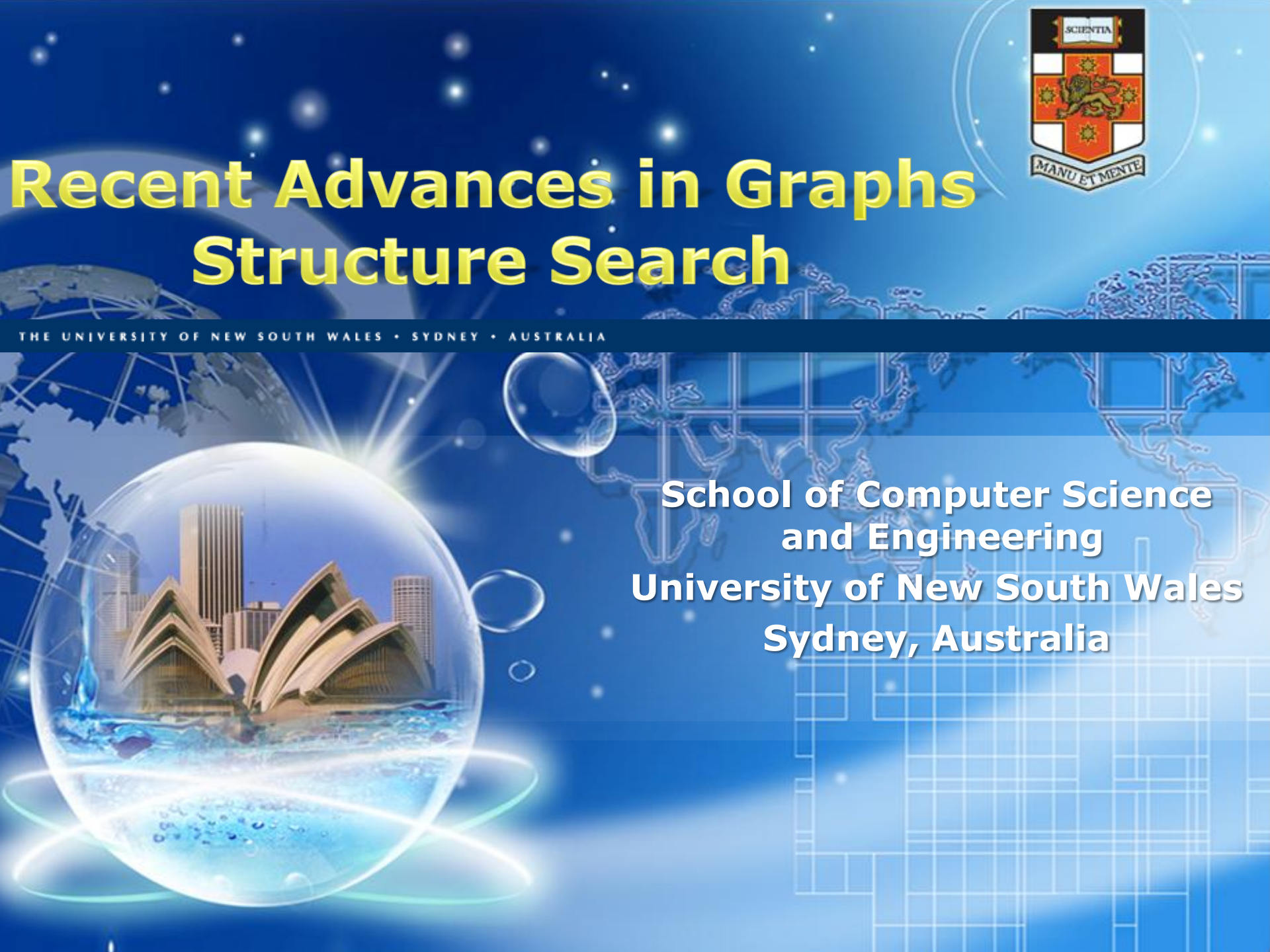
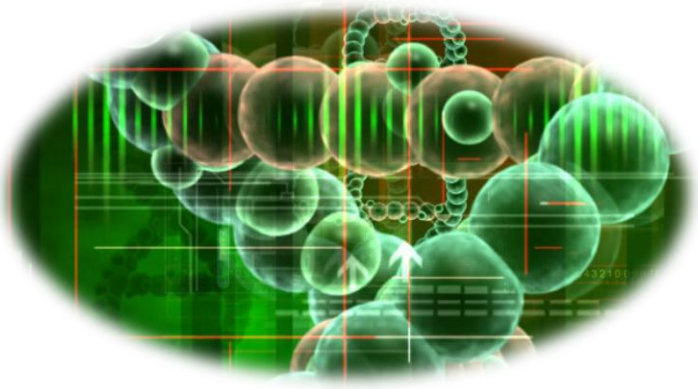
Recent Advances in Graphs Structure Search



THE UNIVERSITY OF NEW SOUTH WALES • SYDNEY • AUSTRALIA

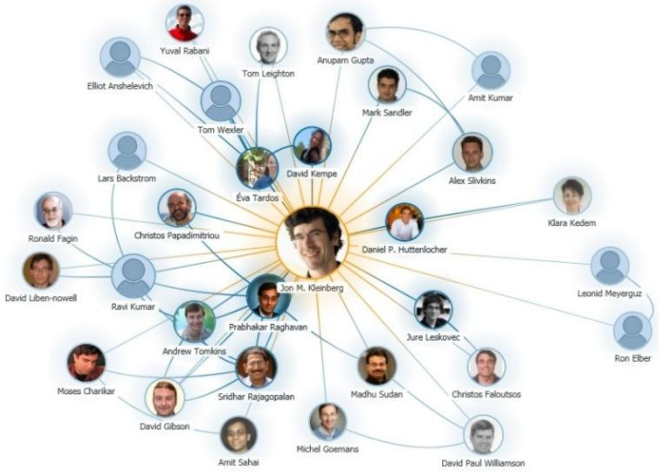
School of Computer Science and Engineering
University of New South Wales
Sydney, Australia

Graphs Can Model Complex Data Structures in Real World!



Chemical Compounds

Social Network

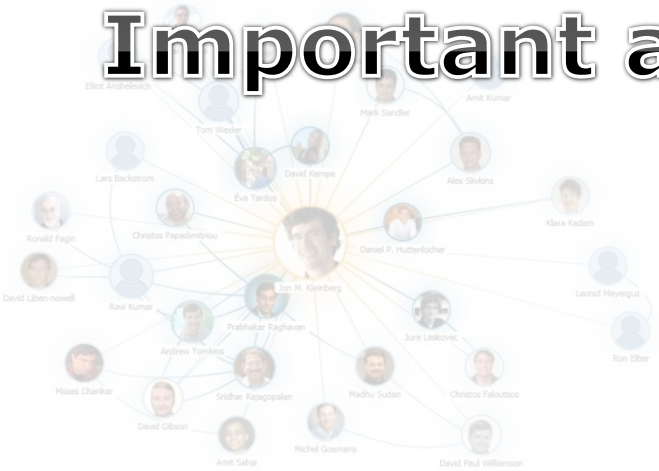


Collaboration Graph

Road Network

Graphs Can Model Complex Data Structures in Real World!

Structure Search on Large Graphs is Important and Imperative!

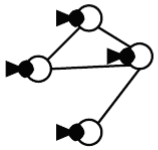


Collaboration Graph

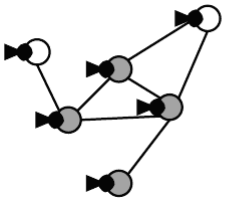
Road Network

Graph Structure Search

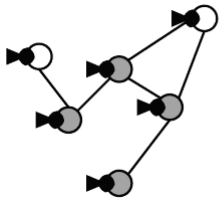
- **Substructure Search**
  - Retrieve graphs in DB which contain the query graph
- **Substructure Similarity Search**
  - Retrieve graphs in DB which have some components of the query graph
- **Superstructure Search**
  - Retrieve graphs in DB which are contained in the query graph
- **Pair-wise Structure Search**
  - Rank node-pairs in a graph based on link structure



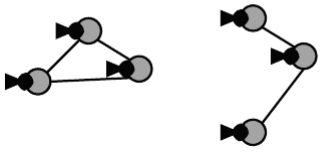
Query Graph



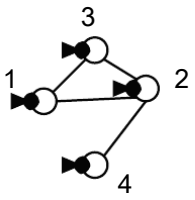
Substructure Search



Substructure Similarity Search

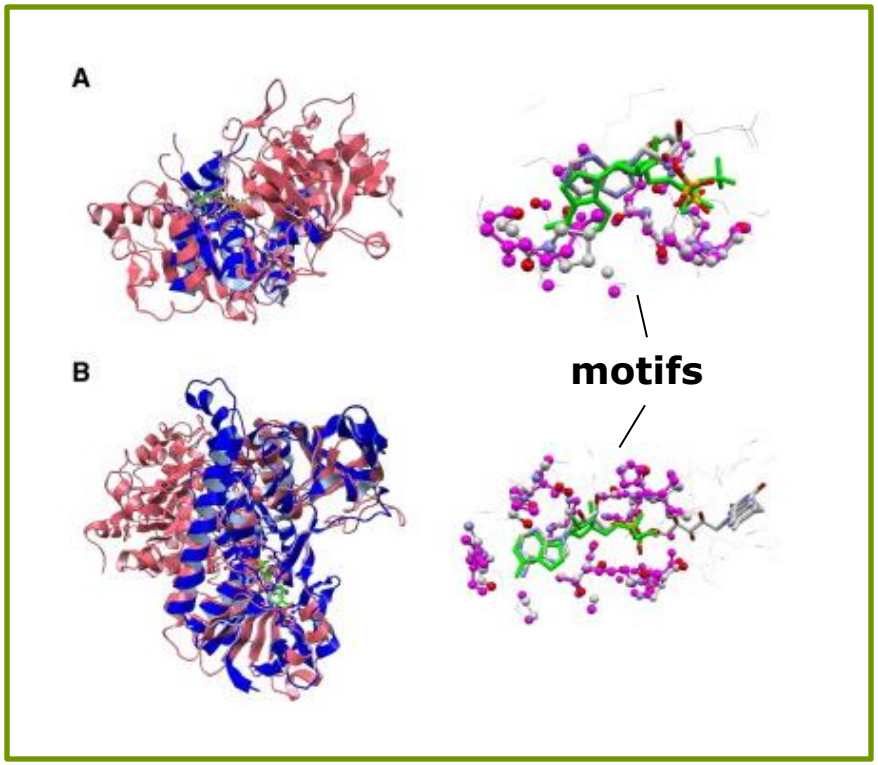


Superstructure Search

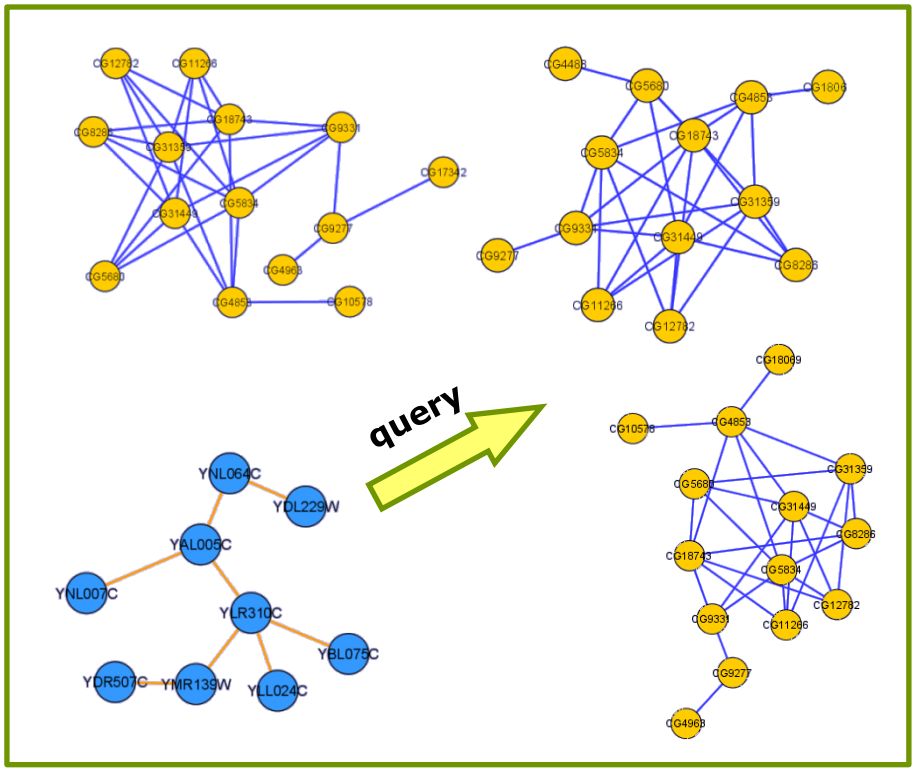


Pair-wise Structure Search

In Real World: Substructure Search (1,2)

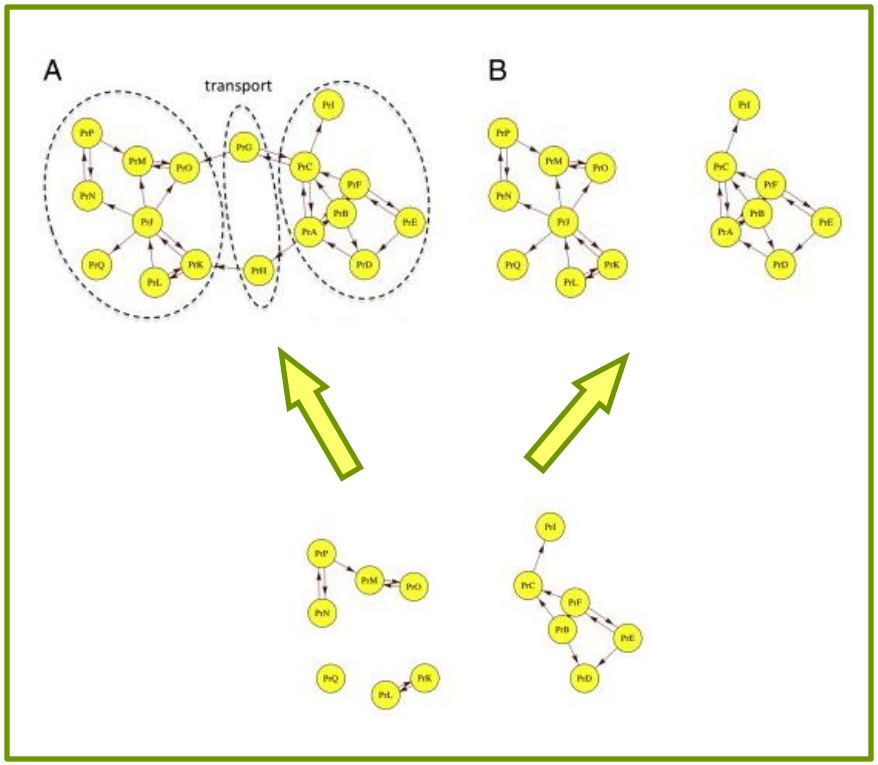


Find Motifs Shared by Different Proteins

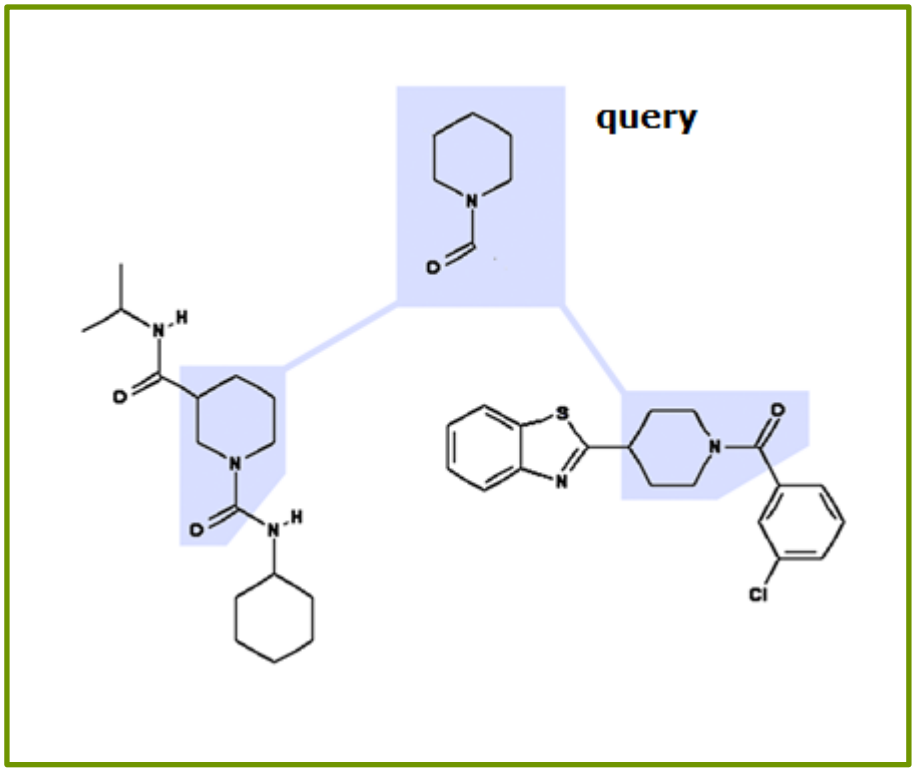


Query a Complex Species

In Real World: Substructure Search (3,4)

