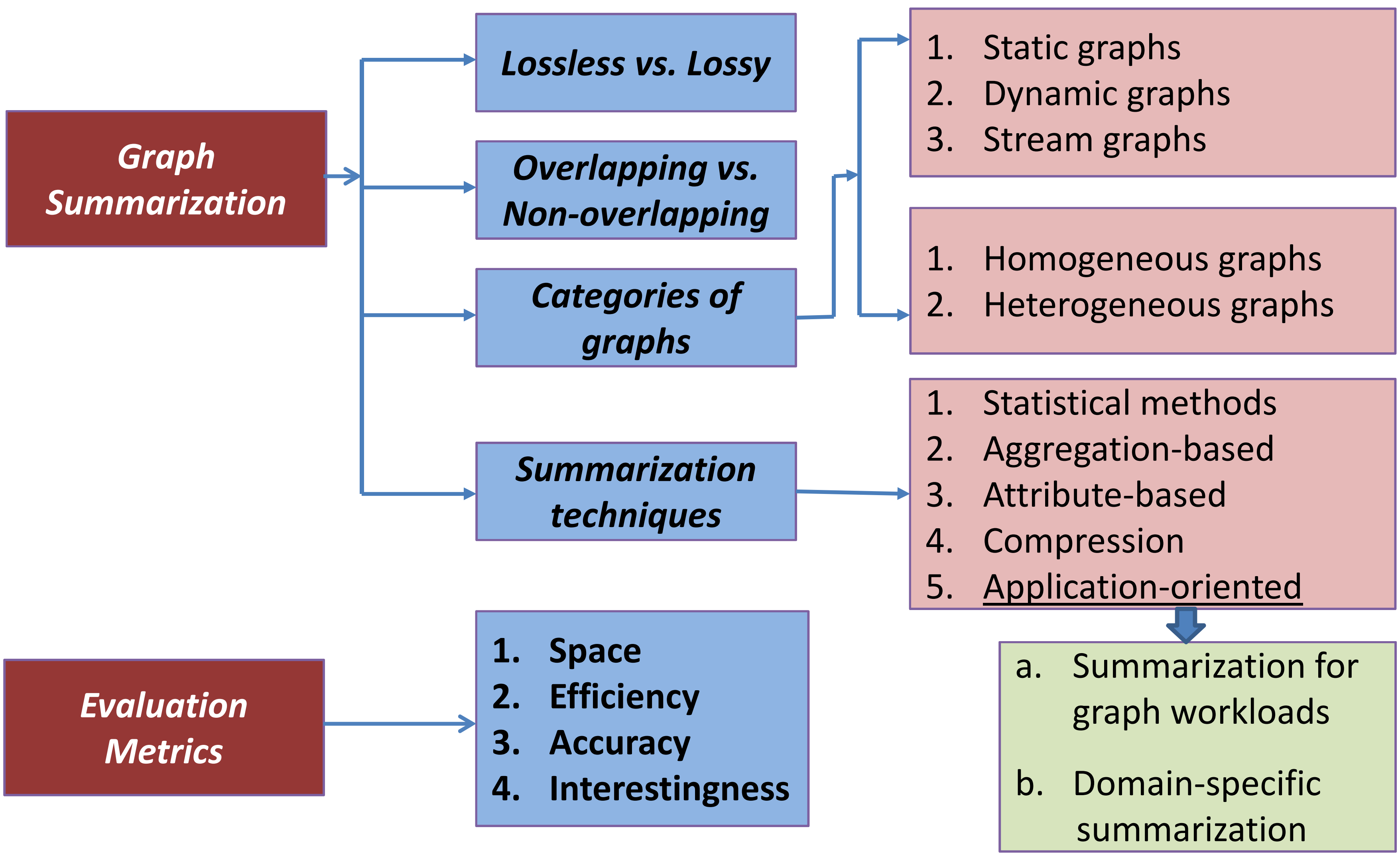


Evaluation Metrics

Categories of Graph Summarization



Categories of Graph Summarization



Graph Summary: Varieties of Graphs

- **Homogeneous graphs**

  - summarize only topology information (nodes + edges)

- **Heterogeneous graphs**

  - nodes and edges have different types and attributes
  - summarization happens at both structural and semantic levels

Graph Summary: Varieties of Graphs

- **Homogeneous graphs**
  - summarize only topology information (nodes + edges)
- **Heterogeneous graphs**
  - nodes and edges have different types and attributes
  - summarization happens at both structural and semantic levels

- **Static graphs**

Types of Temporal/ Evolving Networks

- **Dynamic graphs**
- **Stream graphs**

Graph Summary: Varieties of Graphs

- **Homogeneous graphs**
  - summarize only topology information (nodes + edges)

- **Heterogeneous graphs**
  - nodes and edges have different types and attributes
  - summarization happens at both structural and semantic levels

Types of Temporal/ Evolving Networks

- **Static graphs**
- **Dynamic graphs**
  - Snapshots of graph over time
  - Snapshots are given apriori
  - can perform many passes over snapshots to build summary
- **Stream graphs**
  - Edge-streams arriving in real-time
  - One pass over the stream to incrementally build/ update the summary

Graph Summarization Techniques

- **Statistical methods**
  - degree distribution, hop-plot, clustering coefficient
- **Aggregation-based**
  - grouping of nodes and edges into super-nodes/ super-edges
- **Attribute-based**
  - summary considering both topology and attributes (heterogeneous graphs)
- **Compression**
  - reducing storage space by smartly encoding nodes and edges
- **Application-oriented**
  - summarization for efficient graph querying (e.g., shortest path, graph pattern matching)
  - domain-specific (e.g., bioinformatics, visual querying)

Challenges in Graph Summarization

- **Varieties of graph data**
  - static vs. dynamic vs. stream
  - homogeneous vs. heterogeneous
  - numerical vs. categorical attributes
- **Different objectives**
  - OLAP vs. compression
  - Lossy vs. lossless summary
  - Accuracy vs. efficiency vs. space
- **Different applications/ workloads/ systems**
  - shortest path vs. graph pattern matching
  - main-memory vs. distributed

No unique graph summarization technique!

Other Related Tutorials/ Surveys

- [Tutorial] S.-D. Lin, M.-Y. Ych, and C.-T. Li, Sampling and Summarizing for Social Networks, in SDM 2013
- [Tutorial] D. Koutra, Summarizing Large-Scale Graph Data, in SDM 2017
- [Survey] Y. Liu, A. Dighe, T. Safavi, and D. Koutra, A Graph Summarization: A Survey, in ArXiv

Other Related Tutorials/ Surveys

- [Tutorial] S.-D. Lin, M.-Y. Ych, and C.-T. Li, Sampling and Summ
  SDM 2013
- [Tutorial] D. Koutra, Summarizing Large-Scale Graph Data, in S
- [Survey] Y. Liu, A. Dighe, T. Safavi, and D. Koutra, A Graph Sum

Our New Materials:

- Summarizing dynamic and stream graphs
- Domain-dependent graph summaries

Other Related Tutorials/ Surveys

- [Tutorial] S.-D. Lin, M.-Y. Ych, and C.-T. Li, Sampling and Summ
SDM 2013
- [Tutorial] D. Koutra, Summarizing Large-Scale Graph Data, in S

Our New Materials:

- Summarizing dynamic and stream graphs
- Domain-dependent graph summaries

Specific sub-areas under graph summarization

- . Dighe, T. Safavi, and D. Koutra, A Graph Sum
- Y. Liu, N. Shah, and D. Koutra, An Empirical Comparison of the Summarization Power of Graph Clustering Methods, in ArXiv, 2015
- A. McGregor, Graph Stream Algorithms: A Survey, in SIGMOD Rec., 2014
- C. Chen, C. X. Lin, M. Fredrikson, M. Christodorescu, X. Yan, and J. Han, Mining Large Information Networks by Graph Summarization, in Link Mining: Models, Algorithms, and Applications, 2010
- Y. Tian and J. M. Patel, Interactive Graph Summarization. In Link Mining: Models, Algorithms, and Applications, 2010

Roadmap

- Introduction
- Summarizing Static Graphs
- Summarizing Dynamic Graphs
- Summarizing Heterogeneous Graphs
- Future Work and Conclusion

Summarizing Static Graphs

SIGMOD 08

Graph Summarization with Bounded Error

Saket Navlakha*
Dept. of Computer Science
University of Maryland
College Park, MD, USA-20742
saket@cs.umd.edu

Rajeev Rastogi†
Yahoo! Labs
Bangalore, India
rrastogi@yahoo-inc.com

Nisheeth Shrivastava
Bell Labs Research
Bangalore, India
nisheeths@alcatel-lucent.com

SDM 10

GraSS: Graph Structure Summarization

Kristen LeFevre*   Evimaria Terzi†

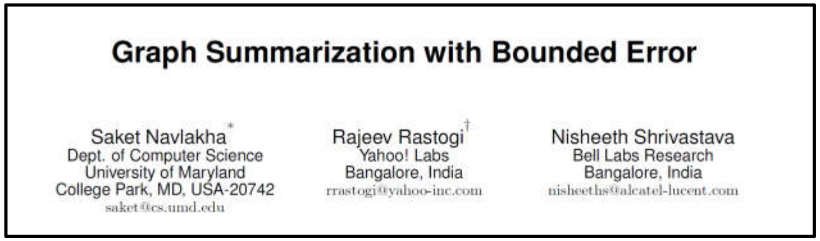
- Summary made of supernodes (set of nodes) and superedges
- Follow the MDL principle

SIGMOD 08:

- Both lossless or lossy (with bounded error)
- Edge corrections

SDM 10:

- Lossy
- Densities
- Number of supernodes predefined
- Answer queries directly on the summary (expected-value semantics)



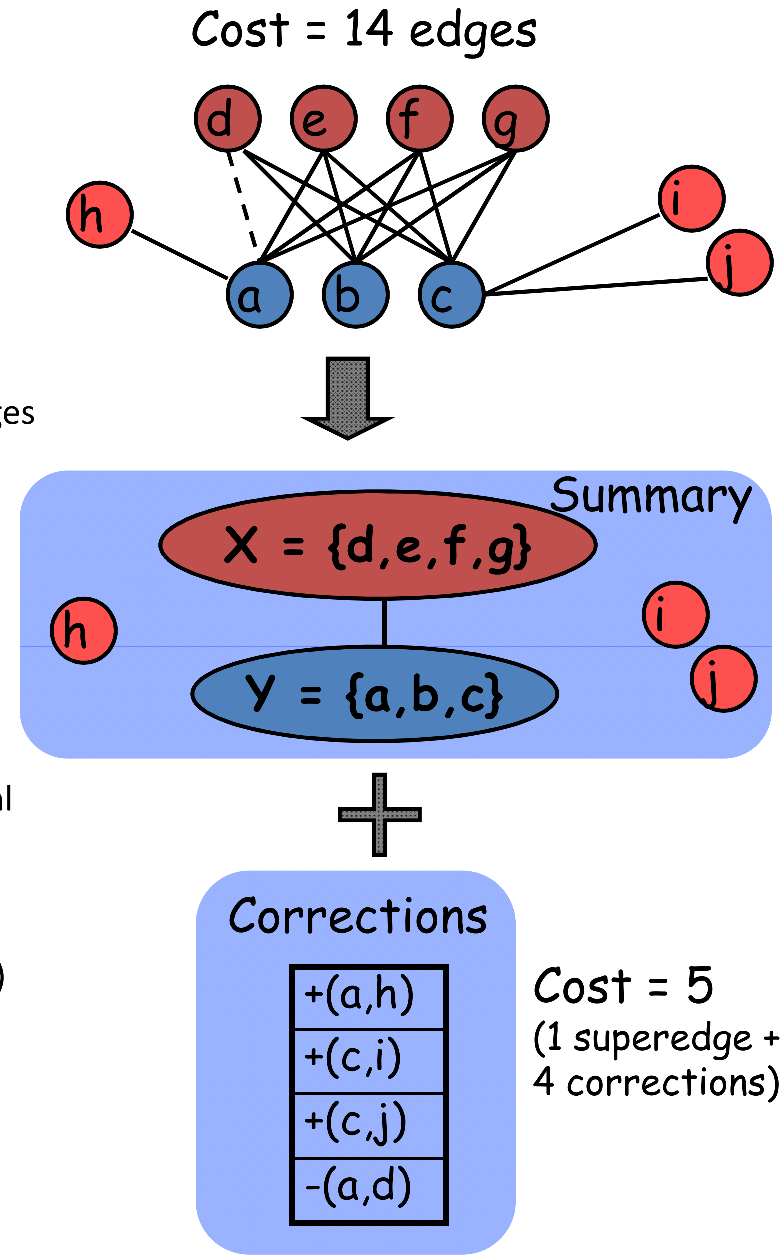
Graph Summarization with Bounded Error

Saket Navlakha[*]
Dept. of Computer Science
University of Maryland
College Park, MD, USA-20742
saket@cs.umd.edu

Rajeev Rastogi[†]
Yahoo! Labs
Bangalore, India
rrastogi@yahoo-inc.com

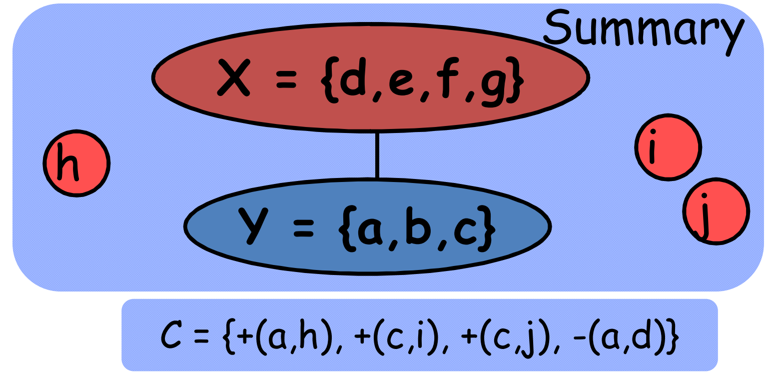
Nisheeth Shrivastava
Bell Labs Research
Bangalore, India
nisheeths@alcatel-lucent.com

