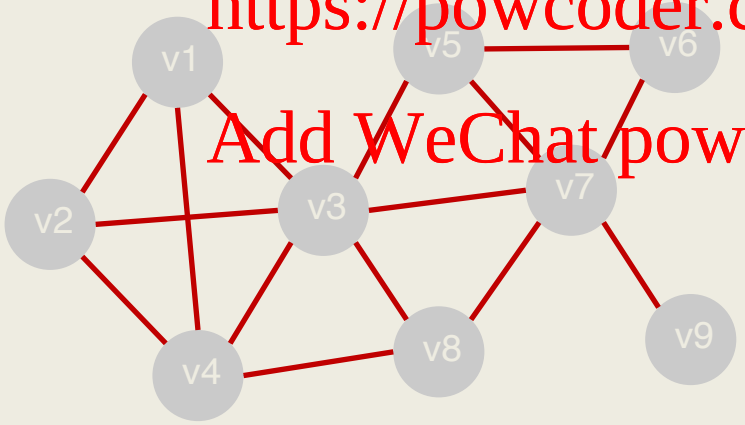
Towards Big Graph Processing: Applications, Challenges, and Advances

Xuemin Lin

UNSW

What is a graph ?

- Vertices: a collection of entities
- Edges: connections between vertices



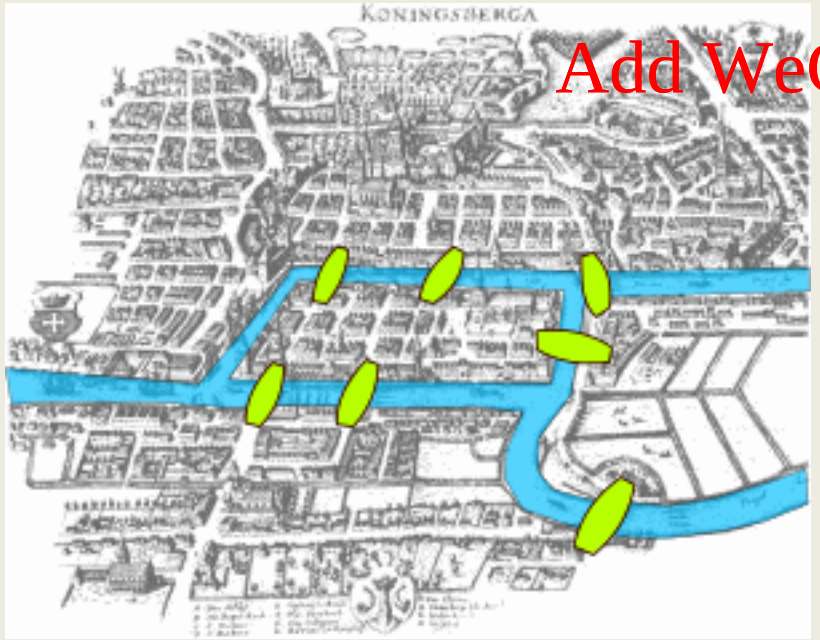
Introduction of

Graphs and Classic Problems

1. Seven Bridges of Königsberg

Leonhard Euler in 1735:

- find a roundtrip through the city to cross each bridge once and only once
- earliest (published) work on networks/graphs



1. Seven Bridges of Königsberg(cont.)

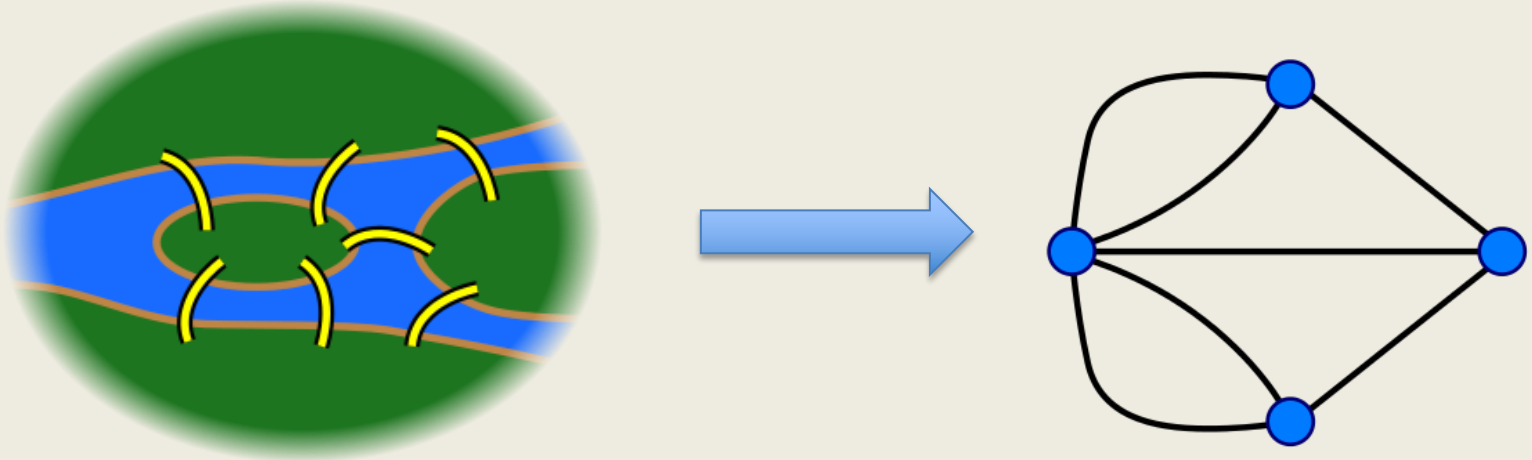
Abstraction:

- replaced each land mass with an abstract "vertex" or node, and each bridge with an abstract connection, an "edge”

Showed that this problem has no solution.

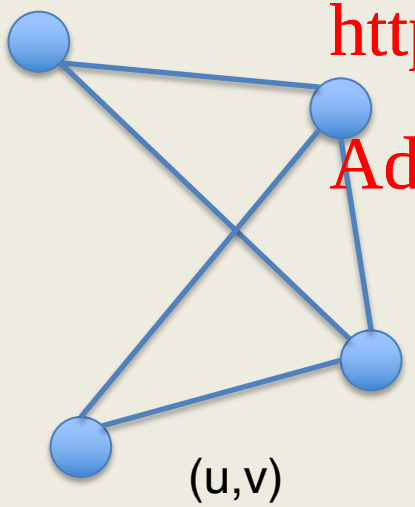
Euler Tour:

- necessary and sufficient conditions for the walk of the desired form: connected and all vertices with even degrees.



Graphs

- Definition:
  - A Graph is a collection of nodes joined by edges.



- G=(V,E)
  - V is a set of vertices, u, v in V are vertices
  - E is a set of edges, (u,v) in E is an edge.

Edges

- Undirected edges
  - u–v: (u,v)=(v,u)
  - e.g., u and v like each other; u and v are co-authors
- Directed edges(arcs)
  - u□v: (u, v) != (v, u)
  - (u, v) is an *in-link (in-edge)* of v, and an *out-link (out-edge)* of u.
  - e.g., u likes v; u is the father of v.
- Other attributes
  - weight (e.g. frequency of communication) – w(u, v)
  - rank (e.g., best friend, second best friend, ...)
  - type (e.g., friend, co-worker, activates, inhibits, ...)