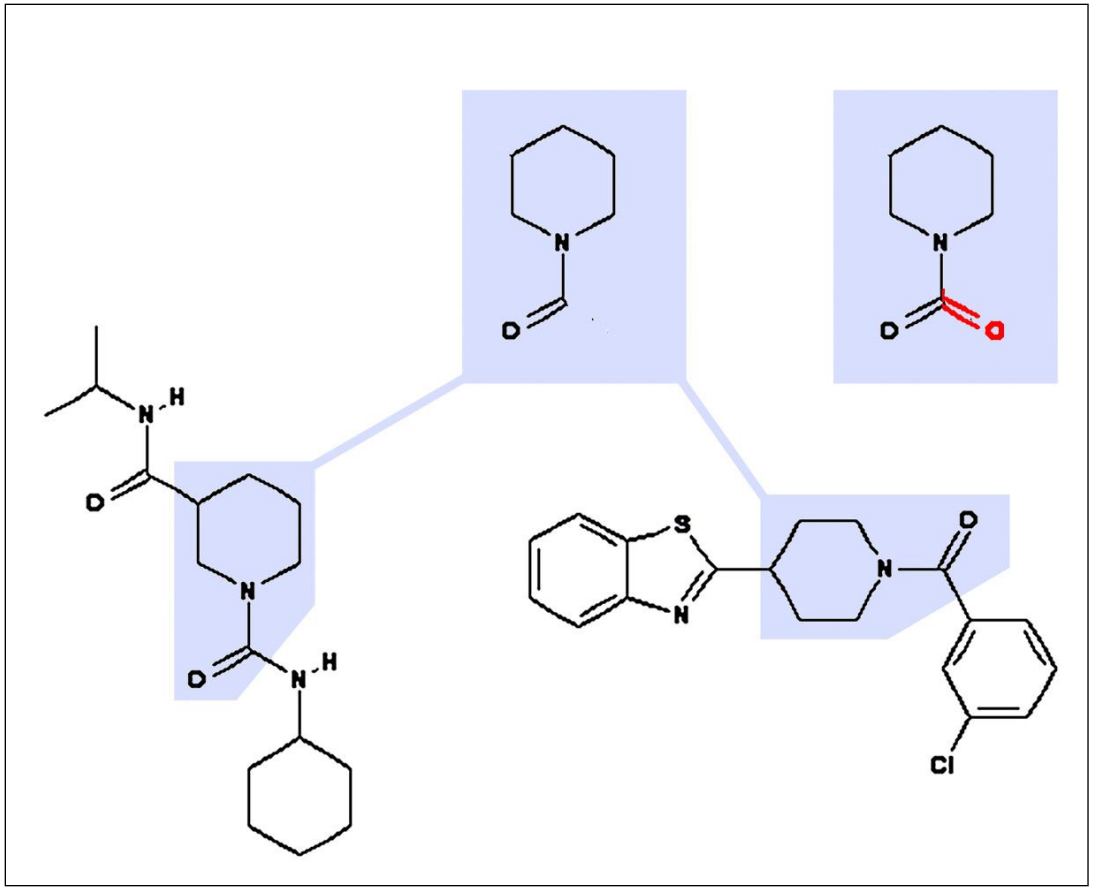


Drug Design



Chemical Compounds Identification

In Real World: Similarity Search

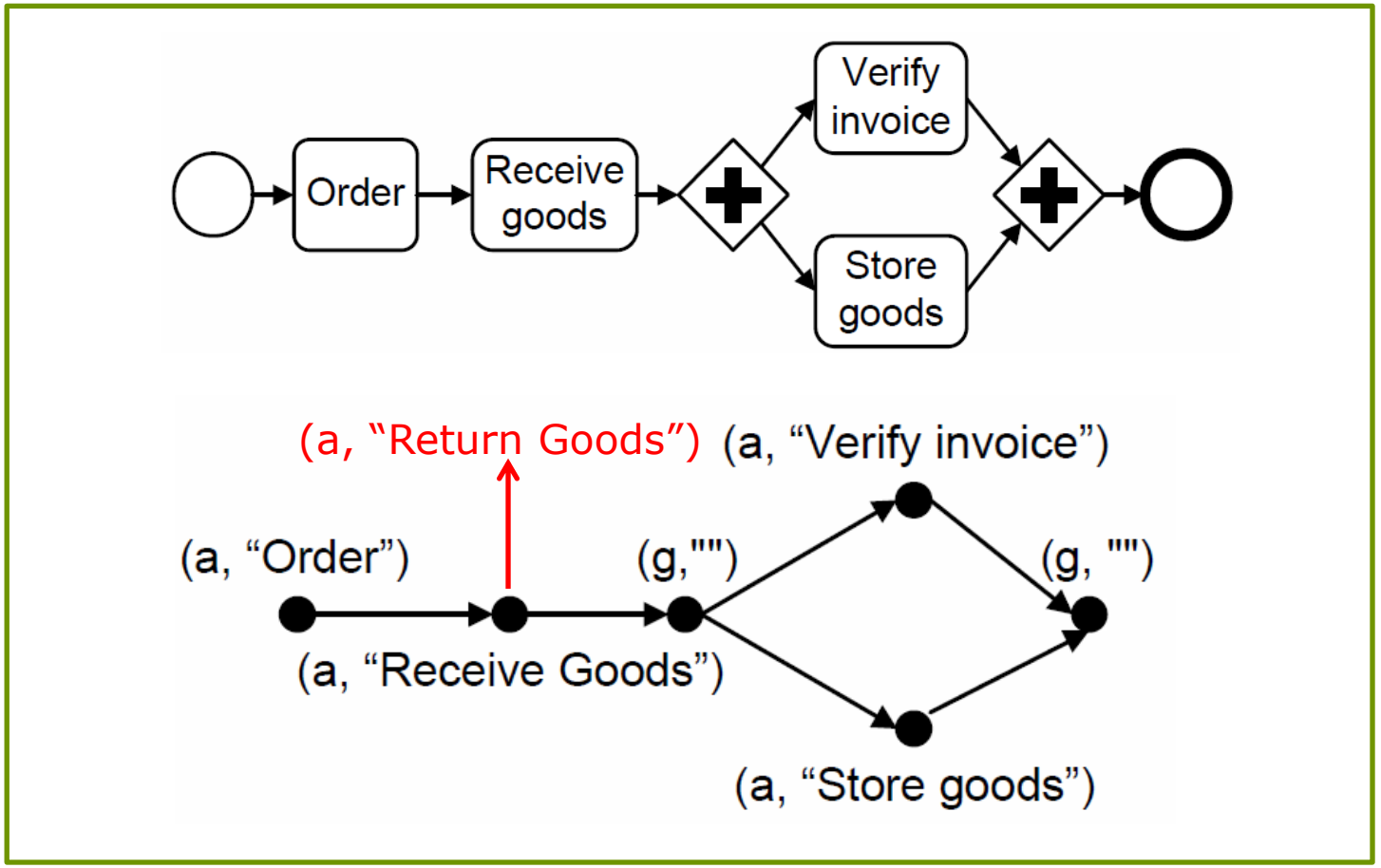


Input Mistake

Exploration Queries

……

In Real World: Graph Similarity Join



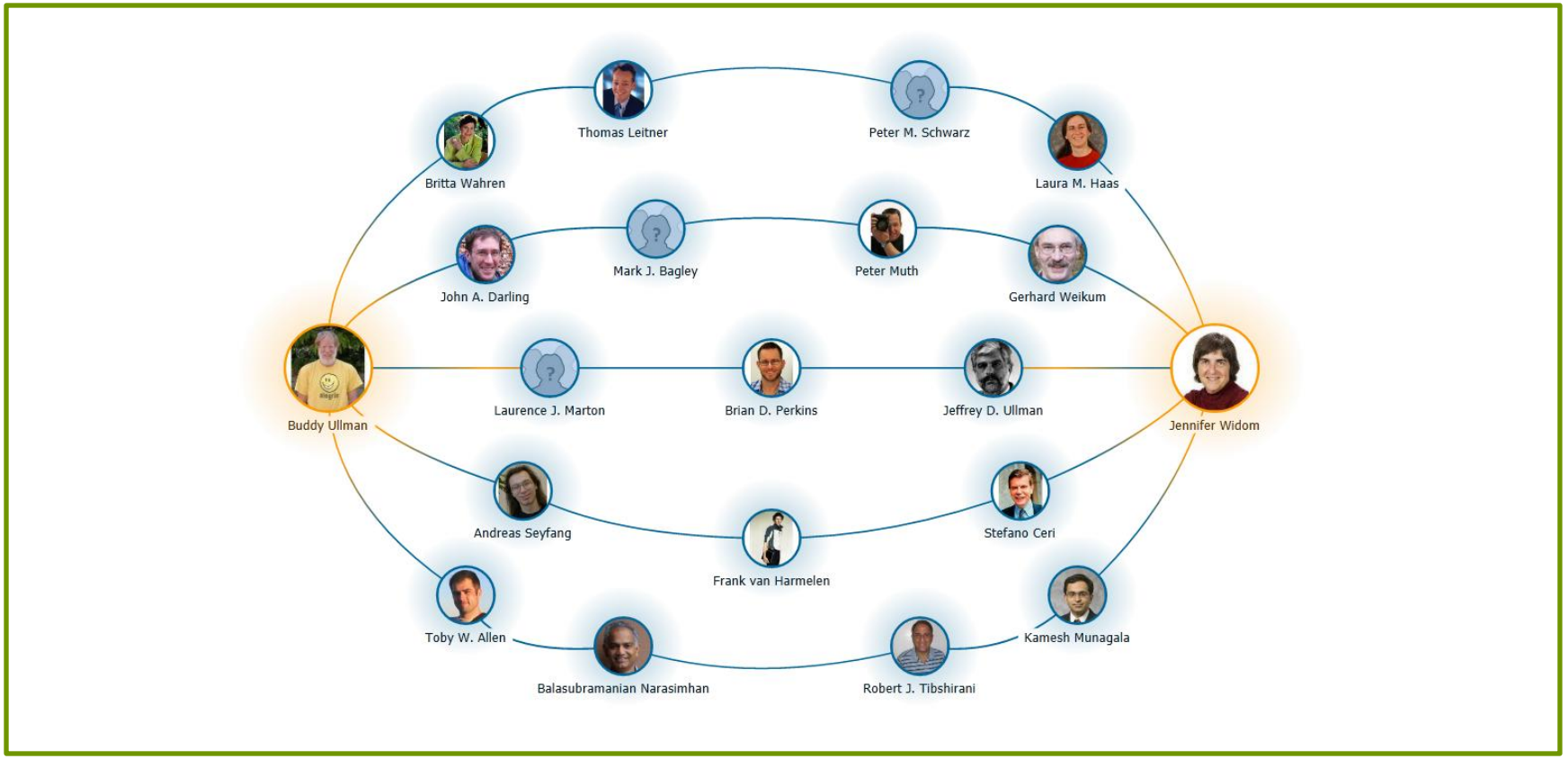
Business Model Repository

In Real World: Pair-wise Structure Search (1,2)

"Find-Similar-Document" Query in Search Engine

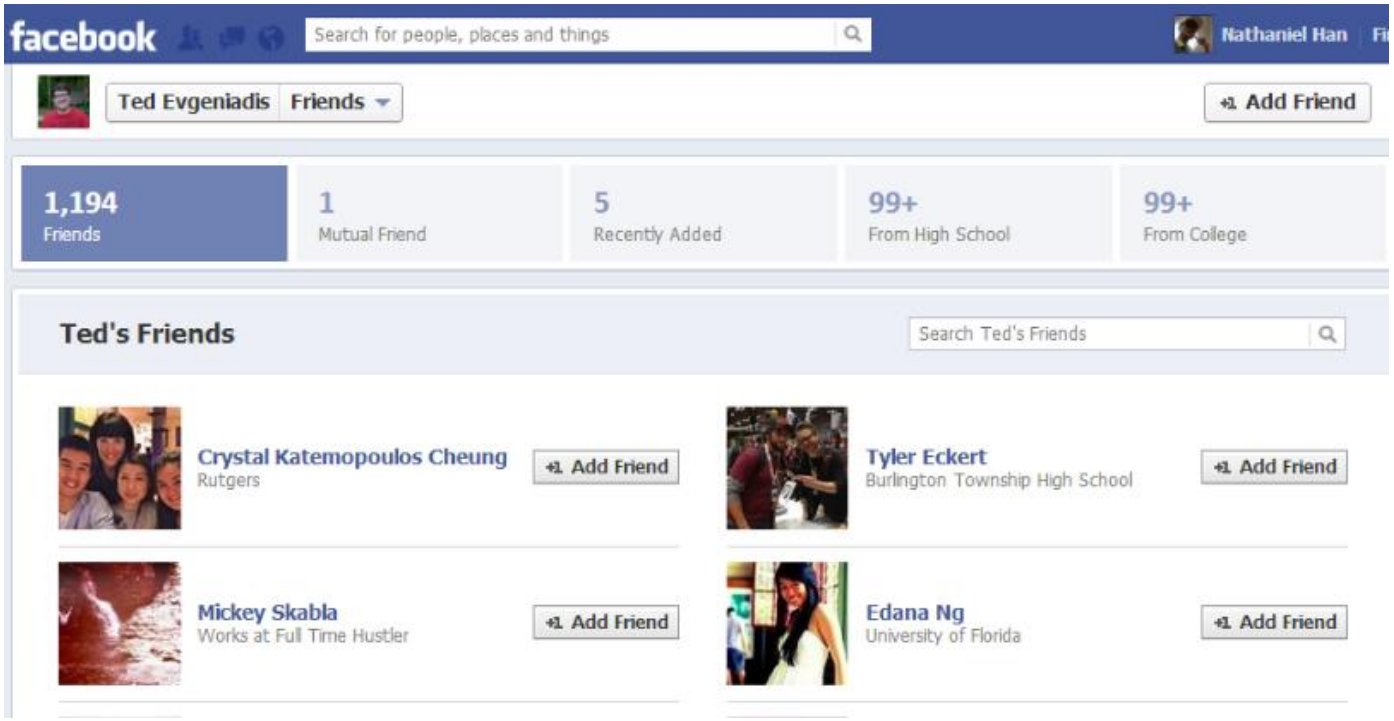
Collaborative Filtering in a Recommender System