- Compression possible (S)
  - Many nodes with similar neighborhoods
    - Communities in social networks; link-copying in webpages
  - Collapse such nodes into *supernodes* (clusters) and the edges into *superedges*
    - Bipartite subgraph to two supernodes and a superedge
    - Clique to supernode with a "self-edge"
- Need to correct mistakes (C)
  - Most superedges are not *complete*
    - Nodes don't have exact same neighbors: friends in social networks
  - Remember edge-corrections
    - Edges not present in superedges (-ve corrections)
    - Extra edges not counted in superedges (+ve corrections)
- Minimize overall storage cost = S+C

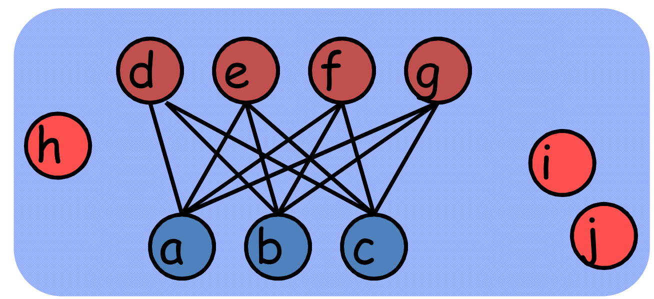


Representation Structure R=(S,C)

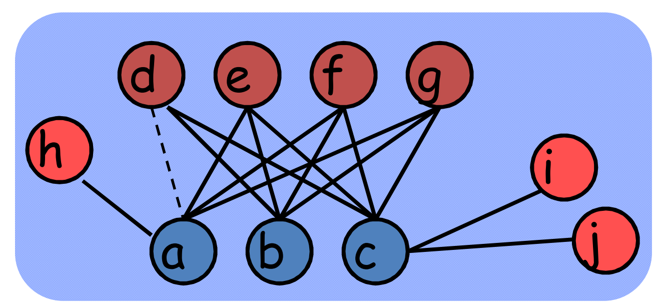
- Summary $S(V_S, E_S)$
  - Each supernode v represents a set of nodes A_v
  - Each superedge (u,v) represents all pair of edges $\pi_{uv} = A_u \times A_v$
- Corrections C: {(a,b); a and b are nodes of G}



- Supernodes are key, superedges/corrections easy
  - A_{uv} actual edges of G between A_u and A_v
  - Cost with (u,v) = $1 + |\pi_{uv} - A_{uv}|$
  - Cost without (u,v) = $|A_{uv}|$
  - Choose the minimum, decides whether edge (u,v) is in S

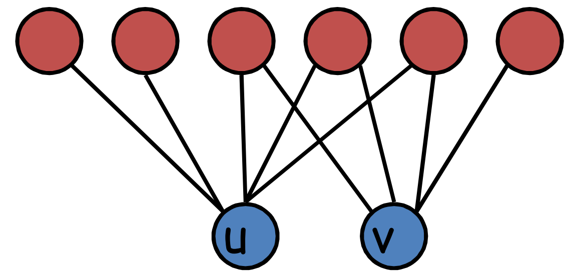


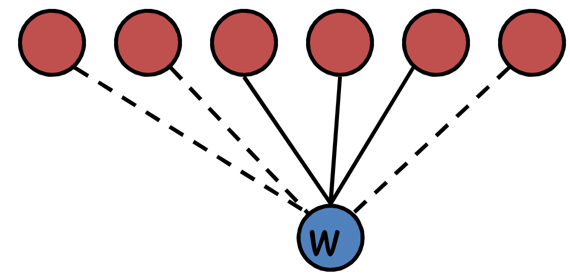
- Reconstructing the graph from R
  - For all superedges (u,v) in S, insert all pair of edges π_{uv}
  - For all +ve corrections +(a,b), insert edge (a,b)
  - For all -ve corrections -(a,b), delete edge (a,b)



Greedy

- Cost of merging supernodes u and v into single supernode w
  - Recall: cost of a superedge (u,x):
    $$c(u,x) = \min\{|\pi_{vx} - A_{vx}|+1, |A_{vx}|\}$$
  - c_u = sum of costs of all its edges = $\Sigma_x c(u,x)$
  - **$s(u,v) = (c_u + c_v - c_w)/(c_u + c_v)$**
- Main idea: recursive bottom-up merging of supernodes
  - If $s(u,v) > 0$, merging u and v reduces the cost of reduction
  - Normalize the cost: remove bias towards high degree nodes
  - Making supernodes is the key: superedges and corrections can be computed later





$c_u = 5$; $c_v = 4$

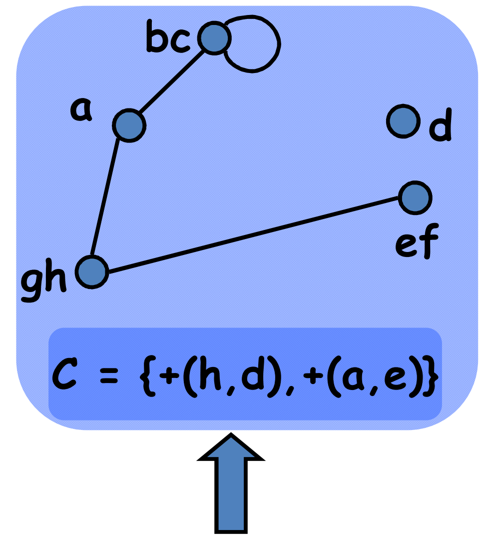
$c_w = 6$ (3 edges, 3 corrections)

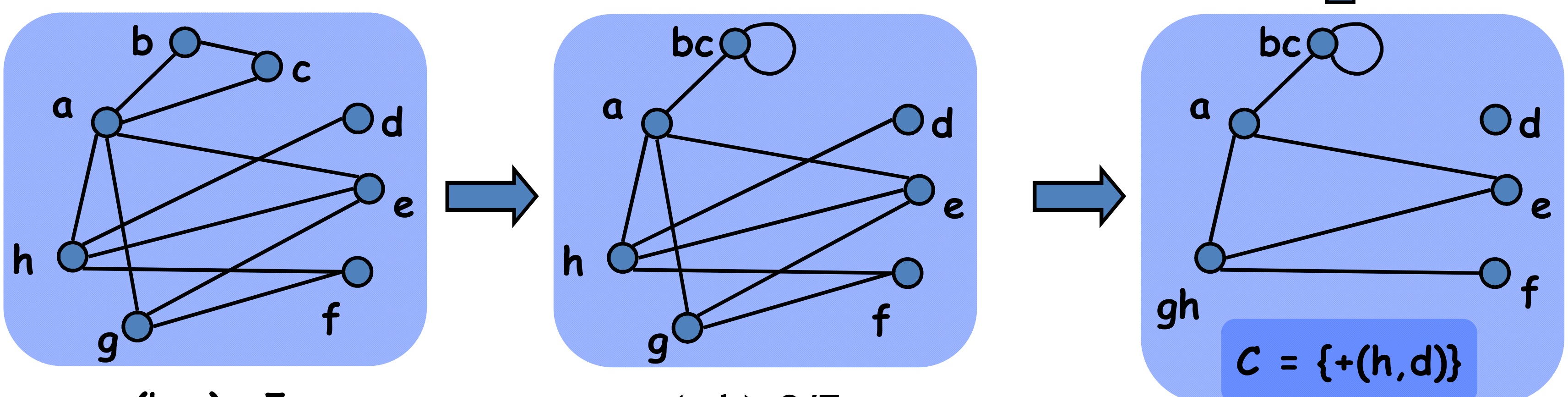
$s(u,v) = 3/9$

Greedy

- Recall: $s(u,v) = (c_u + c_v - c_w)/(c_u + c_v)$
- GREEDY algorithm
  - Start with S=G
  - At every step, pick the pair with max s(.) value, merge them
  - If no pair has positive s(.) value, stop

Cost reduction: 11 to 6





$s(b,c)=.5$
$[\ c_b = 2;\ c_c=2;\ c_{bc}=2\ ]$

$s(g,h)=3/7$
$[\ c_g = 3;\ c_h=4;\ c_{gh}=4\ ]$

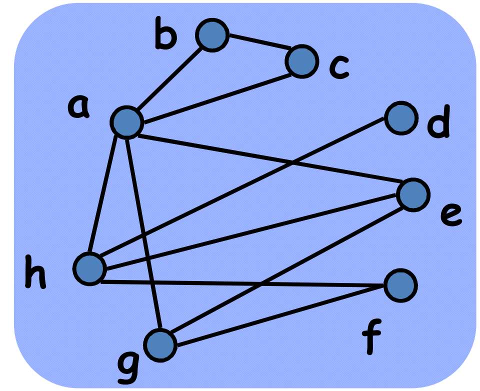
$s(e,f)=1/3$
$[\ c_e = 2;\ c_f=1;\ c_{ef}=2\ ]$

Randomized

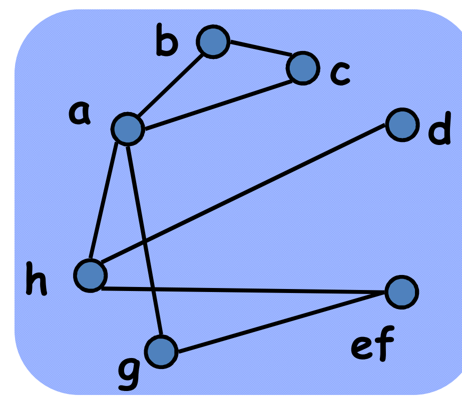
- GREEDY is slow
  - Need to find the pair with (globally) max s(.) value
  - Need to process all pair of nodes at a distance of 2-hops
  - Every merge changes costs of all pairs containing N_w
- Main idea: light weight randomized procedure
  - Instead of choosing the globally best pair,
  - Choose (randomly) a node u
  - Merge the best pair containing u

Randomized

- Randomized algorithm
  - Unfinished set $U=V_G$
  - At every step, randomly pick a node u from U
  - Find the node v with max s(u,v) value
  - If s(u,v) > 0, then merge u and v into w, put w in U
  - Else remove u from U
  - Repeat till U is empty

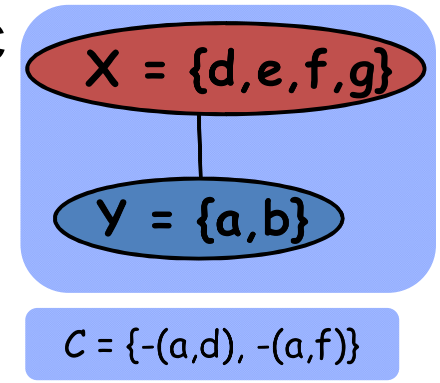


Picked e; s(e,f)=3/5
[c_e = 3; c_f=2; c_{ef}=3]

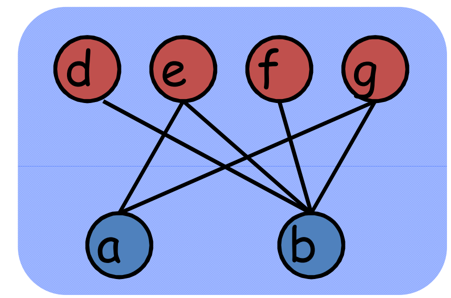


C = {+(a,e)}

Approximate Representation R_ϵ



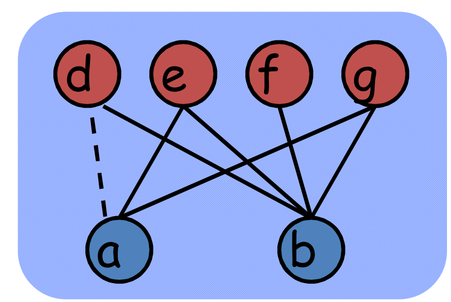
- Approximate representation
  - Recreating the input graph *exactly* is not always necessary
  - Reasonable approximation enough: to compute communities, anomalous traffic patterns, etc.
  - Use approximation leeway to get further cost reduction



- Generic Neighbor Query
  - Given node v, find its neighbors N_v in G
  - Apx-nbr set N'_v estimates N_v with ϵ-accuracy
  - Bounded error: error(v) = $|N'_v \setminus N_v| + |N_v \setminus N'_v| < \epsilon\, |N_v|$
  - Number of neighbors added or deleted is at most ϵ-fraction of the true neighbors

- Intuition for computing R_ϵ
  - If correction (a,d) is deleted, it adds error for both a and d
  - From exact representation R for G, remove (maximum) corrections s.t. ϵ-error guarantees still hold

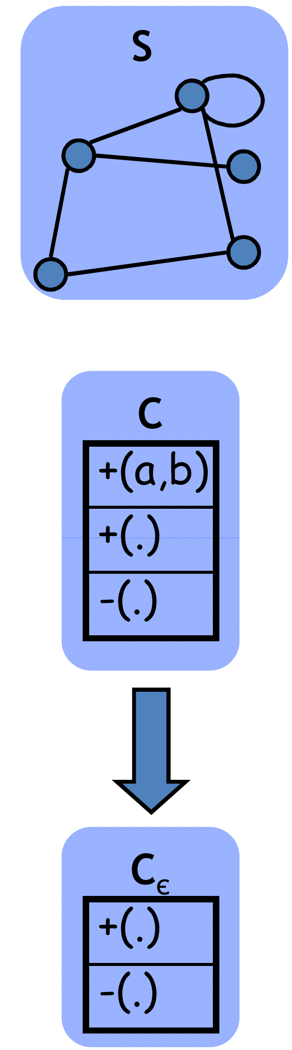
For ϵ=.5, we can remove one correction of a



Computing approx representation

- Reducing size of corrections
  - *Correction graph* H: For every (+ve or –ve) correction (a,b) in C, add edge (a,b) to H
  - Removing (a,b) reduces size of C, but adds error of 1 to a and b
  - Recall bounded error: error(v) = $|N'_v \backslash N_v| + |N_v \backslash N'_v| < \epsilon\ |N_v|$
  - Implies in H, we can **remove** up to $b_v = \epsilon\ |N_v|$ edges incident on v
  - Maximum cost reduction: remove subset M of E_H of max size s. t. M has at most b_v edges incident on v

