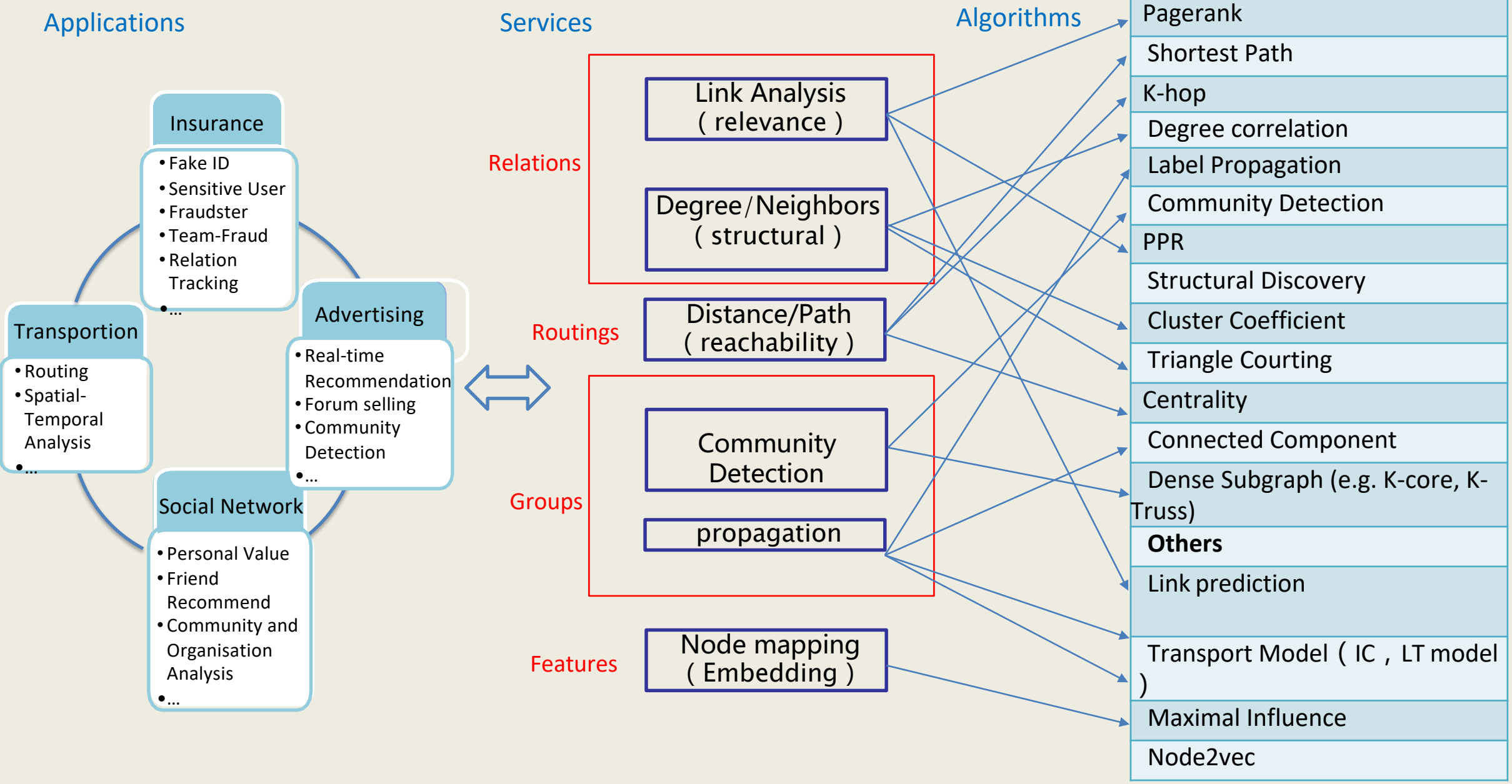
Towards Big Graph Processing: Applications, Challenges, and Advances

Xuemin Lin
UNSW, Australia

Applications



Outline

- Applications
- Graph Matching
- Cohesive Subgraphs
- Graph Similarity and Clustering
- Distributed Graph Computation
- Other Graph Problems
- Industry Collaborations

Some Examples

- ❑ Fraud Detection
- ❑Product Recommendation
- ❑Retail Services
- ❑ Investment Analysis
- ❑ Product Lifecycle Management
- ❑ Anti Money Laundering
- ❑ Cyber Security

…….

Fraud Detection: First-Party Bank Fraud

- Typical Scenario
  - A group of two or more people organize into a **fraud ring**;
  - The ring shares a subset of real contact, combining them to create a number of synthetic identities;
  - Ring members open accounts using these synthetic identities;
  - New accounts are added to the original ones (e.g. credit cards);
  - The accounts are used normally, with regular purchases and timely payments;
  - Banks increase the revolving credit lines over time, due to the observed responsible credit behaviour;
  - One day the ring “busts out”, coordinating their activity, maxing out all of their credit lines, and disappearing.

Fraud Detection: First-Party Bank Fraud

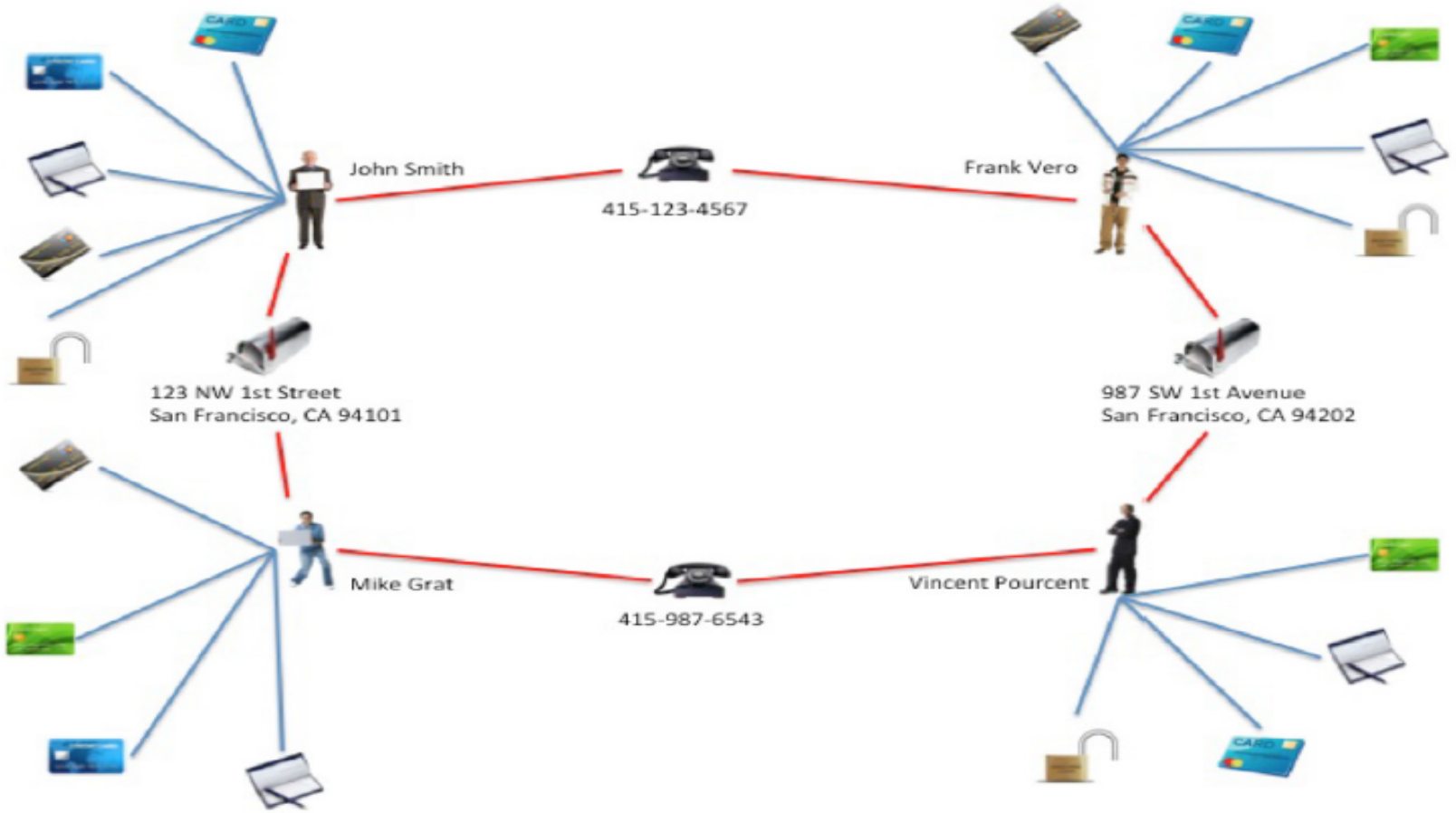


Diagram 1: 2 people sharing 2 pieces of data and creating 4 synthetic identities

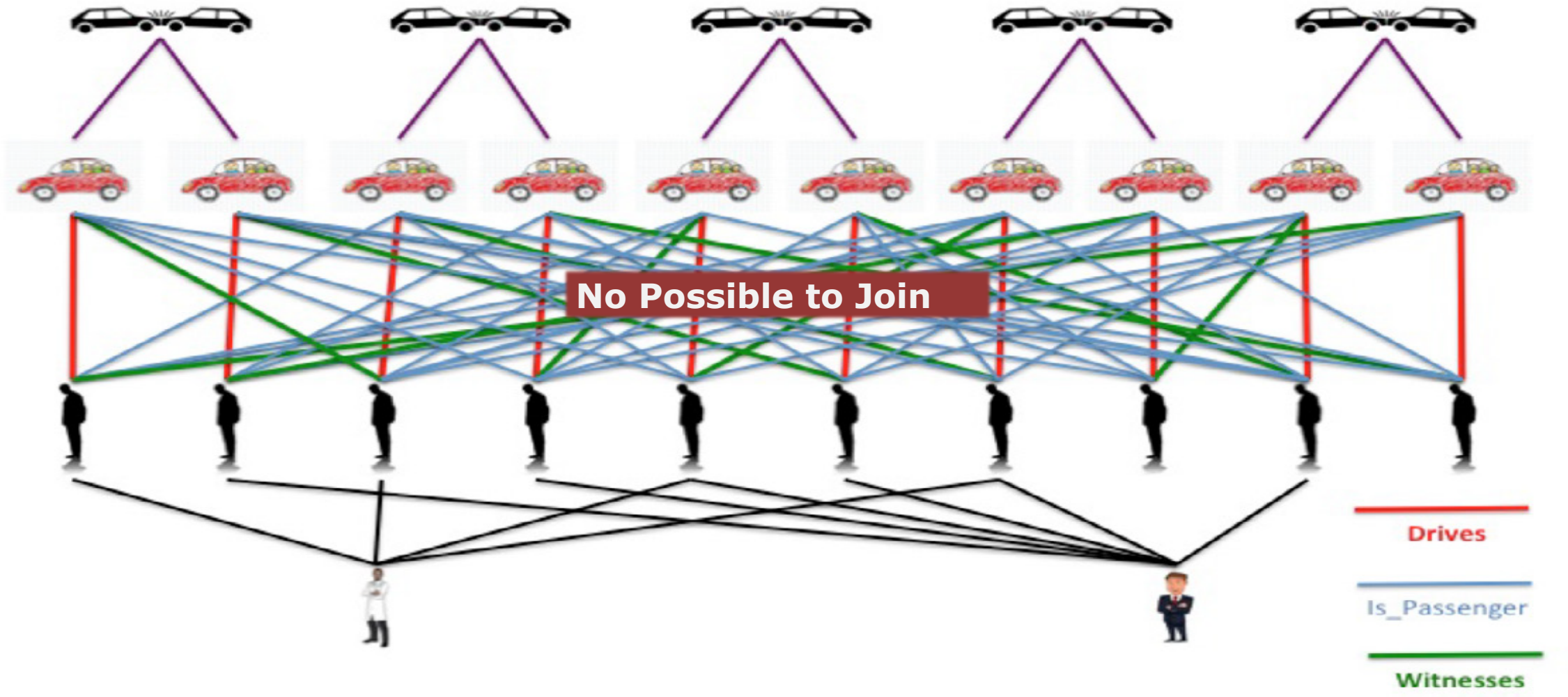
Fraud Detection: Insurance Fraud

- Background:
  - Fraudsters claim the insurance money via faking or exaggerating injury, damage etc.
  - The impact of fraud on the insurance industry is estimated to be $80 billion annually in the US.
  - In the UK, insurers estimate that bogus whiplash claims add $144 per year to each driver’s policy

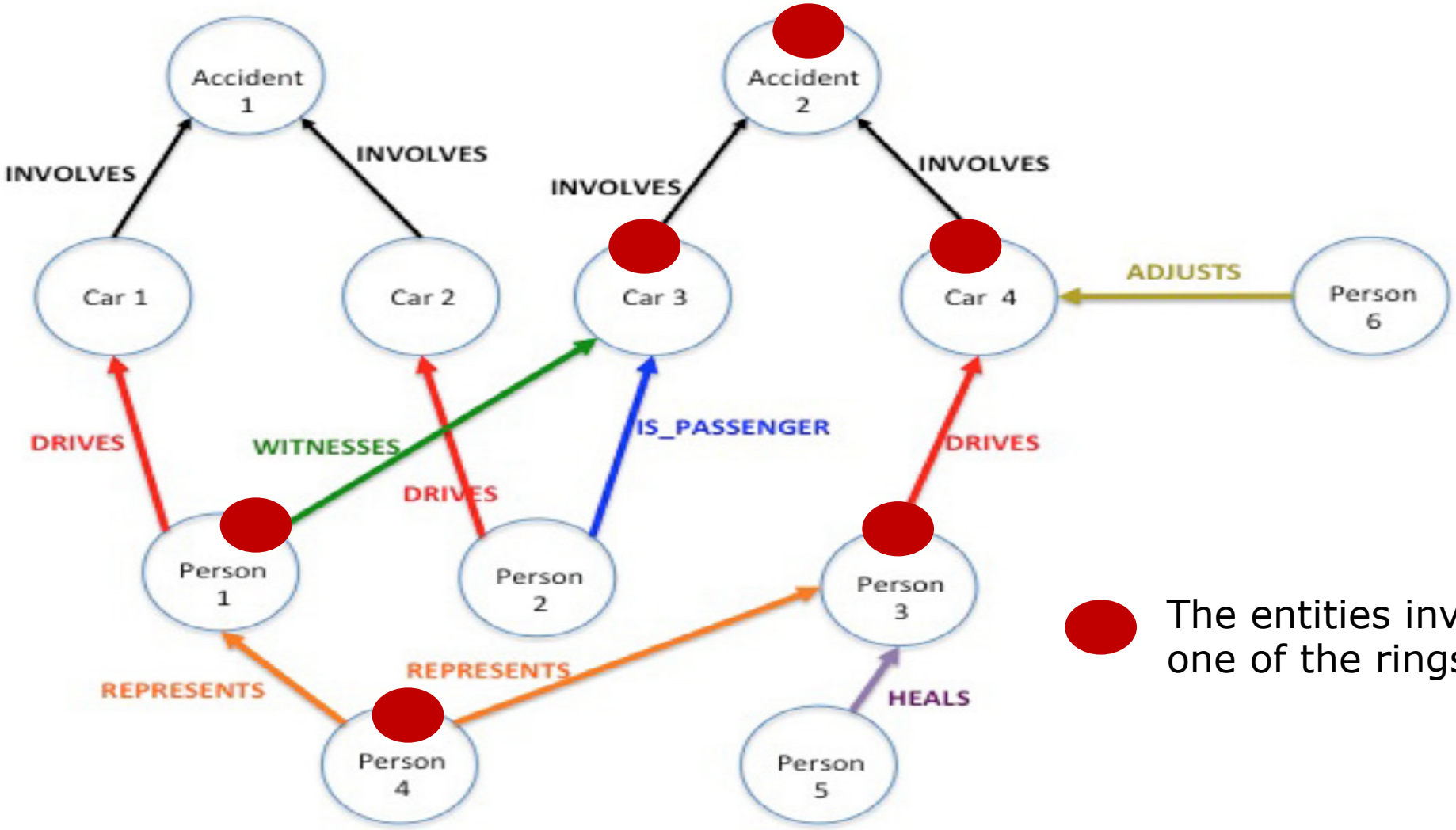
Fraud Detection: Insurance Fraud

- Typical Scenario
  - rings of fraudsters work together to stage fake car accidents and claim **soft tissue** injuries.
  - “Paper collisions”, with fake drivers, fake passengers, fake pedestrians and even fake witnesses.
  - Roles
    - **Providers**
      - Doctors: diagnose false injuries
      - Lawyers: file fraudulent claims
      - Body shops: misrepresent damage to cars
    - **Participants**
      - Drivers, Passengers, Pedestrians and Witnesses

Fraud Detection:
Insurance Fraud



Fraud Detection: Insurance Fraud



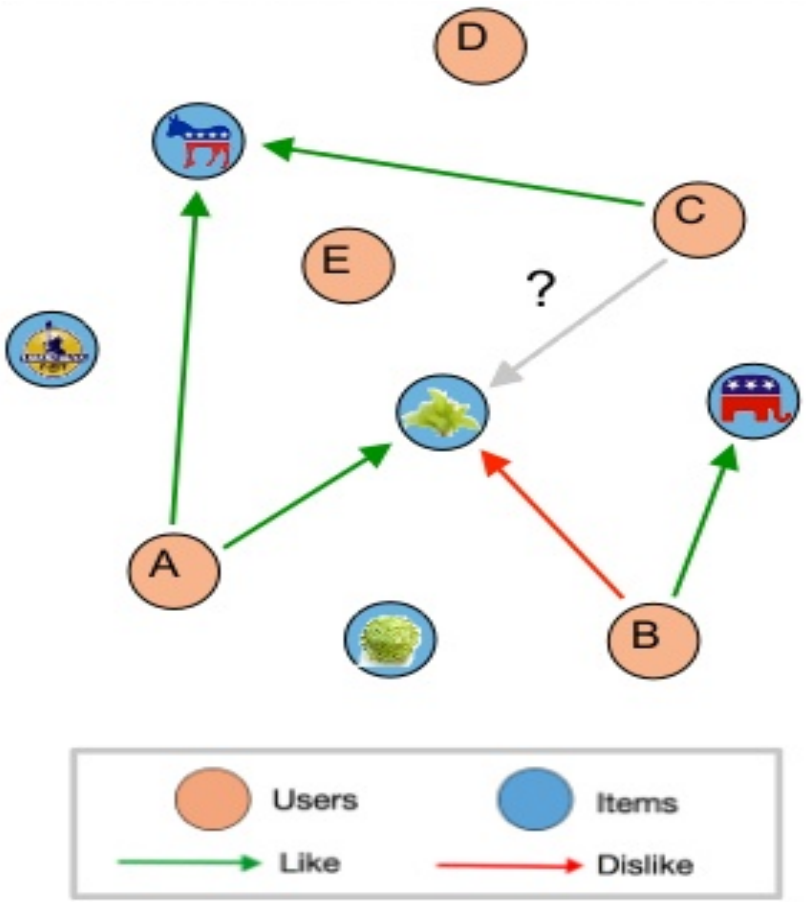
The entities involved in one of the rings.

Fraud Detection: Some Insights

- The data is grow so big and so complex that it is impossible to go with relation database.
- Graph database can naturally model the complex relations of data.
- The fraud detection can often reduce to the mining of certain patterns:
  - Rings (in the previous two examples)
  - Bi-cliques
  - More others.