In Real World: Pair-wise Structure Search (3)



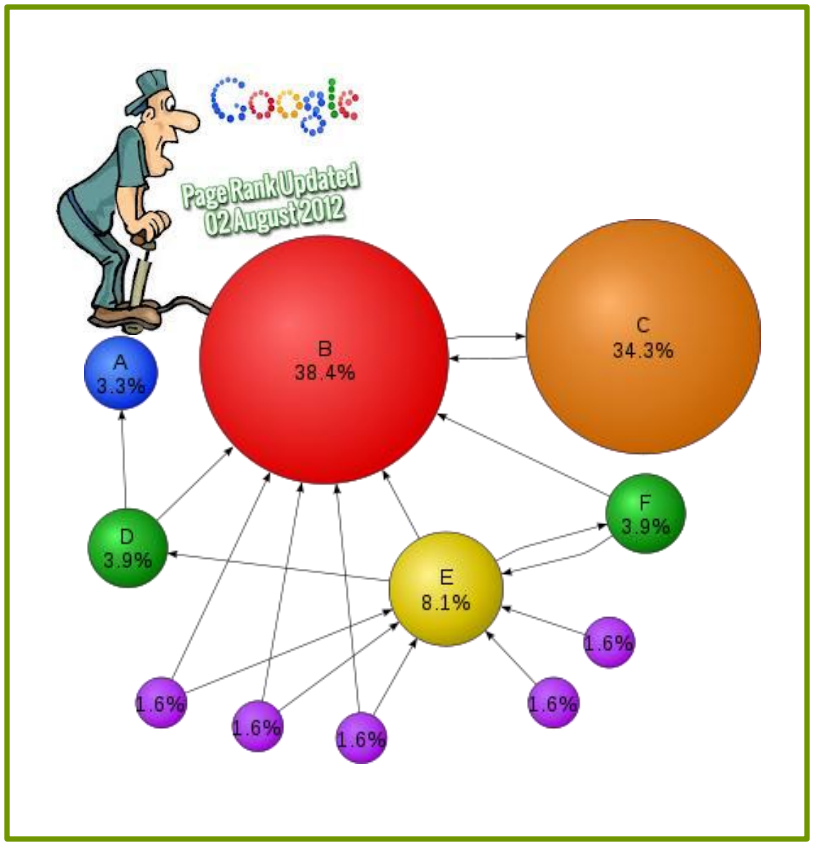
Find Co-author Path Between 2 Collaborators

In Real World: Pair-wise Structure Search (4)

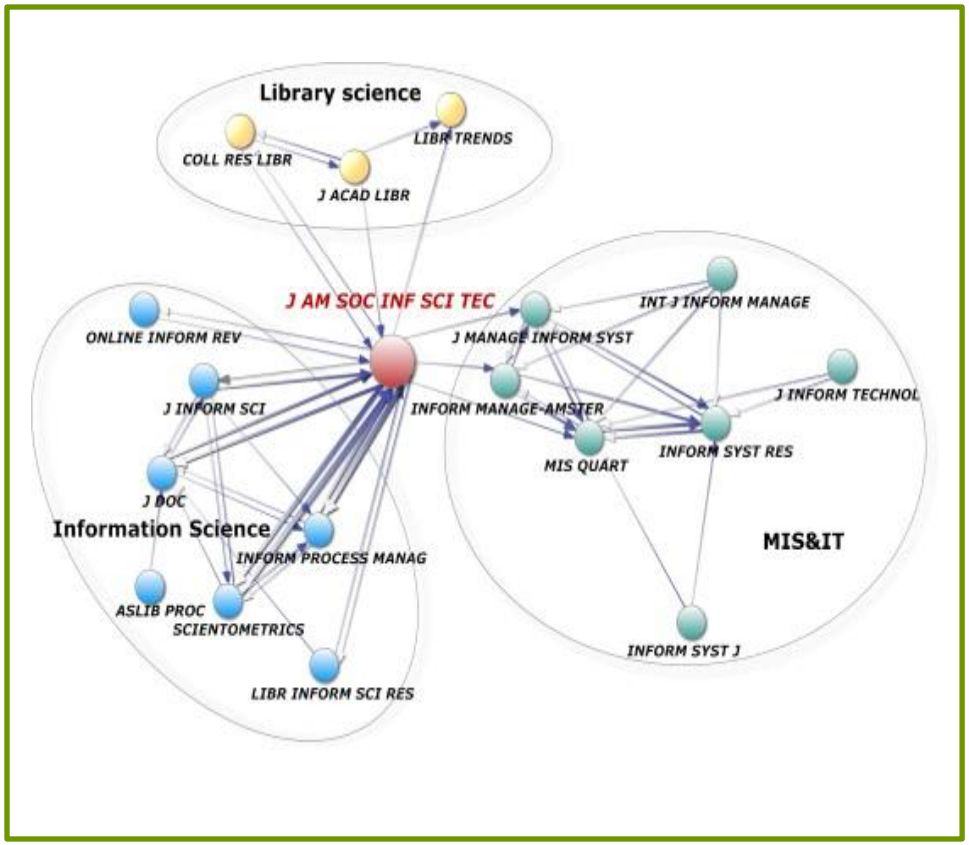


Find Close Friends with Similar Hobbies

In Real World: Pair-wise Structure Search (5,6)



Web Document Ranking



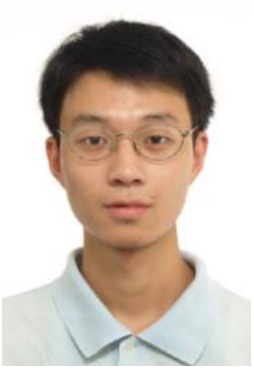
Citation Analysis

Collaborators

Dr Haichuan Shang
CSE, UNSW

Dr Wei Wang
CSE, UNSW

Prof Jeffrey Xu Yu
CUHK



Weiren Yu
CSE, UNSW

Dr Wenjie Zhang
CSE, UNSW

Dr Ying Zhang
CSE, UNSW

Dr Gaoping Zhu
CSE, UNSW

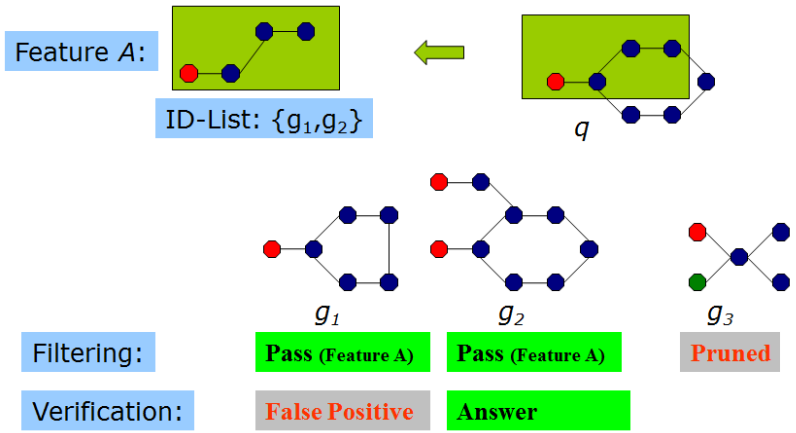
Xiang Zhao
CSE, UNSW

Outline

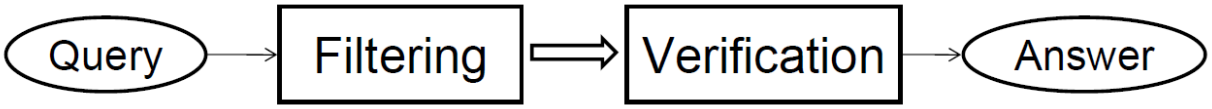
- Substructure Search (VLDB'08)
- Substructure Similarity Search (SIGMOD'10)
- Superstructure Search (SSDBM'10)
- Superstructure Similarity Search (ICDE'10)
- Graph Similarity Join (ICDE'12)
- Similarity All-Matching (SIGMOD'12)
- SimFusion+ Similarity Search (SIGIR'12)
- SimRank Similarity Search (ICDE'13)
- Conclusions and Open Issues

1. Subgraph Structure Search (VLDB'08)

- **Prior Work**
  - gIndex [SIGMOD'04]
  - Tree+Delta [VLDB'07]
  - FG-index [SIGMOD'07]



- **Motivation**
  - Most existing works adopt the "filtering-and-verification" framework



— Filtering: filter false answers by an index and produce a candidate set C

— Verification: verify if $q \subseteq g$, for each $g \in C$ (by sub-Iso test)

  - Verification cost dominates due to sub-Iso

1. Subgraph Structure Search (VLDB'08)

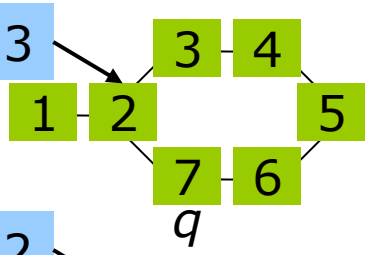
- **Our Goal**
  - Develop an efficient verification algorithm, ***QuickSI***,
    to accelerate both verification and filtering by orders of magnitude.

- **Main Idea of *QuickSI***
  - Encode query graphs: terminate earlier
  - Enforce connectivity
  - Three novel pruning techniques

Example: *QuickSI*

Depth First Traversal

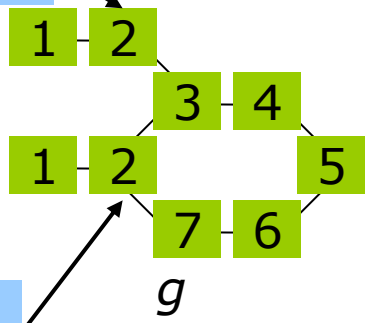
Deg=3



q

Stop here

Deg=2



g

Continue

Deg=3

Access infrequent labels as early as possible

Retain connectivity

Effectively use degree information

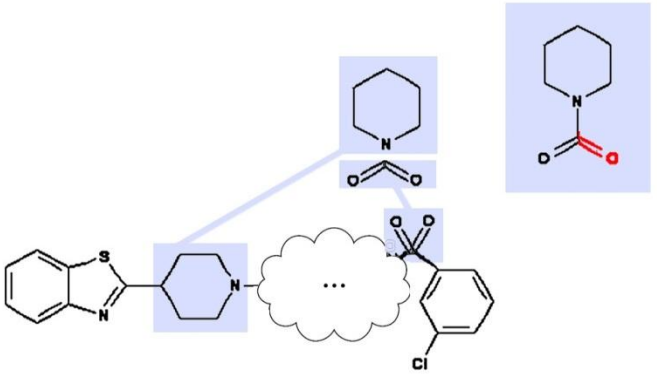
Determine the access order for *q*.

Outline

- Substructure Search (VLDB'08)
- Substructure Similarity Search (SIGMOD'10)
- Superstructure Search (SSDBM'10)
- Superstructure Similarity Search (ICDE'10)
- Graph Similarity Join (ICDE'12)
- Similarity All-Matching (SIGMOD'12)
- SimFusion+ Similarity Search (SIGIR'12)
- SimRank Similarity Search (ICDE'13)
- Conclusions and Open Issues

2. Substructure Similarity Search (SIGMOD' 10)

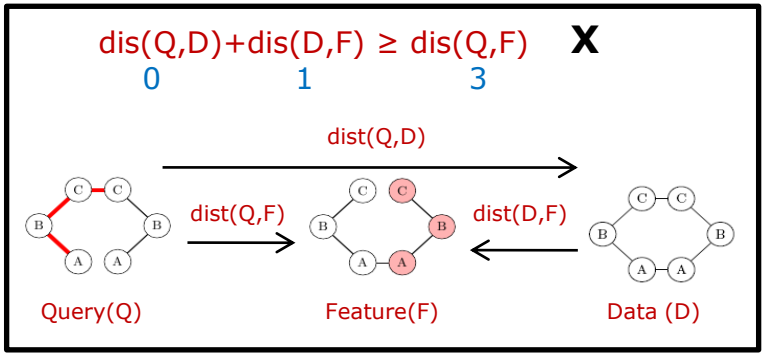
- **Problem Definition**
  - mccs(g1, g2) is the connected common graph of g1 and g2 with the maximal #-edges
  - dis (q, g) := |q| - |mccs(q, g)|
  - The goal is to find all g in D s.t. dis(q,g) ≤σ.



- **Prior Work**
  - Only focus on removing non-promising graphs via "filtering-and-verification"
  - Unable to remove many "low" quality candidates [SIGMOD'07]
- **Challenges**
  - NP-Complete

2. Substructure Similarity Search (SIGMOD' 10)

- **Our Goal**
  - Propose GrafD-Index, to index graph based on their "distances" to features
- **Observation**
  - Triangular inequality does NOT always hold
- **Main Idea**
  - Use *Connectivity Dominance* to safeguard triangular inequality :



Theorem: If the connectivity of mccs(g_1, g_2) or mccs(g_2, g_3) dominates g_2, then dist(g_1, g_3) ≤ dist(g_1, g_2) + dist(g_2, g_3).