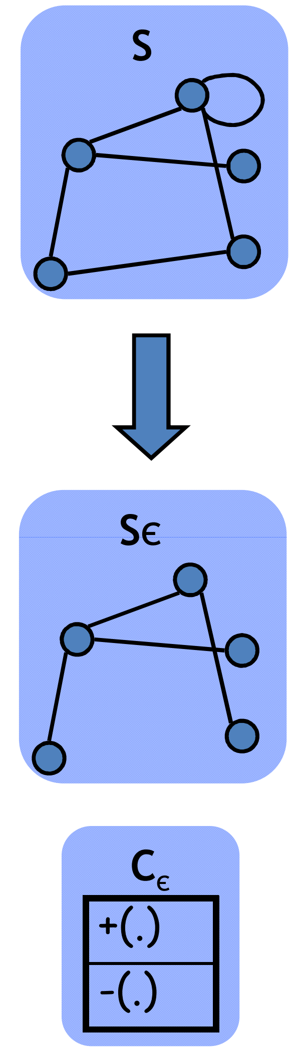
- Same as the b-matching problem
  - Find the matching $M \subset E_G$ s.t. at most b_v edges incident on v are in M
  - For all $b_v = 1$, traditional matching problem
  - Solvable in time $O(mn^2)$ [Gabow-STOC-83] (for graph with n nodes and m edges)



Computing approx representation

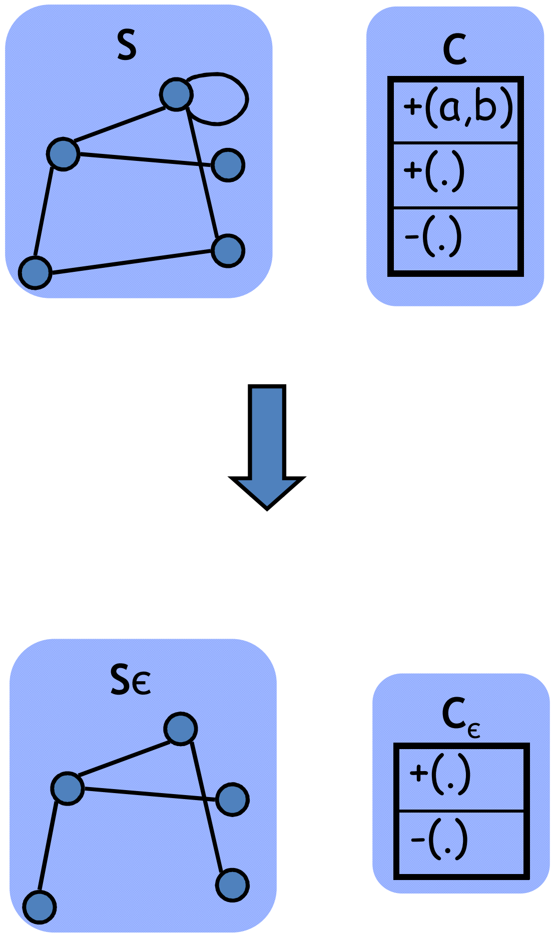
- Reducing size of summary
  - Removing superedge (a,b) implies bulk removal of all pair edges π_{uv}
  - But, each node in A_u and A_v has different b value
  - Does not map to a clean matching-type problem

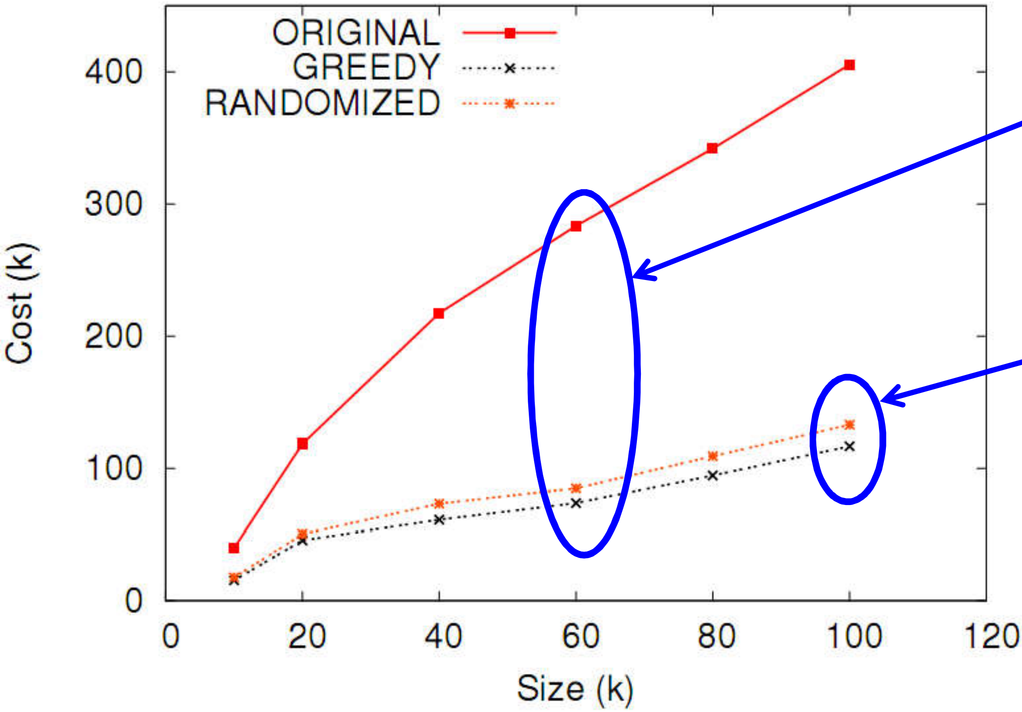
- A greedy approach
  - Pick superedges by increasing $|\pi_{uv}|$ value
  - Delete (u,v) if that doesn't violate ϵ-bound for nodes in $A_u U A_v$
  - If there is correction (a,b) for π_{uv} in C, we cannot remove (u,v); since removing (u,v) violates error bound for a or b



APXMDL

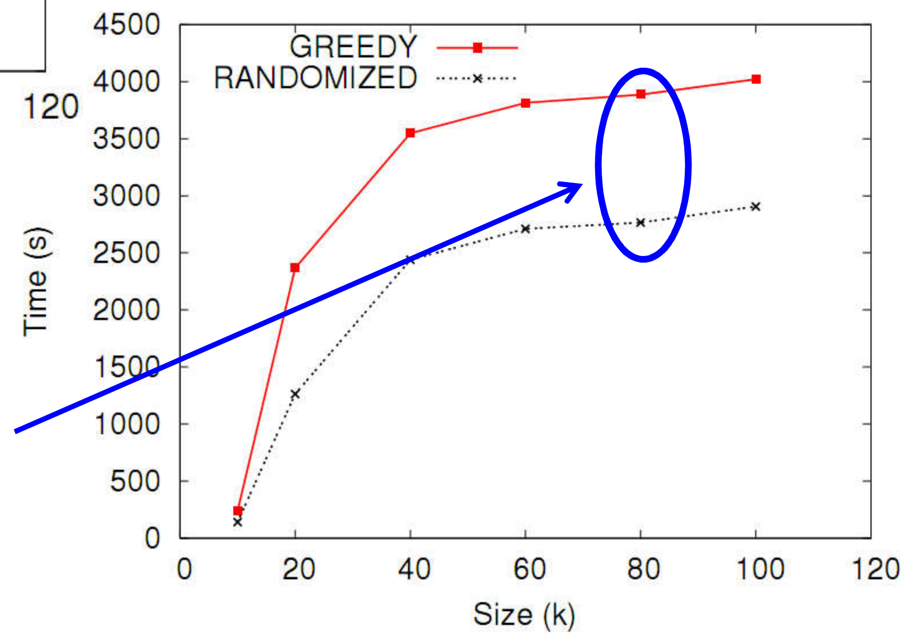
- Compute the R(S,C) for G
- Find C_ϵ
  - Compute H, with E_H=C
  - Find maximum b-matching M for H; C_ϵ=C-M
- Find S_ϵ
  - Pick superedges (u,v) in S having no correction in C_ϵ in increasing $|\pi_{uv}|$ value
  - Remove (u,v) if that doesn't violate ϵ-bound for any node in A_u U A_v
- Axp-representation $R_\epsilon=(C_\epsilon, S_\epsilon)$





Reduces the cost down to 40%

Cost of GREEDY 20% lower than RANDOMIZED

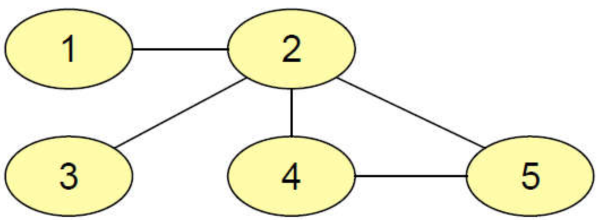


RANDOMIZED is 60% faster than GREEDY

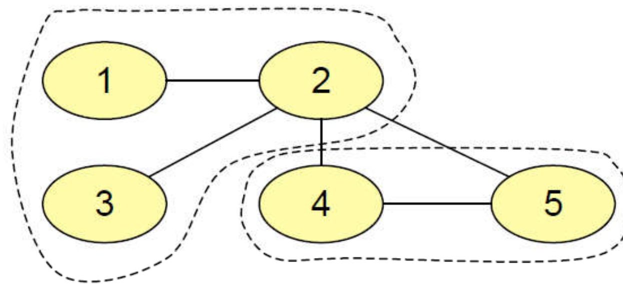
GraSS: Graph Structure Summarization

Kristen LeFevre* Evimaria Terzi†

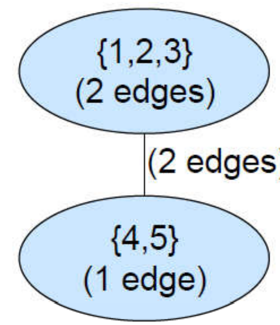
Original graph



Node partition



Summary



	1	2	3	4	5
1	0	1	0	0	0
2	1	0	1	1	1
3	0	1	0	0	0
4	0	1	0	0	1
5	0	1	0	1	0

Adjacency matrix of the original graph

	1	2	3	4	5
1	0	2/3	2/3	1/3	1/3
2	2/3	0	2/3	1/3	1/3
3	2/3	2/3	0	1/3	1/3
4	1/3	1/3	1/3	0	1
5	1/3	1/3	1/3	1	0

Expected adjacency matrix resulting from the summary input graph $G(V,E)$ a summary $\mathbf{S}(G)$ consists of

1. A *partition* of the nodes of V into parts $\mathbf{V}(V) = \{V_1, \ldots, V_k\}$, such that $V_i \subseteq V$ and $V_i \cap V_j = \emptyset$, for $i, j \in \{1, \ldots, k\}$ and $i \neq j$. We refer to each group of nodes V_i as a *supernode* of the summary $\mathbf{S}$.
2. For every supernode $V_i \in \mathbf{V}$, summary $\mathbf{S}$ describes the *number of edges* within the nodes in the supernode. That is, it counts the number of edges in the input graph G that have both their endpoints in the nodes of V_i. For supernode V_i we denote this number by E_i. That is,
$$E_i = |\{e(u,v) \mid u, v \in V_i, e(u,v) \in E\}|.$$
3. For every pair of supernodes $V_i, V_j \in \mathbf{V}$, summary $\mathbf{S}$ also gives the *number of edges* across the two supernodes. That is, it counts the number of edges in the input graph G that have one of their endpoints in a node of V_i and their other endpoint in a node in V_j. For two supernodes V_i and V_j, we denote this number by E_{ij}.
$$E_{ij} = |\{e(u,v) \mid u \in V_i, v \in V_j, e(u,v) \in E\}|.$$

Definition 1. (Graph Reconstruction) *Let* $\mathbf{S}$ *be a summary graph consisting of* k *supernodes* $\mathbf{V} = \{V_1, \ldots, V_k\}$ *and edge counts* E_i, E_{ij} *for* $i, j \in \{1, \ldots, k\}$. *The set of valid reconstructions of* $\mathbf{S}$, *denoted by* $\mathcal{R}(\mathbf{S})$, *is the set of graphs* $G(V, E)$, *such that*

- *For every* $i \in \{1, \ldots, k\}$,

$$|\{e(u, v) \mid u, v \in V_i, e(u, v) \in E\}| = N_i.$$

- *For every* $i, j \in \{1, \ldots, k\}$ *and* $i \neq j$,

$$|\{e(u, v) \mid u \in V_i, v \in V_j, e(u, v) \in E\}| = N_{ij}.$$

Definition 2. (Expected Value Semantics) *Let $\mathcal{R}(\mathbf{S})$ denote the set of all valid reconstructions from summary $\mathbf{S}$, and let $Q()$ denote a query on G with a boolean or real-valued response.*[2] *Under expected value semantics, the answer to $Q()$ is defined to be the real number e such that*

$$e = \frac{\sum_{G \in \mathcal{R}(\mathbf{S})} Q(G)}{|\mathcal{R}(\mathbf{S})|}$$

Note that the above equation assumes uniform probability distribution over all graphs. Incorporating prior knowledge about the graph structure (e.g., giving preference to scale-free graphs) can be easily done by multiplying $Q(G)$ with the prior probability $\mathcal{P}(G)$ of the graph G.

Definition 3. (Expected Adjacency Matrix) *Let* $\mathbf{S}$ *be a summary graph. The expected adjacency matrix* $\overline{A}$ *for* $\mathbf{S}$ *is a* $|V| \times |V|$ *matrix, where all entries are real numbers in the range* $[0, 1]$ *defined as follows:*

$$\overline{A}(u, v) = \frac{|\{G(V, E) \mid G \in \mathcal{R}(\mathbf{S}), (u, v) \in E\}|}{|\mathcal{R}(\mathbf{S})|}$$

Given a graph summary, each of the entries in the expected adjacency matrix is easily computed in closed form.