Product Recommendation

- Goal
  - Online shopping, financial product recommendation
- Solution
  - Put customers, preferences, purchases, products and locations etc. into a graph model
  - Uses connections to make product recommendations
- Challenge
  - Serve many millions of customers and products: scalability & efficiency

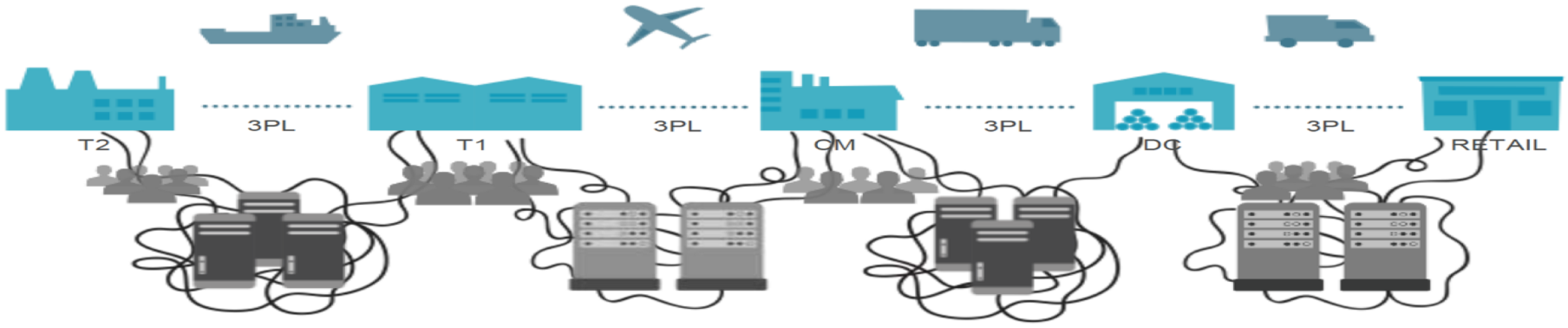


Retail Services:
Ecommerce Delivery Service Routing

- Goal
  - Enable delivery within 60 minutes to compete with Amazon Prime
- Solution
  - With graph, we can identify the fastest delivery route requires support for complex routing queries at scale with fast and consistent performance
- Challenge
  - Calculate best route option in real-time
  - Offer more predictable delivery time

Retail Services:
Supply Chain Visibility

- Goal
  - Visualize the supply chain and easily to explore for the clients
- Solution
  - With graph, we can easily explore the supply chain with graph traversal operations
- Challenge
  - the volume and structure of information to be processed are huge and complex

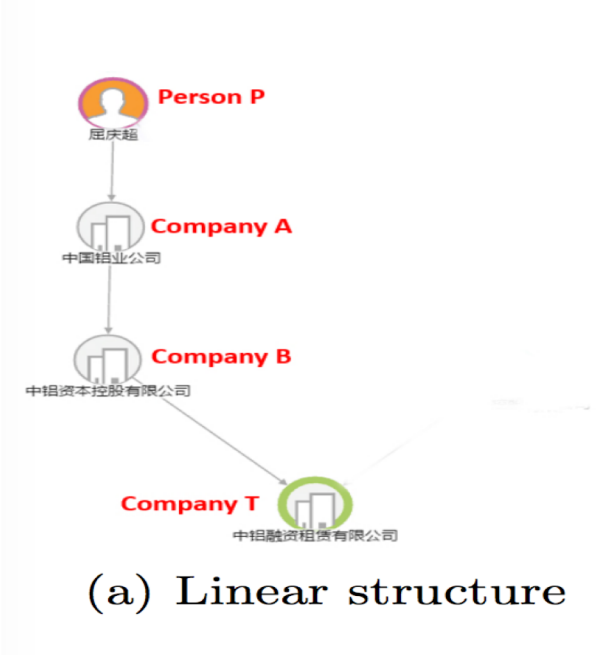


Investment Analysis

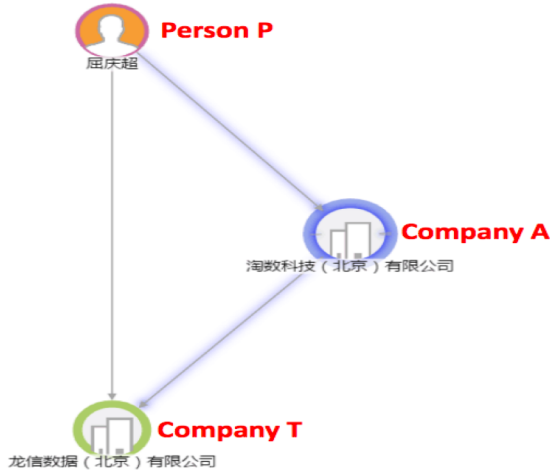
- A complete knowledge of an enterprise facilitate investment analysis, which requires:
  - Own profile: financial status, innovation strength of long credit
  - Ownership and investment relationships
    - The stronger the owner and relevant, the higher the investing potential
    - Linking all companies via their investment relations requires graph database

Investment Analysis

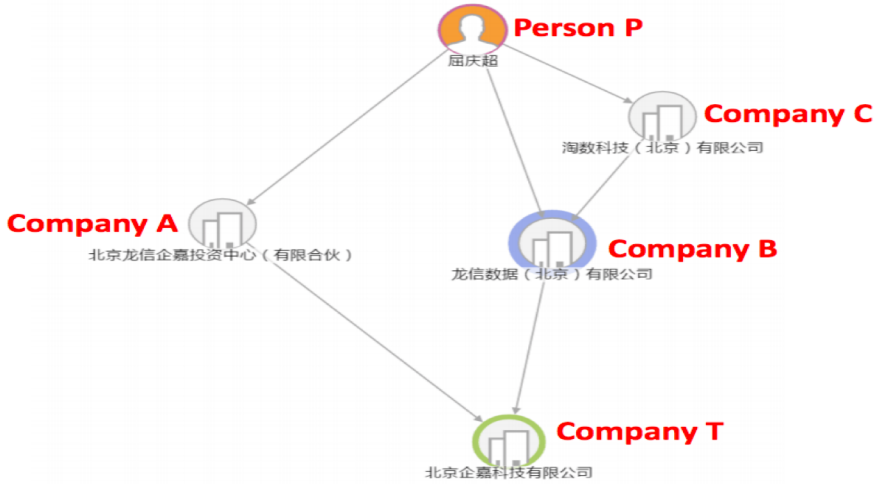
- Reveal an enterprise's real controllers: the person who owns the biggest equity share, directly or indirectly



(a) Linear structure



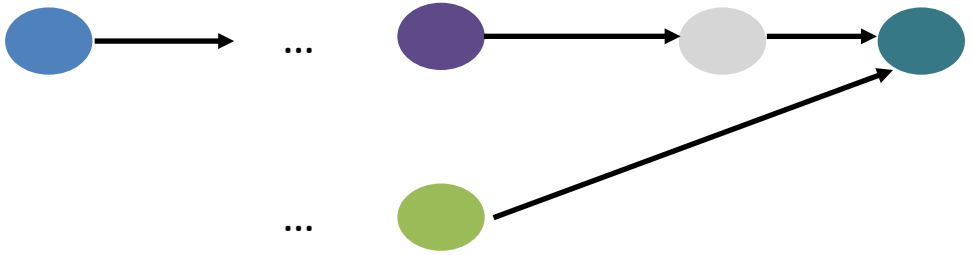
(b) Triangle structure



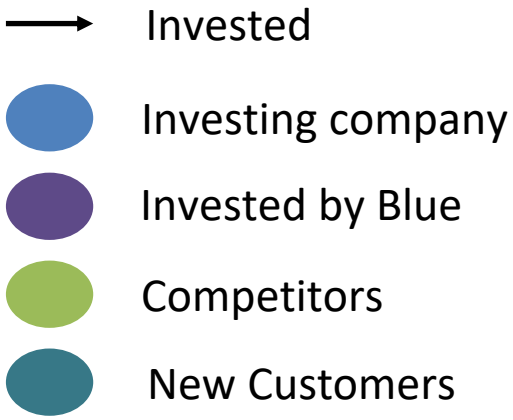
(c) Complex triangle structure

Investment Analysis

- Enterprise path discovery
  - Whether there are paths to reach new customers
  - the potential paths from the competitors to the new customers



Ooops, competitors approached first

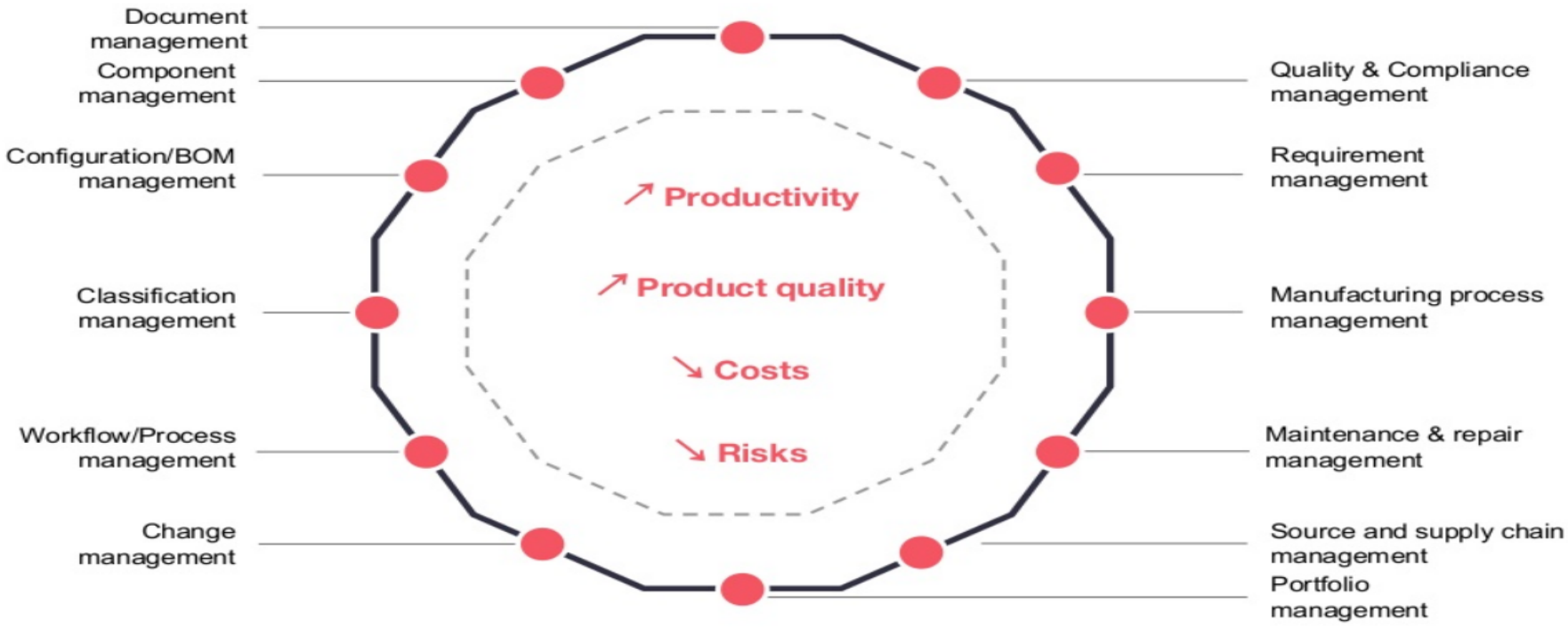


Investment Analysis

- Multidimensional relationships discovery:
  - competitors
  - pattern transfer
  - acquisition
  - investment
- Why this can help?
  - Difficulties of investing a target company? Try negotiate with the competitors' targets.

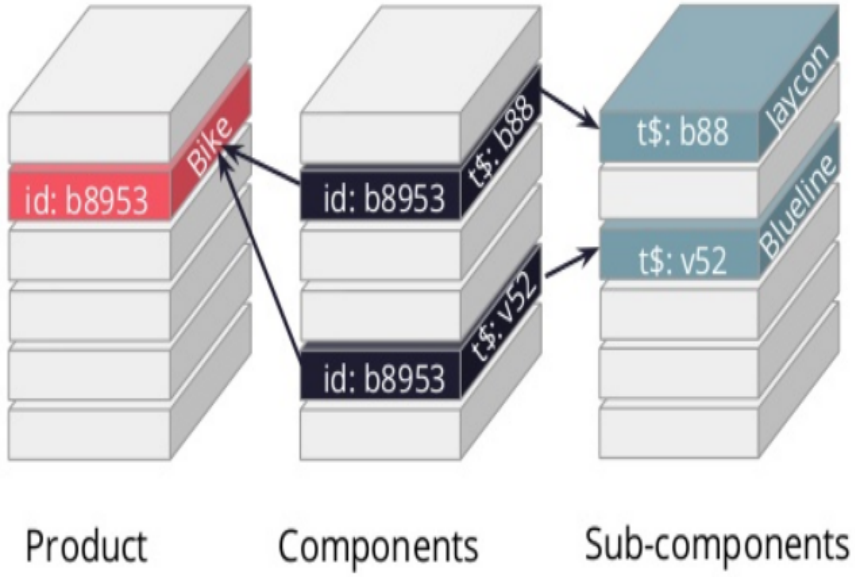
Product Lifecycle Management

- Product lifecycle management is the process of managing the entire lifecycle of a product from design stage to development to go-to-market to retirement to disposal



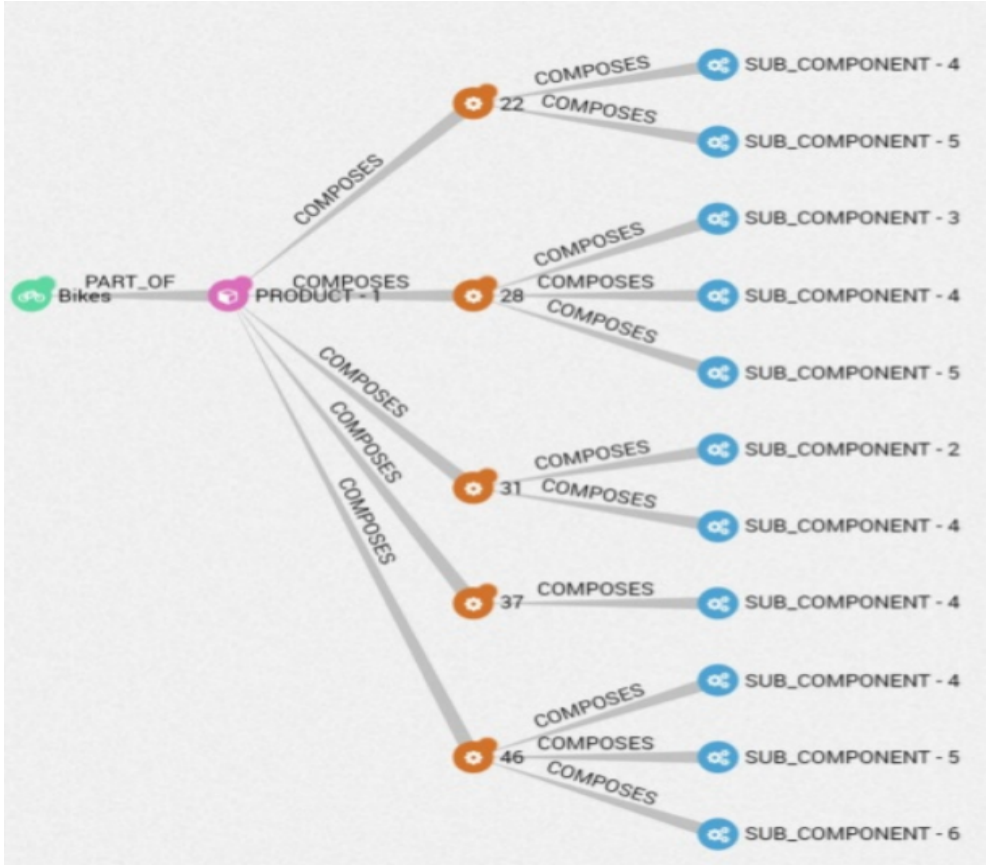
Product Lifecycle Management

- Existing PLM systems rely on RDBMS
  - Siloed data, inability to model real-life complexities and to adapt to changes
  - Inability to scale data model without costs and risks as business evolves
  - Search for information is time-consuming
  - Missing insights regarding dependencies leads to poor decisions and increase risks



Product Lifecycle Management

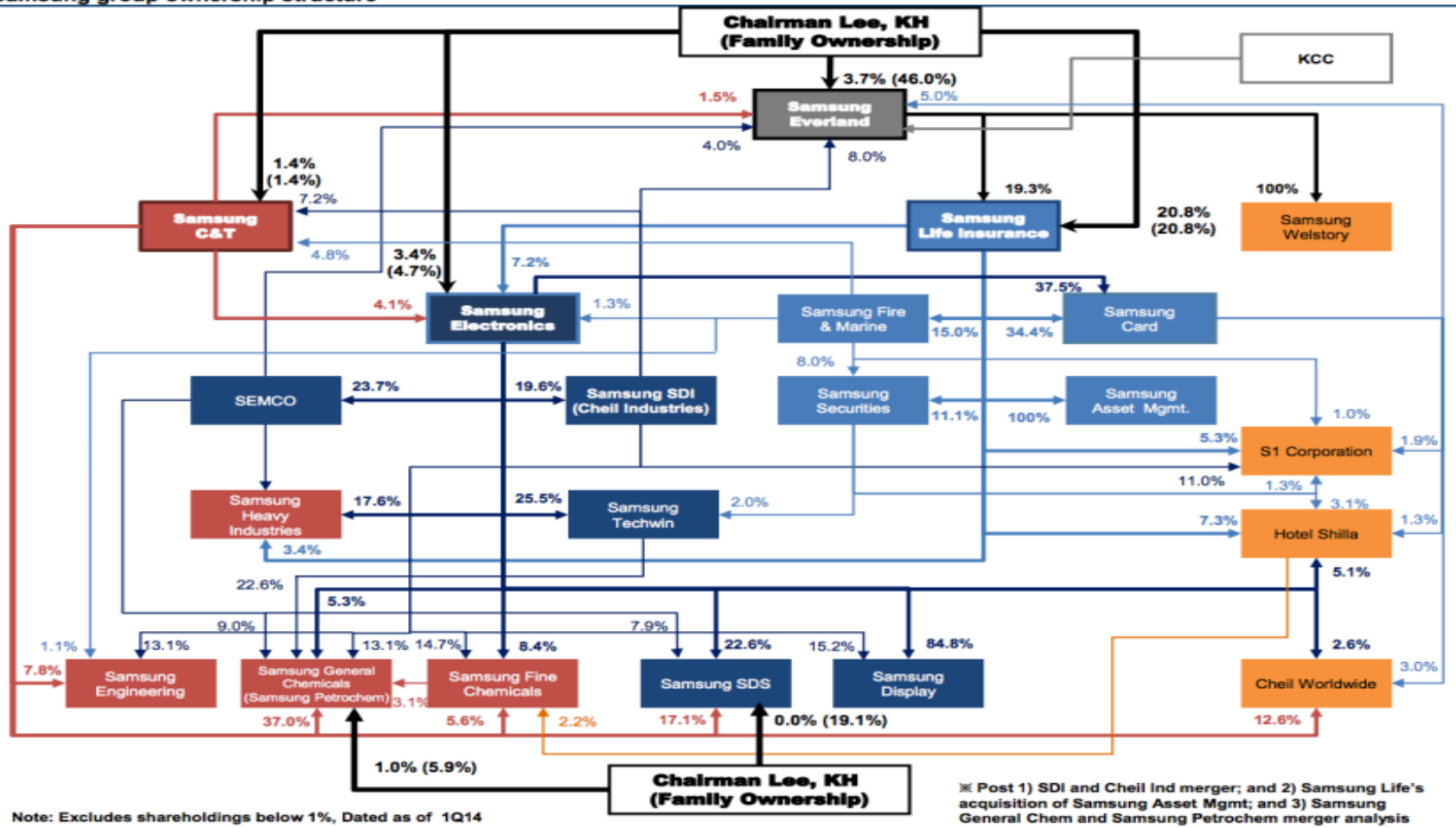
- Represent cross-department data about products and processes as a graph and store it as one
  - Aggregate product hierarchies and connections into a single source of truth
  - Easily edit, or expand your model as your need evolve
  - Saving time by searching and exploring your data
  - Visually track dependencies between entities and get contextual insights



Anti Money Laundering

Samsung Group

Figure 8: **Samsung group ownership structure**

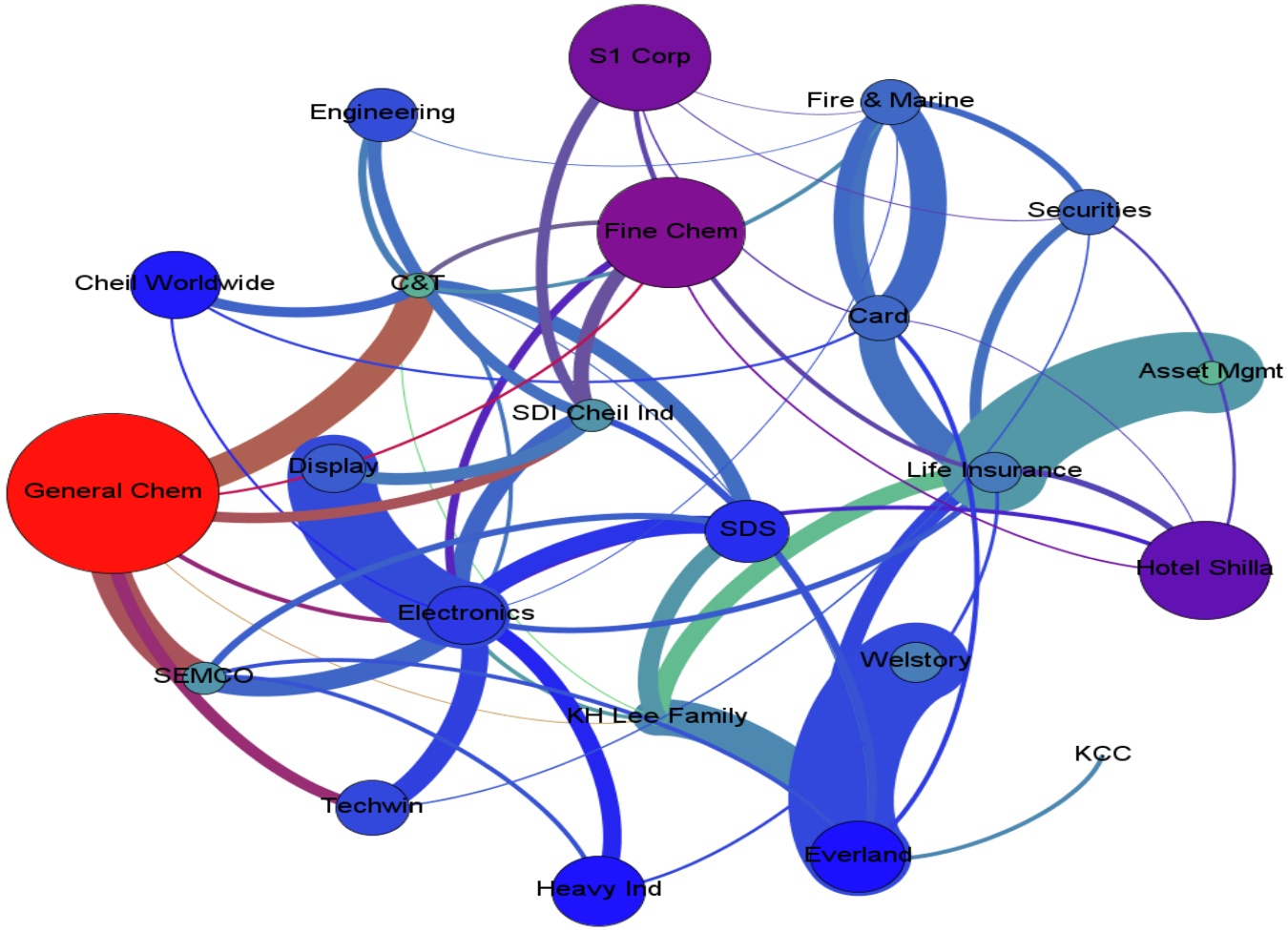


Note: Excludes shareholdings below 1%, Dated as of 1Q14

※ Post 1) SDI and Cheil Ind merger; and 2) Samsung Life's acquisition of Samsung Asset Mgmt; and 3) Samsung General Chem and Samsung Petrochem merger analysis

CREDIT SUISSE

Anti Money Laundering: Graph Model



- More than a pretty picture
- Graph structure for deep quantitative analysis

Anti Money Laundering: Graph Model

- Classical measures of **centrality**, like degree and **betweeness**, and **eccentricity**:
  - to correlate such measures with a propensity of fraud
- Complex circular money flows:
  - the native graph structure can also be used to input, track and calculate transactions across these chains.
  - profit distributions flow from one company to another via many **different paths**