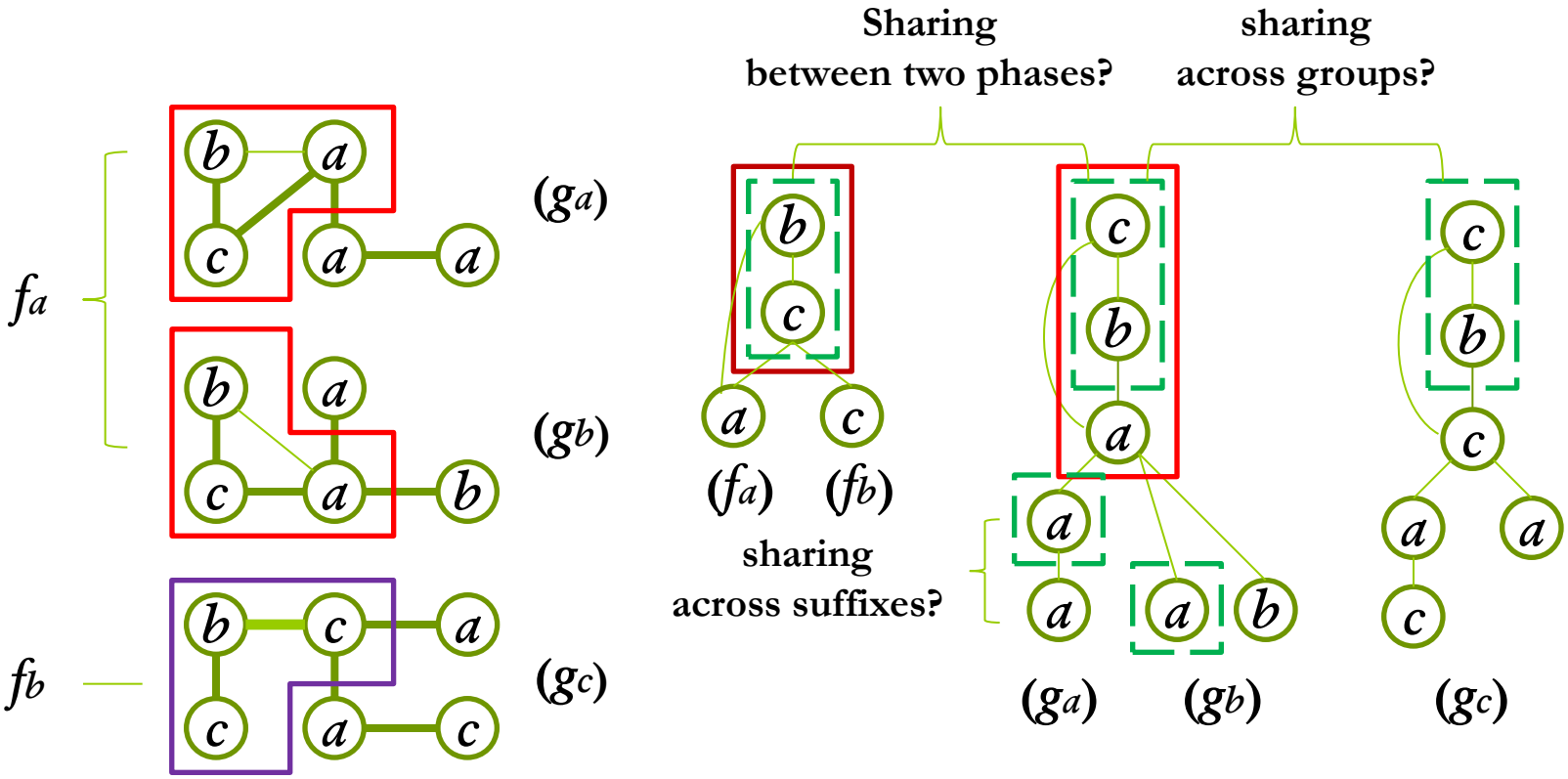
- Exploit GrafD-Index to (1) prune non-promising graphs; (2) include graphs whose similarities are guaranteed to exceed the given similarity threshold.

Outline

- Substructure Search (VLDB'08)
- Substructure Similarity Search (SIGMOD'10)
- Superstructure Search (SSDBM'10)
- Superstructure Similarity Search (ICDE'10)
- Graph Similarity Join (ICDE'12)
- Similarity All-Matching (SIGMOD'12)
- SimFusion+ Similarity Search (SIGIR'12)
- SimRank Similarity Search (ICDE'13)
- Conclusions and Open Issues

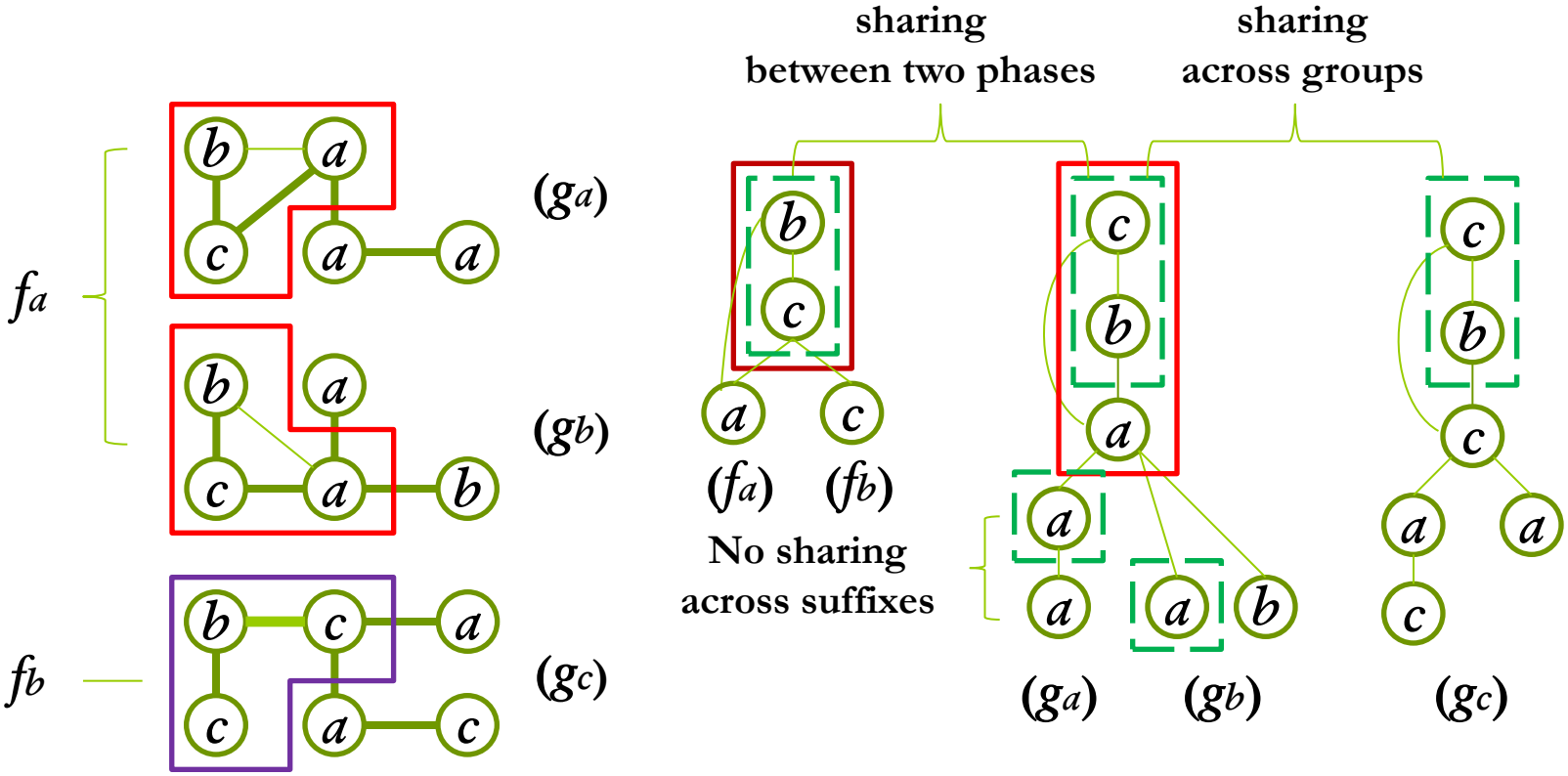
3. Superstructure Search (SSDBM' 10)

- **Prior Work [EDBT'09] GPTree**
  - **Enhanced** Filtering-Verification Framework
  - Share test cost in filtering and verification, respectively.



3. Superstructure Search (SSDBM' 10)

- **Our Work [SSDBM'10]**
  - **Sharing-based** Filtering-Verification Framework
  - Share test cost in filtering and verification, respectively.



3. Superstructure Search (SSDBM' 10)

- **Cost Gain (Computation Sharing)**
  - Given k master groups of data graphs, assume that

    1) all data graphs in each group G_i contain a master feature f_i ($1 \leq i \leq k$);

    2) the subgraph isomorphism test from f_i to a query graph q is cost f_i;
  - The total cost gain from each master group G_i can be represented as

$$gain_{total} = \sum_{i=1}^{k} (|G_i| - 1) cost_{f_i}$$

- **Maximized Gain**
  - Cluster the database into a disjoint set of master groups to maximize the total gain (NP-hardness).

Outline

- Substructure Search (VLDB'08)
- Substructure Similarity Search (SIGMOD'10)
- Superstructure Search (SSDBM'10)
- Superstructure Similarity Search (ICDE'10)
- Graph Similarity Join (ICDE'12)
- Similarity All-Matching (SIGMOD'12)
- SimFusion+ Similarity Search (SIGIR'12)
- SimRank Similarity Search (ICDE'13)
- Conclusions and Open Issues

4. Superstructure Similarity Search (ICDE' 10)

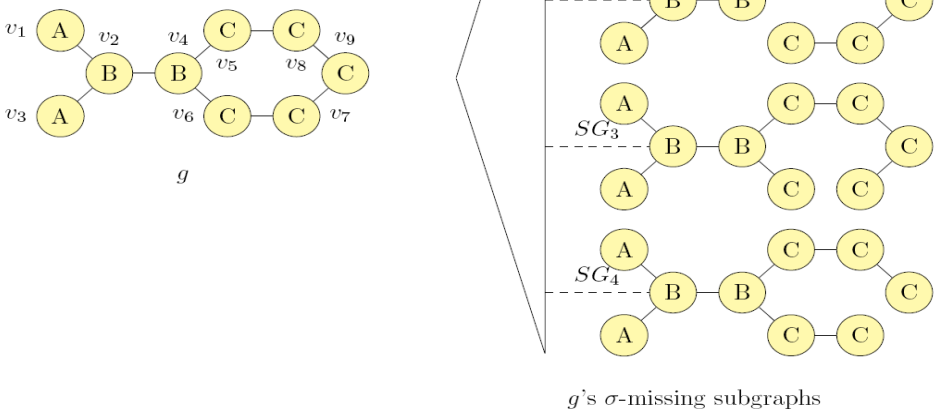
- **Problem Definition**
  - dis(q, g) := |g| − |mccs(q, g)|.
  - The goal is to find all g from D s.t. dis(q, g) ≤ σ.

  Note: dis(q, g) = |q| − |mccs(q, g)| in substructure similarity search

- **Observations**
  - Filtering framework in SIGMOD10 is immediately applicable.
  - Techniques in SIGMOD10 may not be effective for a nearly "super-containment" relationship.
  - Sharing is possible just like PrefIndex.

4. Superstructure Similarity Search (ICDE' 10)

- **Main Idea: *SG-Enum Index***
  - For any g, enumerate all subgraphs with at most σ edges removed, σ-missing subgraphs
  - dis(q, g) ≤σ if and only if q contains a σ-missing subgraph



Key Issues:

- Automorphic subgraphs?
- Prefix-sharing?
  - in one g
  - among different data graphs
- Query processing?

4. Superstructure Similarity Search (ICDE' 10)

- **Main Idea: *SG-Enum Index***

  Top-down Construction:

  - Enumerate all σ-missing subgraphs.
  - Iteratively, choose an edge as follows:
    - — Always select an edge contained by most σ-missing subgraphs.
    - — Split the group into 2: one contains the edge; the other not.

  Bottom-up Construction:

  - Generate a sequence for each σ-missing subgraph.
  - Merge the prefixes by chance.

  Bottom-up among Data Graphs

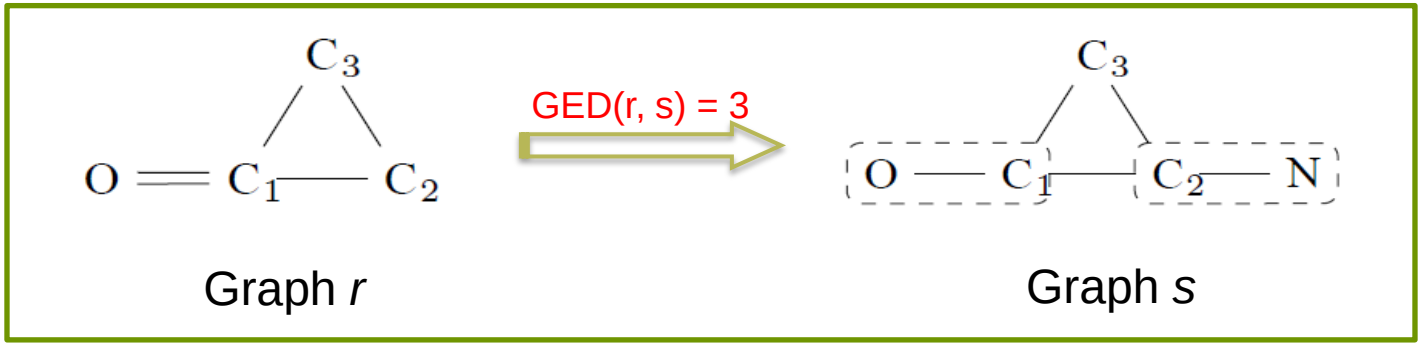
  Query Algorithm: extends ***QuickSI***

Outline

- Substructure Search (VLDB'08)
- Substructure Similarity Search (SIGMOD'10)
- Superstructure Search (SSDBM'10)
- Superstructure Similarity Search (ICDE'10)
- 
  Graph Similarity Join (ICDE'12)
- Similarity All-Matching (SIGMOD'12)
- SimFusion+ Similarity Search (SIGIR'12)
- SimRank Similarity Search (ICDE'13)
- Conclusions and Open Issues

5. Graph Similarity Join (ICDE' 12)

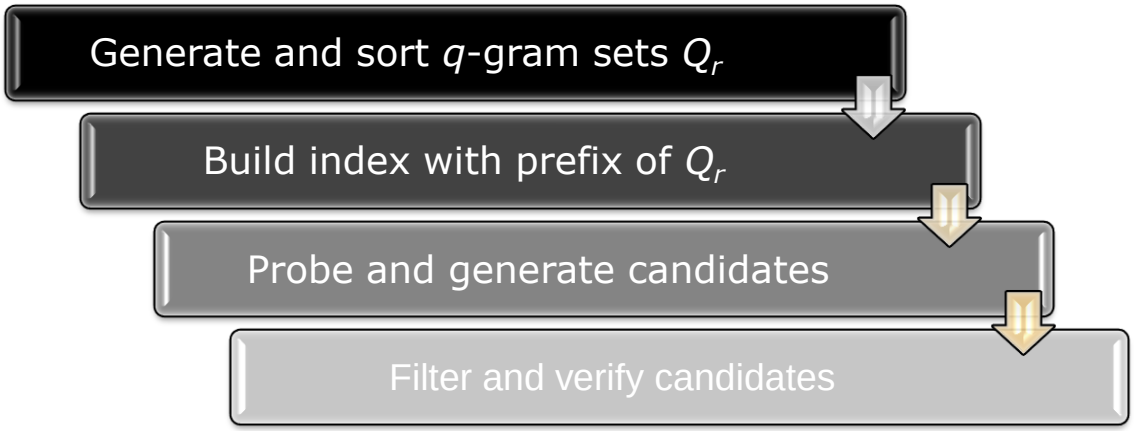
- **Problem Definition**
  - The *graph edit distance* between graphs r and s, denoted by GED(r, s), is the minimum number of edit operations that transform r to a graph isomorphic to s.

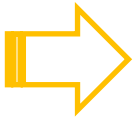


  - A graph similarity join returns pairs of graphs such that their edit distance is no larger than a threshold τ.