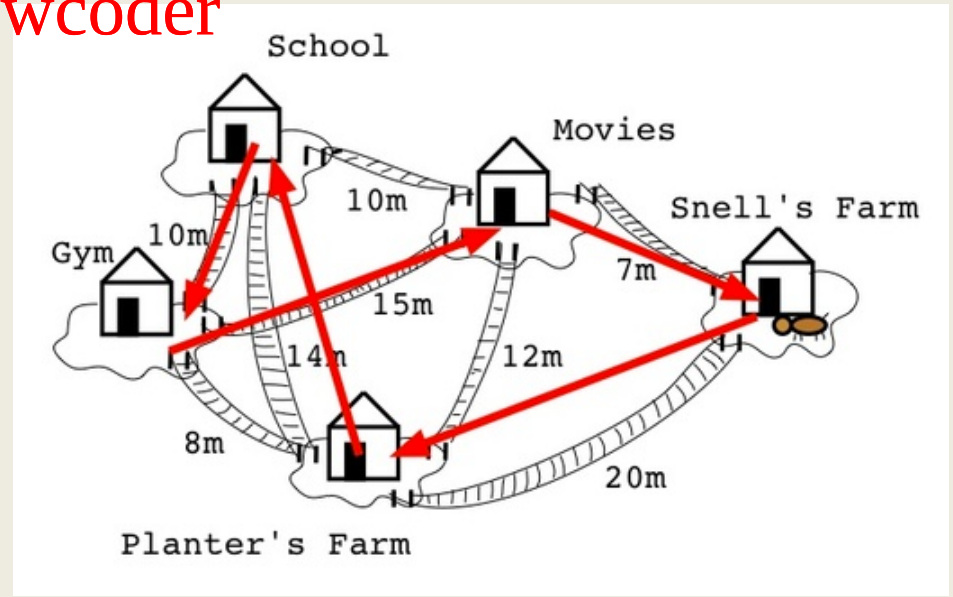
Vertices

- Degree
  - d(v): the number of edges connected to a vertex
  - In degree: number of incoming edges of a vertex
  - Out degree: number of outgoing edges of a vertex
- If edges are weighted, the *degree*, *in degree*, *out degree* are the total weights of *edges*, *incoming edges*, *outgoing edges* of a vertex.
- Other attributes:
  - Weight: (e.g., importance, authority of a person)
    - Type: (e.g., advisor, student, old people)
    - Label: (e.g., name)

2. Travelling salesman

- Problem:
  - Given a list of cities and the distances between each pair of cities, find the shortest route that visits each city exactly once and returns to the origin city.
  - In the theory of computational complexity, the TSP belongs to the class of NP-hard problems.

Thus, it is unlikely that the worst-case running time for any algorithm for the TSP is in PTIME.



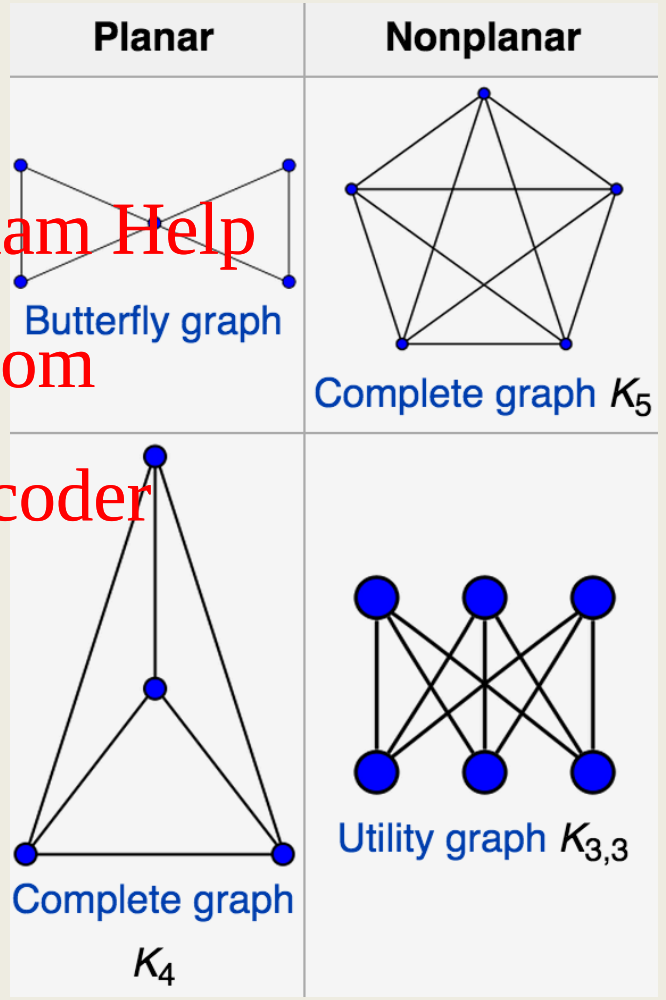
2. Travelling salesman (cont.)

- Heuristic Solutions:
  - Constructive heuristics: Nearest-neighbors (greedy), spanning tree, bitonic tour, etc.
  - for extremely large problems (millions of cities), within a reasonable time which are with a high probability just 2–3% away from the optimal solution

Widely used in several applications:
planning, logistics
the manufacture of microchips.
DNA sequencing, etc.

3. Planar graph

- Definition: can be embedded in the plane
  - i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints.
- Determination:
  - does not contain a subgraph that is a subdivision of K5 (the complete graph on five vertices) or K3,3((complete bipartite graph on six vertices)

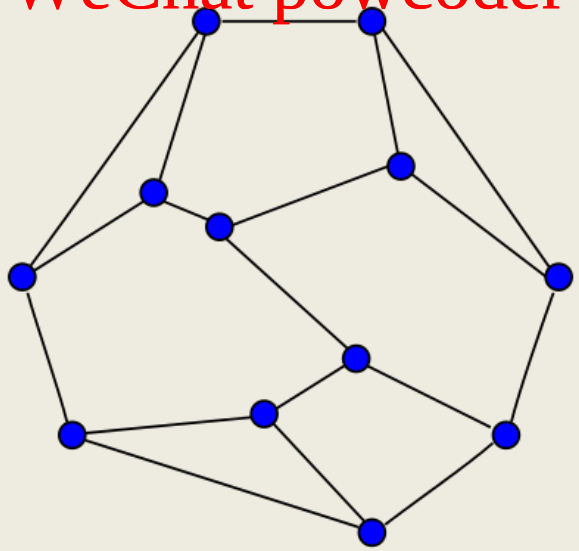


4. Euler's Formula

- Euler's Formula:

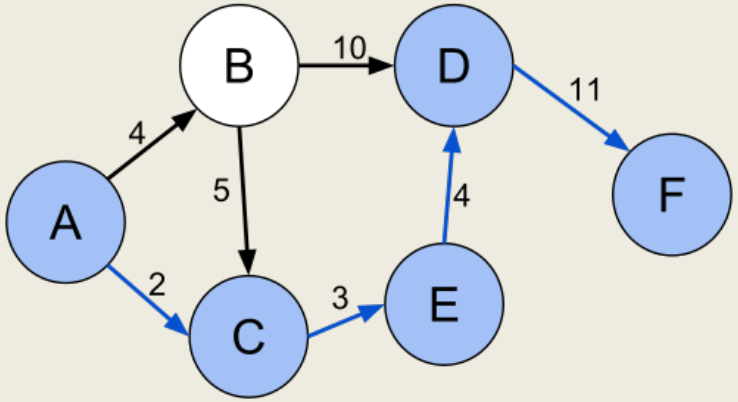
  if a finite, connected, planar graph is drawn in the plane without any edge intersections

  v − e + f = 2. (v is the number of vertices, e is the number of edges and f is the number of faces)



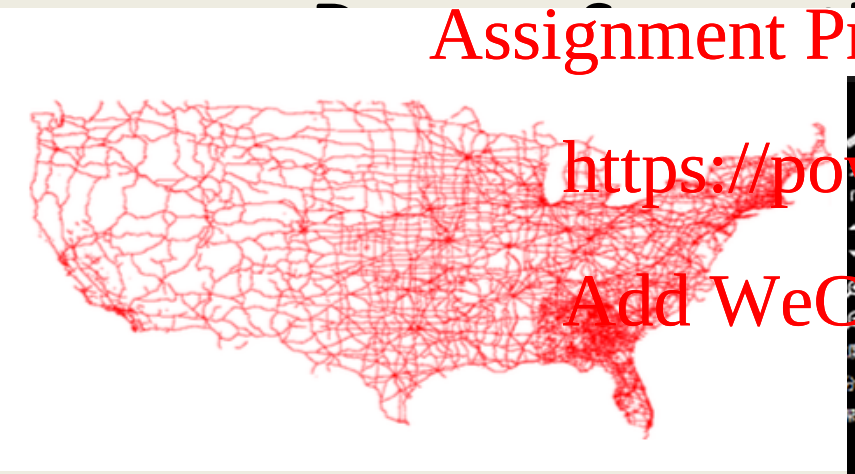
5. Shortest Path

- a.k.a. Geodesic paths:
  - finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.
- Solution:
  - Single Source: Dijkstra
  - All Pair: Floyd-Warshall.



5. Shortest Path(cont.)

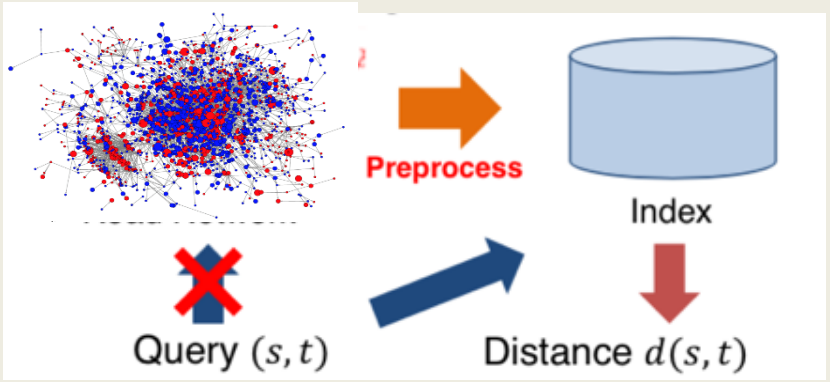
- Applications:
  - Road Network Navigation
  - Join in Social Networks

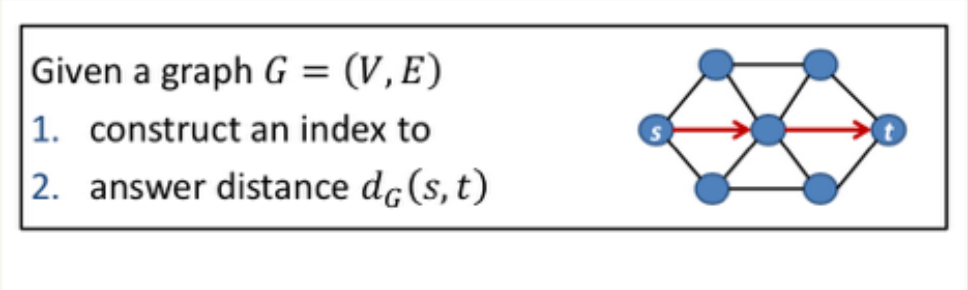


Six Degree of Separation
http://entitycube.research.microsoft.com/6dumapsvg.aspx

5. Shortest Path(cont.)

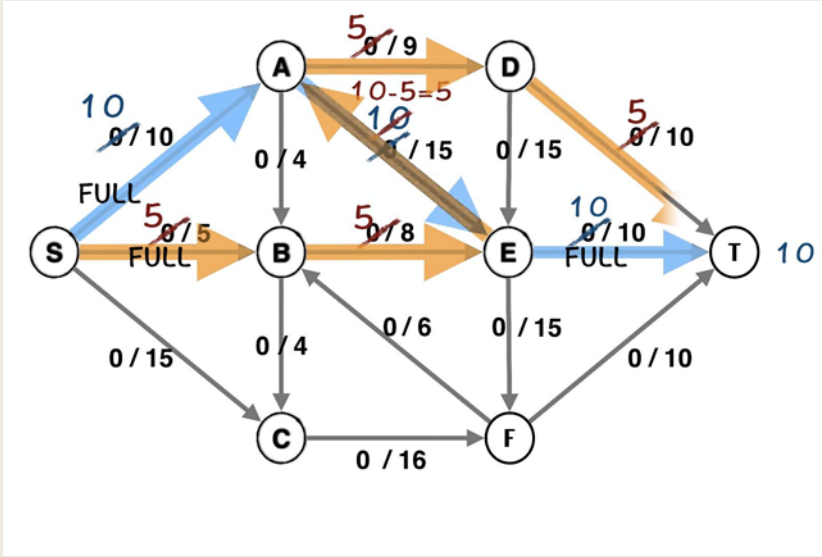
- How to fast get shortest path between two nodes in a large graph?
  - Pre-computed small structure to maintain distances.
  - Distance Oracle:
    - Hop cover for each nodes' reachable node list
      - Intersect of two lists.
    - Landmark based global summary structure.
      - Tree traversal





6. Max-Flow

- Problem:
  - finding a feasible flow through a single-source, single-sink flow network that is maximum.
- Applications:
  - Airline/pipeline/network scheduling
  - Circulation-demand/logistics planning
  - Social network Sybil/Spammer detection