Anti Money Laundering: Some Insights

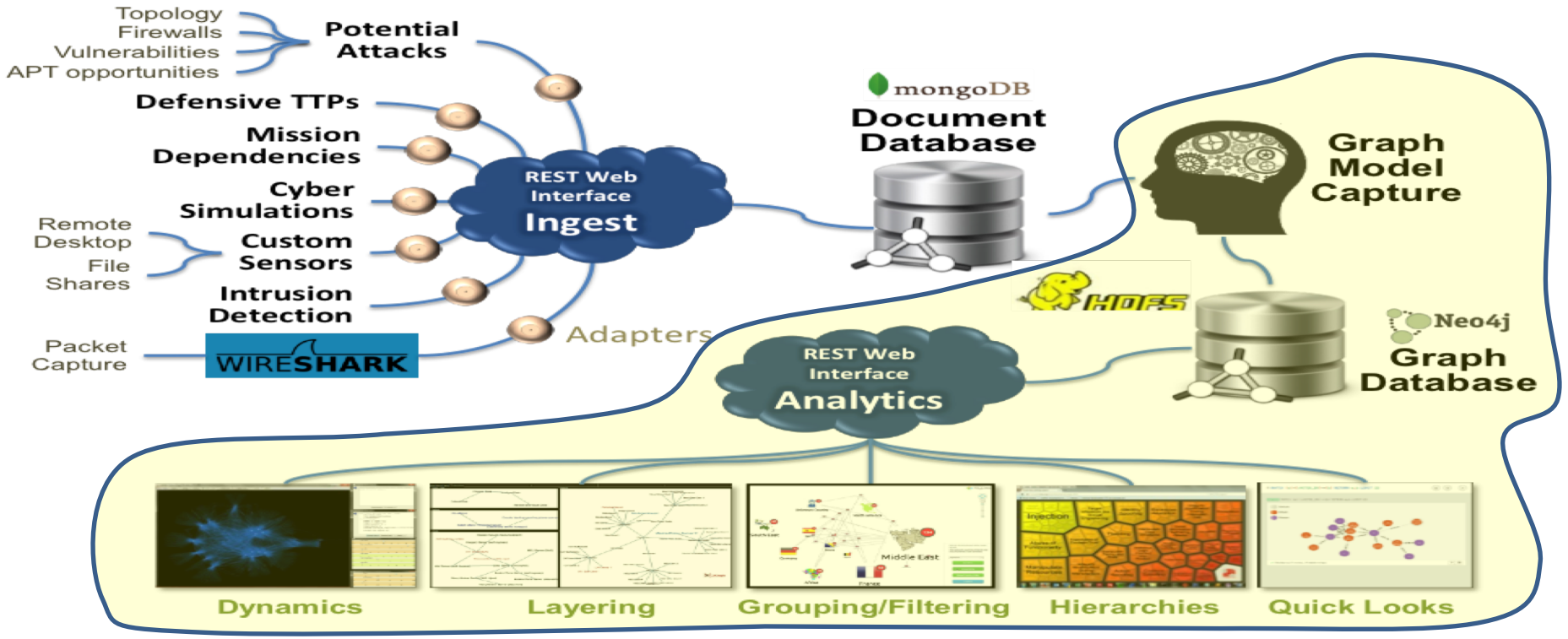
- Global-scale organisation fosters complex transaction flows
  - can be circular, making it impossible to investigate ML using traditional method.
  - Graph not only can visual, but can model the data for deep analysis
    - Graph properties such as **eccentricity** and **betweenness** are useful factors to correlate the potential ML.
    - Path, cycle and pattern analysis can facilitate the AML process

Cybersecurity

- Goal
  - A unified web framework to manage cyber attack relationships in the network
  - Understand how cyber attackers can leverage initial footholds to extend their reach through the network
- Challenge
  - correlate data from numerous sources into a common model
  - flexible and easy to extend, and map naturally to network attack relationships
  - Support various kinds of network attack relationships queries

Cybersecurity: MITRE Case Study

- Solution
  - Graph model

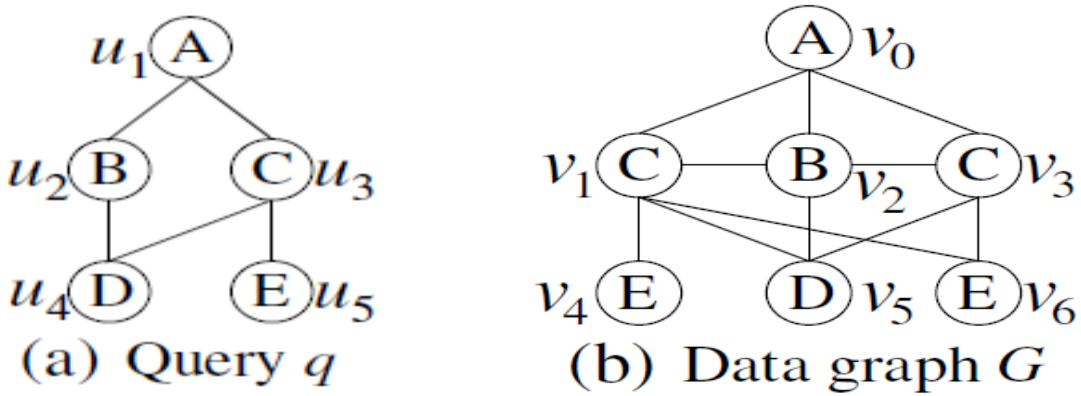


GRAPH PATTERN MATCHING

Introduction

- **Subgraph Matching**

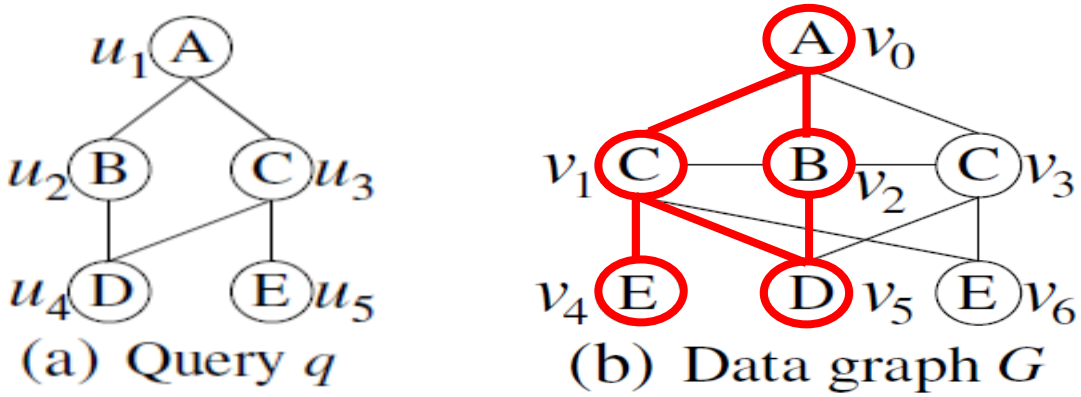
  Given a query ***q*** and a large data graph ***G***, the problem is to extract all subgraph isomorphic embeddings of ***q*** in ***G***.



Introduction

- **Subgraph Matching**

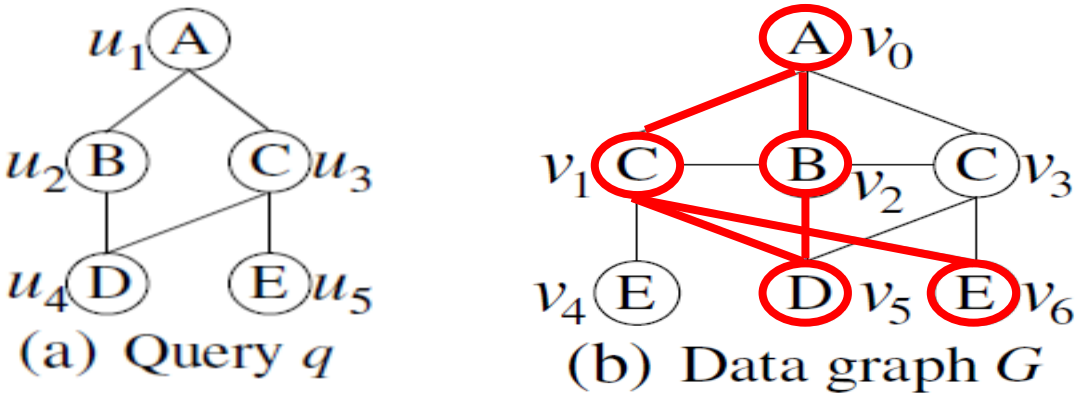
  Given a query ***q*** and a large data graph ***G***, the problem is to extract all subgraph isomorphic embeddings of ***q*** in ***G***.



Introduction

- **Subgraph Matching**

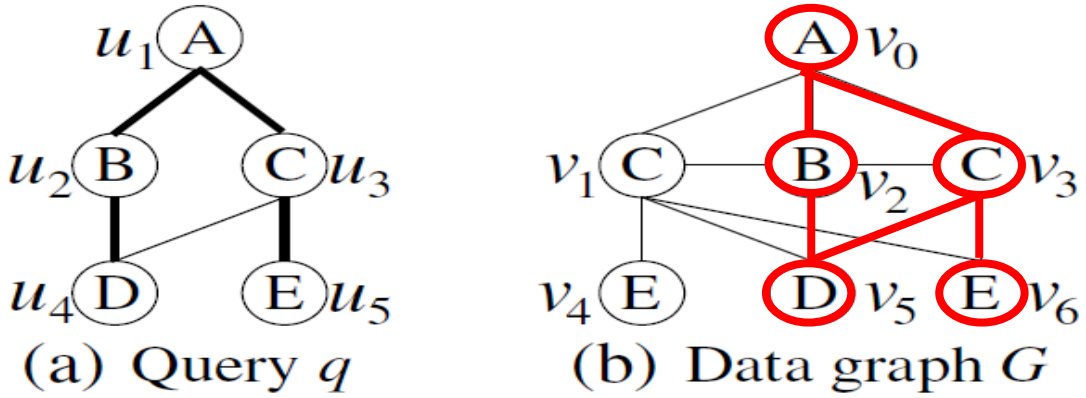
  Given a query ***q*** and a large data graph ***G***, the problem is to extract all subgraph isomorphic embeddings of ***q*** in ***G***.



Introduction

- **Subgraph Matching**

  Given a query ***q*** and a large data graph ***G***, the problem is to extract all subgraph isomorphic embeddings of ***q*** in ***G***.



Graph Pattern Matching: summary

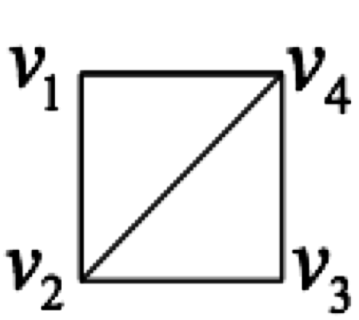
- Related Applications
  - Fraud Detection, Anti Money Laundering, Cyber Security etc.
- Algorithms and Recent Publications
  - Exact Match
    - Tree Sequence (QuickSI), VLDB'08
    - Core-Forest-Leaf Decomposition (CFL Algorithm), Sigmod'16
    - Optimal Distributed Join-based algorithms, VLDB'15, VLDB'17
  - Approximate Match
    - Spanning-tree-based matching algorithm (TreeSpan), SIGMOD'12
  - Supergraph Match
    - Tree-index-based supergraph search algorithm, DGTree, ICDE'16
  - Tree Match
    - Lawler-procedure-based top-k tree searching algorithm, VLDB'15

Distributed Subgraph Enumeration

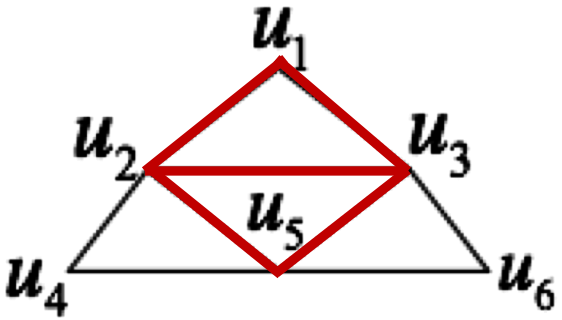
(*VLDB2015 & VLDB2017)*

Problem Statement

Given a pattern graph P, we aim at finding all matches of ***P*** in a large data graph *G.*



P



G

$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 \\ u_1 & u_2 & u_5 & u_3 \end{pmatrix}$$

Related Works

Edge Join [T. Plantenga, JPDC 2012]

- Join edges one by one to get the final results
- Finish in m rounds, not scalable to handle complicated pattern graph
- To many intermediate results

Multiway join [F.Afrati et al., ICDE 2013]

- One-round computation by duplicating edges
- Do not scale well due to memory issues

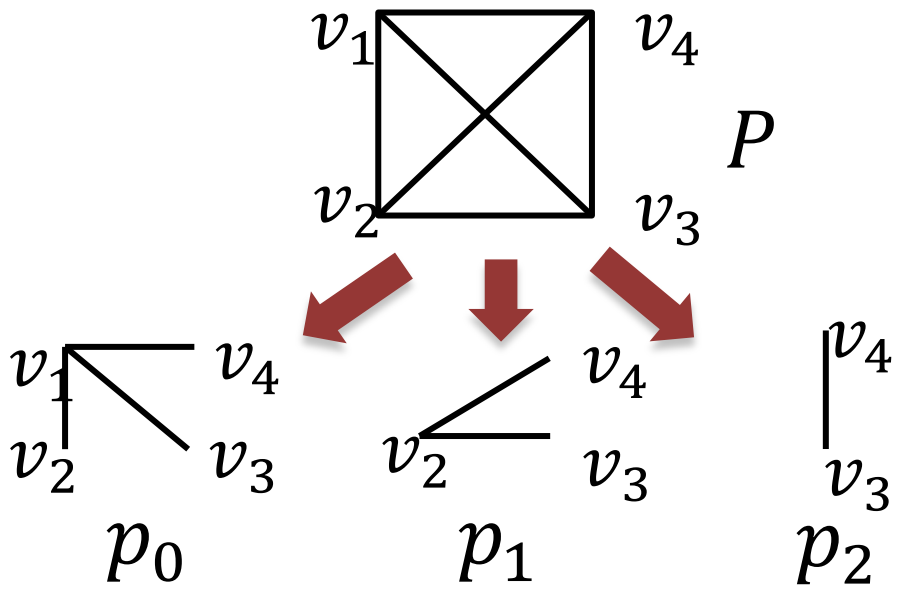
Related Works

PSgL [Shao et al., Sigmod 2014]

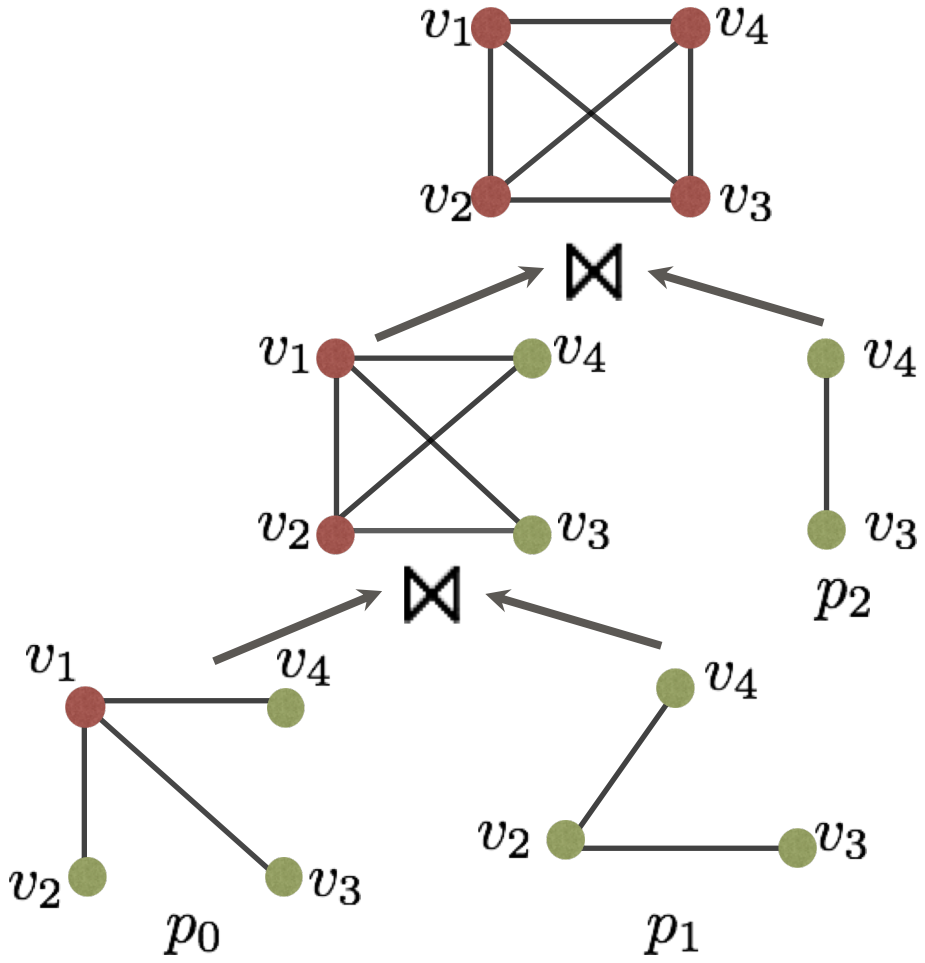
- BFS Search strategy, Vertex-centric computation
- Analogous to decomposing into stars and then processing **Star-based join**.
- Several issues including memory issues

Star-based Join

Star decomposition



Left-deep Join



Trade-off for stars as join units

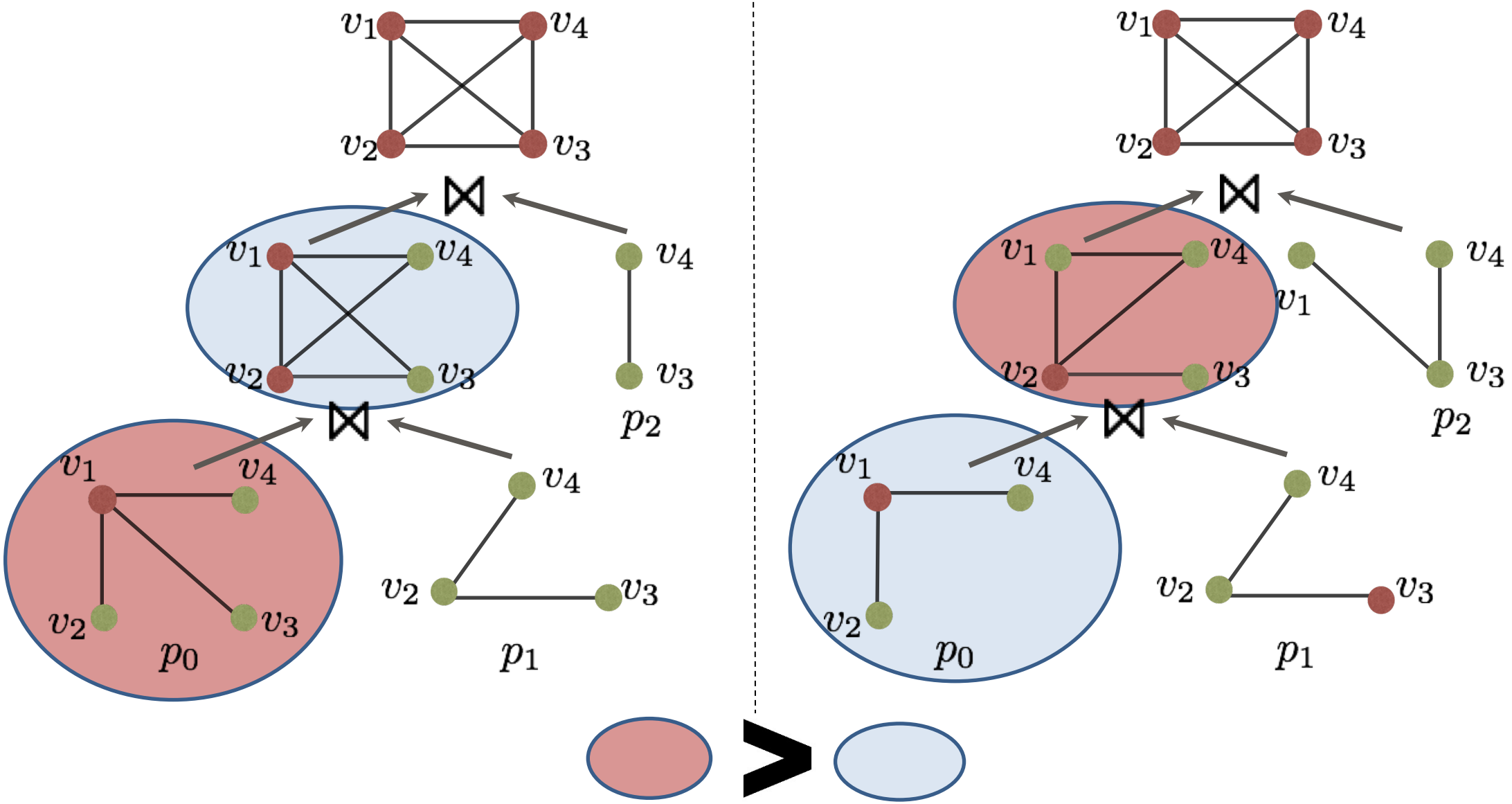
- # of matches to a star is massive if many edges in a star.

A node with 1,000,000 neighbors => 10^{18} matches of a 3-star

Not unusual: a node contains many neighbors in a large social networks or web graphs

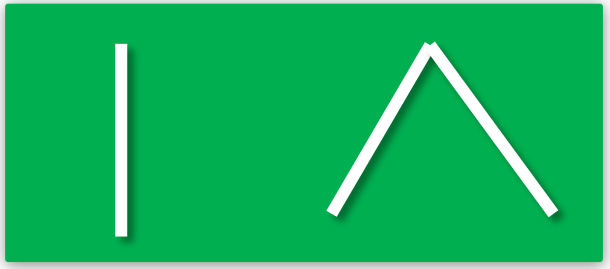
- If star with many edges, a join followed has more pruning power; i.e., less # intermediate results of a join.