5. Graph Similarity Join (ICDE' 12)

- **Observation**
  - Frequent q-grams are shared by many graphs, resulting in repetitive probes of inverted index
- **Main Idea: GSimJoin**:
  - Batch join in filtering-verification framework





Minimum prefix & Minimum edit filtering

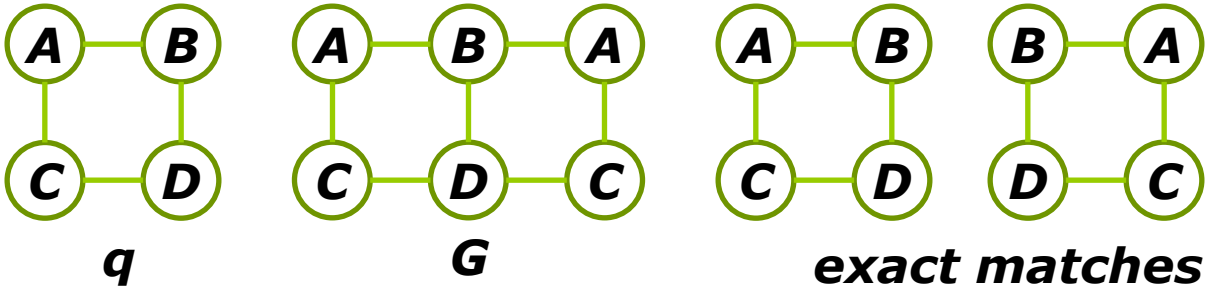
Outline

- Substructure Search (VLDB'08)
- Substructure Similarity Search (SIGMOD'10)
- Superstructure Search (SSDBM'10)
- Superstructure Similarity Search (ICDE'10)
- Graph Similarity Join (ICDE'12)
- Similarity All-Matching (SIGMOD'12)
- SimFusion+ Similarity Search (SIGIR'12)
- SimRank Similarity Search (ICDE'13)
- Conclusions and Open Issues

6. Similarity All-Matching (SIGMOD' 12)

Introduction — Exact All-Matching (I)

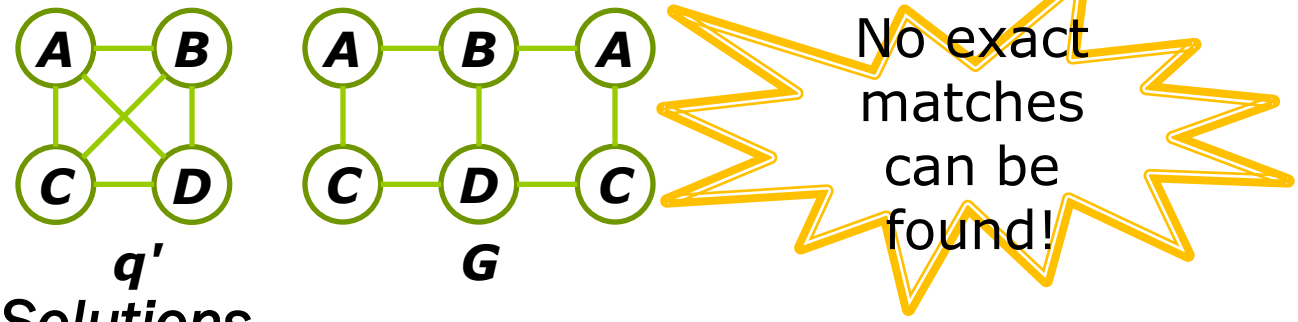
- ***Exact All-Matching***
  - Enumerate all **exact (i.e. isomorphic) matches** of a query graph ***q*** in a data graph ***G***.



- ***Applications***
  - Query biological patterns in PPI networks.
  - Detect suspicious bugs in software programs.

Introduction — Exact All-Matching (II)

- ***Dilemma of Exact All-Matching***
- If ***q*** is issued by user for exploratory purpose …
- If ***G*** is noisy due to imprecise data collection …



- ***Potential Solutions***
- Modify ***q***/***G*** and run exact all-matching again and again.
- Ask system to return approximate results (i.e., similarity all-matching)

SAPPER [VLDB'10 Zhang et al] (I)

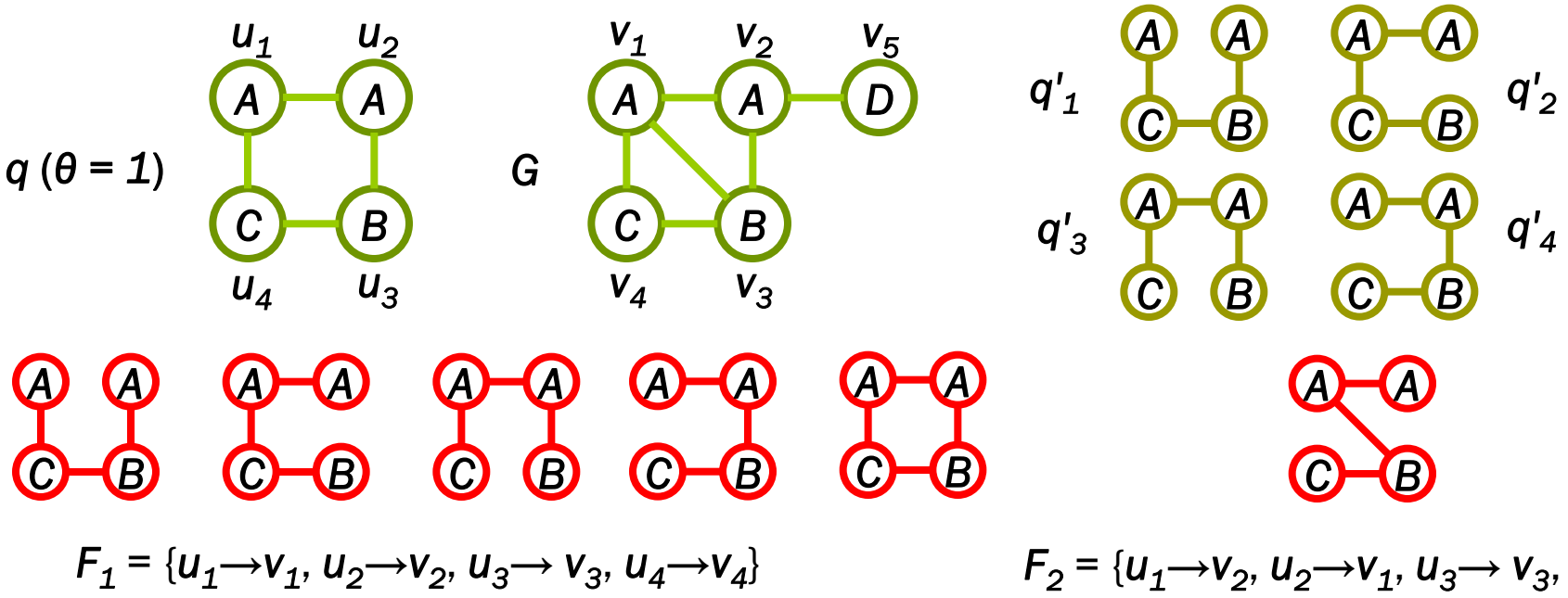
❖ ***Similarity All-Matching***

- Given a query graph q, a data graph G and a similarity threshold θ, enumerate all similarity matches of q in G (i.e., all **connected** subgraphs of G missing at most θ edges in q).

❖ ***Framework***

- Enumerate a set of seeds Q_{SAPPER} (i.e., all connected subgraphs q' of q missing θ edges in q).
- Exact all-matching on each seed q' to obtain exact matches.
- Induce similarity matches based on exact matches of seeds.

SAPPER [VLDB'10 Zhang et al] (II)



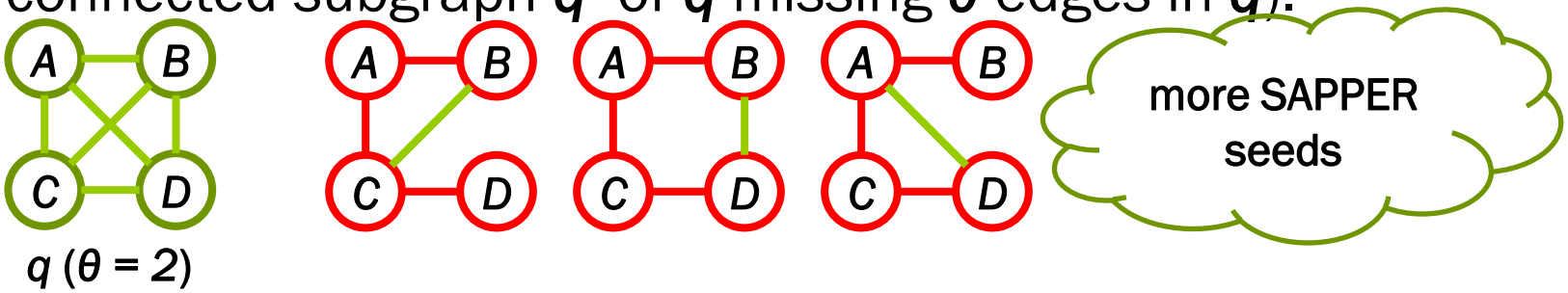
$F_1 = \{u_1 \rightarrow v_1, u_2 \rightarrow v_2, u_3 \rightarrow v_3, u_4 \rightarrow v_4\}$

$F_2 = \{u_1 \rightarrow v_2, u_2 \rightarrow v_1, u_3 \rightarrow v_3, u_4 \rightarrow v_4\}$

- ***Cost Model***
- $C_{total} = C_{seeding} + \sum_{q' \in Q_{SAPPER}} C_{all-matching}(q', G) + C_{inducing}$
- $|Q_{SAPPER}|$ = # of exact all-matching tests

Our Approach — Overview (I)

- ***Tree-based Spanning Search Paradigm — TSpan***
- Enumerate a set of seeds Q_T (i.e., spanning trees of q cover all connected subgraph q' of q missing θ edges in q).



q ($\theta = 2$)

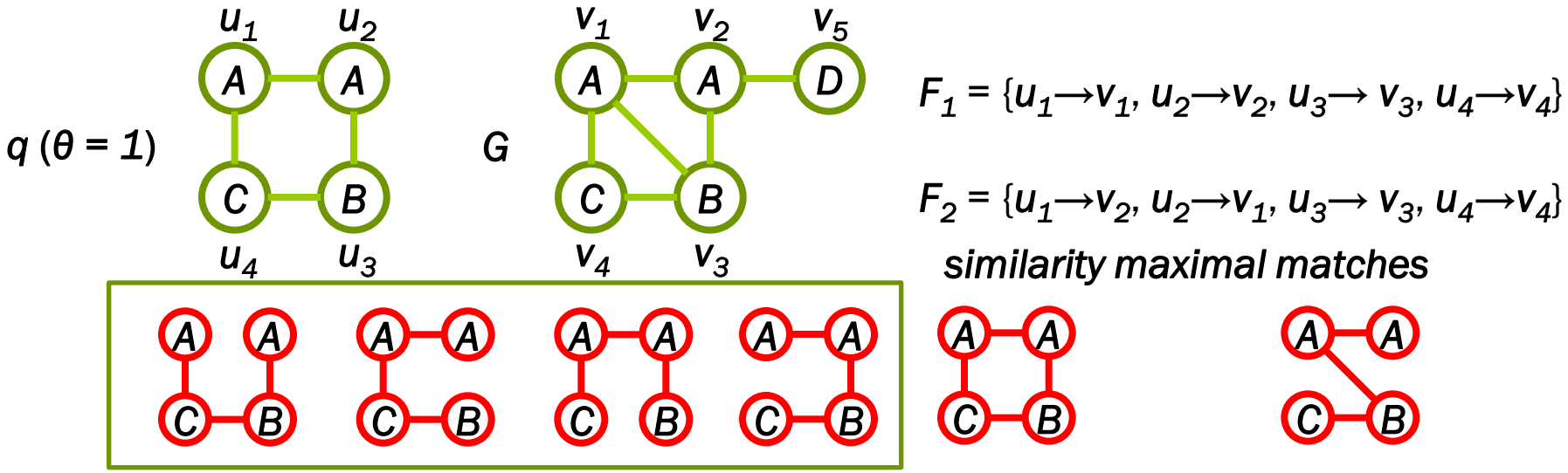
3 all-matching tests on connected subgraphs of q

1 all-matching tests on a spanning tree of q

- ***Primary Contribution***
- Reduce # of exact all-matching tests (i.e., # of seeds).
- Reduce the complexity of exact all-matching test from **graph to graph** to **tree to graph**.

Our Approach — Overview (II)

- *Generating Similarity Maximal Matches*



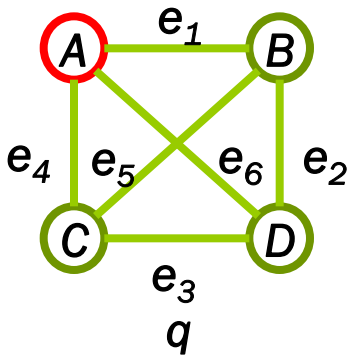
- Generating similarity maximal matches only can reduce # of exact all-matching tests.

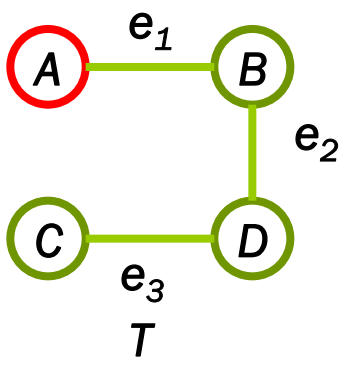
Our Approach — Problem Statement

- ***Similarity Maximal All-Matching***
- Given a query graph q, a data graph G and a similarity threshold θ, enumerate all distinct similarity maximal matches of q in G conforming θ.

Our Approach — Seeding (I)

- ***PRIM Order on Spanning Trees***
- Similar to the basic idea of minimum spanning tree.
- Given a total order on $E(q)$, a spanning tree $T = \{T[0], T[1], \ldots, T[|V(q)|-1]\}$ of q conforms PRIM order ($T[0]$ is head vertex), if and only if each spanning edge $T[i]$ has the smallest order in $E(q) - \{T[1], \ldots, T[i-1]\}$ and connects $\{T[0], T[1], \ldots, T[i-1]\}$.