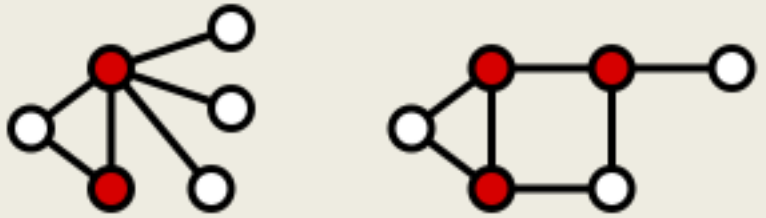


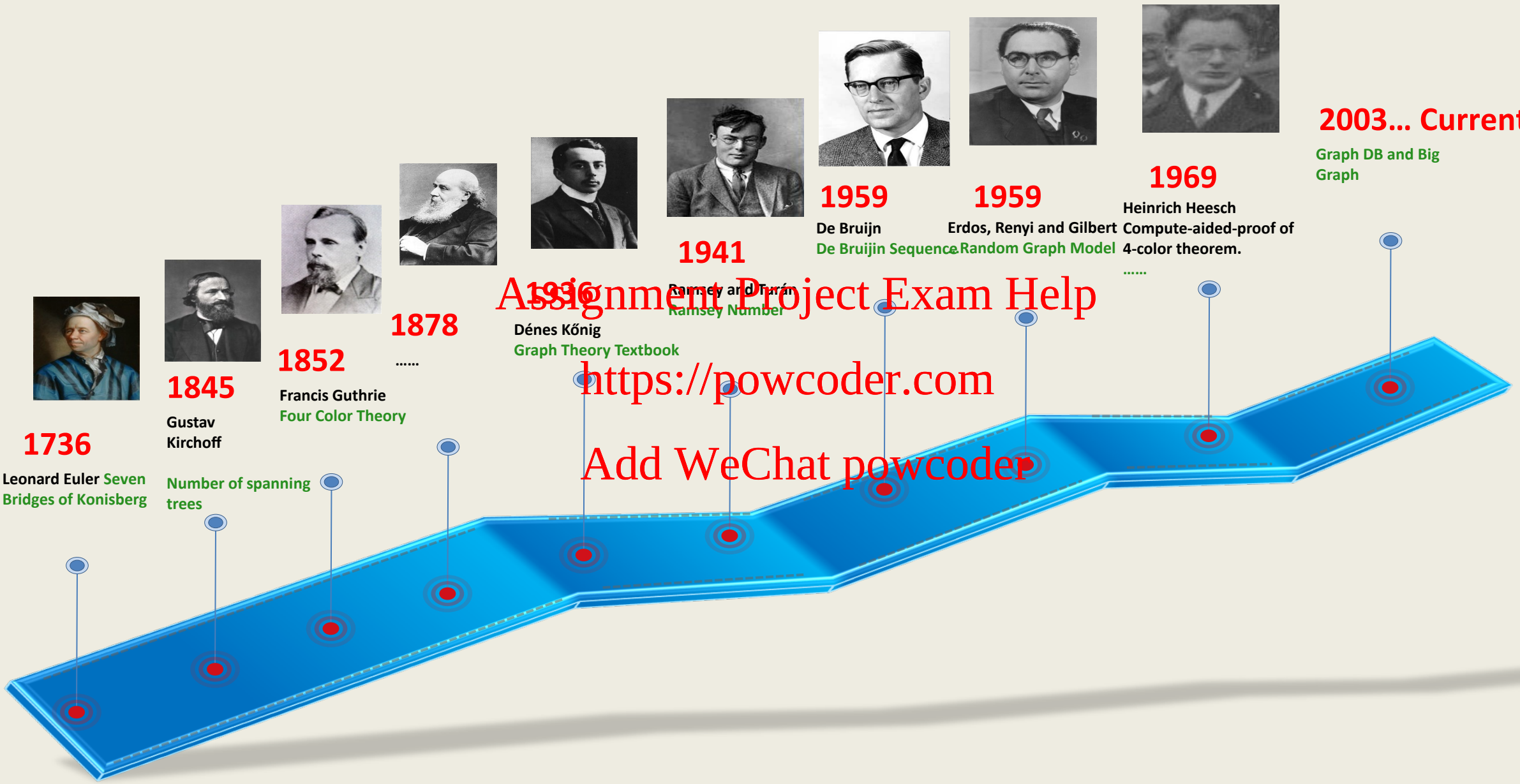
6. Max-Flow (cont.)

- Common Solutions:
  - Linear Programming.
  - Ford-Fulkerson: O(mn)
  - Recent result: close to O (m)