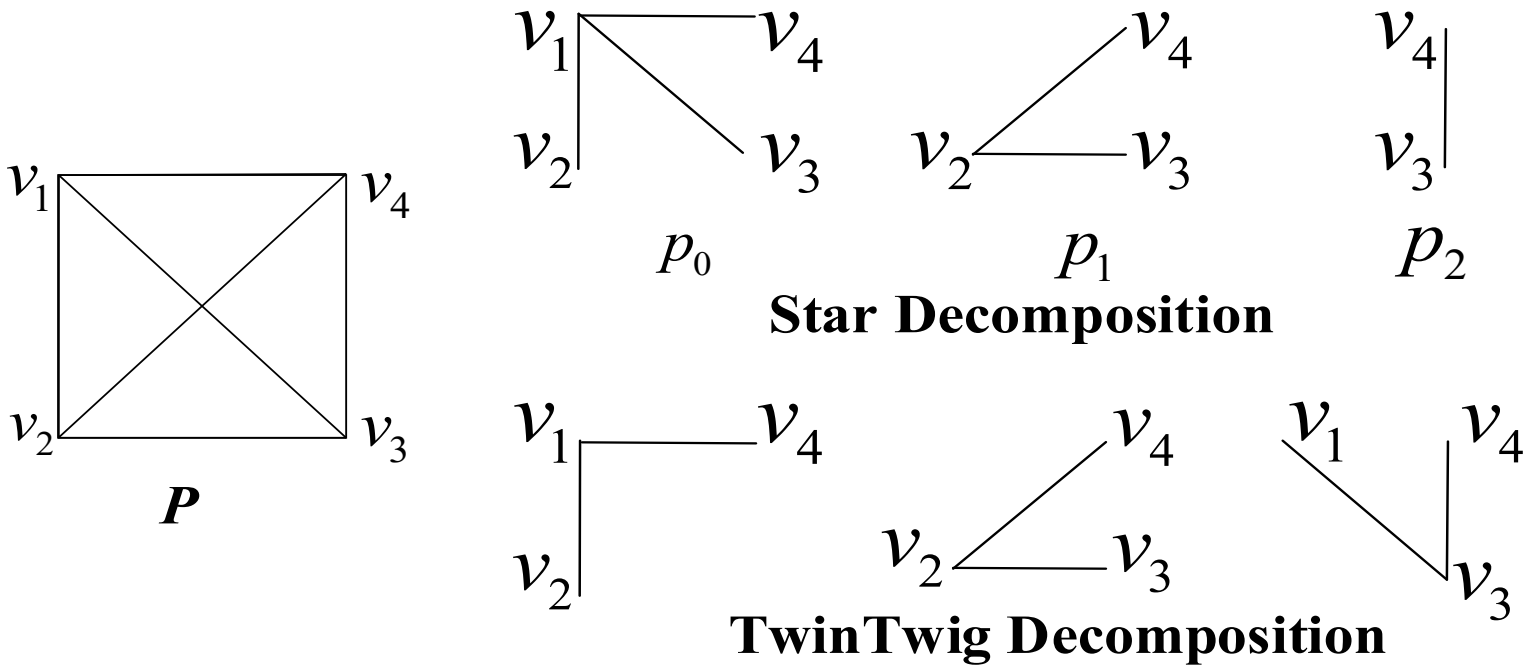
Star-based Join



TwinTwig Join VLDB15'

TwinTwig: A star of at most two edges





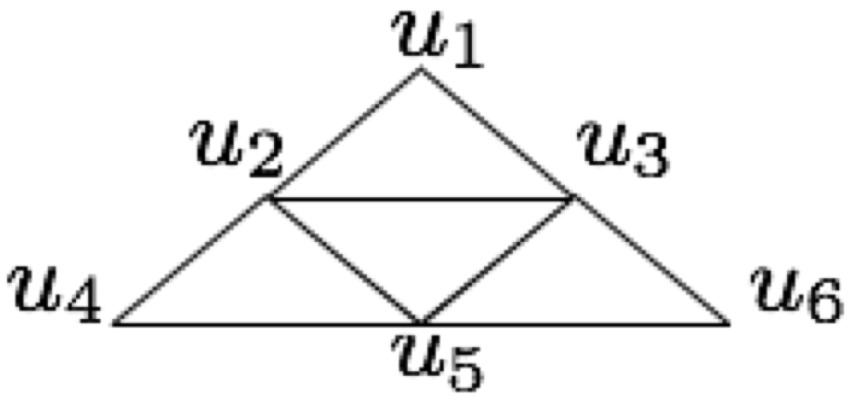
TwinTwig Join VLDB15’

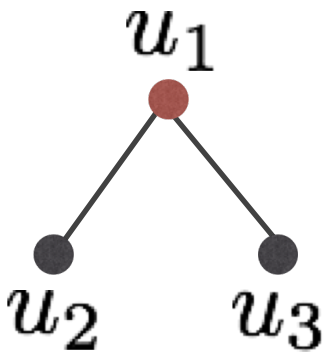
Given any *star-based join*, *star*, there always exists a *TwinTwig join*, *tt*, s.t. the size of intermediate results of tt is no larger than that of star.

The drawbacks of TwinTwig Join

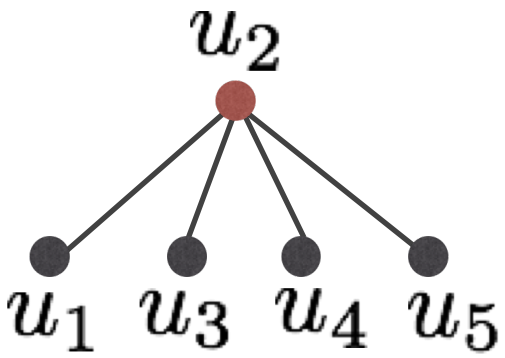
Simple graph storage only supports using **star (TwinTwig)** as join units, which can result in too many intermediate results

(*e.g. a node of degree 1,000,000,* ***O(10^18)*** *stars,* ***O(10^12)*** *TwinTwigs*)





G_{u_1}



G_{u_2}

Left-deep join can result in **sub-optimal** solution

General Algorithmic Framework

Graph Storage $\Phi(G) = \{G_u | u \in V(G)\}$

- *Stored as (u; G_u) for each data node u*
- *G_u is called the local graph of u such that*:

(1) it is connected;

(2) $u \in V(G_u)$;

(3) $\bigcup_{u \in V(G)} E(G_u) = E(G)$

General Algorithmic Framework

The structure into which we decompose the pattern graph are called the ***join unit***

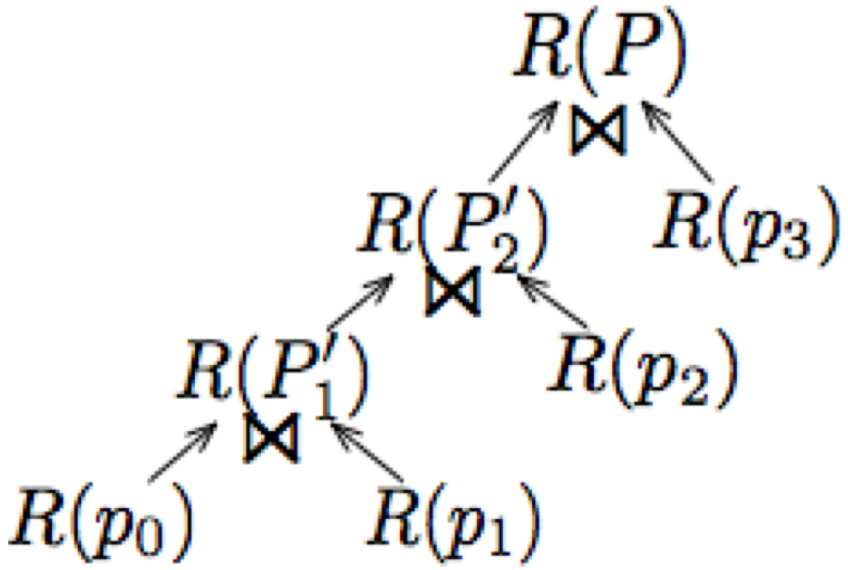
- *e.g.* ***Star*** *in Star-based join,* ***TwinTwig*** *in TwinTwig Join*
- A structure p can be the ***join unit*** iff.

$$R_G(p) = \bigcup_{u \in V(G)} R_{G_u}(p)$$

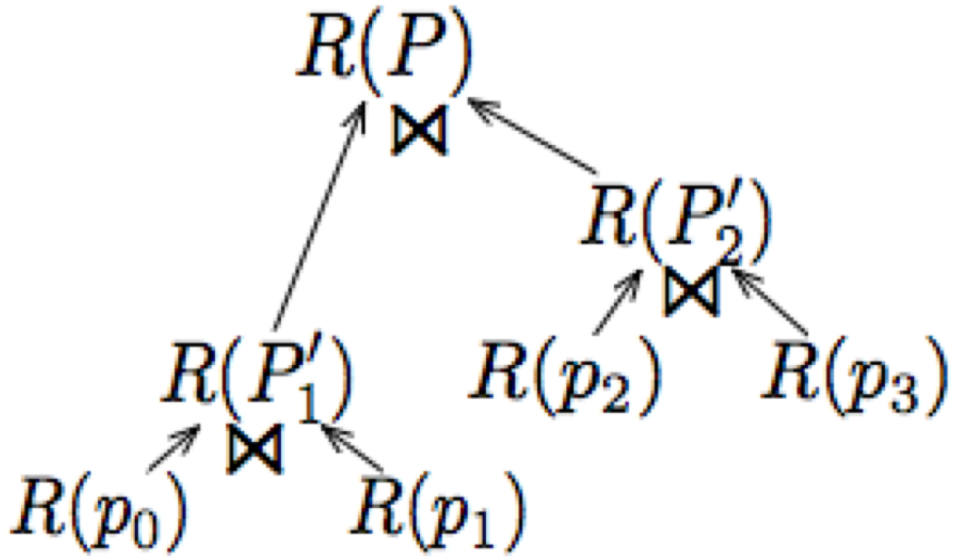
where $R_{\mathcal{G}}(p)$ stands for the matches of p in $\mathcal{G}$

General Algorithmic Framework

Decomposing $P = p_0 \cup p_1 \cup p_2 \cup p_3$



Left-deep tree



Bushy tree

SEED VLDB2017

SCP (Star-Clique-Preserved) graph storage: Use star and clique as the join units

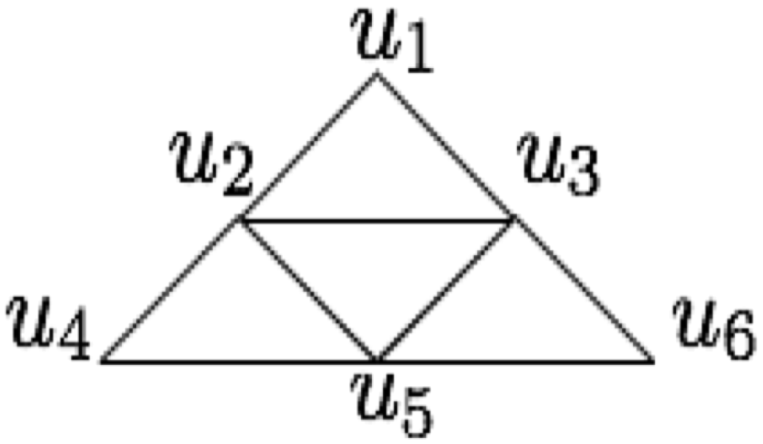
- We can avoid using star if clique is an alternative
- Shorter execution. The 6-clique can now be processed in one single round, instead of 7 rounds in TwinTwig Join

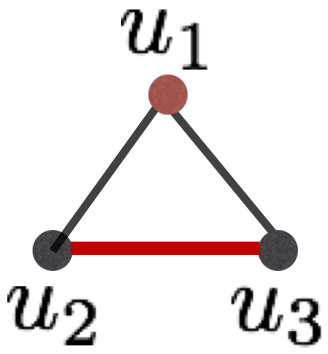
Bushy join plan: Optimality Guarantee

- The search space of an optimal bushy join covers that of a left-deep join

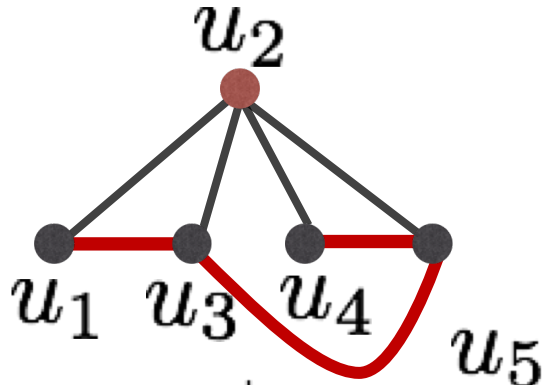
SCP Graph Storage

SCP Graph, each local graph is denoted as G_u^+





$G_{u_1}^+$



$G_{u_2}^+$

We include the edges among the neighbors into the local graph Gu, so that all cliques with u can be fully computed within G_u^+

Optimal Bushy Join

Notations

E_P: the join plan to solve P

$C(E_P)$: the cost of the join plan

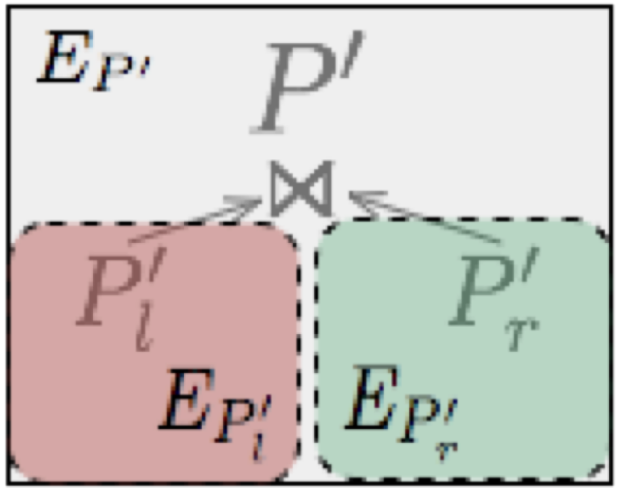
$C(P)$: estimated # matches of P in G (We apply powe-law random graph model for the estimation, while estimation can vary with applications)

We aim at finding a join plan for P , s.t $C(E_P)$ **is minimized**

Optimal Bushy Join

A dynamic programming transform function:

$$R(P') = R(P'_l) \bowtie R(P'_r)$$



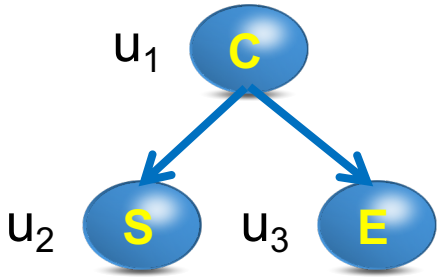
$$C(E_{P'}) = \min_{P'_l \subset P' \wedge P'_r = P' \setminus P'_l} \{C(E_{P'_l}) + C(P'_l) + C(E_{P'_r}) + C(P'_r)\}$$

Optimal Enumeration: Efficient Top-k Tree Matching (VLDB2015)

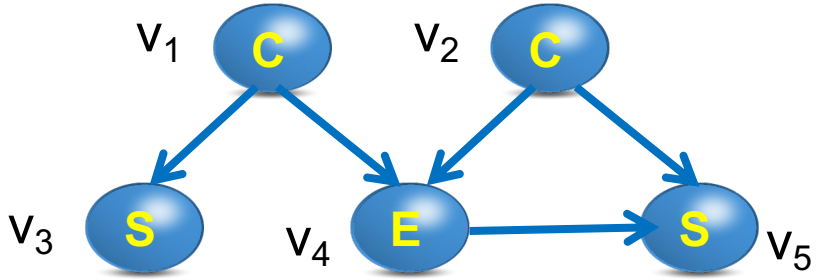
Introduction

- Data Graph: a node-labeled directed graph *G*
  - Query: a node-labeled rooted tree *T*
  - Find *k* mappings from *T* to *G* with the minimum lengths.

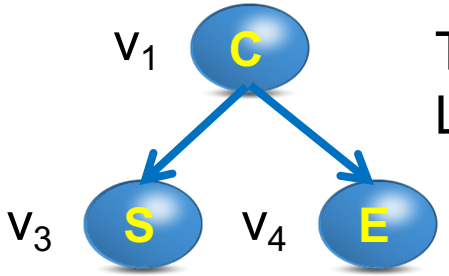
Query Tree *T*:



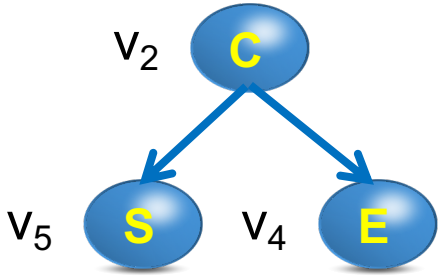
Data Graph *G*:



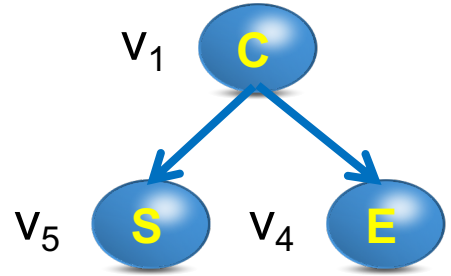
Top-1 Match
Length=2



Top-2 Match
Length=2



Top-3 Match
Length=3



Applications of Top-k Tree Matching

- Top-k tree matching strengthen the relevance of twig-pattern matching results
  - Twig-pattern matching is a core operation in XQuery over XML/Graph data
  - Exponential number of matches

- It is also used as a key building block to retrieve top-k graph pattern matches

State-of-the-art Approach[Gou'08]

- Time Complexity of the existing techniques
  - $O(n_T(d_T + \log k))$ for enumerating each top-i match
    - n_T: number of nodes in T
    - d_T: the maximum node degree in T

- **Our contribution in this paper**: we propose a new enumeration paradigm to enumerate each match in $O(n_T + \log k)$ time

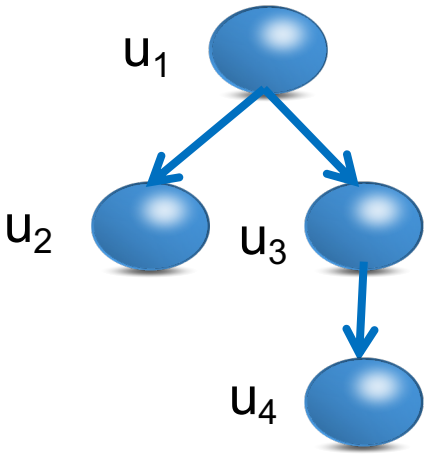
A New Enumeration Strategy

- The solution space: all valid matches in $V_1 \times V_2 \times \ldots \times V_n$
  - $\{u_1, u_2, \ldots, u_n\}$: the set of nodes T
  - V_i: the set of nodes in G having the same label as u_i.
- Adapt Lawler's Procedure to compute top-k matches
  - Suppose $(v_1, v_2, \ldots, v_n)$ is the top-*1* match, then the entire solution space is divided into n subspaces
    - $(V_1 - \{v_1\}) \times V_2 \times \ldots \times V_n$
    - $\{v_1\} \times (V_2 - \{v_2\}) \times \ldots \times V_n$
    - …
    - $\{v_1\} \times \{v_2\} \times \ldots \times (V_n - \{v_n\})$
  - Top-*2* match is the match with lowest score among best matches in these obtained subspaces.

How to efficiently compute the best match in a subspace?

Two Types of Subspaces

- Categorize the n subspaces into two types
  - Let $\{v_1\} \times \{v_2\} \times \ldots x (V_i - U_i) \times \ldots \times V_n$ be divided by $(v_1, v_2, \ldots, v_n)$
  - Type-1: $\{v_1\} \times \{v_2\} \times \ldots \times (V_i - U_i - v_i) \times \ldots \times V_n$
    - » Only one such type of subspace
  - Type-2: $\{v_1\} \times \{v_2\} \times \ldots \times \{v_i\} \times \ldots \times (V_j - v_j) \times \ldots \times V_n$
    - » $(n - i)$ such type of subspaces
- For efficient computation, order the nodes in T in a top-down fashion (e.g., u_1, u_2, u_3, u_4).



An Optimal Approach

- Compute the **score** of the best match in a subspace
  - *One* Type-1 subspace: takes time *O(log k)*
  - *(<n)* Type-2 subspaces: each takes time *O(1)*
- The matching details if needed can be obtained in *O(n)* time.
- Each top-*i* match can be computed in *O(n + log k)* time

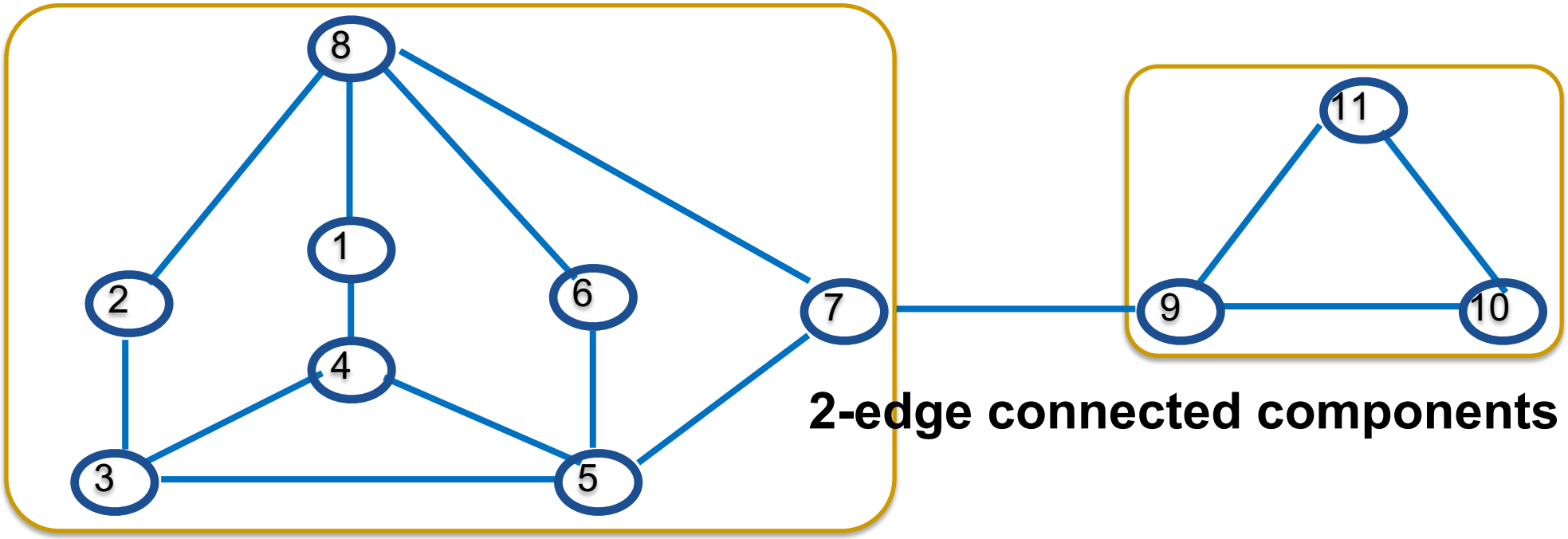
This is optimal, since *log k* usually is much smaller than *n*

Index-based Optimal Algorithms for Computing Steiner Components with Maximum Connectivity

(SIGMOD 2015)

Introduction

- A graph is k-edge connected if it is still connected after removing any set of (k-1) edges from it.
- A k-edge connected component is a maximal k-edge subgraph.