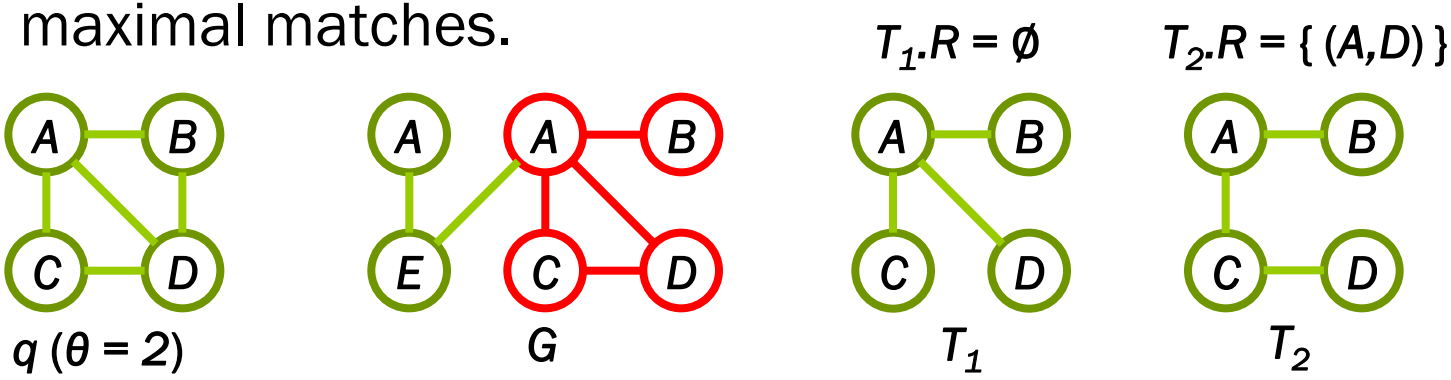




Our Approach — Seeding (II)

❖ ***Avoid Duplicate Results***

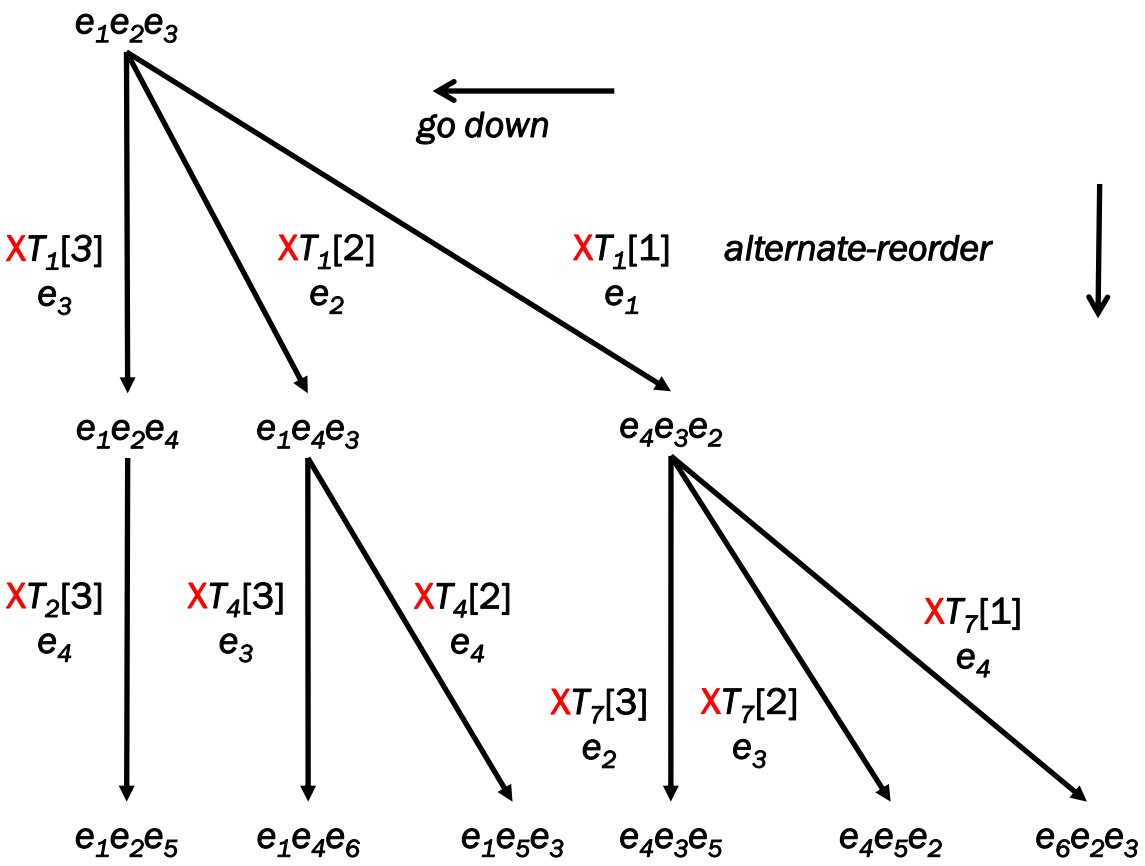
- Two spanning trees of ***q*** may induce duplicate similarity maximal matches.

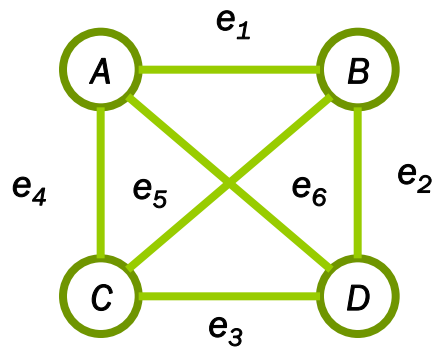


- Associate an edge exclusion set ***T.R*** to each ***T*** in Q_T.
- ***T.R*** is a set of edges in ***E***(***q***) – ***E***(***T***) enforced to be mismatched in the similarity maximal matches induced by ***T***.

Our Approach — Seeding (III)

- Q_T *Enumeration Algorithm*





q (θ =2)

T	T.R	T	T.R
1.$e_1e_2e_3$	{ }	6.$e_1e_5e_3$	{e_2,e_4 }
2.$e_1e_2e_4$	{e_3 }	7.$e_4e_3e_2$	{e_1 }
3.$e_1e_2e_5$	{e_3,e_4 }	8.$e_4e_3e_5$	
4.$e_1e_4e_3$	{e_2 }	9.$e_4e_5e_2$	{e_1,e_3 }
5.$e_1e_4e_6$	{e_2,e_3 }	10.$e_6e_2e_3$	{e_1,e_4}

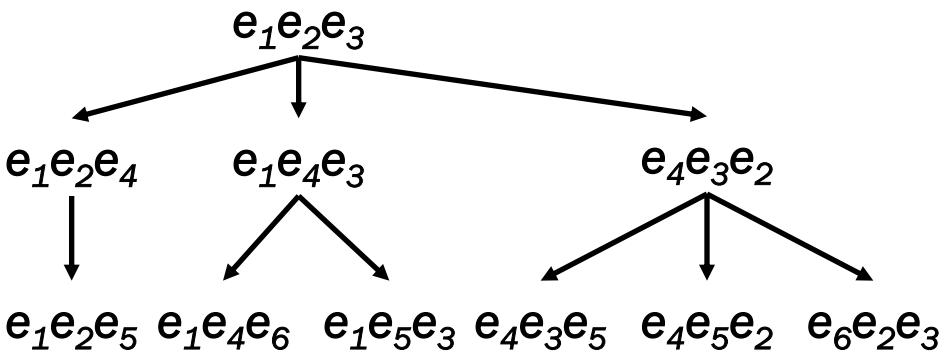
Our Approach — Seeding (IV)

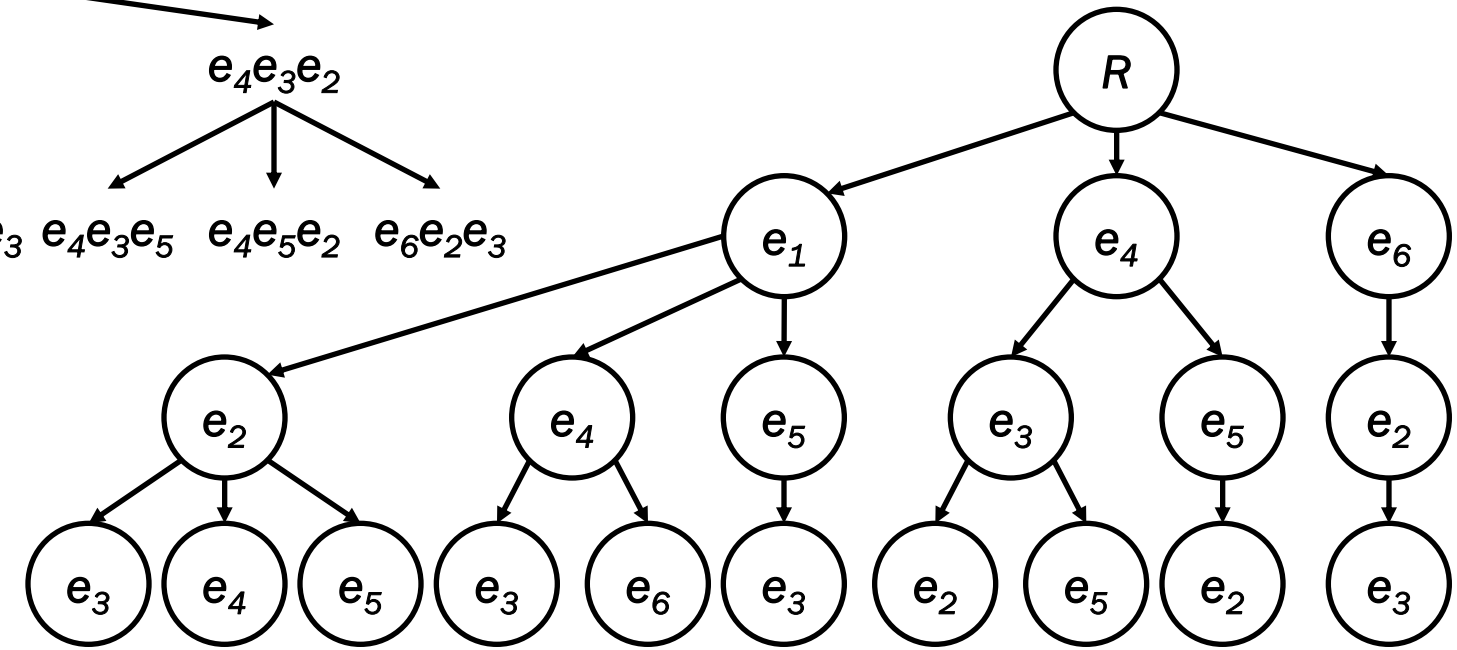
- ***Q_T Enumeration Algorithm***
- ***Correctness*** : Using Q_T to inducing similarity maximal matches neither generates duplicate results nor misses valid results.
- ***Minimality of Q_T*** : Missing any spanning tree in Q_T does not guarantee the completeness of results based on edge exclusion semantics.
- When $|E(q)| = m$, $|V(q)| = n$,
- (1) $|Q_{SAPPER}| \geq |Q_T|$;
- (2) $|Q_T| = |Q_{SAPPER}|$ only when $\theta = 0$ or $m - n + 1$.

$$|Q_{SAPPER}| \geq \sum_{T \in Q_T} \binom{m-n+1-|T.R|}{\theta-|T.R|}$$

Our Approach — Searching (I)

- ***Effectively Storing Q_T***
- Use DFS Traversal Tree to share computation cost.





Our Approach — Searching (II)

- ***Similarity Maximal All-Matching Algorithm Sketch***
- Traverse the DFS Traversal Tree in a depth-first backtrack search fashion.
- **go-down** : Beginning from the initial spanning tree, recursively drill down to extend the current partial match to the next spanning edge $T[i]$ in the current spanning tree T.
- **alternate** : If $T[i]$ can not be extended based on the current partial match and we can still afford to mismatch $T[i]$ by conforming θ, alternate the algorithm from T to the alternative spanning tree T' enumerated by replacing $T[i]$ with $T'[i]$.

Our Approach — Optimizations

- ***Optimizations (I) EnumrateOnDemand Strategy***
- Motivation : further reduce the number of seeds.
- Enumerate an alternative tree ***T'*** based on the current tree ***T*** only when it is feasible to extend the current partial similarity maximal match conforming ***θ*** (1) on the next spanning edge ***T***[***i***] or (2) on the next spanning edge ***T***[***i***]'.

- ***Optimizations (II) Effective Search Order***
- Motivation : terminate all-matching test as early as possible.
- Decide the search order of spanning edges in ***T*** based on the post-filtering candidate sets of each vertex in ***q***.

Outline

- Substructure Search (VLDB'08)
- Substructure Similarity Search (SIGMOD'10)
- Superstructure Search (SSDBM'10)
- Superstructure Similarity Search (ICDE'10)
- Graph Similarity Join (ICDE'12)
- Similarity All-Matching (SIGMOD'12)
- SimFusion+ Similarity Search (SIGIR'12)
- SimRank Similarity Search (ICDE'13)
- Conclusions and Open Issues

7. SimFusion+ Similarity Search (SIGIR' 12)

- A structural similarity measure based on hyperlinks
- Basic Philosophy of SimFusion [Xi et. al, SIGIR 05]
  - *The similarity between two objects is reinforced by the similarity of their related objects.*
- Prior Work
  - trivial solution or divergence issue
  - rather costly to compute SimFusion on large graphs
    - naïve Iteration: matrix-matrix multiplication
    - complexity: $O(Kn^3)$ time, $O(n^2)$ space
  - no incremental algorithms when edges update

Problem Definition

- *Data Space* (vertices) $\mathcal{D} = \{o_1, o_2, \cdots\}$ is a finite set of data objects
- *Data Relation* (edges) Given an entire space $\mathcal{D} = \bigcup_{i=1}^{N} \mathcal{D}_i$
  - Intra-type Relation $\mathcal{R}_{i,i} \subseteq \mathcal{D}_i \times \mathcal{D}_i$ carries info. within one space
  - Inter-type Relation $\mathcal{R}_{i,j} \subseteq \mathcal{D}_i \times \mathcal{D}_j$ carries info. between spaces
- *Unified Relationship Matrix* (URM):

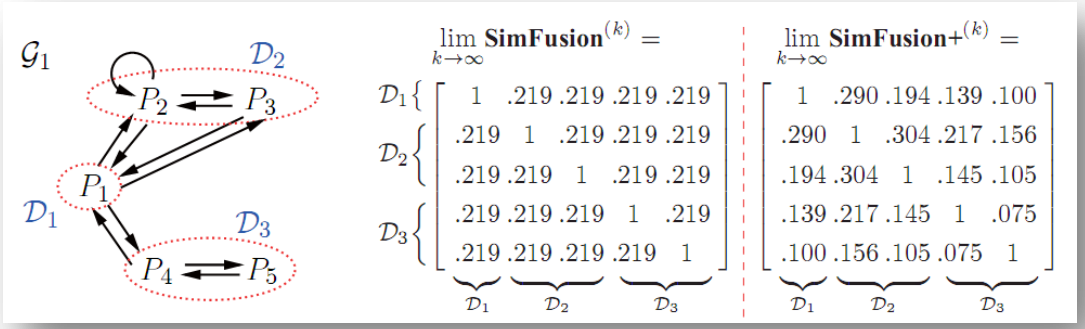
$$\mathbf{L}_{\text{URM}} = \begin{pmatrix} \lambda_{1,1}\mathbf{L}_{1,1} & \lambda_{1,2}\mathbf{L}_{1,2} & \cdots & \lambda_{1,N}\mathbf{L}_{1,N} \\ \lambda_{2,1}\mathbf{L}_{2,1} & \lambda_{2,2}\mathbf{L}_{2,2} & \cdots & \lambda_{2,N}\mathbf{L}_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{N,1}\mathbf{L}_{N,1} & \lambda_{N,2}\mathbf{L}_{N,2} & \cdots & \lambda_{N,N}\mathbf{L}_{N,N} \end{pmatrix} \qquad \mathbf{L}_{i,j}(x,y) = \begin{cases} \frac{1}{n_j}, & \text{if } \mathcal{N}_j(x) = \varnothing; \\ \frac{1}{|\mathcal{N}_j(x)|}, & \text{if } (x,y) \in \mathcal{R}_{i,j}; \\ 0, & \text{otherwise.} \end{cases}$$