Theorem 3.1. *Given summary* $\mathbf{S}$*, the entries of the expected adjacency matrix* $\overline{A}$ *given* $\mathbf{S}$ *can be computed as follows:*

1. *If* $u, v \in V$ *are distinct nodes in the same supernode* V_i*, then*

$$\overline{A}(u, v) = \frac{2E_i}{|V_i|(|V_i| - 1)} \tag{3.1}$$

2. *If* $u, v \in V$ *are distinct nodes in different supernodes,* V_i *and* V_j*, then*

$$\overline{A}(u, v) = \frac{E_{ij}}{|V_i| \times |V_j|} \tag{3.2}$$

3. *Otherwise (if* $u = v$*),*

$$\overline{A}(u, v) = 0 \tag{3.3}$$

	1	2	3	4	5
1	0	2/3	2/3	1/3	1/3
2	2/3	0	2/3	1/3	1/3
3	2/3	2/3	0	1/3	1/3
4	1/3	1/3	1/3	0	1
5	1/3	1/3	1/3	1	0

Example:

Expected degree of node #2:
2/3 + 2/3 + 1/3 + 1/3 = 2

Other measures:
- Expected eigenvector centrality
- Expected number of triangles*

* [Riondato et al., ICDM'14, DMKD]

Query answering

- Queries to the original graph can be approximated directly on the summary.
- The expected adjacency matrix can be seen as a probabilistic (uncertain) graph.
- Expected value sematics

Minimize the reconstruction error

- A summary is good when the expected adjacency matrix is close to the original adjacency matrix
- Define *reconstruction error* as the difference between the two matrices.
- **Problem*:** given an integer *k* find a k-partiton of the nodes s.t. the corresponding summary minimizes reconstruction error.

	1	2	3	4	5
1	0	1	0	0	0
2	1	0	1	1	1
3	0	1	0	0	0
4	0	1	0	0	1
5	0	1	0	1	0

	1	2	3	4	5
1	0	2/3	2/3	1/3	1/3
2	2/3	0	2/3	1/3	1/3
3	2/3	2/3	0	1/3	1/3
4	1/3	1/3	1/3	0	1
5	1/3	1/3	1/3	1	0

$$\mathrm{Re}\left(A \mid \overline{A}\right) = \frac{1}{|V|^2} \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} \left|\overline{A}(i,j) - A(i,j)\right|$$

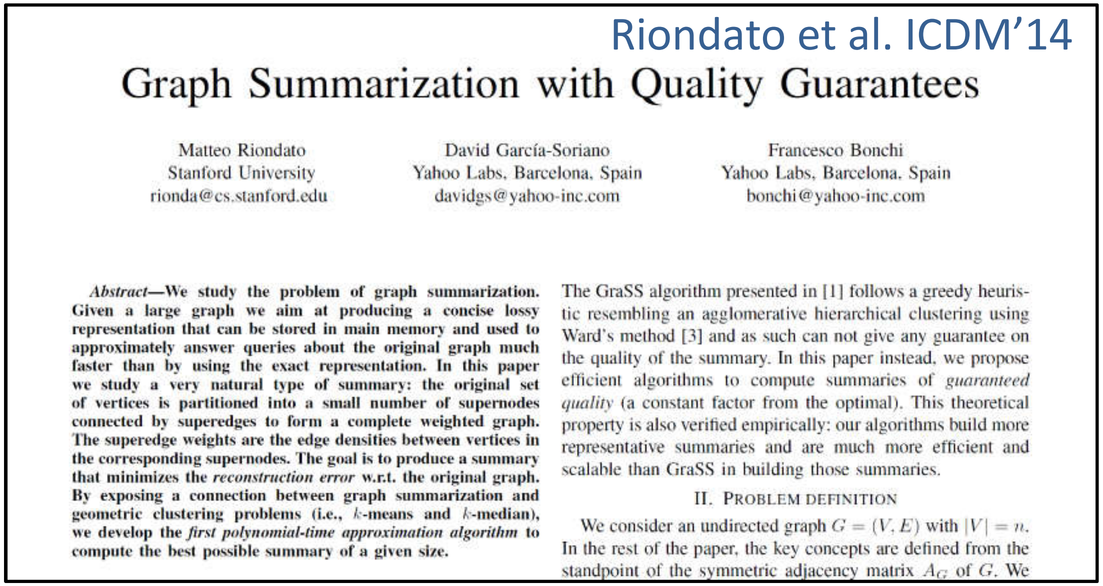
* LeFevre & Terzi also define a MDL-based variant with no *k* parameter.

Greedy algorithm

- *Greedy agglomerative hierarchical clustering*:

1) Put each vertex in a separate supernode;
2) Until the number of supernodes is k:
   - *Merge the two supernodes* whose merging minimizes the reconstruction error;
3) Output the resulting k supernodes;

- Main limitations:
  - no quality guarantees
  - very slow



Riondato et al. ICDM’14

Graph Summarization with Quality Guarantees

Matteo Riondato
Stanford University
rionda@cs.stanford.edu

David García-Soriano
Yahoo Labs, Barcelona, Spain
davidgs@yahoo-inc.com

Francesco Bonchi
Yahoo Labs, Barcelona, Spain
bonchi@yahoo-inc.com

***Abstract*—We study the problem of graph summarization. Given a large graph we aim at producing a concise lossy representation that can be stored in main memory and used to approximately answer queries about the original graph much faster than by using the exact representation. In this paper we study a very natural type of summary: the original set of vertices is partitioned into a small number of supernodes connected by superedges to form a complete weighted graph. The superedge weights are the edge densities between vertices in the corresponding supernodes. The goal is to produce a summary that minimizes the *reconstruction error* w.r.t. the original graph. By exposing a connection between graph summarization and geometric clustering problems (i.e., k-means and k-median), we develop the *first polynomial-time approximation algorithm* to compute the best possible summary of a given size.**

The GraSS algorithm presented in [1] follows a greedy heuristic resembling an agglomerative hierarchical clustering using Ward’s method [3] and as such can not give any guarantee on the quality of the summary. In this paper instead, we propose efficient algorithms to compute summaries of *guaranteed quality* (a constant factor from the optimal). This theoretical property is also verified empirically: our algorithms build more representative summaries and are much more efficient and scalable than GraSS in building those summaries.

II. Problem Definition

We consider an undirected graph $G = (V, E)$ with $|V| = n$. In the rest of the paper, the key concepts are defined from the standpoint of the symmetric adjacency matrix A_G of G. We

Data Mining and Knowledge Discovery manuscript No.
(will be inserted by the editor)

Graph Summarization with Quality Guarantees

Matteo Riondato · David García-Soriano · Francesco Bonchi

DMKD

- Overcome GraSS limitations: fast algorithm with constant-factor approximation guarantee
- Generalize reconstruction error to ℓ_p*-reconstruction error*
- Consider cut-norm error
- Among the contributions: a practical use of extreme graph theory, with the cut-norm and the algorithmic version of Szemerédi’s Regularity Lemma.

We define the *density matrix* of $\mathcal{S}$ as the $k \times k$ matrix $A_{\mathcal{S}}$ with entries $A_{\mathcal{S}}(i,j) = d_G(i,j)$, $1 \leq i,j \leq k$. For each $v \in V$, we also denote by $s(v)$ the unique element w of $\mathcal{S}$ (i.e., a supernode) such that $v \in w$. The density matrix $A_{\mathcal{S}} \in \mathbb{R}^{k \times k}$ can be *lifted* to the matrix $A_{\mathcal{S}}^{\uparrow} \in \mathbb{R}^{n \times n}$ defined as

$$A_{\mathcal{S}}^{\uparrow}(v,w) = A_{\mathcal{S}}(s(v), s(w)) \quad .$$

We justify the use of the lifted matrix in Sect. 3.1. Our lifted matrix is slightly different from the *expected adjacency matrix* defined by LeFevre and Terzi (2010).

Partition-constant matrices. Given a k-partition $\mathcal{P} = \{S_1, \ldots, S_k\}$ of $[n]$, we say that a symmetric $n \times n$ matrix M with real entries is *$\mathcal{P}$-constant* if the $S_i \times S_j$ submatrix of M is *constant*, $1 \leq i,j \leq k$. More formally, M is $\mathcal{P}$-constant if for all pairs (i,j), $1 \leq i,j \leq k$, there is a constant $c_{ij} = c_{ji}$ such that $M(p,q) = c_{ij}$ for each pair (p,q) where $p \in S_i$ and $q \in S_j$. We also say that M is *k-constant*, to highlight the size of the partition. It should be clear from the definition that the lifted adjacency matrix of a k-summary $\mathcal{S}$ of a graph G is $\mathcal{P}_{\mathcal{S}}$-constant for the partition $\mathcal{P}_{\mathcal{S}}$ of the nodes of G into the supernodes of $\mathcal{S}$.

Problem definition. The number of possible summaries is huge (there is one for each possible partitioning of V), so we need efficient algorithms to find the summary that best resembles the graph. This goal is formalized in Problem 1, which is the focus of this work.

Problem 1 (Graph Summarization) Given a graph $G = (V, E)$ with $|V| = n$, and $k \in \mathbb{N}$, find the k-summary $\mathcal{S}^*$, such that $A^{\uparrow}_{\mathcal{S}^*}$ minimizes the error $\mathsf{err}(A_G, A^{\uparrow}_{\mathcal{S}^*})$ for some error function $\mathsf{err} : \mathbb{R}^{n \times n} \times \mathbb{R}^{n \times n} \to [0, \infty)$.

The function err expresses the dissimilarity between the original adjacency matrix of G and the lifted matrix obtained from the summary. Different definitions for err are possible and different algorithms may be needed to find the optimal summary $\mathcal{S}^*$ according to different measures. In the following section we present some of these measures and discuss their properties.

2.1 The ℓ_p-reconstruction error

Let $p \in \mathbb{R}$, $p \geq 1$. Given a graph G with adjacency matrix A_G and a summary $\mathcal{S}$ with lifted adjacency matrix $A^{\uparrow}_{\mathcal{S}}$, the *$\ell_p$-reconstruction error* of $\mathcal{S}$ is defined as the entry-wise p-norm of the difference between A_G and $A^{\uparrow}_{\mathcal{S}}$:

$$\mathsf{err}_p(A_G, A^{\uparrow}_{\mathcal{S}}) = \|A_G - A^{\uparrow}_{\mathcal{S}}\|_p = \left(\sum_{i=1}^{|V|}\sum_{j=1}^{|V|} |A_G(i,j) - A^{\uparrow}_{\mathcal{S}}(i,j)|^p\right)^{1/p} .$$

2.2 The cut-norm error

The *cut norm* of an $n \times m$ matrix A is the maximum absolute sum of the entries of any of its submatrices (Frieze and Kannan, 1999):

$$\|A\|_{\square} = \max_{S,T\subseteq[n]} |A(S,T)| = \max_{S,T\subseteq[n]} \left| \sum_{i\in S, j\in T} A_{i,j} \right| .$$

The *cut distance* between two $n \times n$ matrices A and B is then defined by $\mathsf{err}_{\square}(A,B) = \|A - B\|_{\square}$. The *cut-norm error* of a summary $\mathcal{S}$ with respect to the graph G is therefore

$$\mathsf{err}_{\square}(A_G, A^{\uparrow}_{\mathcal{S}}) = \max_{S,T\subseteq V} |e_G(S,T) - e_{\mathcal{S}^{\uparrow}}(S,T)|,$$

Algorithm: just cluster the rows of the adjacency matrix!

- For ℓ_p-reconstruction error, perform *ℓ_p-clustering* of the *rows of A_G* ($p = 1$: k-median, $p = 2$: k-means). If column i is in cluster j, then vertex i is in supernode V_j.

LEMMA: The summary obtained from the optimal ℓ_1 (resp. ℓ_2) clustering is a *8 (resp. 4) approximation* of the optimal summary for the ℓ_1 (resp. ℓ_2) reconstruction error.

Both k-means and k-median are *NP-hard*;

There are *constant factor approximation algorithms*;
  BOTTLENECK: computing all pairwise distance for the n rows of A_G is expensive (like matrix multiplication);

SOLUTION: Use a *sketch* of the adjacency matrix with n rows and $\log n$ columns;
  Incurs in additional constant error;

Even with the sketch, the approximation algorithms take time $\tilde{O}(n^2)$;
  IDEA: *select $O(k)$ rows of the sketch adaptively*, compute a clustering using them;

In the end, the algorithm runs in time $\tilde{O}(m + nk)$ and obtains a constant-factor approximation.