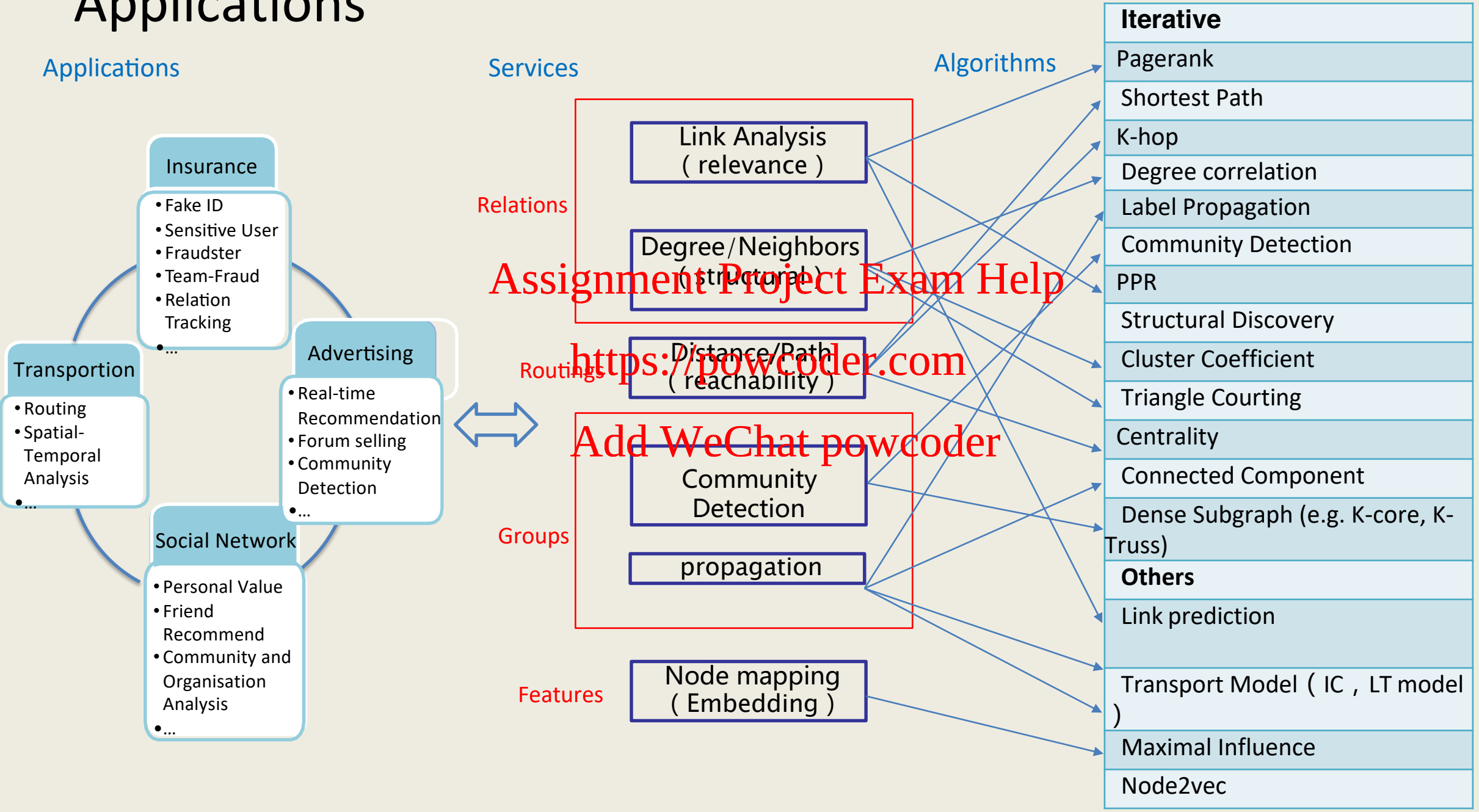
7. Vertex Cover

- Problem:
  - A vertex-cover of an undirected graph G=(V,E) is a subset V' of V, for edge (u,v) in G, u or v is in V'.
  - Minimum Vertex Cover:
    - NP-hard optimization problem.
- Solution:
  - Approximation algorithm:
    - Repeatedly taking both endpoints of an edge into the vertex cover and remove.
    - Maximum Matching