  - $\lambda_{i,j}$ is the *weighting factor* between D_i and D_j
- *Unified Similarity Matrix* (USM):

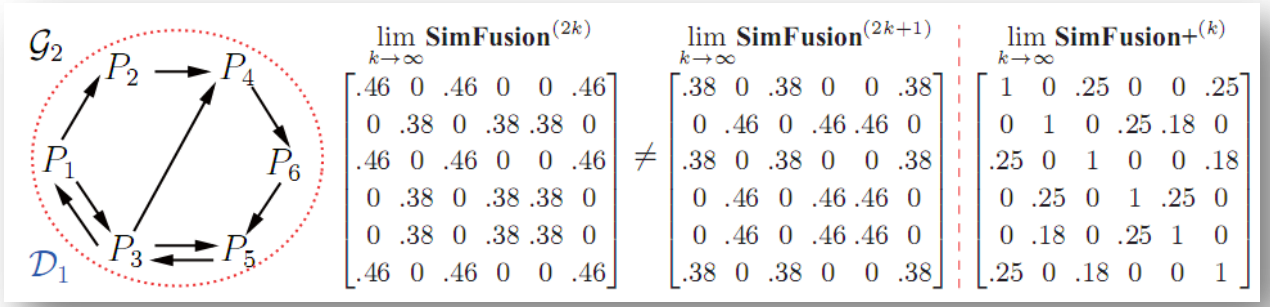
SimFusion similarity

$$\exists\, \mathbf{S} = \begin{pmatrix} s_{1,1} & \cdots & s_{1,n} \\ \vdots & \ddots & \vdots \\ s_{n,1} & \cdots & s_{n,n} \end{pmatrix} \in \mathbb{R}^{n \times n} \quad s.t. \quad \mathbf{S} = \mathbf{L} \cdot \mathbf{S} \cdot \mathbf{L}^T.$$

- Motivation: Two issues of the existing SimFusion model
  - Trivial Solution on Heterogeneous Domain



  - Divergent Solution on Homogeneous Domain



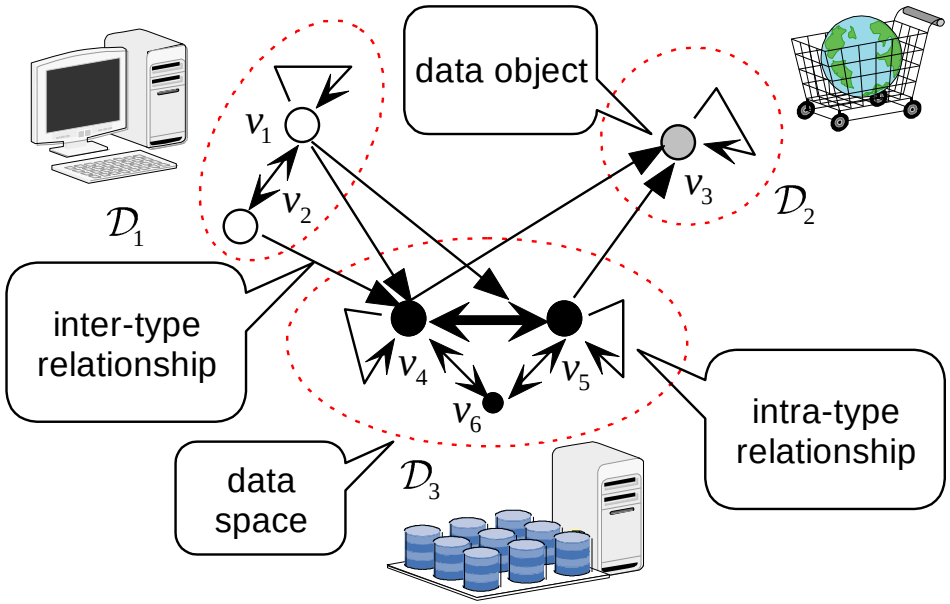
Example:

The root cause of trivial solution:

Row Normalization of URM !!!

Trivial Solution !!!

$S=[1]_{n\times n}$



$$\mathbf{L}_{\text{URM}} = \begin{pmatrix} \frac{1}{8} & \frac{1}{8} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{2} & 0 & 0 \\ \frac{1}{16} & \frac{1}{16} & \frac{1}{4} & \frac{5}{24} & \frac{5}{24} & \frac{5}{24} \\ \frac{1}{10} & \frac{1}{10} & \frac{3}{5} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} \\ \frac{1}{10} & \frac{1}{10} & \frac{3}{5} & \frac{1}{15} & \frac{1}{15} & \frac{1}{15} \\ \frac{1}{10} & \frac{1}{10} & \frac{3}{5} & \frac{1}{10} & \frac{1}{10} & 0 \end{pmatrix}$$

$$\exists \mathbf{S} \in \mathbb{R}^{n\times n} \quad s.t. \quad \mathbf{S} = \mathbf{L}\cdot\mathbf{S}\cdot\mathbf{L}^T.$$

$$\mathbf{\Lambda} = \begin{array}{c} \\ \mathcal{D}_1 \\ \mathcal{D}_2 \\ \mathcal{D}_3 \end{array} \begin{array}{c} \begin{array}{ccc} \mathcal{D}_1 & \mathcal{D}_2 & \mathcal{D}_3 \end{array} \\ \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{8} & \frac{1}{4} & \frac{5}{8} \\ \frac{1}{5} & \frac{3}{5} & \frac{1}{5} \end{pmatrix} \end{array}$$

$$\mathbf{S}_{\text{USM}} = \begin{pmatrix} 1 & s_{1,2} & \cdots & s_{1,n} \\ s_{2,1} & 1 & \cdots & s_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n,1} & s_{n,2} & \cdots & 1 \end{pmatrix}$$

High complexity !!!

$O(Kn^3)$ time

$O(n^2)$ space

Our Contributions

- Revise the existing SimFusion model, avoiding
  - non-semantic convergence
  - divergence issue
- Optimize the computation of SimFusion+
  - O(Km) pre-computation time, plus O(1) time and O(n) space
  - Better accuracy guarantee
- Incremental computation on evolving graphs
  - O(δn) time and O(n) space for updating δ edges

SimFusion+: A Revision of SimFusion

Main Idea

- Use *Unified Adjacency Matrix* (UAM) $\mathbf{A} = \tilde{\mathbf{A}} + 1/n^2$

$$\tilde{\mathbf{A}} = \begin{pmatrix} \lambda_{1,1}\mathbf{A}_{1,1} & \lambda_{1,2}\mathbf{A}_{1,2} & \cdots & \lambda_{1,N}\mathbf{A}_{1,N} \\ \lambda_{2,1}\mathbf{A}_{2,1} & \lambda_{2,2}\mathbf{A}_{2,2} & \cdots & \lambda_{2,N}\mathbf{A}_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{N,1}\mathbf{A}_{N,1} & \lambda_{N,2}\mathbf{A}_{N,2} & \cdots & \lambda_{N,N}\mathbf{A}_{N,N} \end{pmatrix}, \quad \mathbf{A}_{i,j}(x,y) = \begin{cases} \frac{1}{n_j}, & \text{if } \mathcal{N}_j(x) = \varnothing; \\ 1, & \text{if } (x,y) \in \mathcal{R}_{i,j}; \\ 0, & \text{otherwise.} \end{cases}$$

- Replace URM with UAM to postpone "row normalization"
- Revised SimFusion+ Model Original SimFusion

$$\mathbf{S} = \frac{\mathbf{A} \cdot \mathbf{S} \cdot \mathbf{A}^T}{\|\mathbf{A} \cdot \mathbf{S} \cdot \mathbf{A}^T\|_2}, \qquad \mathbf{S} = \mathbf{L} \cdot \mathbf{S} \cdot \mathbf{L}^T$$

squeeze similarity scores in S into [0, 1].

Optimizing SimFusion+ Computation

- Conventional Iterative Paradigm

$$\mathbf{S}^{(k+1)} = \frac{\mathbf{A} \cdot \mathbf{S}^{(k)} \cdot \mathbf{A}^T}{\|\mathbf{A} \cdot \mathbf{S}^{(k)} \cdot \mathbf{A}^T\|_2}.$$

  - Matrix-matrix multiplication, requiring $O(kn^3)$ time and $O(n^2)$ space

- Our approach: Convert SimFusion+ computation into finding the dominant eigenvector of the UAM A.

$$[\mathbf{S}]_{i,j} = [\sigma_{\max}(\mathbf{A})]_i \times [\sigma_{\max}(\mathbf{A})]_j$$

Pre-compute σ_{max}(A) only once, and cache it for later reuse

  - Matrix-vector multiplication, requiring $O(km)$ time and $O(n)$ space

Main Idea Behind Optimization

- Two Properties of Kroneckor product

P1. $vec(\mathbf{BCD}^T) = (\mathbf{D} \otimes \mathbf{B}) \cdot vec(\mathbf{C})$

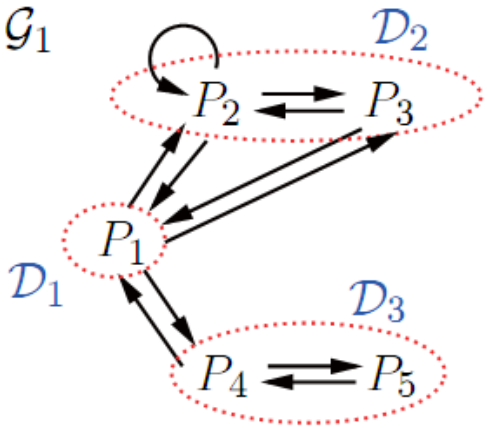
P2. $\sigma_{\max}(\mathbf{A} \otimes \mathbf{A}) = \sigma_{\max}(\mathbf{A}) \otimes \sigma_{\max}(\mathbf{A})$

- Main idea:

$$\mathbf{S}^{(k+1)} = \frac{\mathbf{A} \cdot \mathbf{S}^{(k)} \cdot \mathbf{A}^T}{\|\mathbf{A} \cdot \mathbf{S}^{(k)} \cdot \mathbf{A}^T\|_2}. \quad \xrightarrow{(1)} \quad vec(\mathbf{S}^{(k+1)}) = \frac{(\mathbf{A} \otimes \mathbf{A}) \cdot vec(\mathbf{S}^{(k)})}{\|(\mathbf{A} \otimes \mathbf{A}) \cdot vec(\mathbf{S}^{(k)})\|_2}$$

Power Iteration

$$vec(\mathbf{S}) = \sigma_{\max}(\mathbf{A}) \otimes \sigma_{\max}(\mathbf{A}) \quad \xleftarrow{(2)} \quad \lim_{k \to \infty} vec(\mathbf{S}^{(k)}) = \sigma_{\max}(\mathbf{A} \otimes \mathbf{A})$$



Let $\mathbf{A} = \tilde{\mathbf{A}} + 1/5^2$ with

$$\tilde{\mathbf{A}} = \begin{bmatrix} \frac{1}{2}[1] & \frac{1}{6}[1\ 1] & \frac{1}{3}[1\ 0] \\ \frac{1}{6}\begin{bmatrix}1\\1\end{bmatrix} & \frac{7}{12}\begin{bmatrix}1&1\\1&0\end{bmatrix} & \frac{1}{4}\begin{bmatrix}\frac{1}{2}&\frac{1}{2}\\\frac{1}{2}&\frac{1}{2}\end{bmatrix} \\ \frac{1}{3}\begin{bmatrix}1\\0\end{bmatrix} & \frac{1}{4}\begin{bmatrix}\frac{1}{2}&\frac{1}{2}\\\frac{1}{2}&\frac{1}{2}\end{bmatrix} & \frac{5}{12}\begin{bmatrix}0&1\\1&0\end{bmatrix} \end{bmatrix} = \left[\begin{array}{c|cc|cc} \frac{1}{2} & \frac{1}{6} & \frac{1}{6} & \frac{1}{3} & 0 \\ \hline \frac{1}{6} & \frac{7}{12} & \frac{7}{12} & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{6} & \frac{7}{12} & 0 & \frac{1}{8} & \frac{1}{8} \\ \hline \frac{1}{3} & \frac{1}{8} & \frac{1}{8} & 0 & \frac{5}{12} \\ 0 & \frac{1}{8} & \frac{1}{8} & \frac{5}{12} & 0 \end{array}\right]$$

- Conventional iteration:

$$\mathbf{S}^{(0)} = \mathbf{1} \qquad \mathbf{S}^{(k+1)} = \frac{\mathbf{A}\cdot\mathbf{S}^{(k)}\cdot\mathbf{A}^T}{\|\mathbf{A}\cdot\mathbf{S}^{(k)}\cdot\mathbf{A}^T\|_2}.$$

- Our approach:

$$\sigma_{\max}(\mathbf{A}) = [.431 \quad .673 \quad .451 \quad .322 \quad .232]^T$$

$$[\mathbf{S}]_{1,2} = [\sigma_{\max}(\mathbf{A})]_1 \times [\sigma_{\max}(\mathbf{A})]_2 = .431 \times .673 = .290.$$
$$[\mathbf{S}]_{1,3} = [\sigma_{\max}(\mathbf{A})]_1 \times [\sigma_{\max}(\mathbf{A})]_3 = .431 \times .451 = .194.$$

$$\mathbf{S} = \begin{bmatrix} .186 & .290 & .194 & .139 & .100 \\ .290 & .453 & .304 & .217 & .156 \\ .194 & .304 & .203 & .145 & .105 \\ .139 & .217 & .145 & .104 & .075 \\ .100 & .156 & .105 & .075 & .054 \end{bmatrix}$$

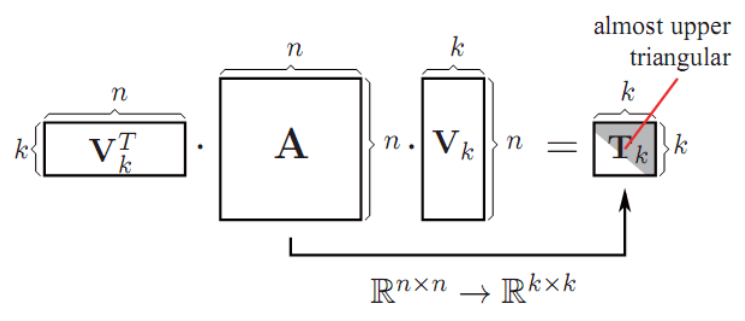
Accuracy Guarantee

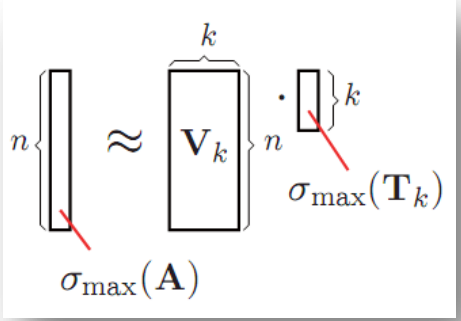
- Conventional Iterations: No accuracy guarantee !!!

$$\mathbf{S}^{(k+1)} = \frac{\mathbf{A}\cdot\mathbf{S}^{(k)}\cdot\mathbf{A}^T}{\|\mathbf{A}\cdot\mathbf{S}^{(k)}\cdot\mathbf{A}^T\|_2}. \qquad \mathbf{S} = \frac{\mathbf{A}\cdot\mathbf{S}\cdot\mathbf{A}^T}{\|\mathbf{A}\cdot\mathbf{S}\cdot\mathbf{A}^T\|_2}$$

Question: $\| S^{(k+1)} - S \| \leq ?$

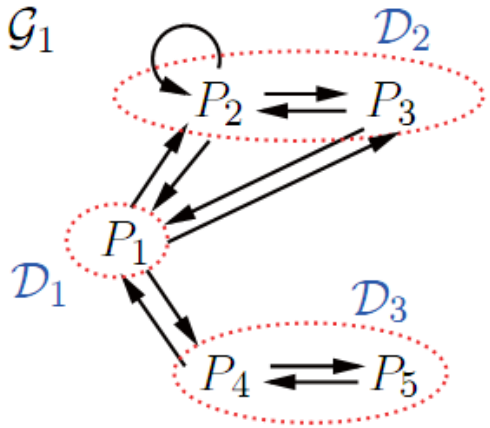
- Our Method: Utilize Arnoldi decomposition to build an order-k orthogonal subspace for the UAM A.