Algorithm 1: Graph summarization with ℓ_p-reconstruction error

Input : $G = (V, E)$ with $|V| = n$, $k \in \mathbb{N}$, $p \in \{1, 2\}$

Output: A $O(1)$-approximation to the best k-summary for G under the ℓ_p-reconstruction error

```
// Create the n × O(log n) sketch matrix (Indyk, 2006)
S ← createSketch(A_G, O(log n), p)
// Select O(k) rows from the sketch (Aggarwal et al, 2009)
R ← reduceClustInstance(A_G, S, k)
// Run the approximation algorithm by Mettu and Plaxton (2003) to
obtain a partition.
𝒫 ← getApproxClustPartition(p, k, R, S)
// Compute the densities for the summary
D ← computeDensities(𝒫, A_G)
return (𝒫, D)
```

Roadmap

- Introduction
- Summarizing Static Graphs
- Summarizing Dynamic Graphs
- Summarizing Heterogeneous Graphs
- Future Work and Conclusion

IEEE BigData 2016

Scalable Dynamic Graph Summarization

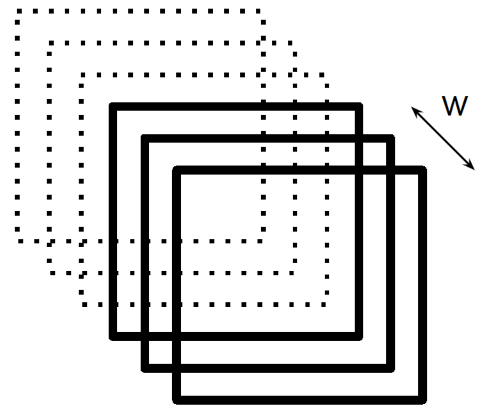
Ioanna Tsalouchidou
Web Research Group, DTIC
Pompeu Fabra University, Spain
ioanna.tsalouchidou@upf.edu

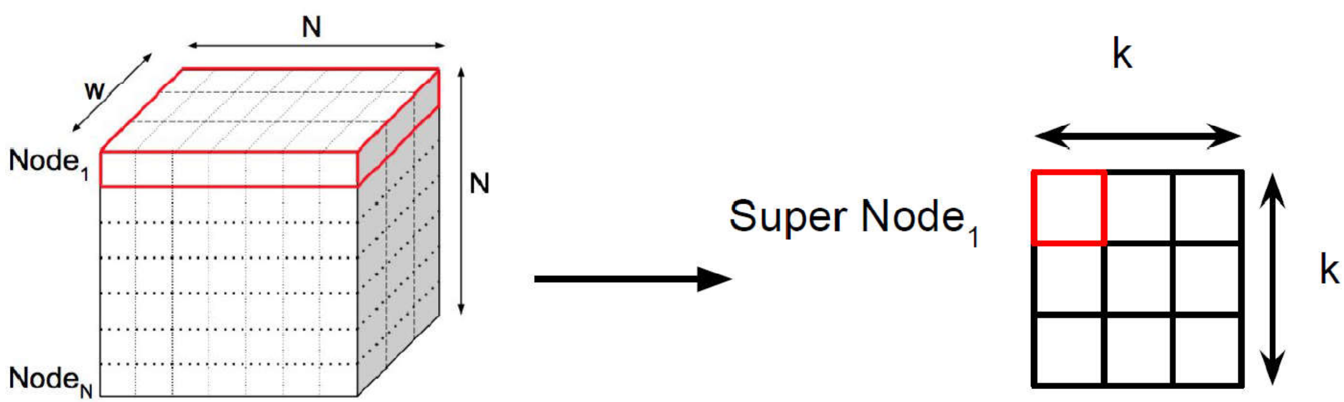
Gianmarco De Francisci Morales
Qatar Computing Research Institute
gdfm@acm.org

Francesco Bonchi
Algorithmic Data Analytics Lab
ISI Foundation, Turin, Italy
francesco.bonchi@isi.it

Ricardo Baeza-Yates
Web Research Group, DTIC
Pompeu Fabra University, Spain
rbaeza@acm.org

- Extends the GraSS framework to dynamic graphs
- Dynamic graph = a tensor with one dimension increasing in time
- Potentially infinite stream of static graphs
- Define a sliding tensor window
- Summarize the tensor within the tensor window





Overview and contributions

At each time-stamp :

- A new adjacency matrix arrives
- The sliding window is updated (one adjacency matrix exits the window)
- Summary is created for the current window, by clustering nodes to creat supernodes (following Riondato et al.)
- **Output:** one summary at every time-stamp

Contributions:

- Two online algorithms for summarizing dynamic, large-scale graphs
- Distributed, scalable algorithms, implemented in Apache Spark

B. Tensor summarization

We consider next a time series of w static graphs as described before. The time series of static graphs can be expressed as a time series of adjacency matrices $A_{G^t} \in [0,1]^{NN}$, where $t \in T$ or as a 3-order tensor $\mathcal{A}_G^W \in [0,1]^{NNw}$ as depicted in Figure 1(a). Similarly to the static graph case, given $k \leq N$ we define as k-summary of the tensor $\mathcal{A}_G^W$ the adjacency matrix $A_{G'} \in [0,1]^{kk}$ which is uniquely identified by a k-partition $S = \{S_1, ..., S_k\}$ of V:

$$A_{G'}(S_i, S_j) = \frac{\sum_{t=0}^{w} \sum_{k \in S_i, l \in S_j} \mathcal{A}_G^W(k,l,t)}{w|S_i||S_j|}, S_i \neq S_j \quad (1)$$

and

$$A_{G'}(S_i, S_j) = \frac{2\sum_{t=0}^{w} \sum_{k \in S_i, l \in S_j} \mathcal{A}_G^W(k,l,t)}{w|S_i||S_j - 1|}, S_i = S_j. \quad (2)$$

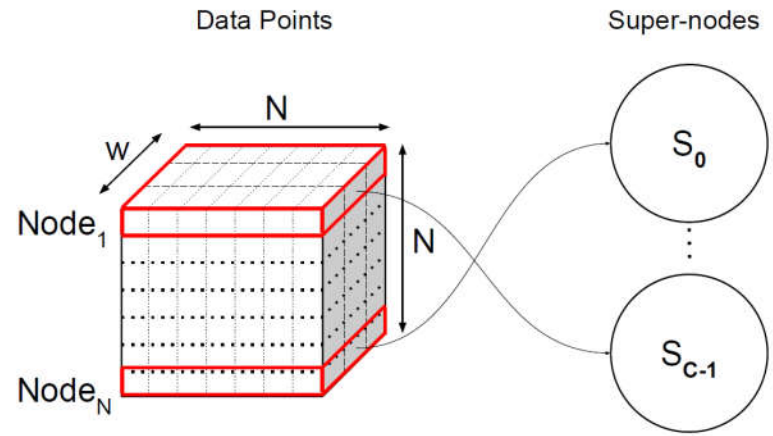
The reconstruction error for tensor summarization is defined as follows:

$$RE(\mathcal{A}_G^W | A_{G'}) = \frac{\sum_{t=0}^{w-1} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |\mathcal{A}_G^W(V_i, V_j, t) - A_{G'}(s(V_i), s(V_j))|}{wN^2}. \quad (3)$$

Algorithms

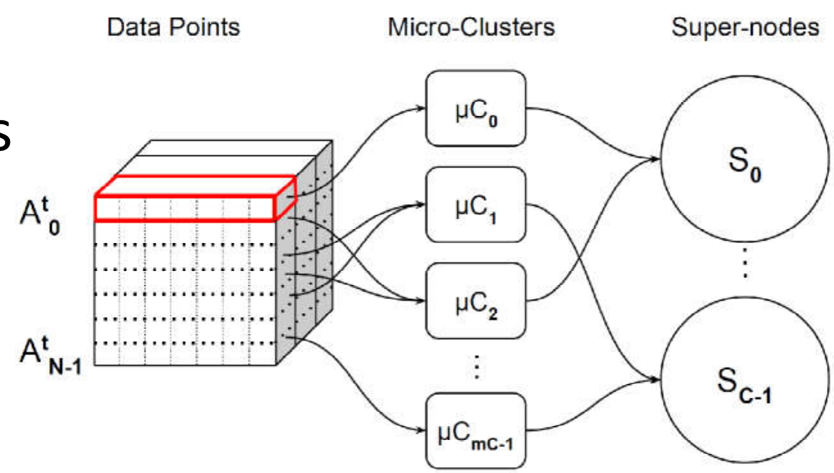
Baseline:

- Standard k-means clustering at each timestamp
- *N* points each with *wN* values
- Observation: *(w-1)N²* unchanged at every new timestamp



Two-level clustering:

- adjacency matrix to micro-clusters
- keep statistics in the micro-clusters
- run maintenance algorithm
- micro-clusters to supernodes



KDD'15

TimeCrunch: Interpretable Dynamic Graph Summarization

Neil Shah
Carnegie Mellon University
neilshah@cs.cmu.edu

Danai Koutra
Carnegie Mellon University
danai@cs.cmu.edu

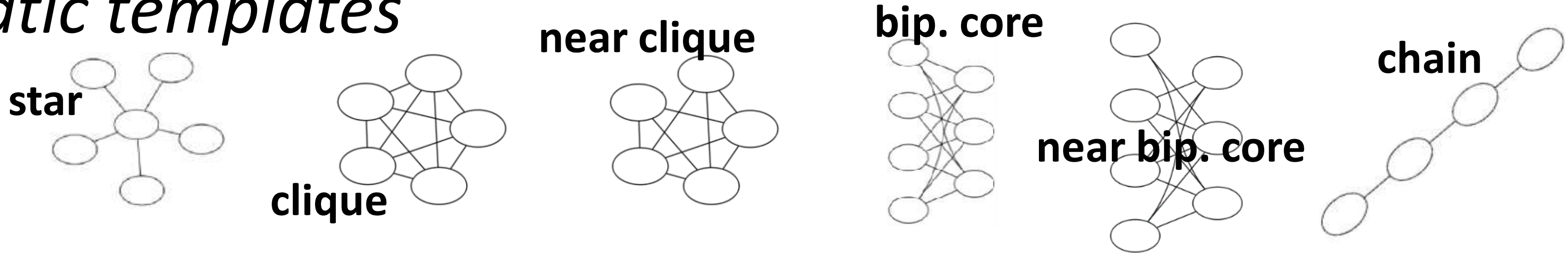
Tianmin Zou
Carnegie Mellon University
tzou@andrew.cmu.edu

Brian Gallagher
Lawrence Livermore Lab
bgallagher@llnl.gov

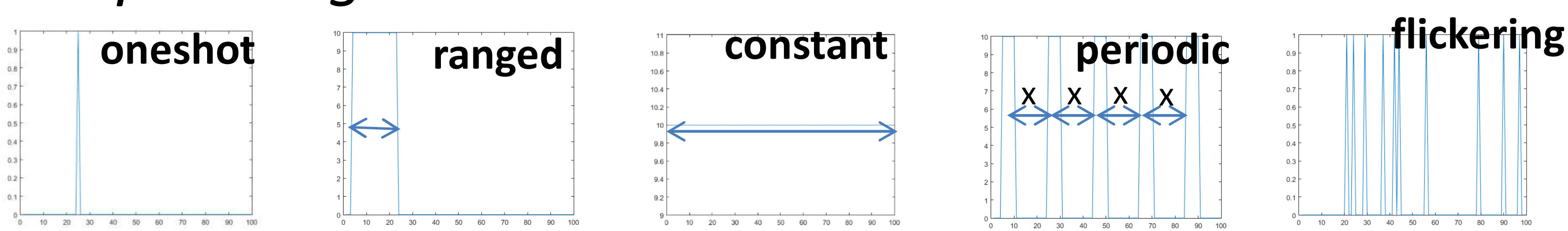
Christos Faloutsos
Carnegie Mellon University
christos@cs.cmu.edu

1) Use a dictionary of **temporal templates:**

- *Static templates*



- *Temporal signatures*

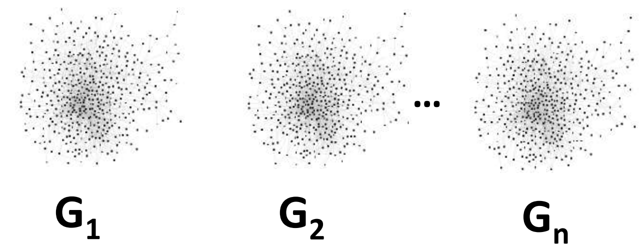


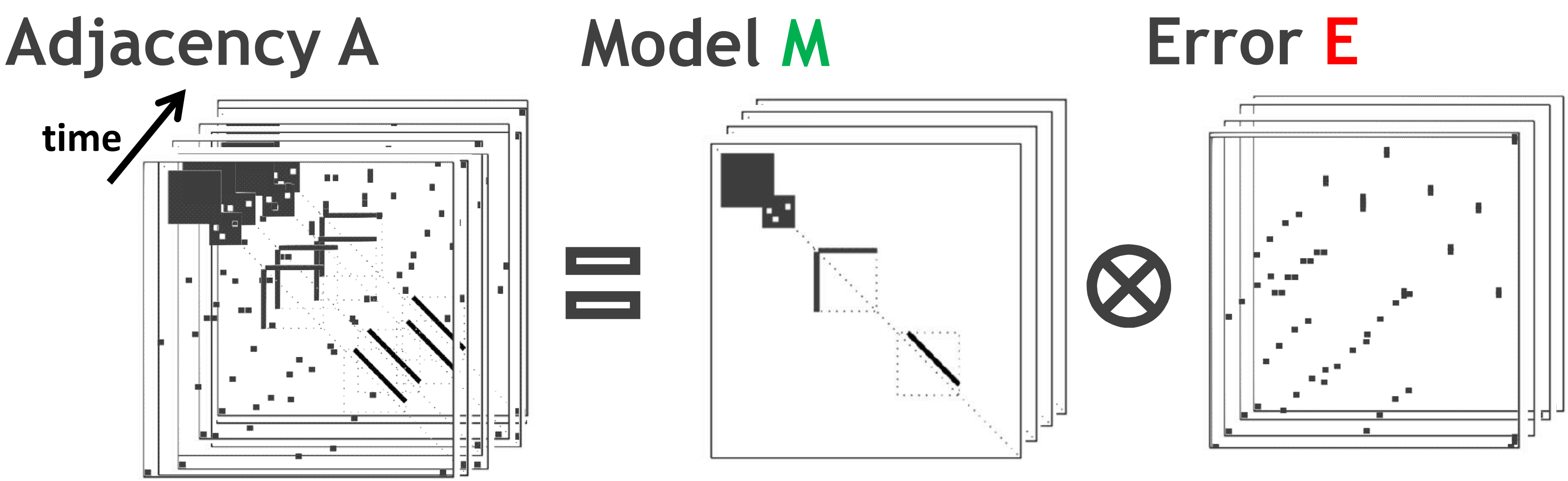
1) Get the shortest lossless description (MDL)