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

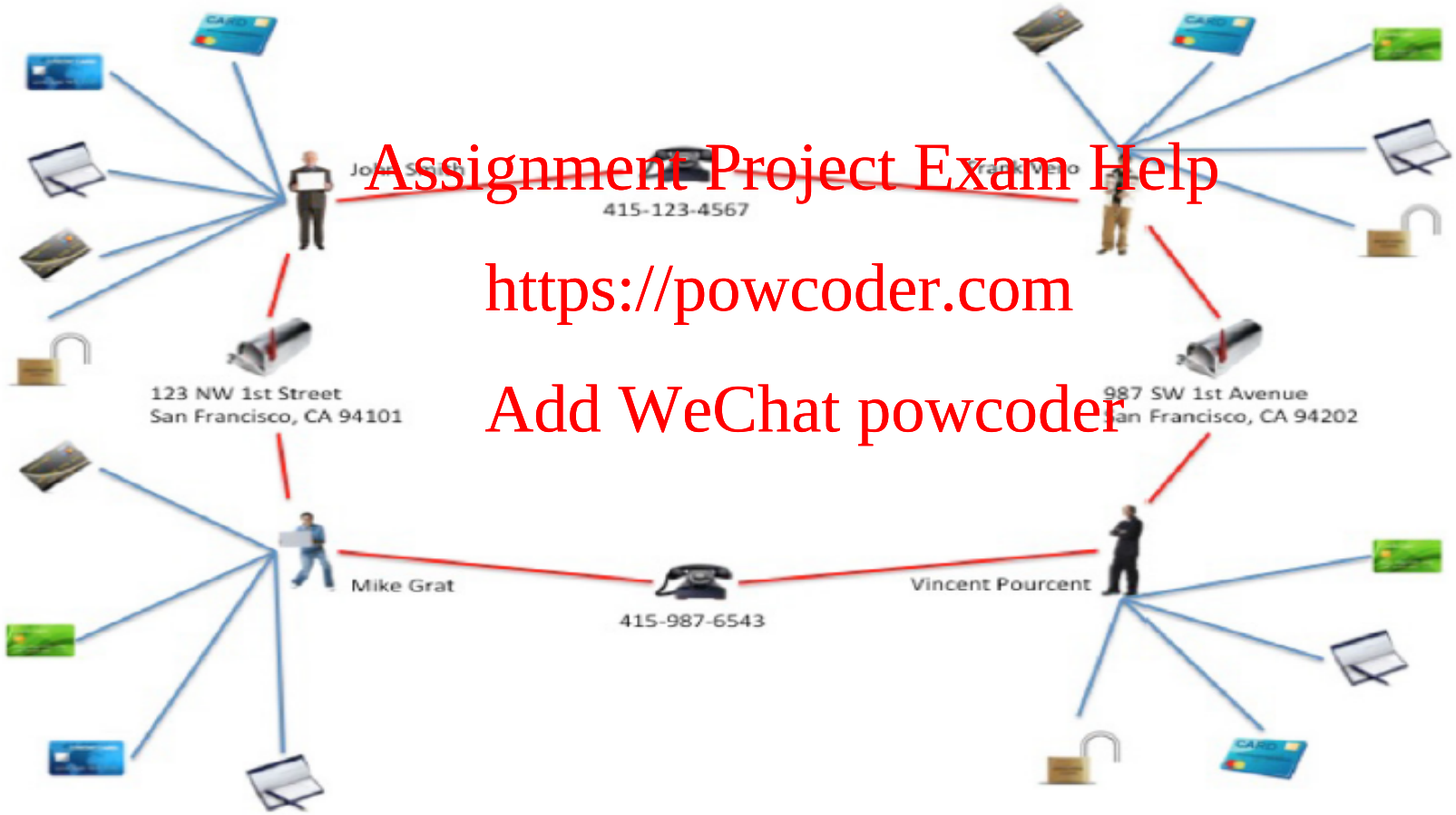


Diagram 1: 2 people sharing 2 pieces of data and creating 4 synthetic identities

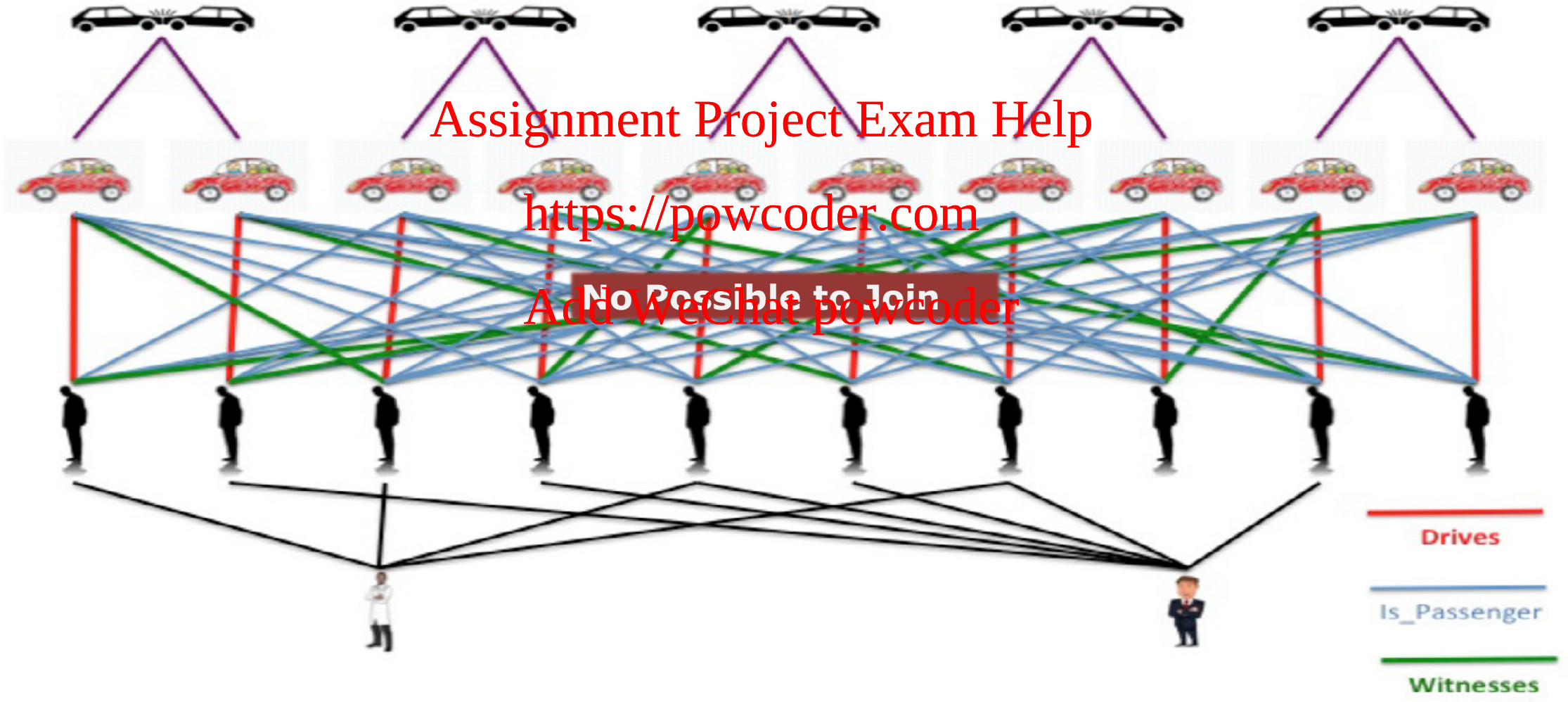
Fraud Detection: Insurance Fraud

- 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

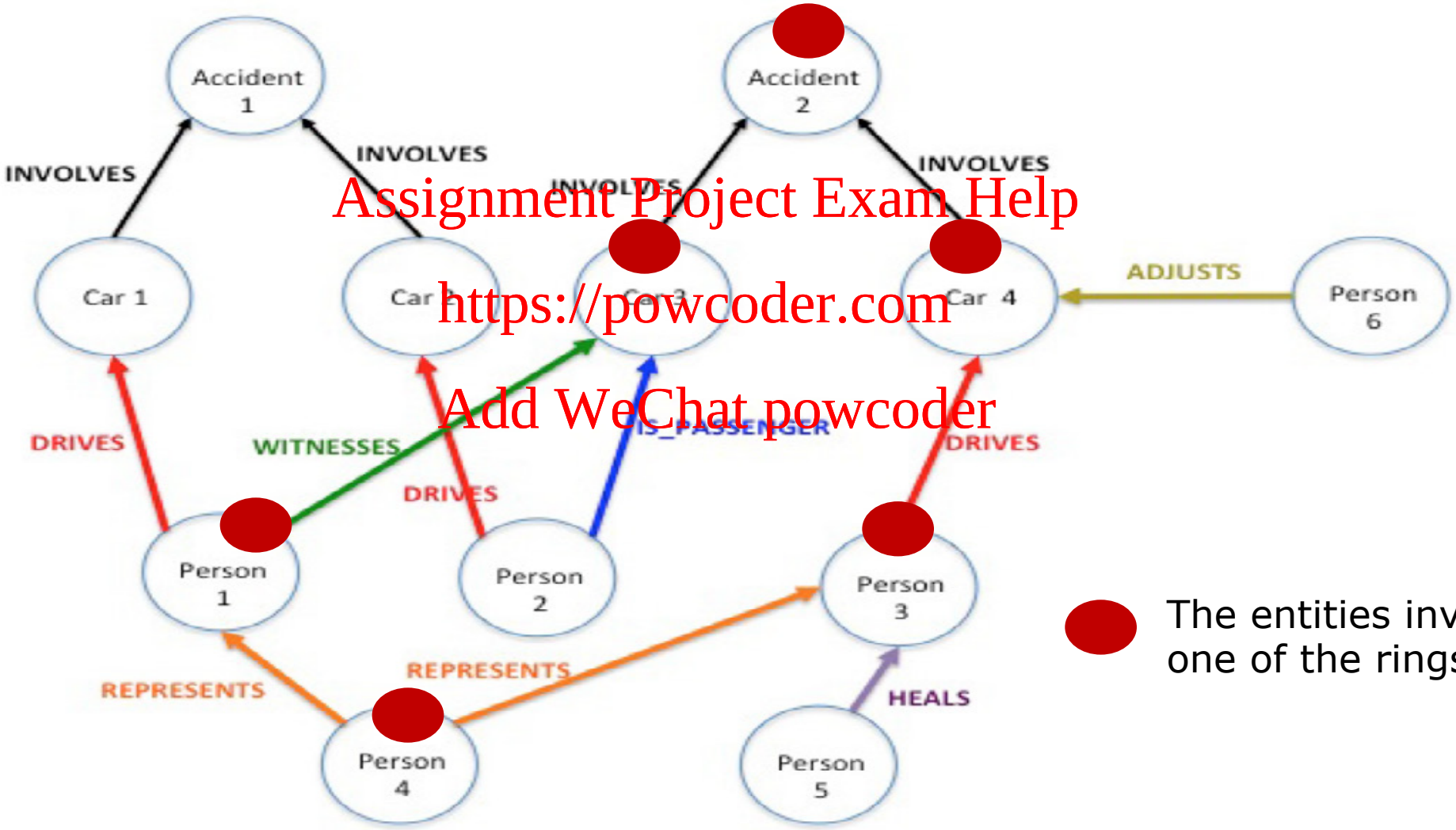
Fraud Detection: Insurance Fraud

- 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



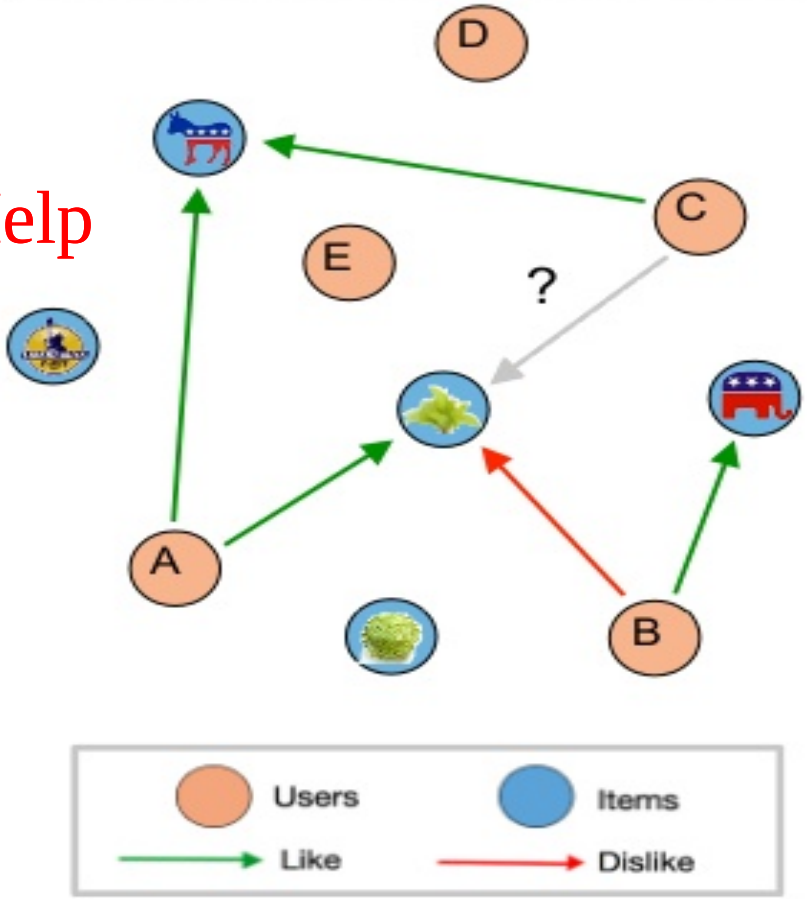
The entities involved in one of the rings.