Introduction

- Applications of computing k-edge connected components
  - Community detection
  - Robustness in a communication network
  - Graph visualization
  - ……

- Algorithms of computing k-edge connected components for a fixed k
  - Deterministic algorithm: $O(h \times l \times |E|)$
    - Our work in SIGMOD'13
  - Randomized algorithm: $O(t \times |E|)$
    - By Takuya Akiba, Yoichi Iwata, Yuichi Yoshida in CIKM'13

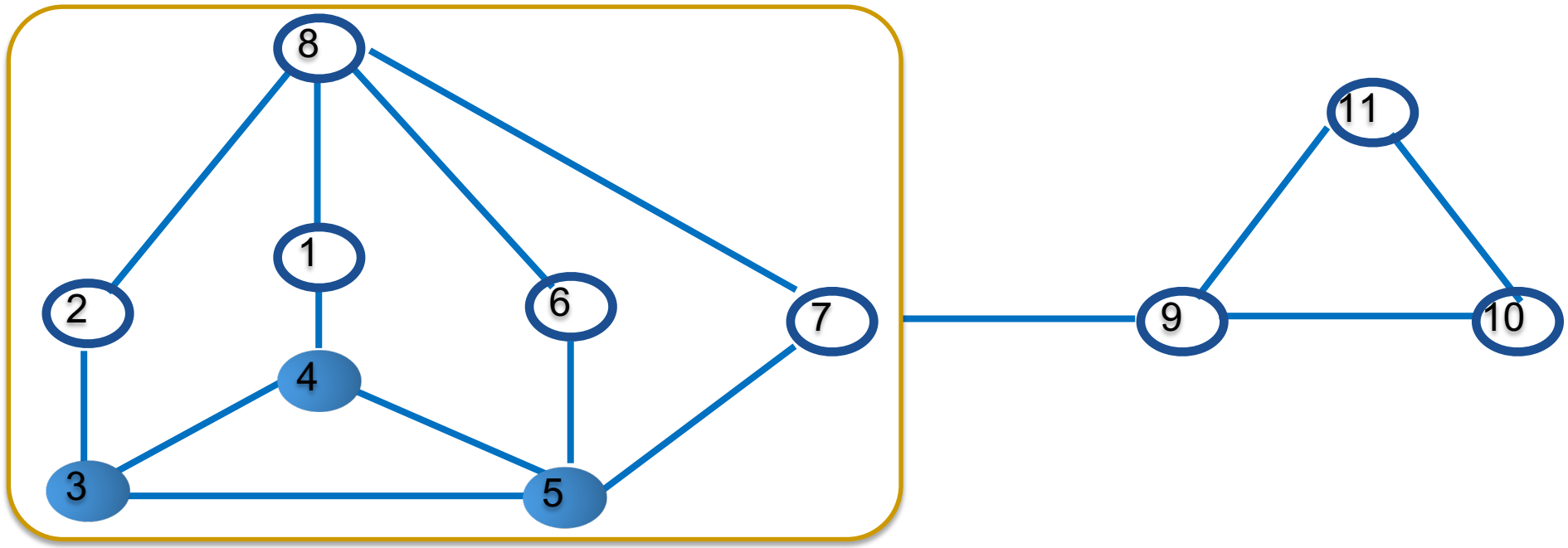
Steiner Maximum-Connected Component (SMCC) and Steiner-Connectivity

1. *SMCC* of q: maximal subgraph of G containing $q \subseteq V(G)$ with maximum connectivity.
2. *Steiner-connectivity* of q: the connectivity of the SMCC of q. Denoted sc(q).

Applications:

- Potential customer prediction
- Product promotion
- Research team assembling

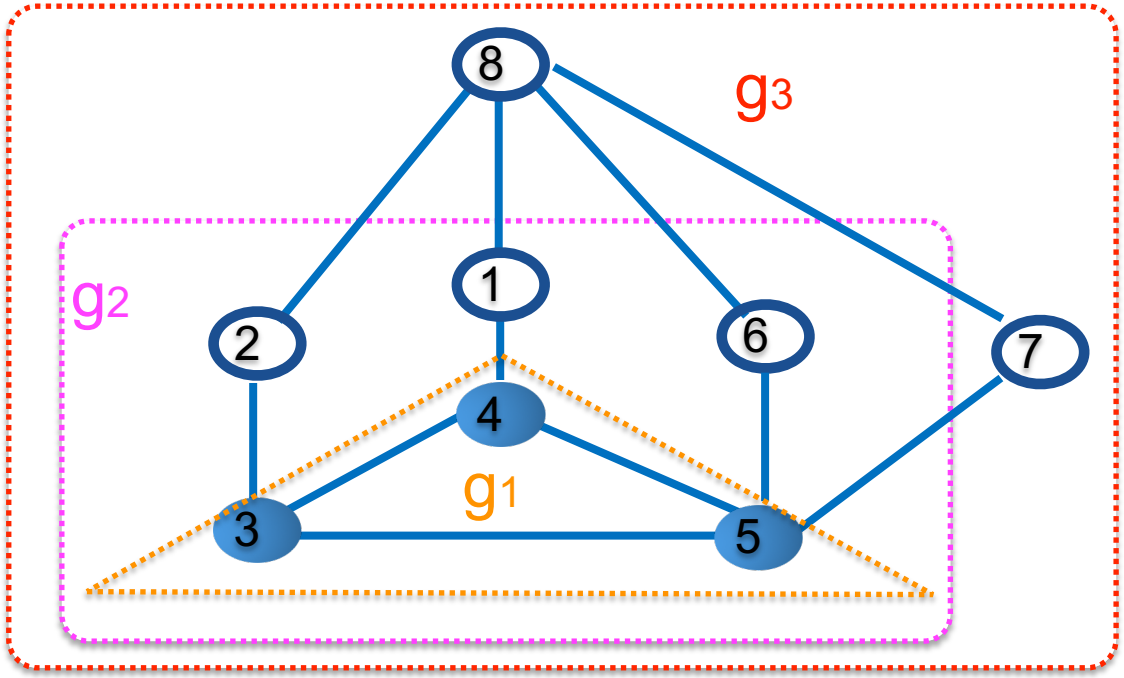
Steiner Maximum-Connected Component (SMCC) and Steiner-Connectivity



the SMCC of {3,4,5}

Challenges

- Challenge I: the subgraph induced by q or the entire graph may not be the solution since the connectivity of subgraphs is not monotonic .



Challenges

- Challenge-II: enumerating subgraphs requires exponential time.
  - Enumerate all subgraphs containing q, and choose the maximal one with maximum connectivity.

Contribution

- Problem 1: computing the SMCC of q.
  - Solve in time O (|r|), r is the result vertex set.
- Problem 2: computing the Steiner-connectivity of q
  - Solve in O(|q|)

Two Phase Computation:

Phase-I: offline index construction

- Step-1: compute $sc(u,v)$ for all edges (u,v) in G
- Step-2: compute max-spanning tree based on these $sc(u,v)$

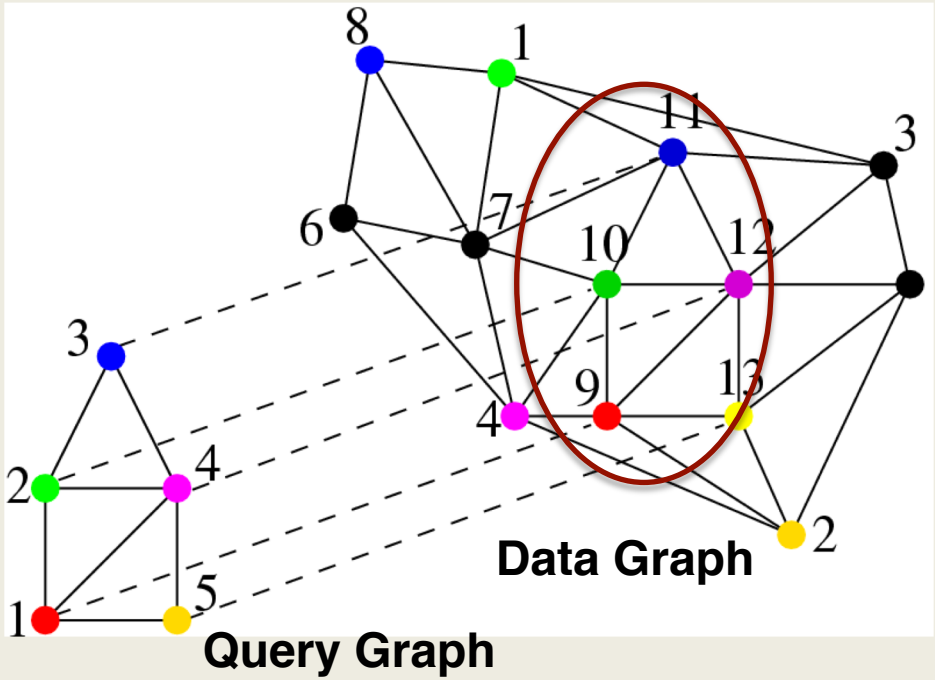
Phase-II: online query processing

- Step-1: compute $sc(q)$
- Step-2: start from any vertex $u \in q$ to extend result set to include all vertices v with $sc(u,v) \geq sc(q)$

Remark

- Optimal External-Algorithm?
  - Optimal I/O Semi External-Algorithms (VLDB Journal)
    - Not only vertices but also edges in components
  - Open for external-algorithms.

Exact Subgraph Match



Problem Definition:

- Given a query graph and data graph, find all subgraphs in the data graph that match the query graph.
- Data graph can be a large graph, or a set of medium-sized graphs.
- The vertices (edges) may or may not be labelled.

Applications:

Protein Interactions, Graph Classification, Chemical Compound search, Anomaly detection etc.

Challenges:

- Subgraph Isomorphism is NP complete.
- Graph data and results can be huge (MB-scale graph can produce PB-scale results).

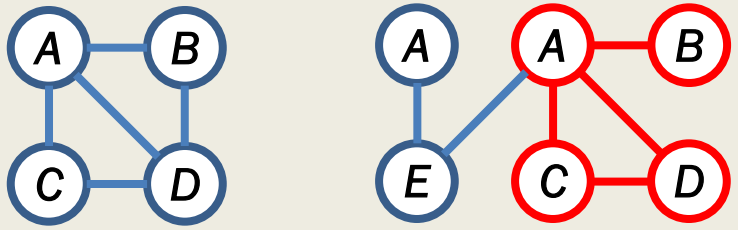
Algorithms and publications:

- Tree Sequence (QuickSI), VLDB'08
- Core-Forest-Leaf Decomposition (CFL Algorithm), Sigmod'16 (enumeration)
- Optimal Join-based algorithms, VLDB'15, VLDB'17

Contributions

- CFL is 100 times faster than the state-of-the-art，thousands of times faster than Neo4j
- Distributed join-based algorithm has high scalability, can process billion-scale graphs in a small cluster (10 computing nodes).

Approximate Subgraph Match



Query Graph *Data Graph*

(Allow one edge mis-matched)

There is no exact match in the above example. However, if it is allowed to mis-match one edge, we can find the red-marked match (miss (B, D) edge).

Problem Definition:

- Regarding exact matching, here we allow at most a certain number of mis-matched edges.

Applications:

Exact matching is too constrained, in addition, the data graph may be incomplete, and it may be unclear to users whether a query is exactly what they want.

Challenges:

- Harder than exact match (equal to processing multiple exact matches).
- The solution set can be huge.

Algorithms and publications

- Spanning-tree-based matching algorithm (TreeSpan), SIGMOD'12

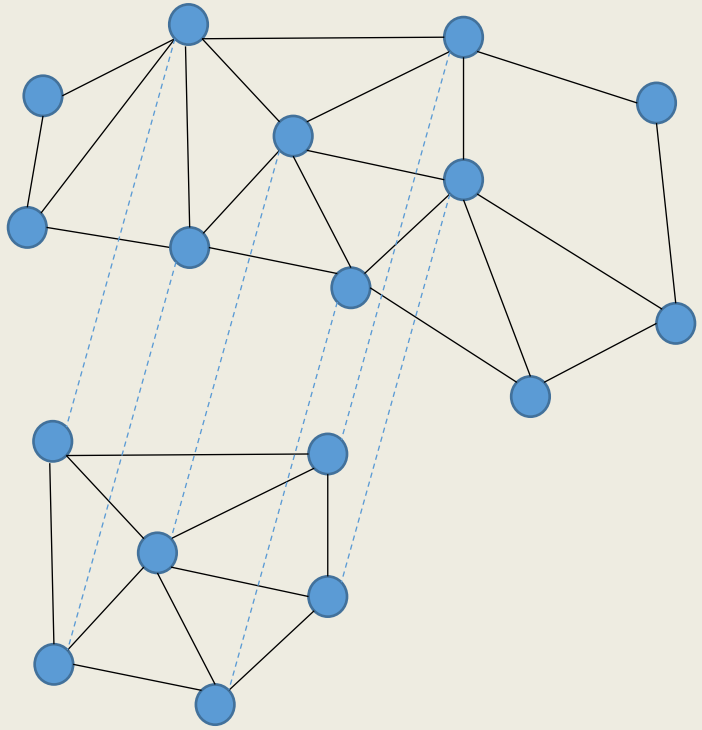
Contributions:

- Minimum-cost spanning tree index
- Targeting maximum match to reduce the size of solution set
- 4 orders of magnitude faster than the state-of-the-art

Supergraph Match

Query Graph

Data Graph



Problem Definition:

- Given a query graph and a graph database, search all graphs in the database that are contained by the query graph.
- Can be seen as the transposed query of subgraph pattern matching.

Applications:

Protein-protein interaction analysis, Chemical compound search

Challenges:

- Complexities: Subgraph isomorphism, NP complete
- Data scale：up to the scale of hundreds of vertices, compared to tens in the state-of-the-art

Algorithms and publications:

- Tree-index-based supergraph search algorithm, DGTree，ICDE’16

Contributions:

- Propose the index called DGTree that is free from the costly subgraph mining procedure.
- Based on DGTree, propose the algorithm that is free from verification
- 8 times faster than the state-of-the-art

COHESIVE SUBGRAPHS

k-Core, k-Truss, k-Edge Connected,
k-Vertex Connected, Cliques

Cohesive Subgraph Search: summsry

- Related Applications
  - Product Recommendation, Investment Analysis etc.
- Algorithms and Recent Publications
  - Clique
    - In-memory approximate algorithm, ICDE 2015
    - Graph partition based I/O efficient search algorithm, VLDBJ 2016
  - K-ECC
    - Graph decomposition based algorithm , SIGMOD 2013
    - I/O efficient algorithm to compute the k-edge connected component for all k values, VLDBJ 2015
  - Steiner Maximum-Connected Component
    - Index-based in-memory algorithm, SIGMOD 2015
    - Index-based semi-external algorithm, VLDBJ 2017

Cohesive Subgraph Search

- Algorithms and Recent Publications
  - K-Core
    - Pruning based search algorithm, ICDE 2018
    - upper and lower bound based pruning rules on uncertain graph, ICDE 2018
    - I/O efficient core decomposition, ICDE 2016 (Best Paper Award).
  - Critical Users
    - Iteratively find a best user to anchor based on the heuristics, such that the corresponding subgraph (e.g., k-core) is the largest, VLDB 2017, ICDE 2018.
    - Iteratively find a best user and delete, such that the corresponding k-core is the smallest, AAAI 2017.
  - Multi-dimensional Subgraph Search
    - Solid pruning rules based algorithm on search tree with fast search orders, VLDB 2017.

I/O Efficient Core Graph Decomposition at Web Scale

ICDE2016 Best Paper Award

Definition k-Core

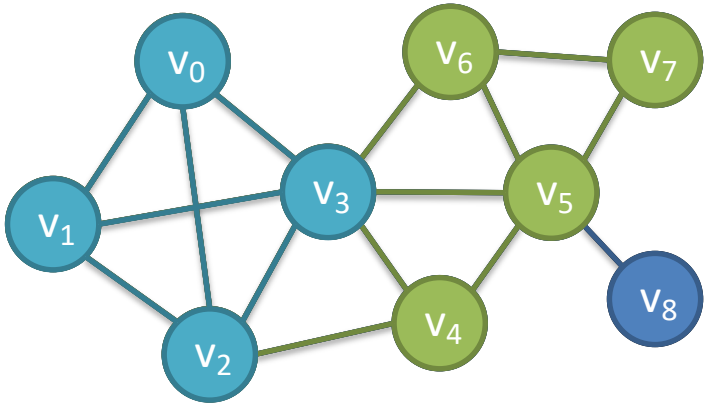
The k-core of graph G is a maximal subgraph in which every node has a degree of at least k.

Definition Core Number

Given a graph G, the core number of v is the largest k, such that v is contained in a k-core.

Problem Statement

Given a graph G, Core Decomposition computes the the k-cores of G for all $1 \leq k \leq k_{max}$



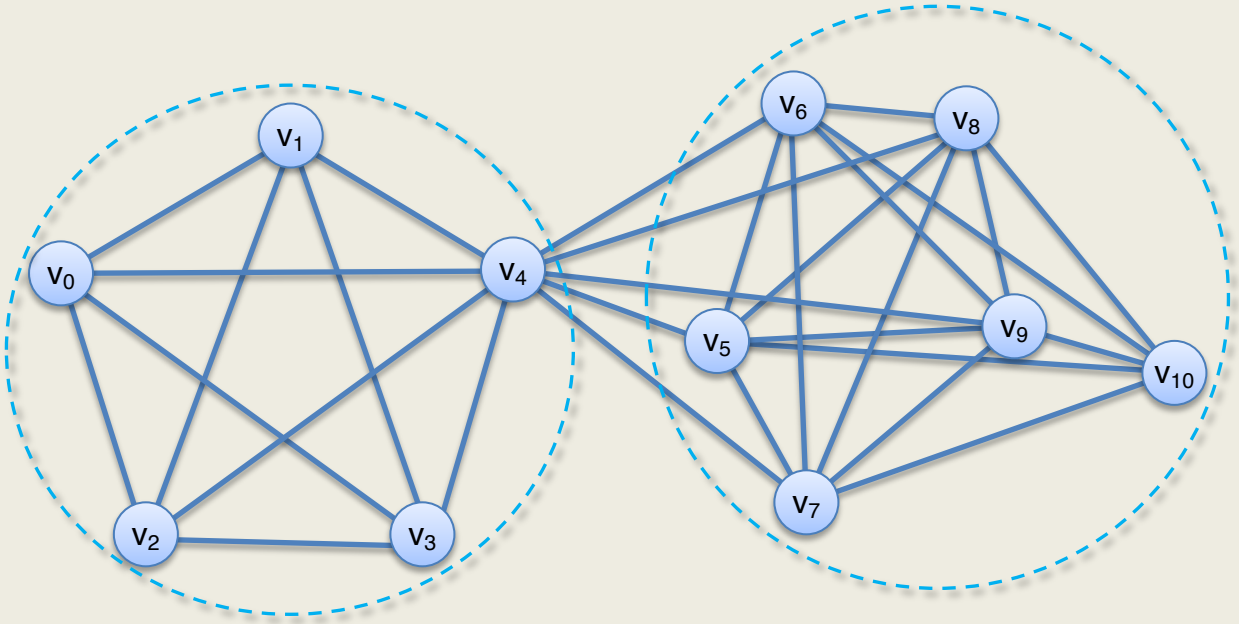
Core Decomposition of Graph G

1-Core = { v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8 }

2-Core = { v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7 }

3-Core = { v_0, v_1, v_2, v_3 }

Diversified Top-k Clique Search



Maximal Clique:

- A maximal subgraph such that there exists an edge between any two nodes

Problem Definition:

- Find k maximal cliques such that the nodes covered by the returned cliques are maximum

Applications:

- Community detection in social networks
- Anomaly detection in complex networks
- Gene expression and motif discovery in molecular biology

Challenges:

- NP-Hard
- Huge memory consumption

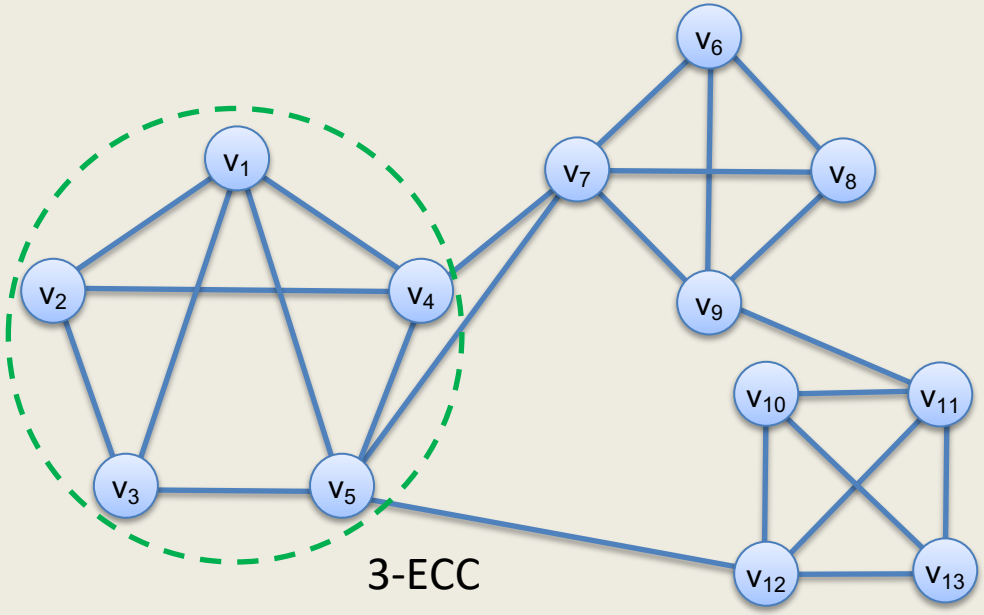
Publications:

- In-memory approximate algorithm, ICDE 2015
- Graph partition based I/O efficient search algorithm , VLDBJ 2016

Contributions:

- Our algorithm is 20x faster than the state-of-the-art

K-Edge Connected Component Search



K-Edge Connected Component:

- K-edge Connected：a graph is connected after the removal of any (k-1) edges
- Maximal subgraph which is k-edge connected

Problem Definition:

- Given a graph G and an integer k, computes the k-edge connected component of G

Applications:

- Social behaviour mining
- Community detection
- Graph visualization

Challenges:

- Input graph is very large

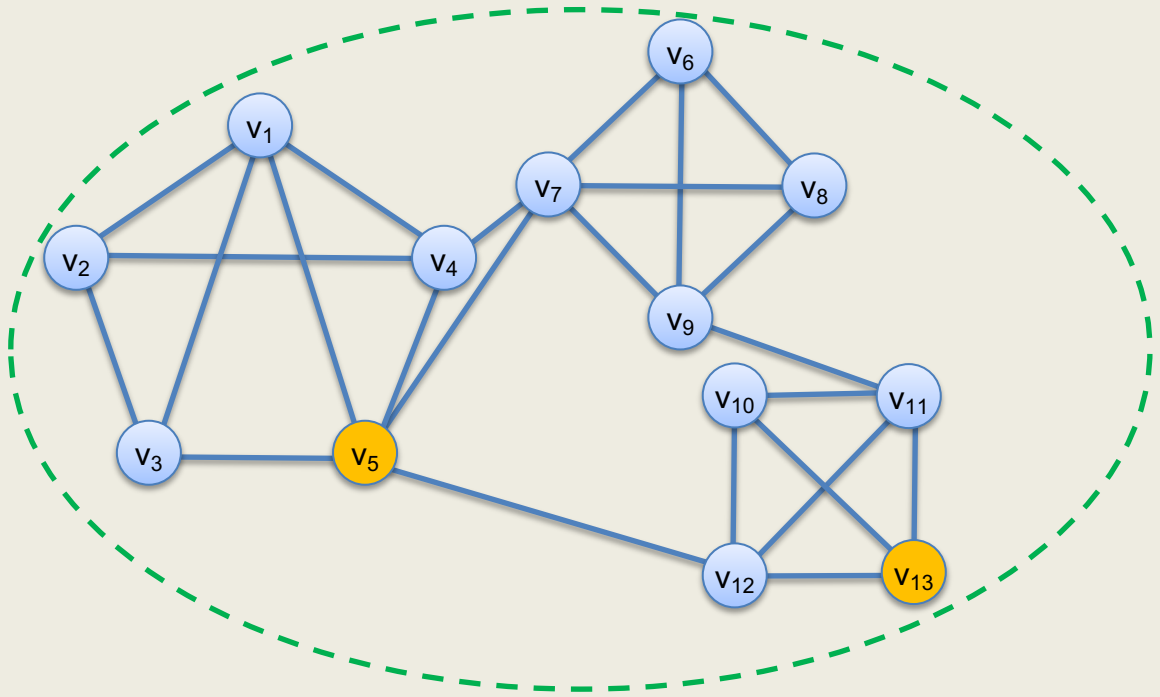
Publications:

- Graph decomposition based algorithm ，SIGMOD 2013
- I/O efficient algorithm to compute the k-edge connected component for all k values， VLDB 2015

Contributions:

- Our in-memory algorithm outperforms the state-of-the-art by several orders of magnitude
- I/O efficient algorithm can handle billion-edges scale graphs on a single consumer grade computer

Steiner Maximum-Connected Component Search



Problem Definition:

- Given a set of nodes in G, compute the k-edge connected component with the maximum k value that contains the given nodes (Steiner Maximum-Connected Component)

Applications:

- Potential customer prediction
- Product promotion
- Research team assembling

Challenges:

- Input graph can be very large

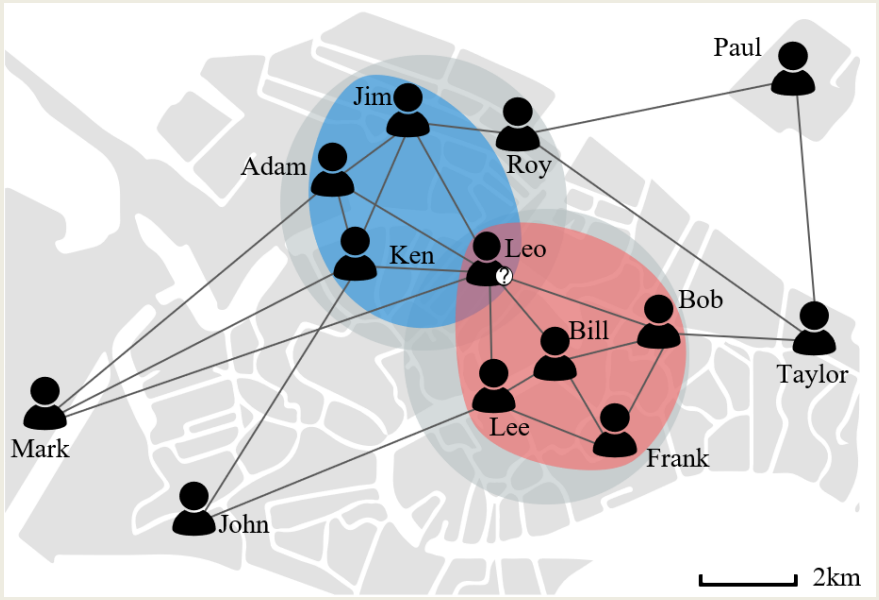
Publications:

- Index-based in-memory algorithm , SIGMOD 2015
- Index-based semi-external algorithm , VLDBJ 2017

Contributions:

- Our in-memory algorithm outperforms the state-of-the-art by several orders of magnitude
- I/O efficient algorithm can handle billion-edges scale graphs on a single consumer grade computer

Efficient Computing of Radius-Bounded *k*-Cores



Problem definition:

- Given a graph *G*, the *k*-core of *G* is a maximal subgraph where each node has at least *k* neighbors in the subgraph.
- Given a geo-social network (as shown above), a query vertex *q* (Leo), a radius *r* (2km) and *k* (3), find *k*-cores containing the query vertex *q* covered by circles with radius *r*.

Applications:

- Personalized event recommendation
- Social marketing

Challenges:

- The location of the radius-bounded circle of a RB-*k*-core is unknown.
- It is time-consuming to verify all the candidate subgraphs individually.

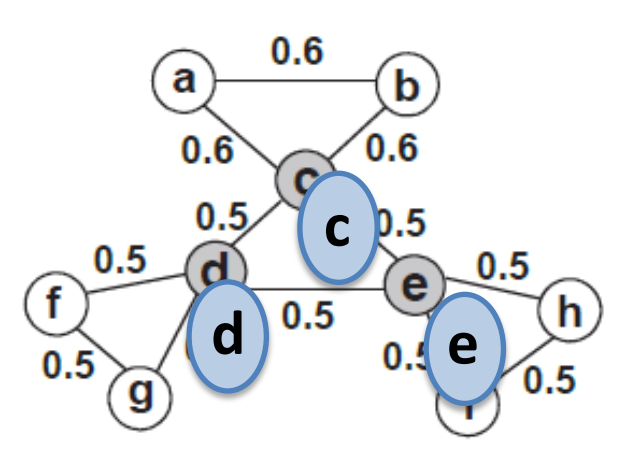
Publications:

- Pruning based search algorithm, ICDE 2018

Contributions:

- Our techniques can be applied to solve the SAC search problem published in VLDB 2017 and achieve a speed-up twice.

K-Core Search on Uncertain Graphs



Problem Definition：

- Given an uncertain graph(edge with probability), find a vertex set where the probability of every node to be in a k-core is no less than θ(given by users)
- Used in undirected graph and easily to extend to directed graph

Example：

- P(d is not a 2-core) = P(d not in 2-core(d,f,g) and not in 2-core(c,d,e)) = 0.766
- (c,d,e) is the result set of k=2,θ=0.23

Applications：

- Cohesive subgraph detection
- Vital vertices detection
- Influential communities detection on social network

Hardness:

- NP-hardness problem
- Prohibitive to enumerate all possible result sets

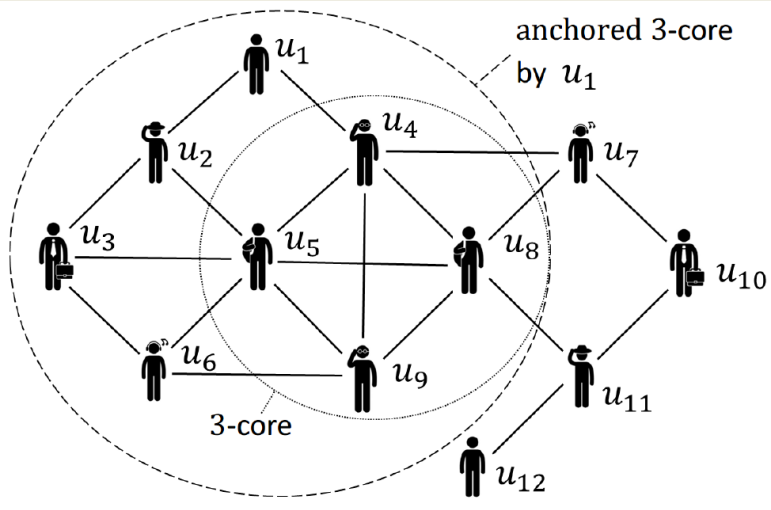
Algorithm and Published Paper:

- Search algorithm based upper and lower bound pruning rules, ICDE 2018

Contributions：

- Advanced algorithm about one order faster than baseline on average
- Propose a new k-core model on uncertain graph, where result sets outperform previous models on average influences test of social networks

Find Critical Users to Prevent Network Unraveling



Example：

- A vertex represents a user.
- An edge represents there is a relationship between two users.
- The number of vertices in 3-core represents the stability of network.
- Encourage the engagement of u_1 can greatly enlarge the 3-core, as the anchored 3-core by u_1 shows.

Applications：

- Efficiently find the critical users to reinforce the network.

Algorithms and Publications：

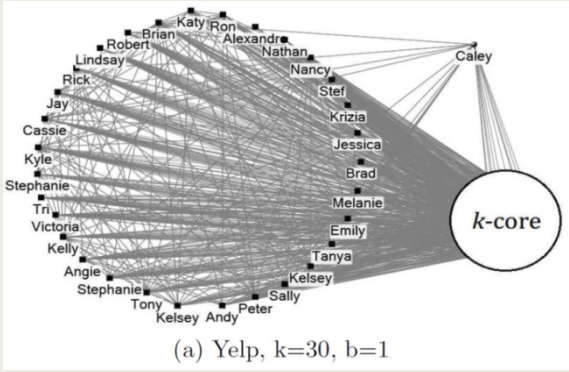
- Iteratively find a best user to anchor based on the heuristics, such that the corresponding subgraph (e.g., *k*-core) is the largest, VLDB 2017, ICDE 2018.

Contributions：

- The proposed algorithms outperform the state-of-the-art algorithms by 3 orders of magnitude in runtime.

Problem:

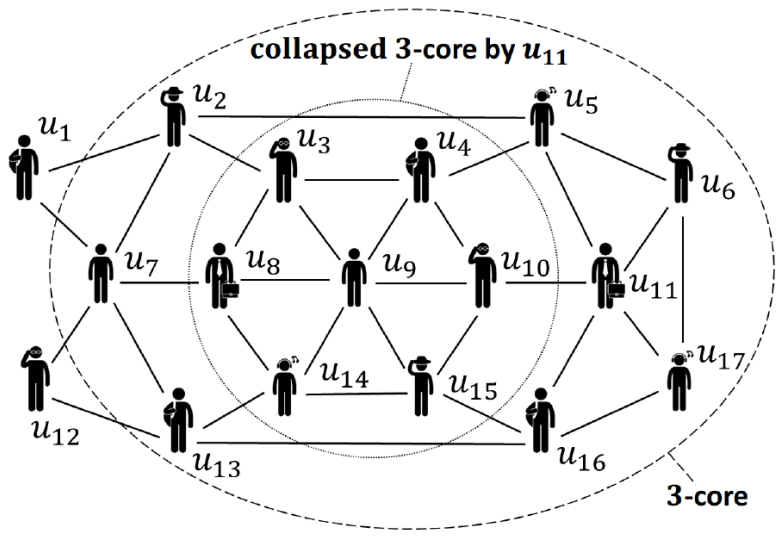
- *k*-core: a maximal subgraph where each vertex is adjacent to at least *k* vertices in the subgraph.
- Given a network *G*, find *b* critical users whose engagement can significantly prevent network unraveling, i.e., find *b* vertices whose persistent existence leads to the *k*-core with the largest number of vertices.
- NP-hard, NP-hard to approximate.



(a) Yelp, k=30, b=1

- In Yelp, the engagement of Caley can enhance the engagement of other 31 users.

Find Critical Users to Reinforce Communities



Example:

- A vertex represents a user.
- An edge represents there is a relationship between two users.
- The number of vertices in 3-core represents the stability of network.
- The leave of u_{11} can greatly collapse the 3-core, as the collapsed 3-core by u_{11} shows.

Applications：

- Efficiently find the critical users to reinforce the communities.

Algorithms and Publications：

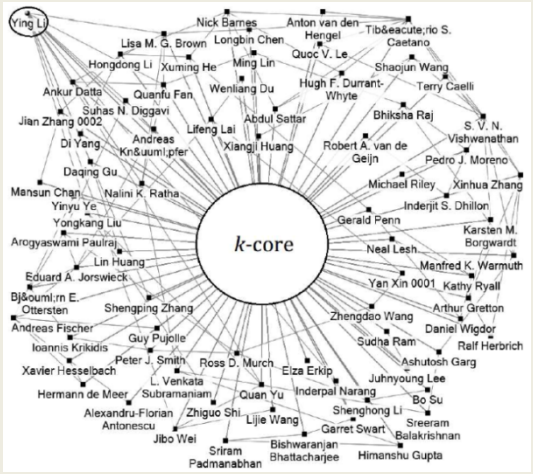
- Iteratively find a best user and delete, such that the corresponding *k*-core is the smallest, AAAI 2017.

Contributions：

- Our algorithm outperforms the state-of-the-art algorithm by 4 times in runtime.

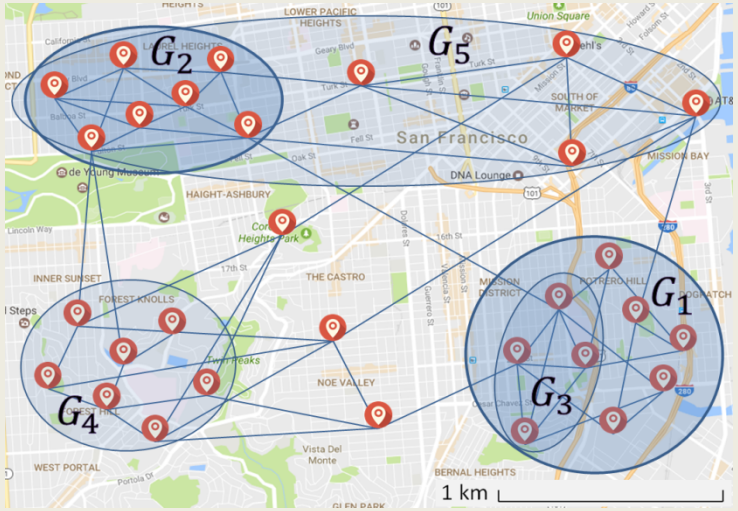
Problem:

- *k*-core: a maximal subgraph where each vertex is adjacent to at least *k* vertices in the subgraph.
- Given a network *G*, find *b* critical users whose engagement can significantly reinforce the communities, i.e., find *b* vertices whose leave leads to the *k*-core with the smallest number of vertices.
- NP-hard.



- In DBLP, the co-author network, the leave of Ying Li will break the engagement of other 74 users (who also leave the k-core).

Multi-dimensional Subgraph Search



Example:

- A vertex represents a user with her location.
- An edge represents there is a relationship between two users.
- A maximal (*k*,*r*)-core requires each user has at least *k* friends in the core and every user pair is similar, such as G_1 or G_2.

Problem:

- Given an attributed graph *G*, find every subgraph satisfying the following conditions: (1) a vertex inside is adjacent to at least *k* other vertices inside; (2) every vertex pair inside is similar; (3) the subgraph is maximal, i.e., no supergraph of it can satisfy conditions 1 and 2.

Applications:

- Find all the communities where users are cohesive and similar.
- Applicable to all types of attribute, such as locations and hobbies.

Complexity:

- NP-hard.

Algorithms and Publications:

- Solid pruning rules based algorithm on search tree with fast search orders, VLDB 2017.

Contributions:

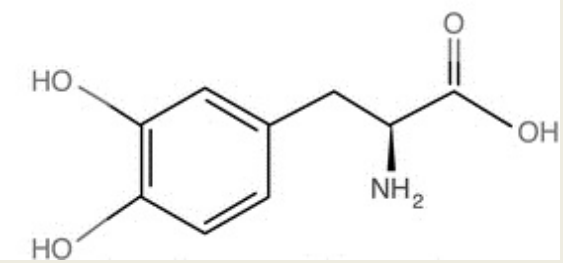
- The proposed algorithms can find all the required communities within 100 seconds on networks with over 1 million vertices.

GRAPH SIMILARITY AND CLUSTERING

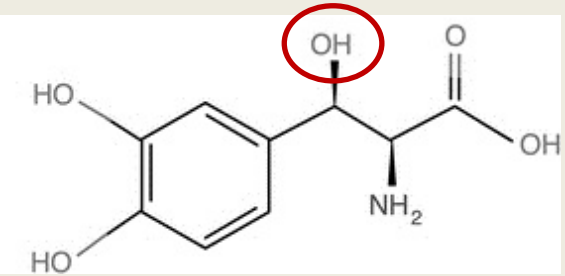
Graph Similarity and Clustering: Summary

- Related Applications
  - Product Recommendation, Investment Analysis etc.
- Algorithms and Recent Publications
  - Graph Edit Distance
    - Path-match-based filtering algorithm, ICDE'12
    - Subgraph-match-based filtering algorithm, VLDB'14
  - SimRank
    - Adjustable clustering strategy for SimRank (OIP-SR), ICDE'13
    - HyperLink-based SimRank compute (MEMO-SR), VLDB'13
    - Incrementally SimRank compute (Inc-SR), ICDE'14
  - Graph Clustering
    - Core clustering and non-core clustering (pSCAN), ICDE'16
    - Dynamic maintenance of clustering structures (dSCAN), TKDE'17
    - Similarity-sequence-index-based algorithm (GS-SCAN) VLDB'18

Graph Similarity - Edit distance



Levodopa



Droxidopa

Problem Definition:

- **Edit：for Vertex: Add/Delete/Alter Label;**
  for Edge: Add/Delete/Alter Label;
- Given a graph pair (r, s), their **edit distance** is defined as the minimum edit operations to turn r into s.
- Given two graph sets (R, S), searching for all pairs of (r, s) such that their edit distance is within a given threshold.

Applications:

- Chemistry: Find similar organic compound.
- Biology: Find similar protein-DNA interactions.
- Finger print recognition: Identify criminal suspect.

Challenges:

- Graph edit distance problem is NP hard.

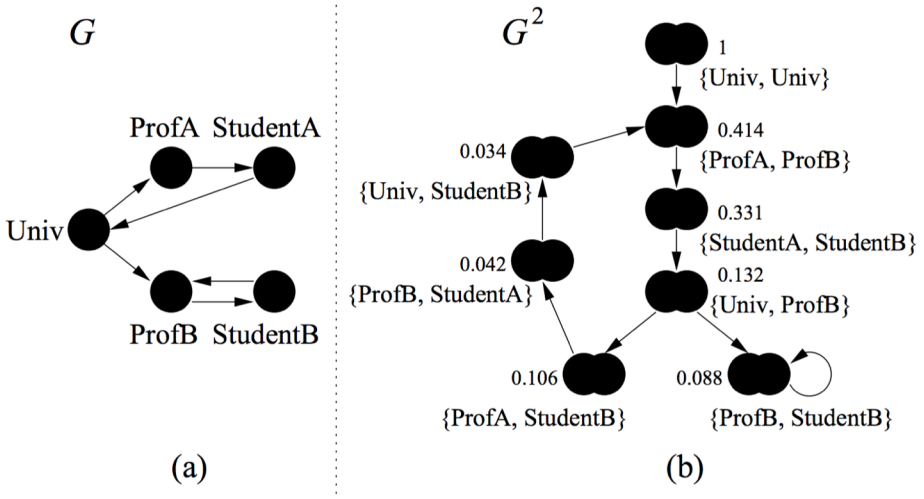
Algorithms and publications

- Path-match-based filtering algorithm, ICDE'12
- Subgraph-match-based filtering algorithm, VLDB'14

Contributions:

- Subgraph-match-based algorithm is 40 times faster than previous work.

Graph Similarity - SimRank



Problem definition:

- In terms of structure, two vertices are similar when their neighbours also similar.
- Given two vertices a, b, and their in neighbours, I(a), I(b), we can compute their SimRank as:

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

Applications:

Collaborative filtering, Page rank, Graph clustering, Link prediction

Challenges:

- Hard to find proper ranking metrics in practice.
- The complexity is O(Kmn), where K is the iteration, m is the number of edges, and n is the number of vertices.
- Hard to incrementally update while graph data changes

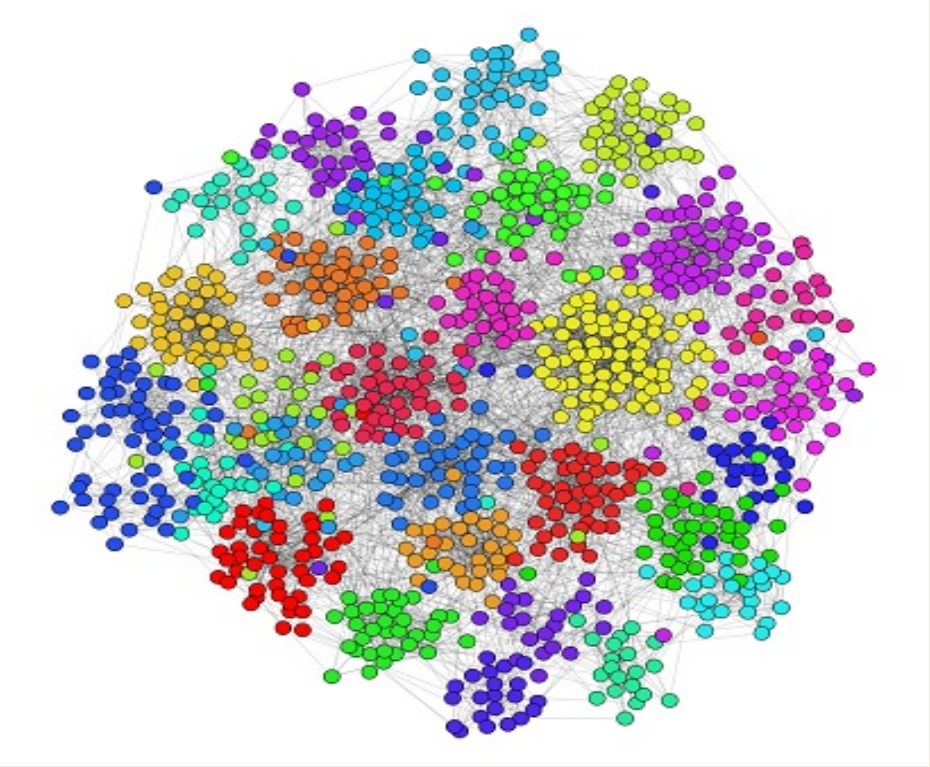
Algorithms and publications

- Adjustable clustering strategy for SimRank (OIP-SR), ICDE'13
- HyperLink-based SimRank compute (MEMO-SR), VLDB'13
- Incrementally SimRank compute (Inc-SR), ICDE'14

Contributions:

- OIP-SR improves the complexity, with 10 times of performance gain.
- MEMO-SR enriches the semantics of SimRank.
- Inc-SR can apply to dynamic graph.