- Better compression → better summary

Formal goal

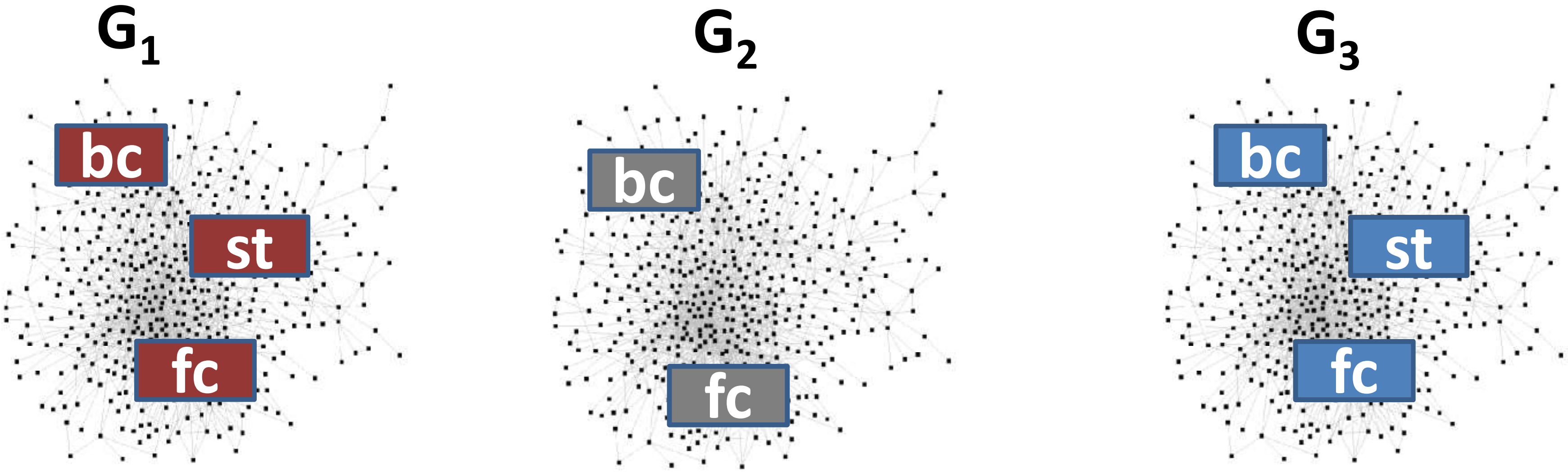
- **Given** a dynamic graph G
  temporal templates **Φ,**
- **Find** the smallest model M
  s.t. min L(G,M) = L(**M**) + L(**E**)





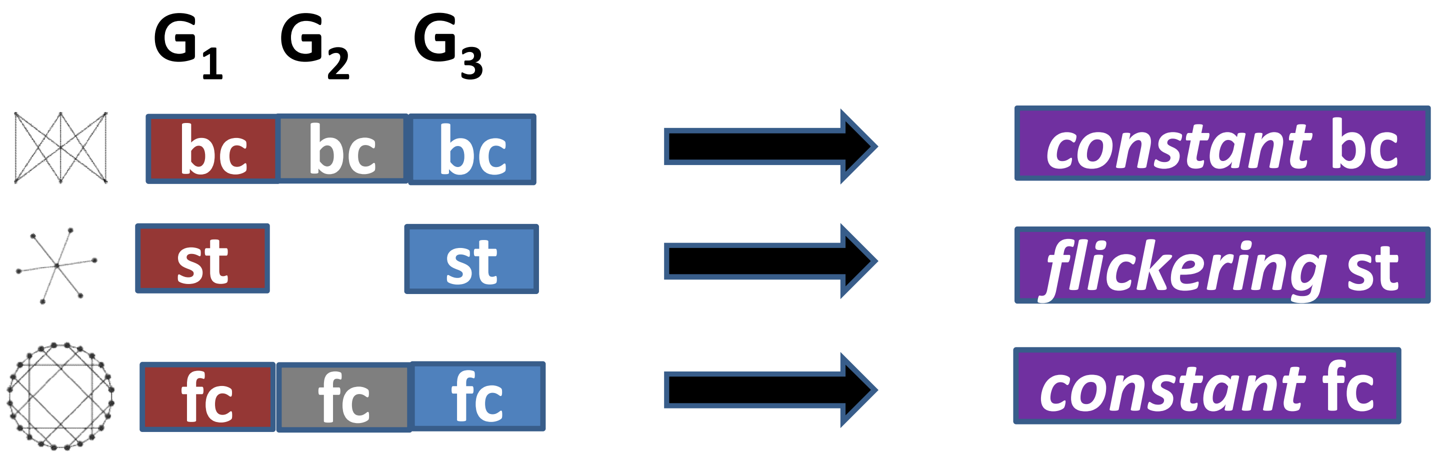
Proposed algorithm: **TimeCrunch**

- **Step 1:** Generate static subgraph instances



Proposed algorithm: **TimeCrunch**

- **Step 1:** Generate static subgraph instances
- **Step 2:** Stitch *static instances* together to form *temporal instances*



Proposed algorithm: **TimeCrunch**

- **Step 1:** Generate static subgraph instances
- **Step 2:** Stitch *static instances* together to form *temporal instances*
- **Step 3:** Compose the dynamic graph summary

Temporal instances	MDL savings
constant bc	$$$$
flickering st	
constant fc	$$$

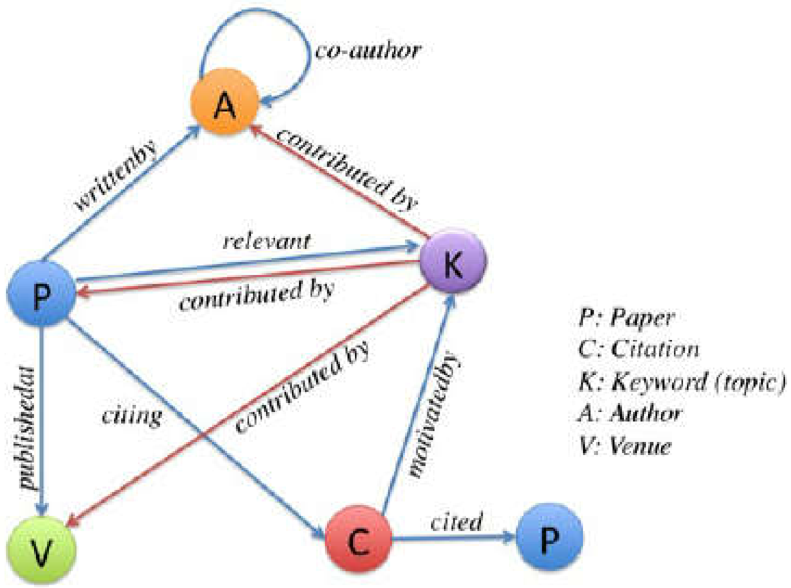
Summary

constant bc
~~*flickering st*~~
constant fc

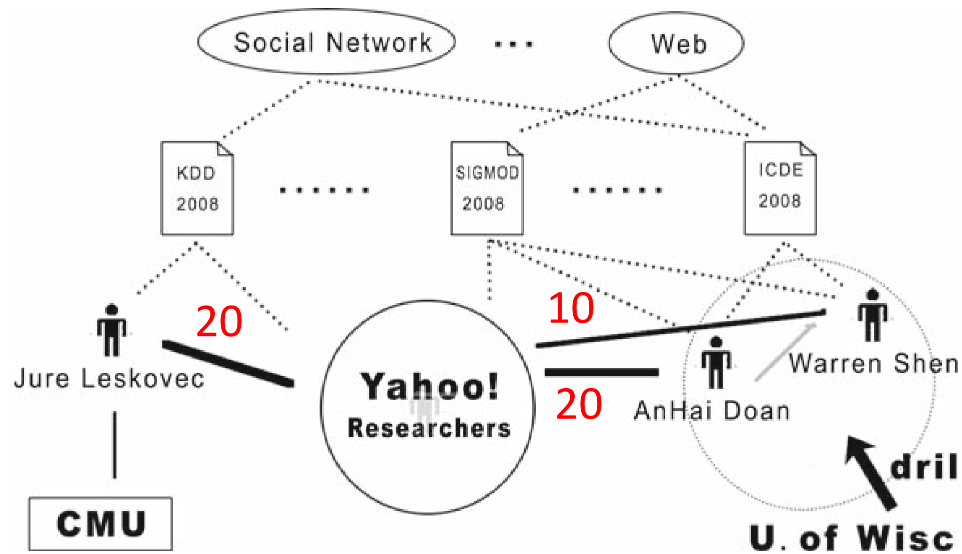
Roadmap

- Introduction
- Summarizing Static Graphs
- Summarizing Dynamic Graphs
- Summarizing Heterogeneous Graphs
- Future Work and Conclusion

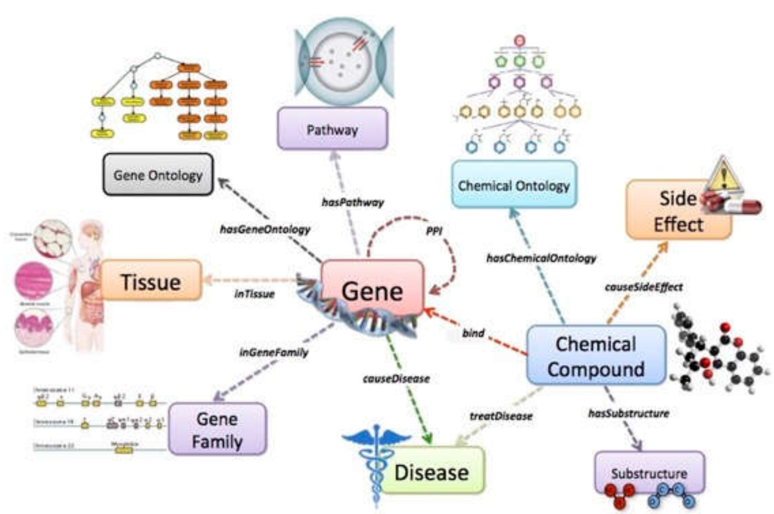
Heterogeneous Graphs



Collaboration Network



Collaboration Network



Biological Network

Roadmap

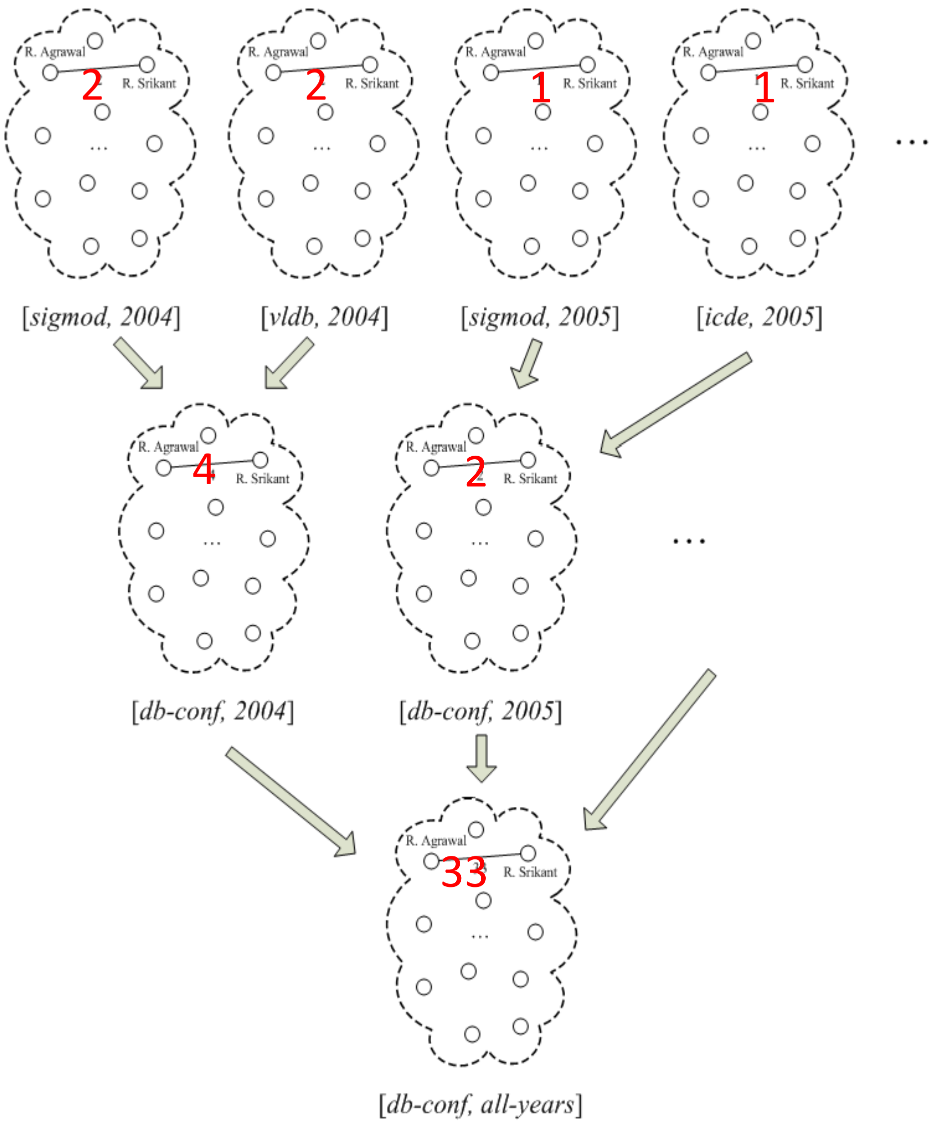
- **Summarizing Heterogeneous Graphs**
  - Graph OLAP
  - SNAP

- **Graph OLAP: Towards Online Analytical Processing on Graphs:**
  [C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]

- **[SNAP] Efficient Aggregation for Graph Summarization:**
  [Y. Tian, R. A. Hankins, J. M. Patel, SIGMOD 2008]