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.

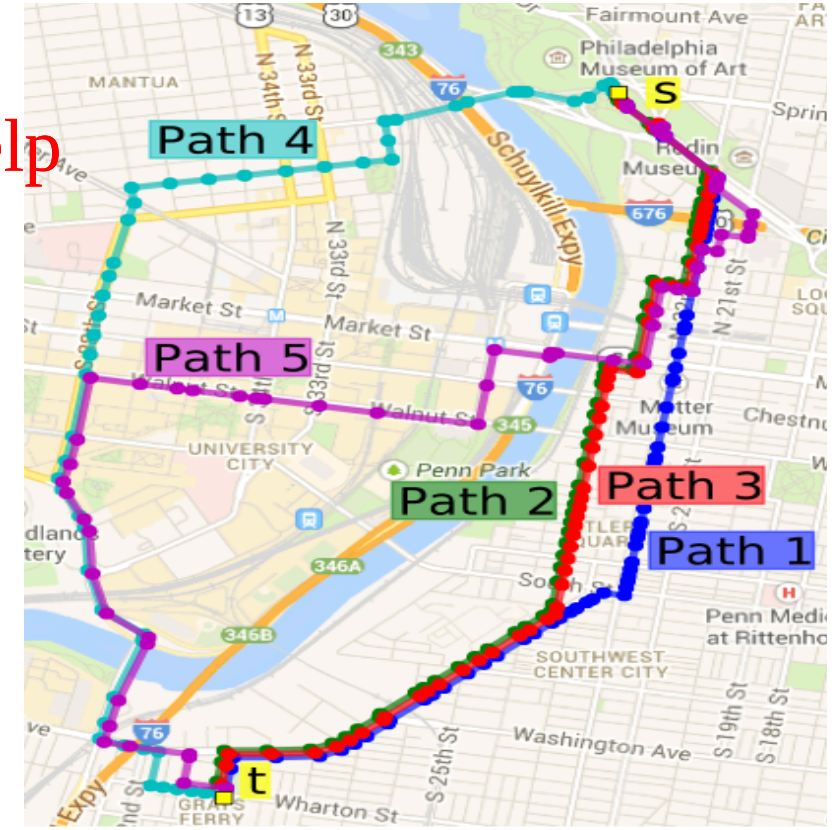
Product Recommendation

- 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



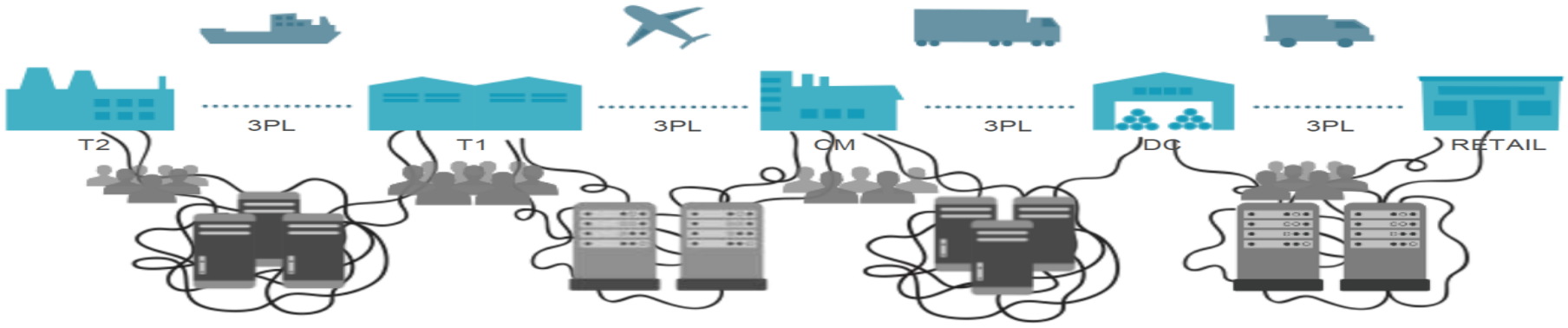
Retail Services:
Ecommerce Delivery Service Routing