Graph Clustering



Problem Definition

- Given a graph, divide the vertices to different groups based on their similarities.

Applications:

Community detection, Graph partition, Graph visualisation, Hidden structure detection, Gene array.

Challenges:

- Costly to compute pair-wise similarities.
- Parameters are hard to adjusted.
- Graph data is large, and evolving.

Algorithms and publications:

- Core clustering and non-core clustering (pSCAN), ICDE'16
- Dynamic maintenance of clustering structures (dSCAN), TKDE'17
- Similarity-sequence-index-based algorithm (GS-SCAN) VLDB'18

Contributions:

- pSCAN is over 10 times faster than the state-of-the-art (SCAN++)
- GS-SCAN proposes a novel data structure, resulting in quite clustering regardless of the parameters, and is 10-1000 times faster than pScan.

DISTRIBUTED GRAPH COMPUTING

Distributed Graph Computing - SGC Model

Scalable Graph Computing (SGC)：

- m: #edges，n: #vertices，t: #machines
- A distributed graph algorithm belongs to SGC if it satisfies:

Metrics	per machine	Total
Disk	$O(m/t)$	$O(m)$
Mem	$O(1)$	$O(t)$
Comm.	$O(m/t)$	$O(m)$
CPU	$O(m/t)$	$O(m)$
Iteration	$\log(n)$	

Applications:

- The SGC algorithms can easily scale out, that is the performance of the algorithm increases nearly linearly with the number of machines

Challenges:

- Hard to design a SGC algorithm, especially targeting O(logn) iterations

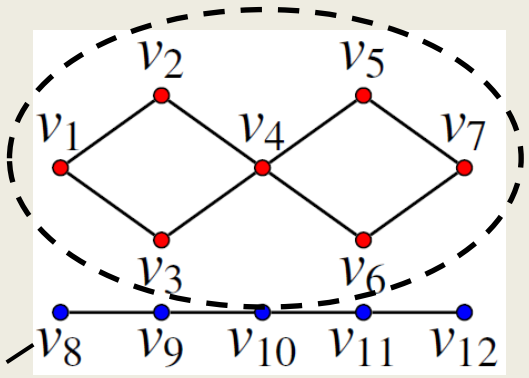
Algorithms and publications:

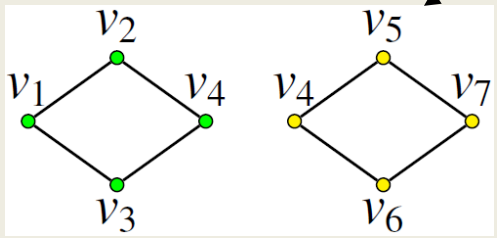
- Scalable Graph Computing Model (SGC), SIGMOD'14

Contributions:

- Design and implement two SGC operators, N and E
- Implement many graph algorithms using N and E in MapReduce
- Empirically prove that it reaches better scalability than the state-of-the-art

Distributed Graph Computing – Connected Component





Problem definition:

- Connected component (CC): the maximal subgraph that is connected
- Double connected component (DCC): the maximal subgraph that is a connected component after removing any vertex.
- Given a large graph, compute CC and DCC

Applications:

- CC is the initial steps of many graph computations.
- Aid the analysis of big graphs, help to reveal the anchor vertices.

Challenges:

- Big graph (billion-scale vertices)

Algorithms and publications:

- CC based on graph decomposition and DCC based on label propagation，ICDE'16

Contributions:

- Novel algorithmic framework that reduces a lot of communication
- 50 times faster than the state-of-the-art

Big Graph

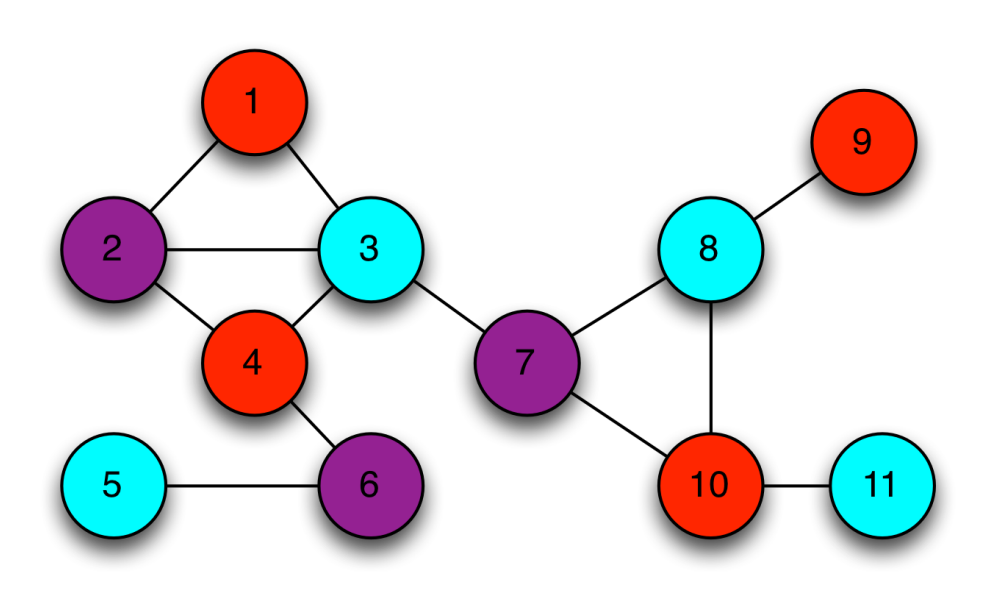
- Algorithms and Recent Publications
  - Distributed Graph Computing Model
    - Scalable Graph Computation Model, SIGMOD’14
  - Distributed Connected Component
    - CC based on graph decomposition and DCC based on label propagation，ICDE’16
  - Distributed Scalable Subgraph Enumeration
    - Optimal Distributed Join-based algorithms, VLDB’15, VLDB’17

Other Graph problems

Other Graph Problems – Coloring, Eccentrality, Shortest Path, etc.

- Related Applications
  - Product Recommendation, Investment Analysis, Anti Monty Laundering, Retail Service etc.
- Algorithms and Recent Publications
  - Graph Coloring
    - Dynamic graph colouring based on local update, VLDB 2018
  - Eccentricity computation
    - Local search algorithm based on lower and upper bounds, ICDE 2018
  - Shortest Path
    - Two-hop-labelling shortest path query on road network, SIGMOD 2018
    - Efficient Top-k shortest path join, EDBT 2015

Graph Colouring



Graph colouring:

- Colour the graph with minimum colours such that any two neighbouring vertices bear different colours.

Problem definition:

- Process graph colouring when the graph is constantly changing (insertion and deletion)

Applications:

- Channel assign in wireless network
- Airline control and management
- Community detection in social network
- An initial step for the other graph, such as maximum clique and minimum cut

Challenges:

- NP hard
- Large graph data
- Frequent update

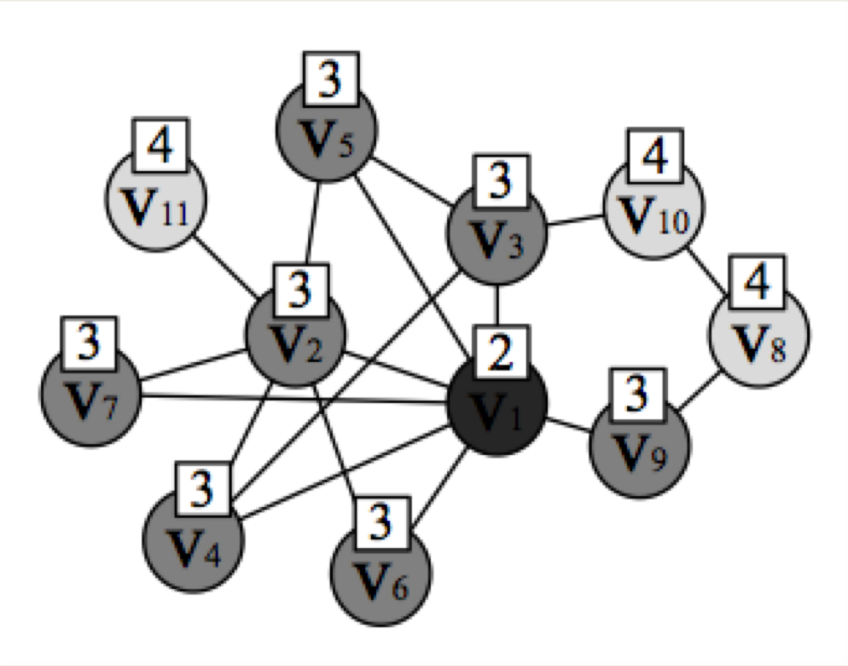
Algorithms and publications:

- Dynamic graph colouring based on local update ， VLDB 2018

Contributions:

- The algorithm uses ½ less colours than previous work, and offers ms response to graph update.

Eccentricity computation



Example：

- V8‘s eccentricity is 4, as longest shortest distance from V8 is 4, towards V11.
- The smaller the eccentricity is, the more centric (darker) the vertex is.

Problem definition:

- The eccentricity of a vertex in the graph is the longest shorted distance this vertex can reach the other vertices.
- It requires to solve APSP problem for computing the eccentricity of all vertices in the graph, while the error rate is high in small-world graph model.

Applications:

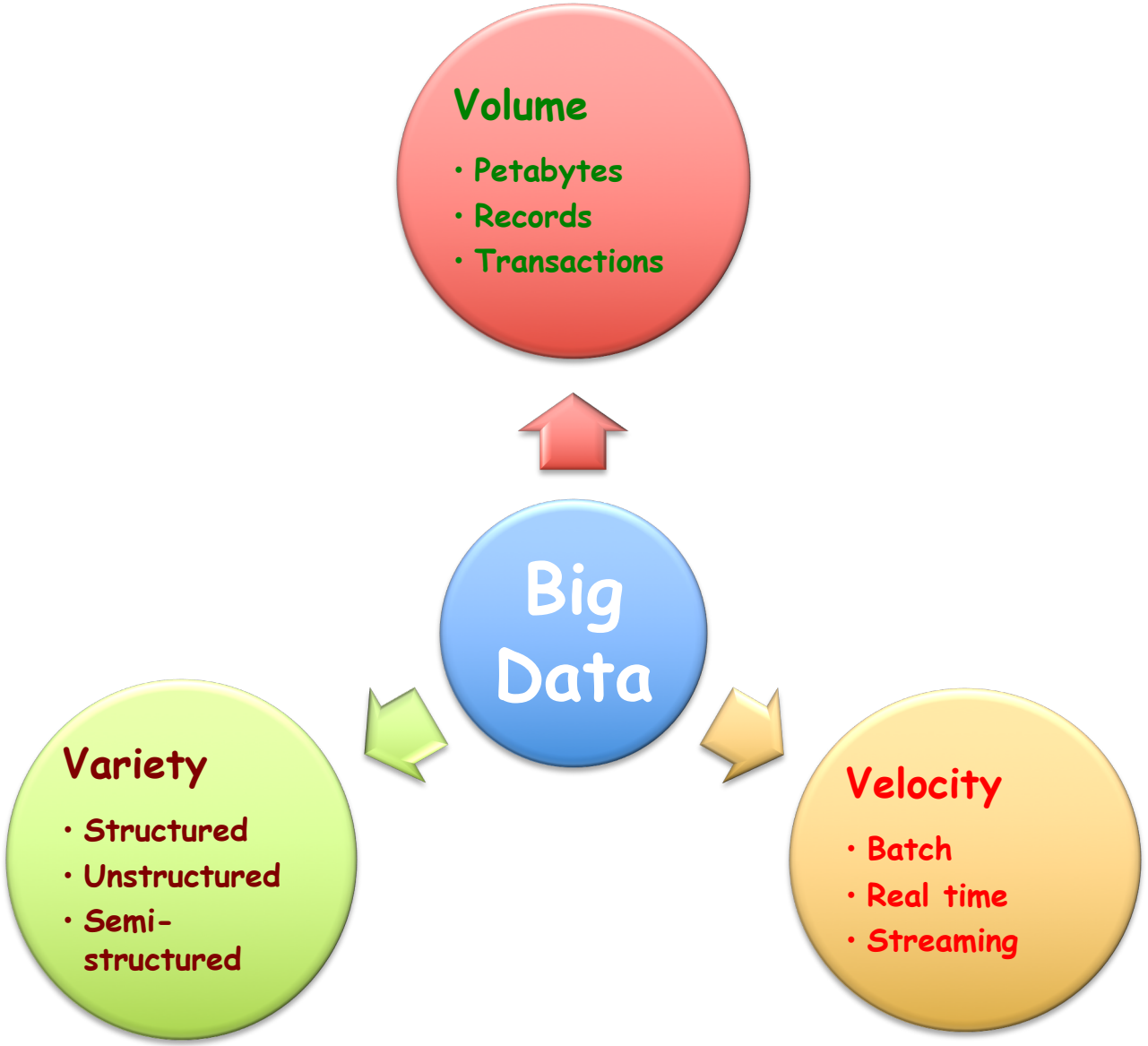
- Vertex ordering (via their importance)
- Compute some other graph features (diameter, radius etc.)

Algorithms and publications:

- Local search algorithm based on lower and upper bounds, ICDE 2018

Contributions:

- Three orders of magnitude faster than previous work.


Volume
• Petabytes
• Records
• Transactions
Big Data
Variety
• Structured
• Unstructured
• Semi-structured
Velocity
• Batch
• Real time
• Streaming

Major Research Issues

New Computing Platform/Architecture

New Graph Analytics Models

New Processing Algorithms & Indexing Techniques

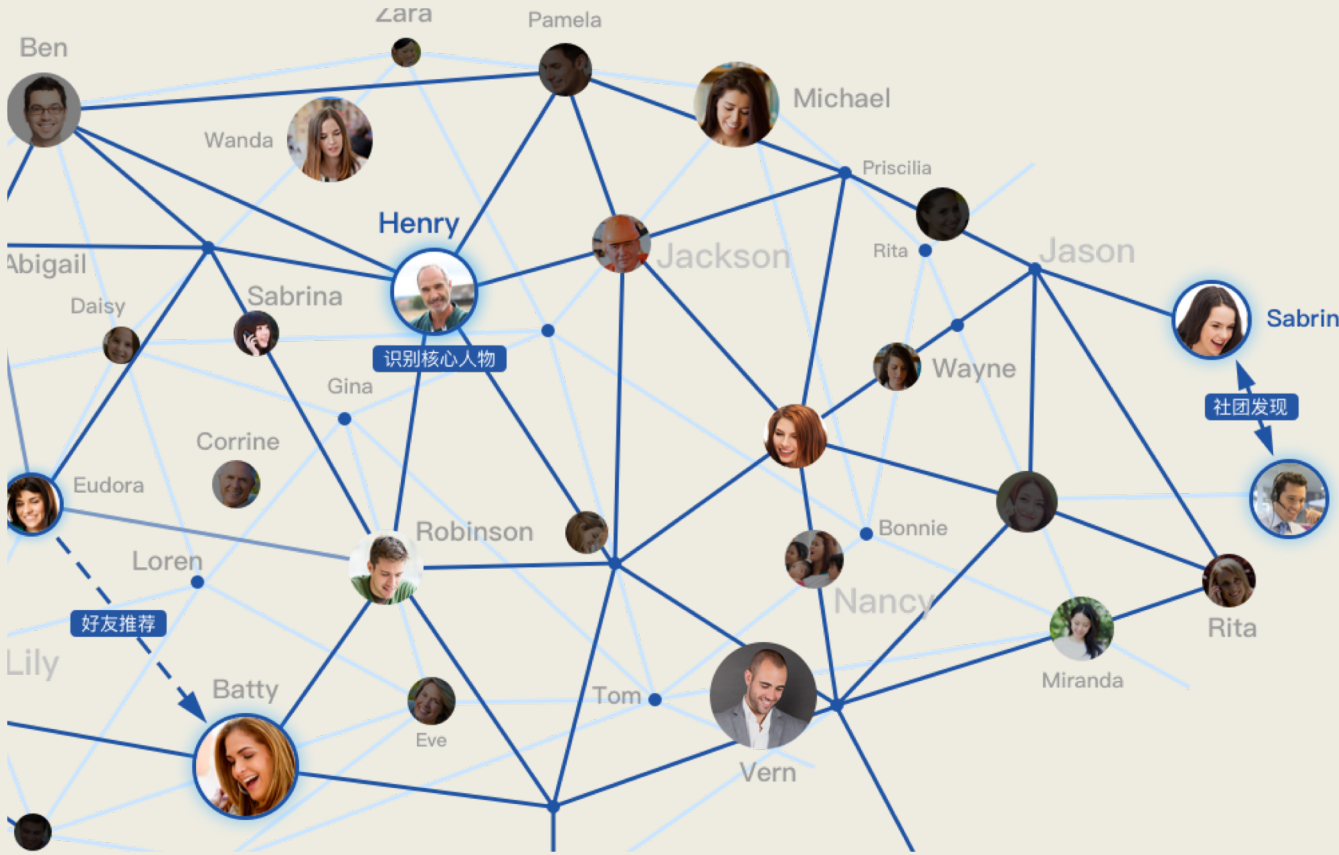
Graph Processing System

- Primitive operators
- Query language
- Distributed techniques
- Storage
- etc

Research Collaborations: Huawei Cohesive Subgraph Exploration

- Cohesive subgraph：
  - Subgraphs whose internal vertices are densely connected, while there are few edges in between.
  - Core member detection, friend recommendation, community detection.

- Main Challenges：
  - Big Graph Data, and computing complexities
  - High-dimensional graph data: Heterogeneous edges
  - Dynamic Graph: Graph data is evolving



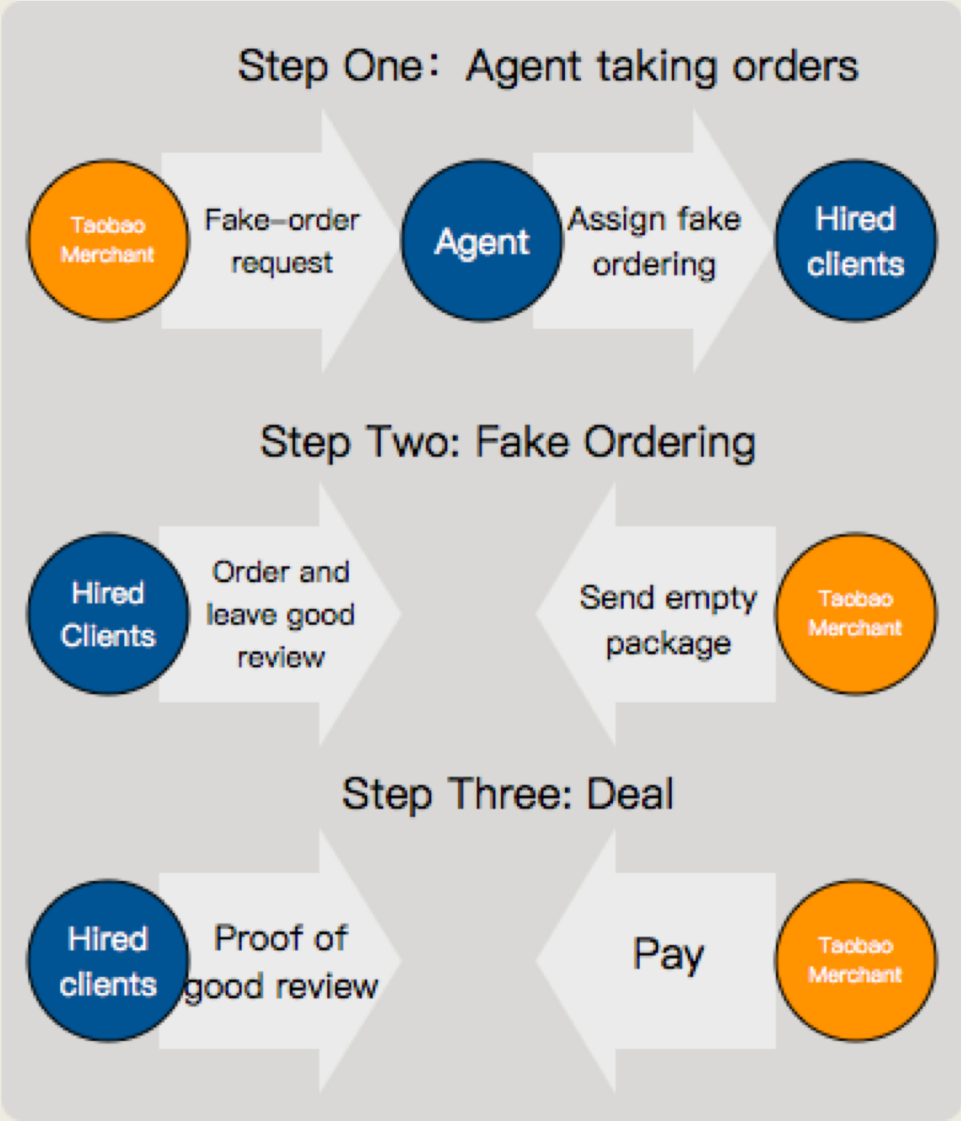
One of the applications on Huawei public cloud

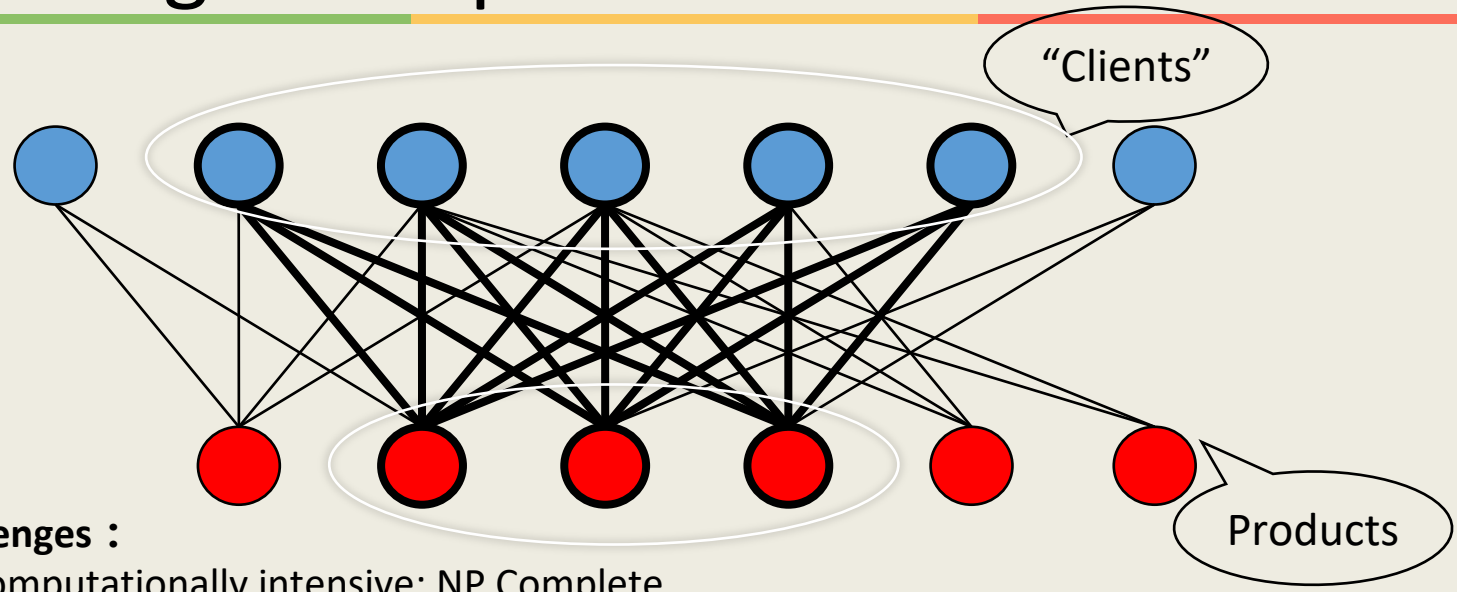
Alibaba Big Graph Computing - FLASH Language

- FLASH Query Language：
  - Only three operators: **F**iler, **L**oc**A**l, Pu**SH**
  - Great expression power: Capable of expression over 100 graph computing cases.
  - Convenient Programming: Programming most cases within 10 lines.