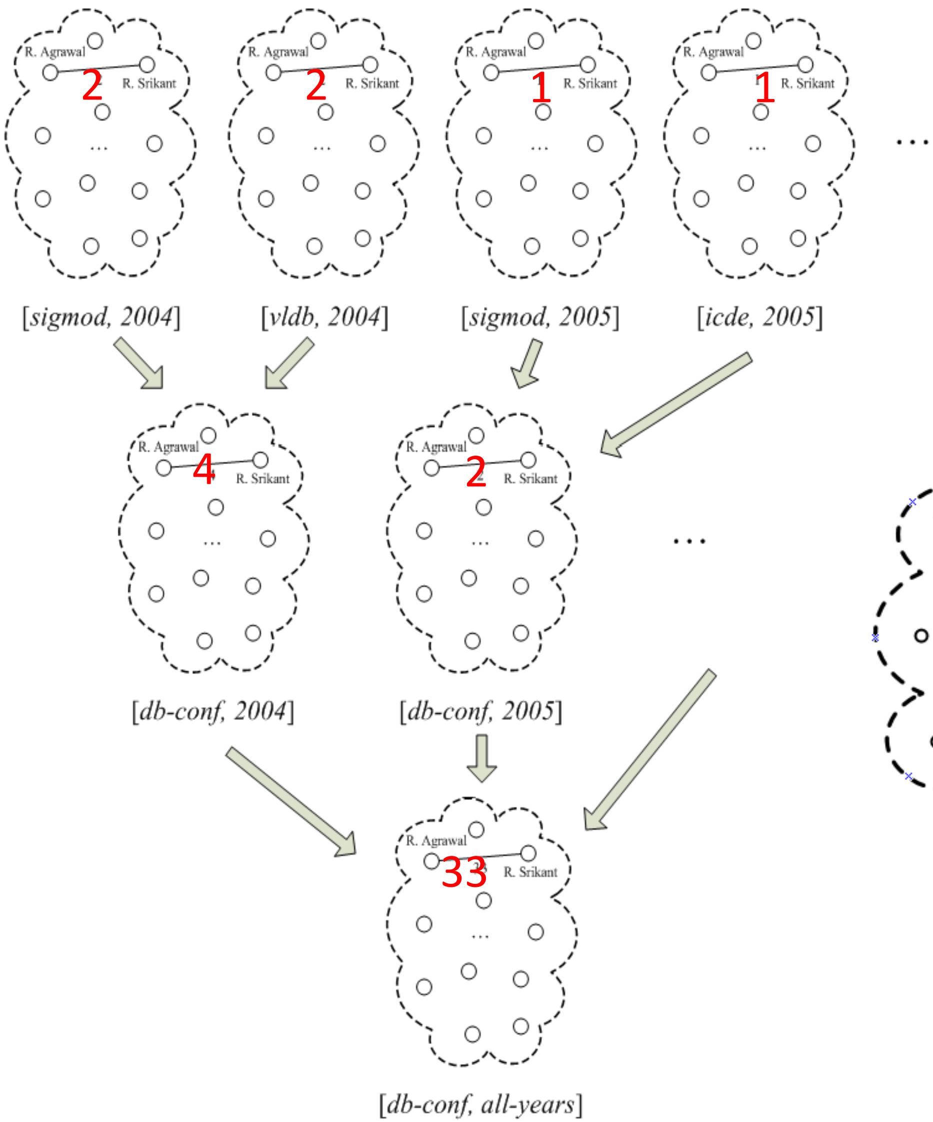
Graph OLAP

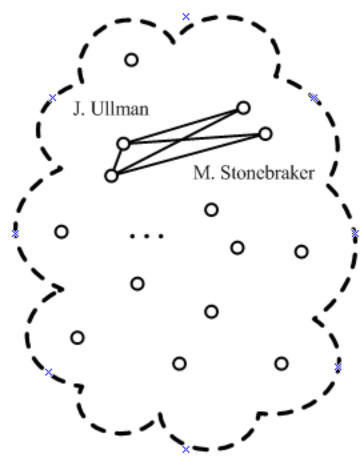
[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]



Graph OLAP

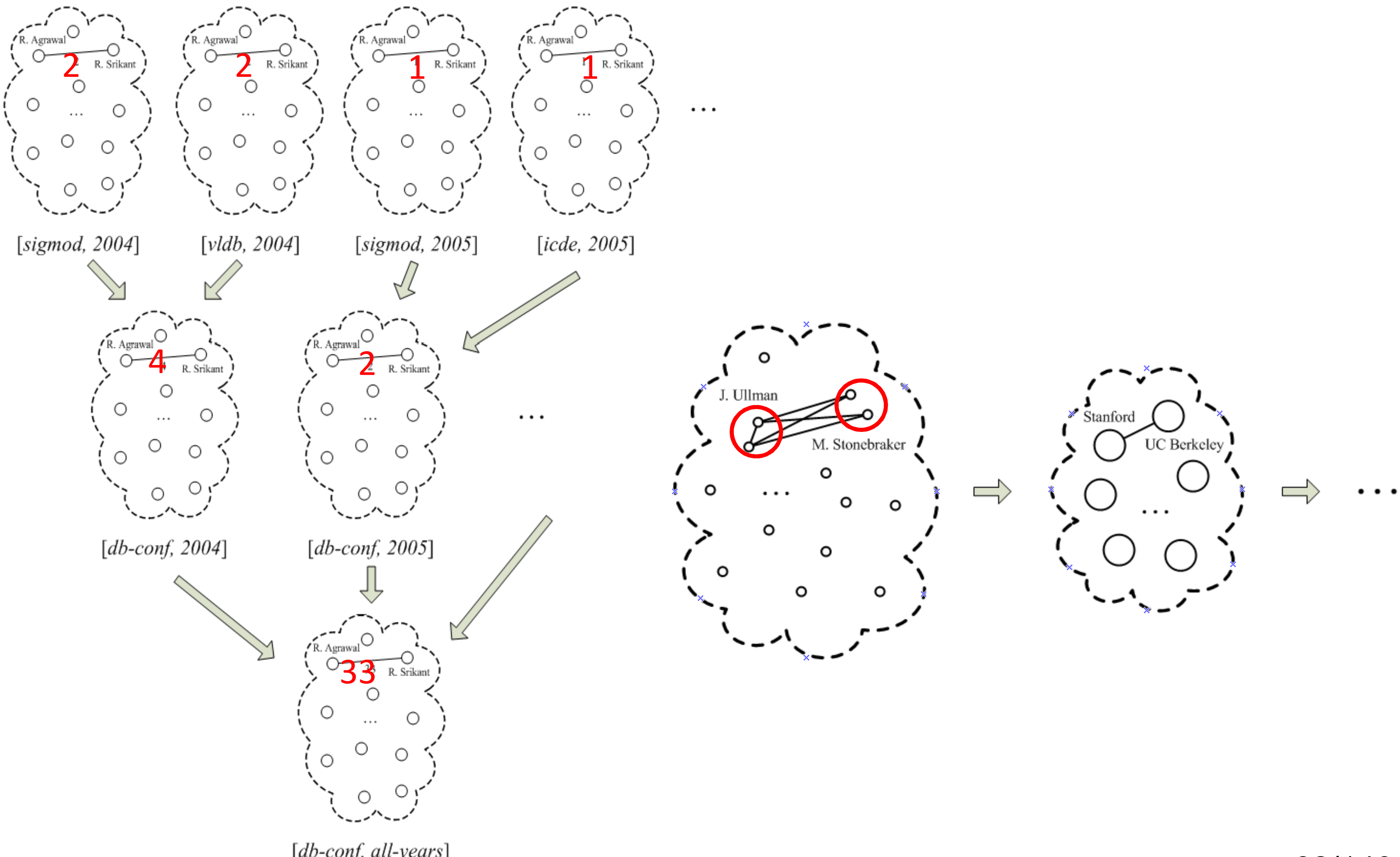
[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]





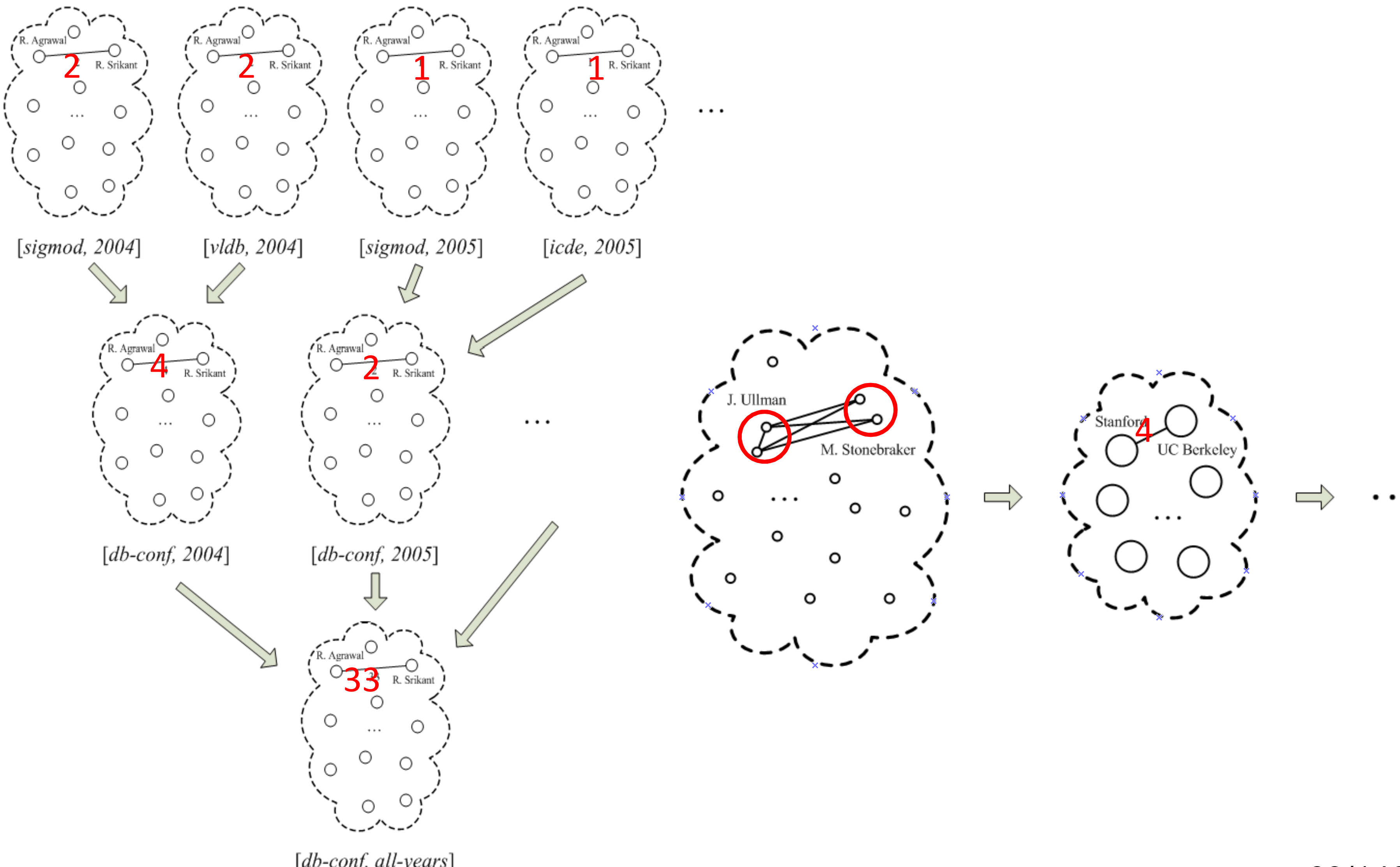
Graph OLAP

[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]



Graph OLAP

[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]



Graph OLAP

[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]

- Collection of network snapshots

  $\mathcal{G} = \{\mathbf{G_1}, \mathbf{G_2}, \ldots, \mathbf{G_N}\}$

- Each snapshot $\mathbf{G_i} = (I_{1,i}, I_{2,i}, \ldots, I_{k,i}; G_i)$

- $I_{1,i}, I_{2,i}, \ldots, I_{k,i}$ are k informational attributes describing the snapshot

- $G_i = (V_i, E_i)$ is an attributed graph, with attributes attached with its nodes V_i and edges E_i

- Since $G_1, G_2, \ldots, G_N$ represent different observations of a network, $V_1, V_2, \ldots, V_N$ correspond to the same set of objects

Graph OLAP

[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]

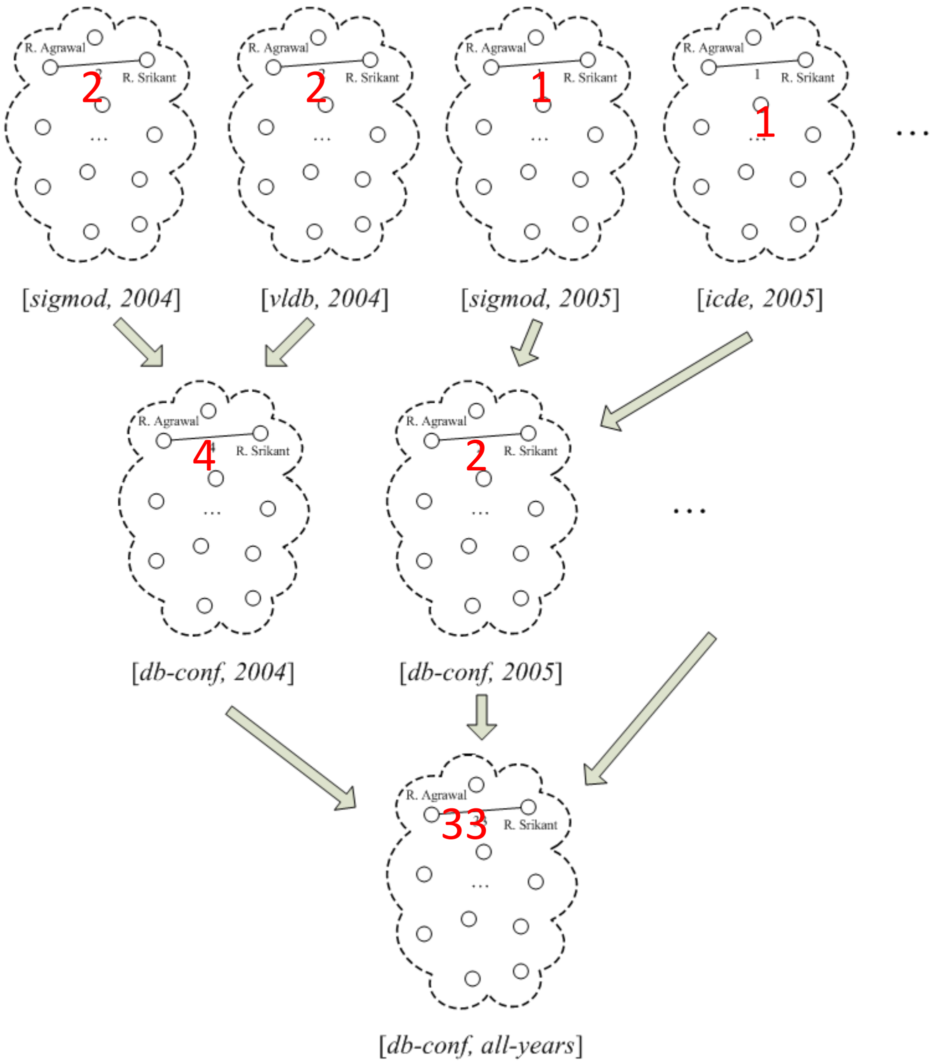
- Two Types of OLAP

  - Informational OLAP (I-OLAP)