Due to T_k small size and almost "upper-triangularity", computing $\sigma_{max}(T_k)$ is less costly than $\sigma_{max}(A)$.

Example



Let $\mathbf{A} = \tilde{\mathbf{A}} + 1/5^2$ with

$$\tilde{\mathbf{A}} = \begin{bmatrix} \frac{1}{2} & \frac{1}{6} & \frac{1}{6} & \frac{1}{3} & 0 \\ \frac{1}{6} & \frac{7}{12} & \frac{7}{12} & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{6} & \frac{7}{12} & 0 & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{3} & \frac{1}{8} & \frac{1}{8} & 0 & \frac{5}{12} \\ 0 & \frac{1}{8} & \frac{1}{8} & \frac{5}{12} & 0 \end{bmatrix}$$

Given $\epsilon = 0.05$

$$\epsilon_k = 2 \times |\delta_k \times [\sigma_{\max}(\mathbf{T}_k)]_k|$$

(2)

k	$\mathbf{T}_k$	$\mathbf{v}_{k+1}$	δ_k	$\sigma_{\max}(\mathbf{T}_k)$	ϵ_k
0	–	$[.447\ .447\ .447\ .447\ .447]^T$	–	–	–
1	$[1.08]$	$[.125\ .750\ -.125\ -.125\ -.625]^T$	*.298*	$[1]$	*.596*
2	$\begin{bmatrix} 1.08 & .298 \\ .298 & .190 \end{bmatrix}$	$[-.089\ .044\ .710\ -.697\ .032]^T$	*.359*	$\begin{bmatrix} .957 \\ .290 \end{bmatrix}$	*.208*
3	$\begin{bmatrix} 1.08 & .298 & 0 \\ .298 & .190 & .359 \\ 0 & .359 & -.083 \end{bmatrix}$	$[-.881\ .329\ .137\ .280\ .135\ .231]^T$	*.231*	$\begin{bmatrix} .945 \\ .316 \\ .090 \end{bmatrix}$	*.041*

(1)

$$\mathbf{A}\mathbf{V}_k - \mathbf{V}_k\mathbf{T}_k = \delta_k \mathbf{v}_{k+1}\mathbf{e}_k^T,$$

$$\hat{\mathbf{S}}_3 = \begin{bmatrix} .206 & .301 & .203 & .146 & .103 \\ .301 & .440 & .296 & .213 & .151 \\ .203 & .296 & .199 & .143 & .102 \\ .146 & .213 & .143 & .102 & .073 \\ .103 & .151 & .102 & .073 & .051 \end{bmatrix}$$

(3)

$$[\hat{\mathbf{S}}_k]_{i,j} = [\mathbf{V}_k \cdot \sigma_{\max}(\mathbf{T}_k)]_i \times [\mathbf{V}_k \cdot \sigma_{\max}(\mathbf{T}_k)]_j$$

Edge Update on Dynamic Graphs

- Incremental UAM

  Given old G =(D,R) and a new G'=(D,R'), the incremental UAM is a list of edge updates s.t. $\bar{\mathbf{A}} = \mathbf{A}' - \mathbf{A}$

- Main idea

  To reuse $\bar{\mathbf{A}}$ and the eigen-pair (α_p, ξ_p) of the old A to compute $\mathbf{S}'$

  $\bar{\mathbf{A}}$ is a sparse matrix when the number δ of edge updates is small.

- Incrementally computing SimFusion+

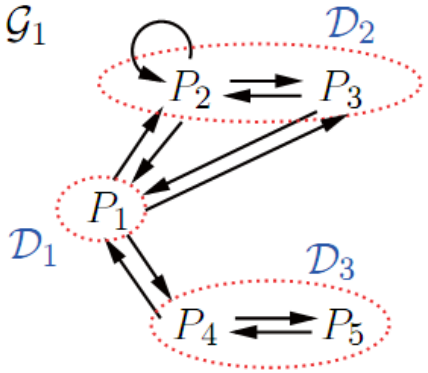
$$[\mathbf{S}']_{i,j} = [\boldsymbol{\xi}']_i \cdot [\boldsymbol{\xi}']_j \; with \; [\boldsymbol{\xi}']_i = [\boldsymbol{\xi}_1]_i + \sum_{p=2}^{n} c_p \times [\boldsymbol{\xi}_p]_i$$

$$c_p = \frac{\boldsymbol{\xi}_p^T \cdot \boldsymbol{\eta}}{\alpha_p - \alpha_1} \; and \; \boldsymbol{\eta} = \bar{\mathbf{A}} \cdot \boldsymbol{\xi}_1$$

O(δn) time

O(n) space

Example



Suppose edges (P1,P2) and (P2,P1) are removed.

$$\bar{\mathbf{A}} = \begin{bmatrix} 0 & -\frac{1}{6} & 0 & 0 & 0 \\ -\frac{1}{6} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

p	α_p	$\boldsymbol{\xi}_p$	c_p
1	*1.184*	$[.431\ .673\ .451\ .322\ .232]^T$	–
2	*.503*	$[.708\ -.522\ -.242\ .388\ .132]^T$	*.062*
3	*-.480*	$[-.256\ -.020\ .095\ .716\ -.641]^T$	*-.018*
4	*-.366*	$[-.021\ -.507\ .853\ -.119\ .017]^T$	*-.025*
5	*.242*	$[.497\ .127\ .037\ -.467\ -.719]^T$	*.069*

$$\boldsymbol{\eta} = \bar{\mathbf{A}} \cdot \boldsymbol{\xi}_1 = [-.112 \quad -.072 \quad 0 \quad 0 \quad 0]^T.$$

$$\begin{aligned} c_2 &= \boldsymbol{\xi}_2^T \cdot \boldsymbol{\eta}/(\alpha_2 - \alpha_1) = -.0419/(.503 - 1.184) = .062, \\ c_3 &= \boldsymbol{\xi}_3^T \cdot \boldsymbol{\eta}/(\alpha_3 - \alpha_1) = .030/(-.480 - 1.184) = -.018. \end{aligned}$$

$$\boldsymbol{\xi}' = \boldsymbol{\xi}_1 + \sum_{p=2}^{5} c_p \times \boldsymbol{\xi}_p = [.327\ .703\ .485\ .326\ .266]^T$$

$$\mathbf{S}' = \begin{bmatrix} .107 & .230 & .159 & .107 & .087 \\ .230 & .494 & .341 & .230 & .187 \\ .159 & .341 & .235 & .158 & .129 \\ .107 & .230 & .158 & .107 & .087 \\ .087 & .187 & .129 & .087 & .071 \end{bmatrix}$$

Outline

- Substructure Search (VLDB'08)
- Substructure Similarity Search (SIGMOD'10)
- Superstructure Search (SSDBM'10)
- Superstructure Similarity Search (ICDE'10)
- Graph Similarity Join (ICDE'12)
- Similarity All-Matching (SIGMOD'12)
- SimFusion+ Similarity Search (SIGIR'12)
- 
  SimRank Similarity Search (ICDE'13)
- Conclusions and Open Issues

8. SimRank Similarity Search (ICDE' 13)

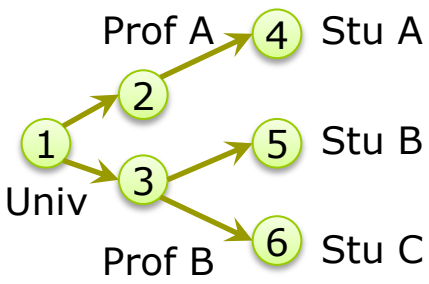
- A powerful model for assessing node-pair similarities [SIGKDD'02]
- Given G=(V,E), SimRank similarity is defined by
  - s (a, a) = 1,
  - s (a, b) = 0, if I(a) = ∅ or I(b) = ∅,
  - otherwise:

$$s(a,b) = \frac{c}{|I(a)| \cdot |I(b)|} \cdot \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s\left(I_i(a), I_j(b)\right)$$

High complexity !!!

$O(Kd^2n^2)$ time

$O(n^2)$ space



Setting c=0.8, we compute s(2,3) and s(4,5).

$\because\ I(2) = I(3) = \{1\},\quad I(4) = \{2\},\quad I(5) = \{3\}.$

$\therefore\ s(2,3) = \frac{0.8}{1\times 1} \times s(1,1) = 0.8$

$s(4,5) = \frac{0.8}{1\times 1} \times s(2,3) = 0.64$

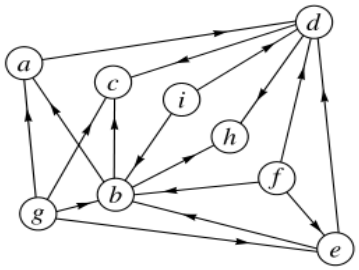
8. SimRank Similarity Search (ICDE' 13)

- **Prior Work** [VLDB J.'10]

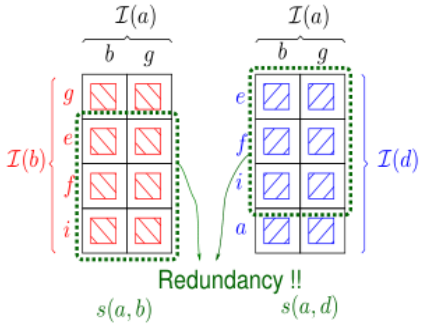
$$s^{(k+1)}(a,b) = \frac{c}{|I(a)| \cdot |I(b)|} \cdot \sum_{i=1}^{|I(a)|} \underbrace{\sum_{j=1}^{|I(b)|} s^{(k)}\left(I_i(a), I_j(b)\right)}_{=Partial^{s^{(k)}}_{I(a)}(I_j(b))}$$

  - Accuracy Estimate: $\left|s^{(k)}(a,b) - s(a,b)\right| \leq c^{k+1}$
  - Accelerate SimRank via Partial Sum Memoization to $O(Kdn^2)$ time

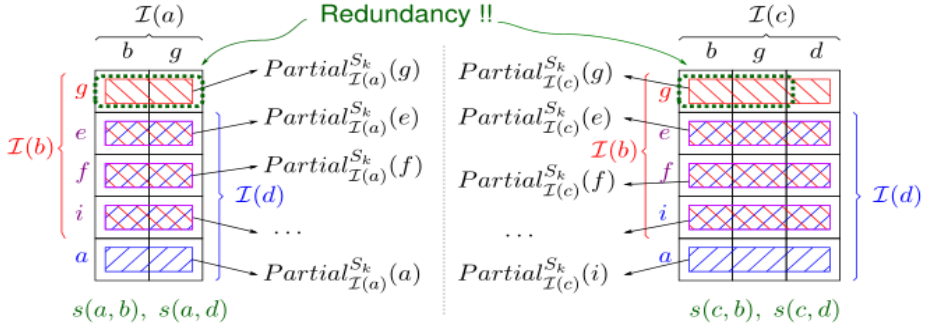
- **Our Observation**
  - Computations among different partial sums may have duplicate redundancy
  - The convergence rate is slow if a high accuracy is desirable.



(a) A paper citation network



(b) Naive method



(c) *Partial sums memorizing* leads to redundancy among different partial sums

- **Main Idea (1) : Speed up SimRank computation**
  - Share the common sub-summations among different partial sums, e.g,

Conventional Approach

$$Partial^{s(k)}_{I(a)}(g) = s(b,g) + s(g,g)$$

$$Partial^{s(k)}_{I(c)}(g) = s(b,g) + s(g,g) + s(d,g)$$

Duplicate Computation

(three additions)

Our Approach

$$Partial^{s(k)}_{I(a)}(g) = s(b,g) + s(g,g)$$

$$Partial^{s(k)}_{I(c)}(g) = Partial^{s(k)}_{I(a)}(g) + s(d,g)$$

(2 additions)

- **Main Idea (2) : Accelerate SimRank Convergence**
  - replace geometric series of SimRank with exponential kernel

	Conventional Approach	Our Approach
Series form	$\mathbf{S} = (1-C)\cdot\sum_{i=0}^{\infty} C^i\cdot\mathbf{Q}^i\cdot(\mathbf{Q}^T)^i$	$\hat{\mathbf{S}} = e^{-C}\cdot\sum_{i=0}^{\infty}\frac{C^i}{i!}\cdot\mathbf{Q}^i\cdot(\mathbf{Q}^T)^i$
Closed form	$\mathbf{S} = C\cdot(\mathbf{Q}\cdot\mathbf{S}\cdot\mathbf{Q}^T)+(1-C)\cdot\mathbf{I}_n$	$\frac{d\hat{\mathbf{S}}(t)}{dt} = \mathbf{Q}\cdot\hat{\mathbf{S}}(t)\cdot\mathbf{Q}^T, \qquad \hat{\mathbf{S}}(0) = e^{-C}\cdot\mathbf{I}_n.$
Iterative form	$\mathbf{S}_0 = \mathbf{I}_n$ $\mathbf{S}_{k+1} = C\cdot(\mathbf{Q}\cdot\mathbf{S}_k\cdot\mathbf{Q}^T)+(1-C)\cdot\mathbf{I}_n$	$\begin{cases}\mathbf{T}_{k+1} = \mathbf{Q}\cdot\mathbf{T}_k\cdot\mathbf{Q}^T\\ \hat{\mathbf{S}}_{k+1} = \hat{\mathbf{S}}_k + e^{-C}\cdot\frac{C^{k+1}}{(k+1)!}\cdot\mathbf{T}_{k+1}\end{cases}$ with $\begin{cases}\mathbf{T}_0 = \mathbf{I}_n\\ \hat{\mathbf{S}}_0 = e^{-C}\cdot\mathbf{I}_n\end{cases}$
Accuracy	$\|\mathbf{S}_k - \mathbf{S}\|_{\max} \le C^{k+1}$	$\|\hat{\mathbf{S}}_k - \hat{\mathbf{S}}\|_{\max} \le \frac{C^{k+1}}{(k+1)!}$

Conclusions

- Substructure search and its similarity search (VLDB'08, SIGMOD'10)
- Superstructure search and its similarity search (SSDBM'10, ICDE'10)
- Graph Similarity Join (ICDE'12)
- Similarity All-Matching (SIGMOD'12)
- SimFusion+ and SimRank Similarity Search (SIGIR'12, ICDE'13)

Open Issues

- Scalability Issue on Large Graphs
- Parallel Computation on Map Reduce
- Extend to Signed Networks (non-negative adjacency matrix)
- Improve I/O Efficiency
- Develop Novel Similarity Models
- …

Thank you!

Questions?