- Progress：
  - Prototyping system are deployed for production use (in distributed context)
  - System optimizations and tunings

Alibaba Big Graph Computing - Biclique Fraud detection





Challenges：

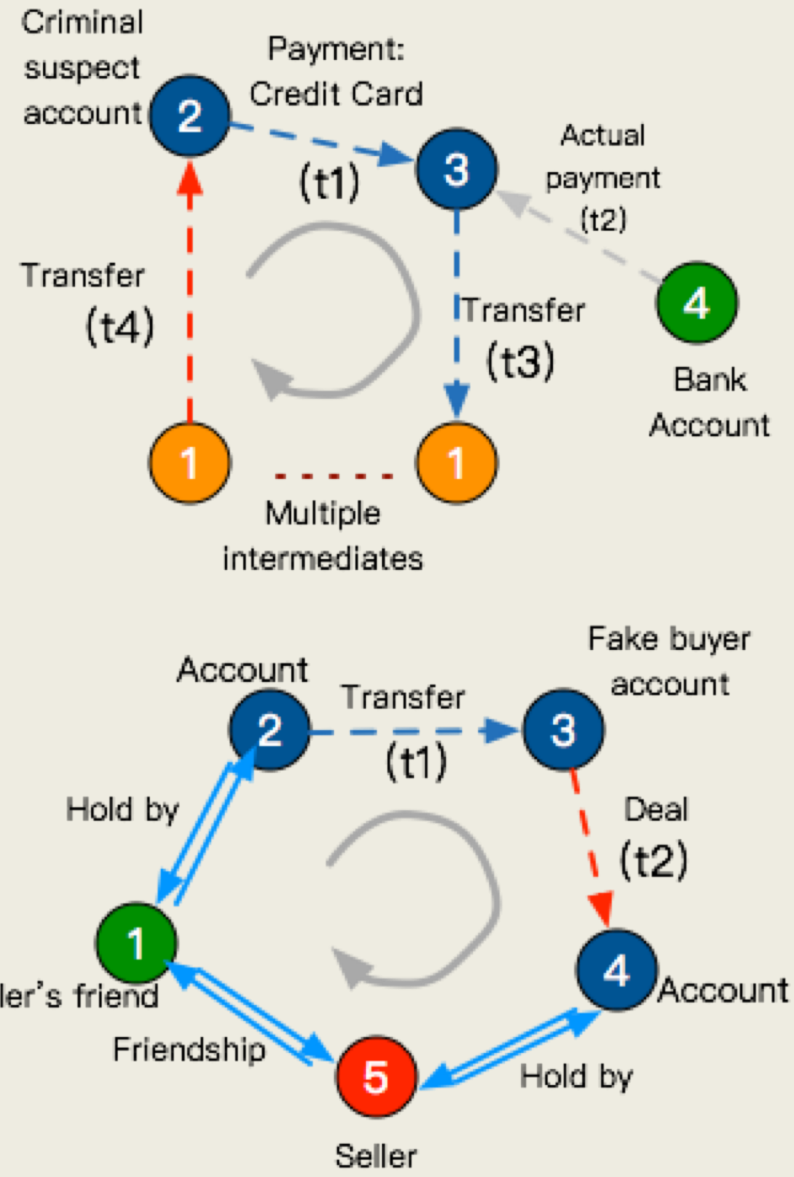
- Computationally intensive: NP Complete
- Big Data：100-million-scale users and products, billion-scale purchasing edge
- Dynamic Data: Frequent buying and selling on Taobao

Solutions: In the client-product bi-graph, fake ordering is modelled as a bi-clique, which infers that there exists a potential fake ordering when a group of people simultaneously purchase a set of products.

Alibaba's "Double 11" shopping festival use case：

- Efficiency: Reduce the computation on billion-scale graph from days to hours
- Effectiveness: Recall over 60% issued deals, improved traditional approach by 40%

Alibaba Big Graph Computing- RT Cycle Detection



Applications：

- Fraud detection
- Money laundering detection

Challenges:

- Large-scale dynamic graph (100-million-scale vertices, billion-scale edges)
- High demand on real-time processing: 10-50ms response time, while previous system takes 50s

Solutions:

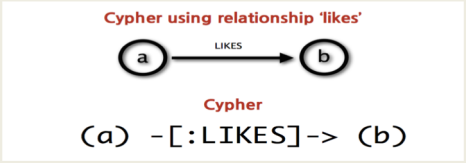
- Real-time cycle detection on each edge
- Light-weighted index structure (easy to maintain and update) and efficient query algorithm

Real-data simulation：

- Detect 6-edge cycles within 10ms

Graph System

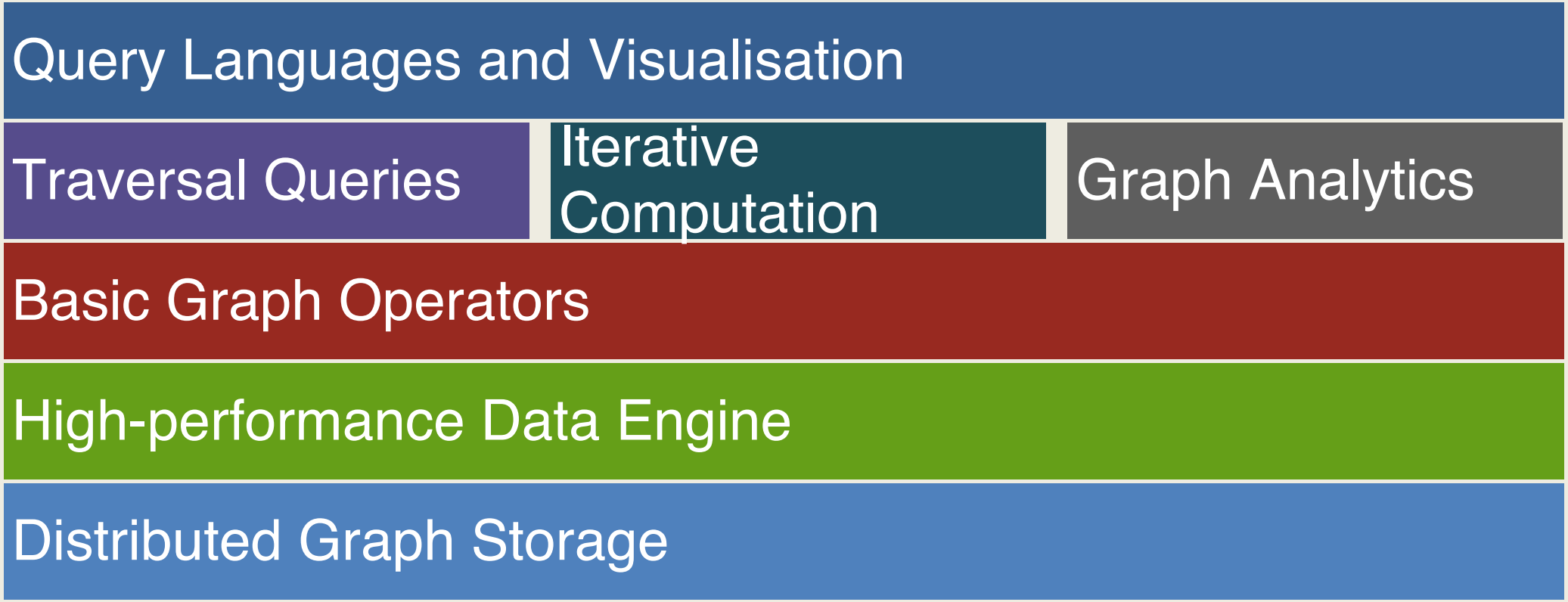
- Graph system should support
  - **Traversal Queries**
    - Explore the neighbouring information of given nodes in **real time**.
  - **Iterative Computation**:
    - BFS, Page rank, Shortest path, Connected component, Label propagation etc
  - **Graph Analytics**
    - Graph pattern matching, Community detection, Similarity, Graph Clustering etc.



Graph System

- Existing systems support each type of query very efficiently, but none covers the whole picture
  - **Combine several systems**
    - Complexities of maintenance are rising
    - Overheads of data transformation
  - **Simulate graph processing in general engine**
    - Spark's GraphX, Flink's Gelly, etc.
    - The performance is not competitive to the graph-specific systems

Graph Processing System Development



References:

1. F. Bi, L. Chang, X. LIN, W. Zhang, An Optimal and Progressive Approach to Online Search of Importance-based Top-K Communities, to appear in VLDB2018.
2. B. Lv, L. Qin, X. LIN, L. Chang, J. Yu, Supergraph Search in Graph Databases vis Hierarchical Feature-Tree, to appear in TKDE (accept in April, 2018. Subbmited before my EIC term.)
3. D. Wen, L. Qin, Y. Zhang, X. LIN, J. Yu, I/O Efficient Core Graph Decomposition: Application to Degeneracy Ordering, to appear in TKDE (Best Paper Award in ICDE 2016).
4. D. Ouyang, L. Qin, L. Chang, X. LIN, Y. Zhang, Q. Zhu, When Hierarchy Meets 2-Hop-Labeling: Efficient Shortest Distance Queries on Road Networks, to appear in SIGMOD 2018.
5. J. Yang, W. Zhang, S. Yang, Y. Zhang, X. LIN, L. Yuan, Efficient Set Containment Join, to appear in VLDB Journal (accepted in March, 2018).
6. K. Wang, X. Cao, X. LIN, W. Zhang, L. Qin, Efficient Computing of Radius-Bounded k-Cores, to appear in ICDE 2018.
7. Y. Peng, Y. Zhang, W. Zhang, X. LIN, L. Qin, Efficient Probabilistic K-Core Computation on Uncertain Graphs, to appear in ICDE 2018.
8. F. Zhang, Y. Zhang, L. Qin, W. Zhang, X. LIN, Efficient Reinforcing Social Networks over User Engagement and Tie Strength, to appear in ICDE2018.
9. J. Qin, Y. Wang, C. Xiao, W. Wang, X. LIN, Y. Ishikawa, GPH: Similarity Search in Hamming Space, to appear in ICDE2018.
10. W. Li, M. Qiao, L. Qin, Y. Zhang, L. Chang, X. LIN, Exacting Eccentricity for Small-World Networks, to appear in ICDE2018.Y. Chen, X. Zhao, X. LIN, Y. Wang, D. Guo, Efficient Frequent Sungraph Mining on Uncertain Graphs, to appear in TKDE (accepted in Jan, 2018, submitted before my EIC term).
11. W. Yu, X. LIN, W. Zhang, and J. McCann. Dynamical SimRank Assessment on Time-Varying Networks. to appear in VLDB Journal. (33 pages, Accepted in OCT 2017).
12. X. Zhao, C. Xiao, X. LIN, Wenjie Zhang, Yan Wang, Efficient Structure Similarity Searches: A Partition-Based Approach, to appear in VLDB Journal.
13. W. Liu, Z. Liu, I. W. Tsang, W. Zhang, X. LIN, Doubly Approximate Nearest Neighbor Approximation, to appear in AAAI 2018
14. L. Yuan, L. Qin, X. LIN, L. Chang, W. Zhang, Effective and Efficient Dynamic Graph Coloring, to appear in VLDB 2018.
15. D. Wen, L. Qin, L. Chang, Y. Zhang, X. LIN, Efficient Structural Graph Clustering: An Index-Based Approach, to appear in VLDB2018.

References:

16. F. Zhang, Y. Zhang, L. Qin, W. Zhang, **X. LIN**, When Engagement Meets Similarity: Efficient (k, r)-Core Computation on Social Networks, to appear in **VLDB2017**.
17. Fan Zhang, Wenjie Zhang, Ying Zhang, Lu Qin, **XUEMIN LIN**, "OLAK: An Efficient Algorithm to Prevent Unraveling in Social Networks", to appear in 42nd International Conference on Very Large Data Bases **(VLDB), 2017**
18. F. Zhang, Y. Zhang, L. Qin, W. Zhang, **X. LIN**, Finding Critical Users for Social Network Engagement: The Collapsed k-Core Problem, to appear in **AAAI2017**.
19. T. Gao, X. Cao, G. Cong, J. Lu, **X. LIN**, Distributed Algorithms on Exact Personalized PageRank, to appear in **SIGMOD2017**.
20. L. Yuan, L. Qin, **X. LIN**, L. Chang, W. Zhang, I/O Efficient ECC Graph Decomposition via Graph Reduction, to appear in **VLDB Journal**.
21. L. Chang, C. Zhang, **X. LIN**, L. Qin Scalable Top-K Structural Diversity Search, Short Paper, Proceedings of the 32st International Conference on Data Engineering **(ICDE2017), 2017**
22. L. Lai, Q. Lu, **X. LIN**, Y. Zhang, L. Chang, Scalable Distributed Subgraph Enumeration, **VLDB 2017**.
23. F. Bi, L. Chang, X. LIN, L. Qin, W. Zhang, Efficient Subgraph Matching by Postponing Cartesian Products, **SIGMOD 2016**.
24. H. Wei, J.X. Yu, C. Lu, X. LIN, Speedup Graph Processing by Graph Ordering, to appear in **SIGMOD 2016**.
25. . Yuan, L. Qin, X. LIN, L. Chang, W. Zhang, I/O Efficient ECC Graph Decomposition via Graph Reduction, **VLDB2016**.
26. B. Lyu, L. Qin, X. LIN, L. Chang, J.X. Yu, Scalable Supergraph Search in Large Graph Databases, **ICDE 2016**.
27. X. Feng, L. Chang, X. LIN, L. Qin, W. Zhang, Computing Connected Components with Linear Communication Cost in Pregel-like Systems, **ICDE 2016**.
28. L. Chang, W. Li, X. LIN, L. Qin, W. Zhang, pScan: Fast and Exact Structural Graph Clustering, **ICDE 2016**.
29. D.-W. Choi, J. Pei, X. LIN, Finding the Minimum Spatial Keyword Cover, **ICDE 2016**.
30. D. Wen, L. Qin, Y. Zhang, X. LIN, J.X. Yu, I/O Efficient Core Graph Decomposition at Web Scale, **ICDE 2016** (**BEST PAPER AWARD**).
31. W. Zhang, X. LIN, Y. Zhang, K. Zhu, G. Zhu, Efficient Probabilistic Supergraph Search, to appear in **IEEE Transactions on Knowledge and Engineering** (TKDE, accepted in Nov 2015)
32. L. Yuan, L. Qin, X. LIN, L. Chang, W. Zhang, Diversified Top-K Clique Search, to appear in **VLDB J**, (accepted in Oct 2015)

References:

33. L. Chang, X.LIN, Q. Lu, J .X. Yu, W. Zhang, Index-based Optimal Algorithms for Computing Steiner Components with Maximum Connectivity, **SIGMOD** 2015.
34. L. Chang, X. LIN, Q. Lu, J.X. Yu, J. Pei, Efficiently Computing Top-k Shortest Path Join, to apeear in **EDBT**2015.
35. L. Lai, L. Qin, X. LIN, L. Chang, Scalable Subgraph Enumeration in MapReduce, to appear in **VLDB** 2015.
36. L. Chang, X. LIN, W. Zhang, J.X. Yu, Y. Zhang, L. Qin, Optimal Enumeration: Efficient Top-k Tree Matching, to appear in **VLDB** 2015.
37. X. Wang, Y. Zhang, W. Zhang, X. LIN, W. Wang, Selectivity Estimation On Streami SpatioTextual Data Using Local Correlations, to appear in **VLDB**2015.
38. J. Wang, S. Song, X. LIN, X. Zhu, J. Pei, Clean Structured Even Logs: A Graph Repair Approach, in **ICDE**2015.
39. L. Yuan, Lu. Qin, X. LIN, L. Chang, W. Zhang, Diversified Top-K Clique Search, to appear in **ICDE**2015.
40. X. Wang, Y. Zhang, W. Zhang, X. LING, W. Wang, AP-Tree: Efficiently Support Continuous Spatial-Keyword Queries Over Stream, to appear in **ICDE**2015
41. Z. Zhang, J.X. Yu, L. Qin, L. Chang, X. LIN, I/O Efficient: Computing SCCs in Massive Graphs, to appear in **VLDB Journal** (accepted in Sept, 2014).
42. W. Yu, X. LIN, W. Zhang, J. A. McCann, Fast All-Pairs SimRank Assessment on Large Graphs and Bipartite Domains, to appear in **TKDE**.

References:

43. X. Wang, Y. Zhang, W. Zhang, X. LIN, Efficiently Identify Local Frequent Keyword Co-occurrence Patterns in Geo-tagged Twitter Stream, **SIGIR** 2014 (short paper): 1215-1218.
44. L. Qu, J.X. Yu, L. Chang, H. Cheng, C. Zhang, X. LIN, Scalable Big Graph Processing in MapReduce, **SIGMOD** 2014: 827-838
45. W. Yu, X. LIN, W. Zhang, Fast Incremental SimRank on Link-Evolving Graphs, **ICDE** 2014: 304-315.
46. C. Zhang, Y. Zhang, W. Zhang, X. LIN, M.A. Cheema, X. Wang, Diversified Spatial Keyword Search on Road Networks, **EDBT** 2014: 367-378.
47. X. Zhao, C. Xiao, X. LIN, Q. Liu, W. Zhang, A Partition-Based Approach to Structure Similarity Search, **PVLDB** 7(3): 169-180 (2013)
48. W. Yu, X. LIN, W. Zhang, L. Chang, J. Pei, More is Simpler: Effectively and Efficiently Assessing Node-Pair Similarity based on Hype-links. **PVLDB** 7(1): 13-24 (2013)
49. W. Yu, X. LIN, IRWR: Incremental Random Walk with Restart, **SIGIR** 2013: 1017-1020.
50. L. Chang, J. Yu, L. Qin, X. LIN, C. Liu, W. Liang, Efficiently Computing k-Edge Connected Components via Graph Decomposition, **SIGMOD** Conference 2013: 205-216
51. Z. Zhang, J. Yu, L, Qin, L. Chang, X. LIN, I/O Efficient: Computing SCCs in Massive Graphs, **SIGMOD** Conference 2013: 181-192
52. X. Zhao, X. Chuan, X. LIN, W. Wang, Y. Ishikawa, Efficient Processing of Graph Similarity Queries with Edit Distance Constraints, to appear in VLDB Journal, **VLDB J**. 22(6): 727-752 (2013)
53. Weiren Yu, XUEMIN LIN, Wenjie Zhang, Towards Efficient Computation on SimRank Computation on Large Graphs, **ICDE** 2013: 601-612. (**One of the Best Papers**)

References:

54. Chengyuan Zhang, Ying Zhang, Wenjie Zhang, XUEMIN LIN, Inverted Linear Quadtree: Efficient Top K Spatial Keyword Search, **ICDE** 2013: 901-912
55. Weiiren Yu, XUEMIN LIN, Wenjie Zhang, Ying Zhang, Jiajin Le, SimFusion+: Extending SimiFusion Towards Efficient Estimation on Large Dynamic Networks, **SIGIR** 2012: 365-374
56. Yuanyuan Zhu, Lu Qin, Jeffrey Yu, Yiping Ke, XUEMIN LIN, High Efficiency and Quality: Large Graphs Matching, **VLDB J**. 22(3): 345-368 (2013)
57. Gaoping Zhu, XUEMIN LIN, Ke Zhu, Wenjie Zhang, Jeffrey Xu Yun, TreeSpan: Efficiently Computing Similarity All-Matching, **SIGMOD** Conference 2012: 529-540
58. Xiang Zhao, Chuan Xiao, XUEMIN LIN, Wei Wang, Efficient Graph Similarity Joins with Edit Distance Constraints, **ICDE** 2012: 834-845
59. Y. Luo, W. Wang, X. LIN, X. Zhou, J. Wang, K. Li, SPARK2: Top-k Keyword Query in Relational Databases, IEEE Transactions on Knowledge and Data Enigneering, 23(12), 1763-1780, 2011 (**Spotlight Paper**)
60. H. Shang, X. LIN, Y. Zhang, J.X. Yu, W. Wang, Connected Substructure Similarity Search , pages 903-914, **SIGMOD** 2010.
61. H. Shang, K. Zhu, X. LIN, Y. Zhang, R. Ichise, Similarity Search on Supergraph Containment , pages 637-648, **ICDE** 2010.
62. H. Shang, Y. Zhang, X. LIN, J. Yu, Taming Verification Hardness: an efficient algorithm for testing subgraph isomorphism, pages 364-375, **VLDB**2008.
63. Y. Luo, W. Wang, X. LIN, SPARK: A Keyword Search Engine on Relational Databases (demo) **ICDE** 2008: pages 1552-1555.
64. J. Chen, J.X. Yu, X. LIN, H. Wang, P.S. Yu, Fast Computing Reachability for Large Graphs with Hogh Compresion Rate , in the proceedings of EDBT08, pages 193-204, France.
65. B. Ding, J.X. Yu, S. Wang, L. Qing, X. Zhang, X. LIN, Finding Top-k Min-Cost Connected Trees in Databases, **ICDE**'07 (Best Student Paper Award), pages 836-845, 2007.

Thank you!

Questions?