  - Topological OLAP (T-OLAP)

Graph OLAP

[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]



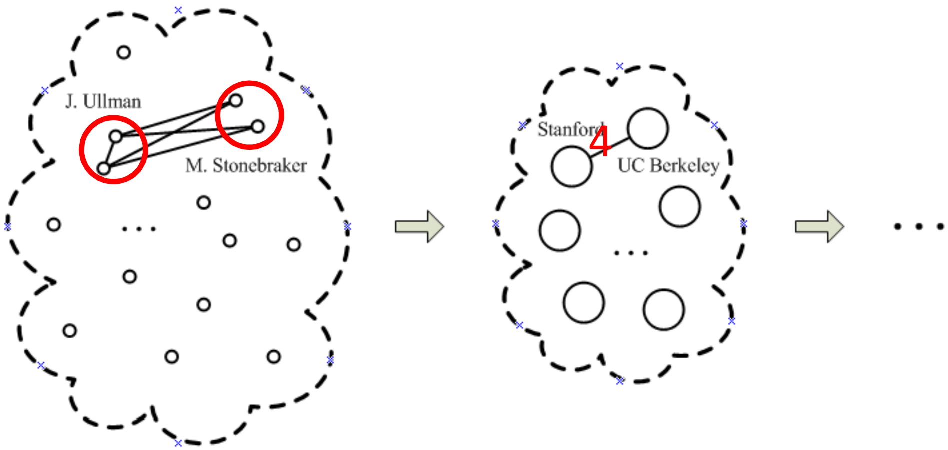
I-OLAP (Informational OLAP)

- Dimensions come from informational attributes attached at the **whole** snapshot level, so-called *Info-Dims*
- Overlay multiple pieces of information
- Do not change the objects whose interactions are being looked at

- In the underlying snapshots, each node is a researcher
- In the summarized view, each node is still a researcher

Graph OLAP

[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]



T-OLAP (Topological OLAP)

- Dimensions come from the **node/edge** attributes inside individual networks, so-called *Topo-Dims*
- Zoom in/Zoom out
- Network topology changed: “generalized” nodes and “generalized” edges

- In the underlying network, each node is a researcher
- In the summarized view, each node becomes an institute that comprises multiple researchers

Graph OLAP

[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]

- Graph OLAP Measures

  - Aggregated graph, node count, average degree, maximum flow, shortest path, centrality, etc.

Graph OLAP

[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]

- Graph OLAP Measures

  - Aggregated graph, node count, average degree, maximum flow, shortest path, centrality, etc.

- Graph OLAP Operations

	Graph I-OLAP	**Graph T-OLAP**
Roll-up	Overlay multiple snapshots to form a higher-level summary via I-aggregated graph	Shrink the topology and obtain a T-aggregated graph that represents a compressed view, whose topological elements (i.e., nodes and/or edges) have been merged and replaced by corresponding higher-level ones
Drill-down	Return to the set of lower-level snapshots from the higher-level overlaid (aggregated) graph	A reverse operation of roll-up
Slice/dice	Select a subset of qualifying snapshots based on Info-Dims	Select a subgraph of the network based on Topo-Dims

Graph OLAP

[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]

- Graph OLAP Measures Classification – How to combine and leverage intermediate results?

- **Distributive**
  - The computation of high-level cells can be directly built on low-level cells
  - *collaboration frequency*

- **Algebraic**
  - Not distributive, but can be easily derived from several distributive measures
  - *maximum flow*

- **Holistic**
  - Neither distributive nor algebraic
  - *centrality*

Graph OLAP

[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]

- Graph OLAP Measures Classification – How to combine and leverage intermediate results?

- **Distributive** ← SUM, COUNT

  - The computation of high-level cells can be directly built on low-level cells

  - *collaboration frequency*

- **Algebraic** ← AVG = *f* (SUM, COUNT)

  - Not distributive, but can be easily derived from several distributive measures

  - *maximum flow*

- **Holistic** ← MEDIAN, MODE, RANK

  - Neither distributive nor algebraic

  - *centrality*

Graph OLAP

[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]

- Optimal Computation of graph OLAP measures

  - Bottom up
  - Top down

Graph OLAP

[C. Chen, X. Yan, F. Zhu, J. Han, P. S. Yu, ICDM 2008]

- Optimal Computation of graph OLAP measures

  - Bottom up
  - Top down

- **Distributive** ← Localization

  - The computation of high-level cells can be directly built on low-level cells
  - *collaboration frequency*

- **Algebraic** ← Attenuation

  - Not distributive, but can be easily derived from several distributive measures
  - *maximum flow*

- **Holistic** ← Constraint Pushing

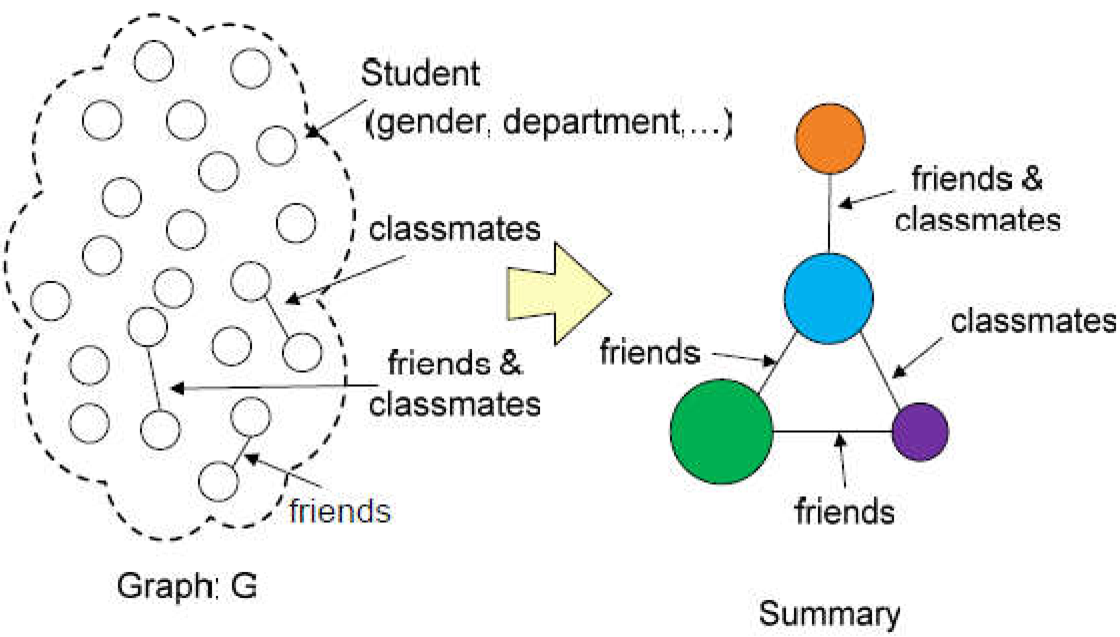
  - Neither distributive nor algebraic
  - *centrality*

SNAP & k-SNAP

[Y. Tian, R. A. Hankins, J. M. Patel, SIGMOD 2008]

- SNAP – **S**ummarization by Grouping **N**odes on **A**ttributes and **P**airwise Relationships

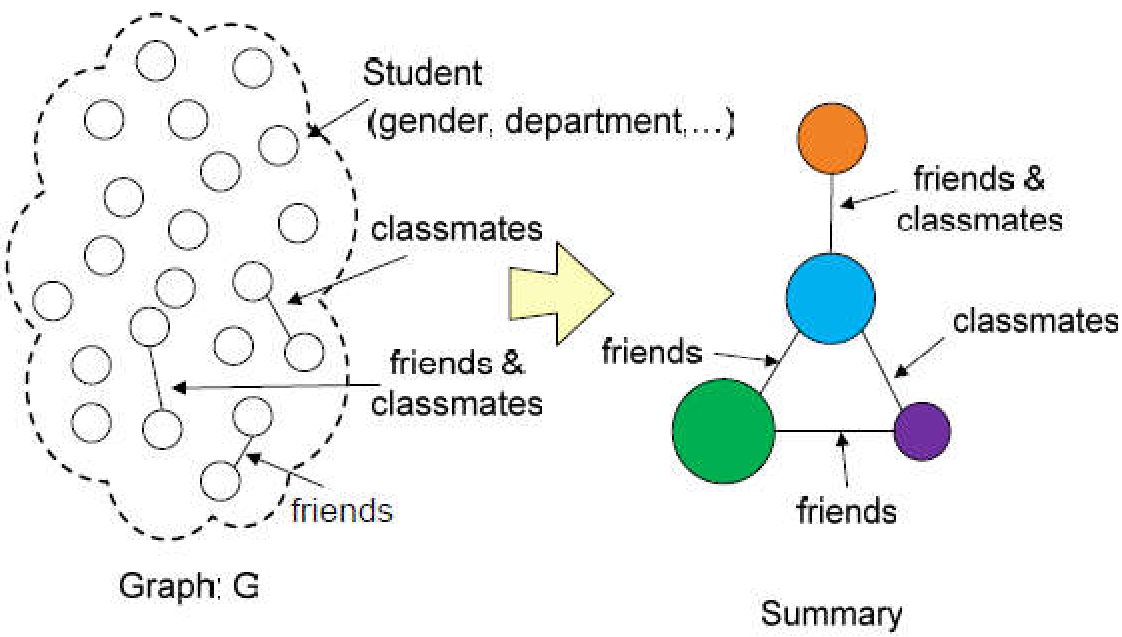
- Similar to T-OLAP (Topological OLAP)



SNAP & k-SNAP

[Y. Tian, R. A. Hankins, J. M. Patel, SIGMOD 2008]

- SNAP – **S**ummarization by Grouping **N**odes on **A**ttributes and **P**airwise Relationships
- Similar to T-OLAP (Topological OLAP)

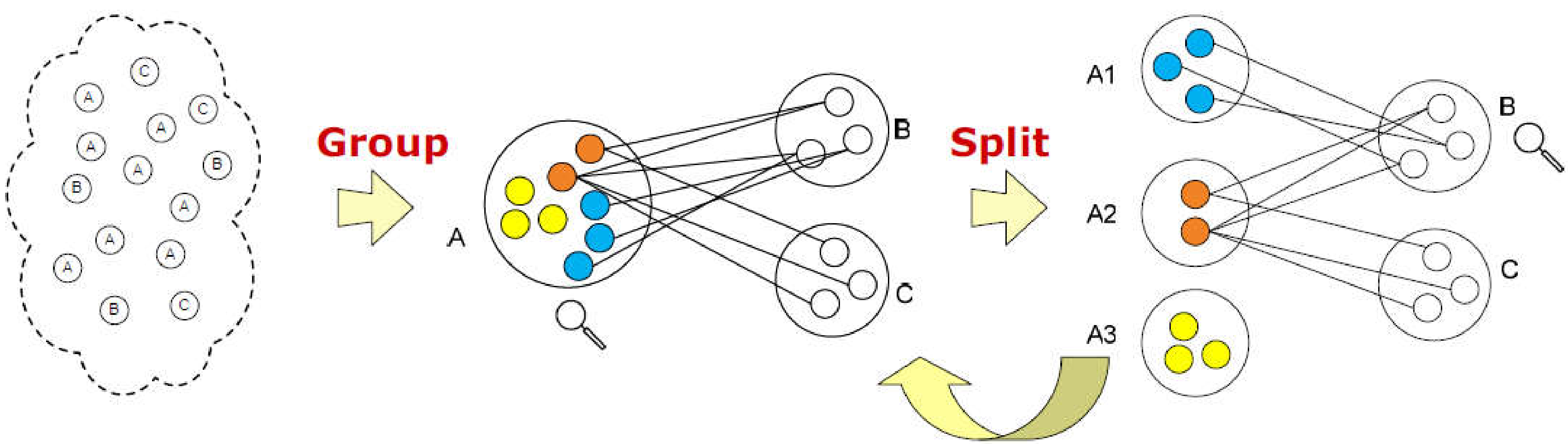


- Group nodes by user-selected node attributes & relationships
- Nodes in each group are homogenous w.r.t. attributes & relationships
- The grouping with the minimum # groups

SNAP & k-SNAP

[Y. Tian, R. A. Hankins, J. M. Patel, SIGMOD 2008]

- Algorithm for SNAP – Top-Down approach
- **Step 1:** group nodes just based on user-selected attributes.
- **Iterative Step:**
  while a group breaks homogeneity requirement for relationships
  split the group based on its relationships with other groups



Open Research Problems

- Scalable, high quality attribute-aware summaries
- Application-driven summarization
- Summary maintenance
- Summarization of uncertain graphs
- Summarizing a set of graphs
- Differential summaries on massive networks