- 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

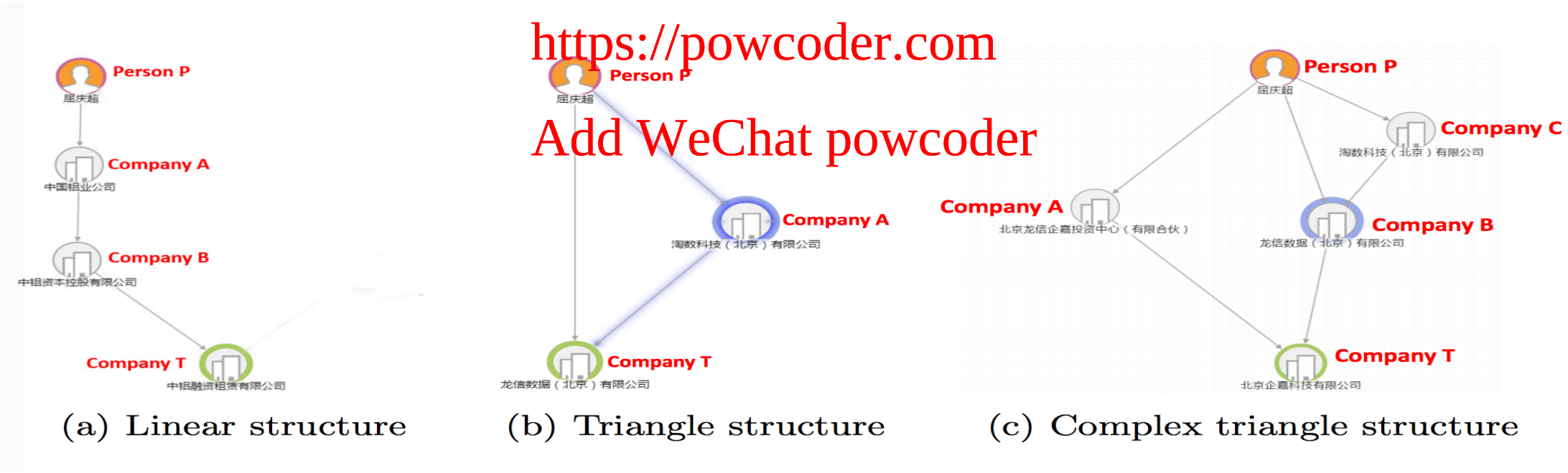


Investment Analysis

- 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

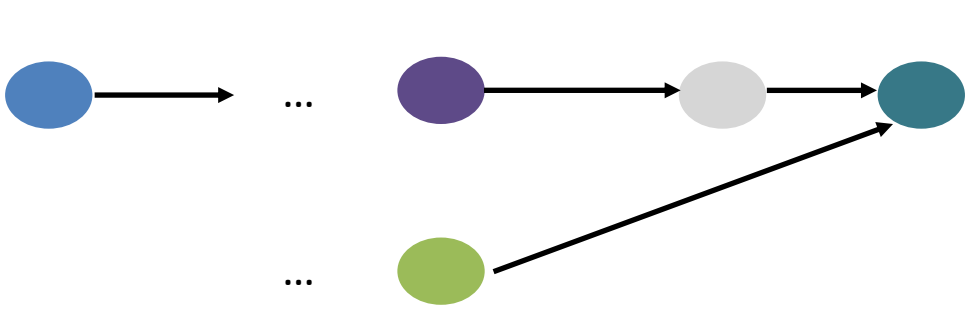
- Reveal an enterprise's real controllers: the person who owns the biggest equity share, directly or indirectly



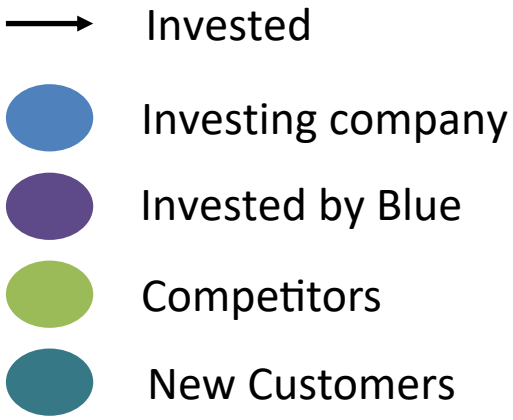
(a) Linear structure (b) Triangle structure (c) Complex triangle structure

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

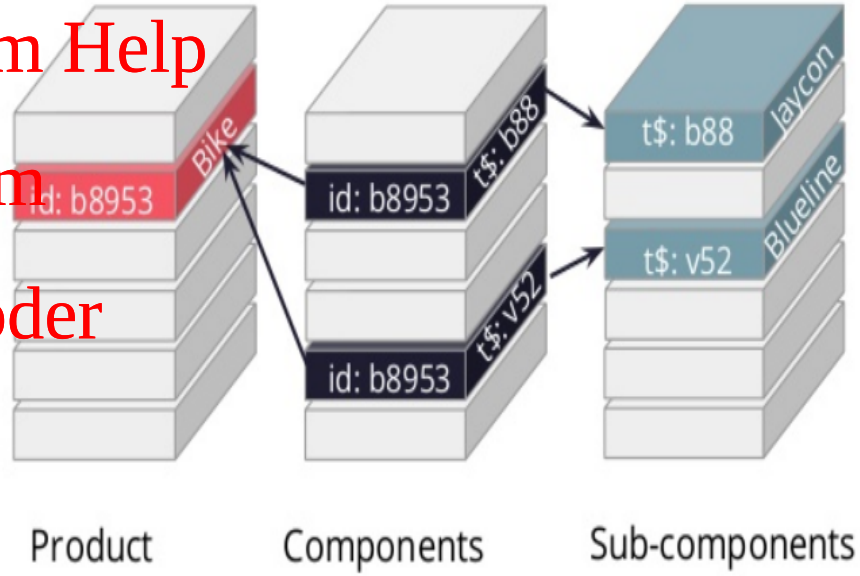
- Multidimensional relationships discovery:
  - competitors
  - pattern transfer
  - acquisition
  - investment
- Why this can help?
  - Difficulties of investing a target company? Try negotiate with the competitors' targets.

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

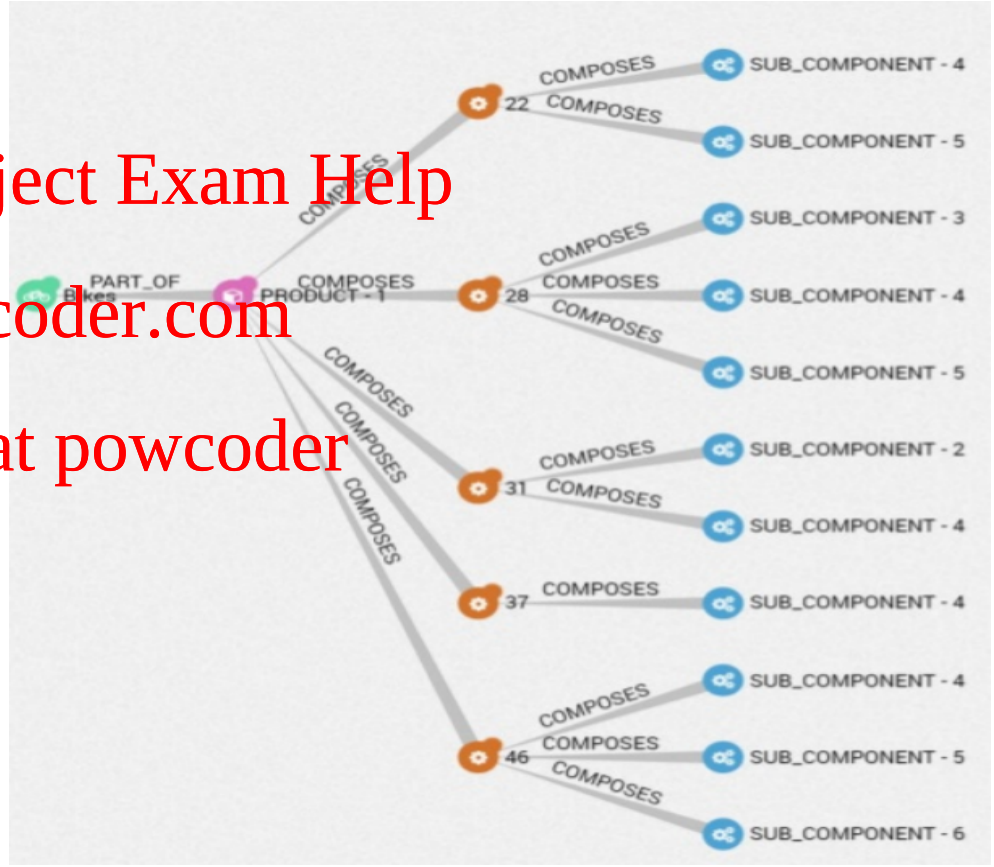
Product Lifecycle Management

- 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

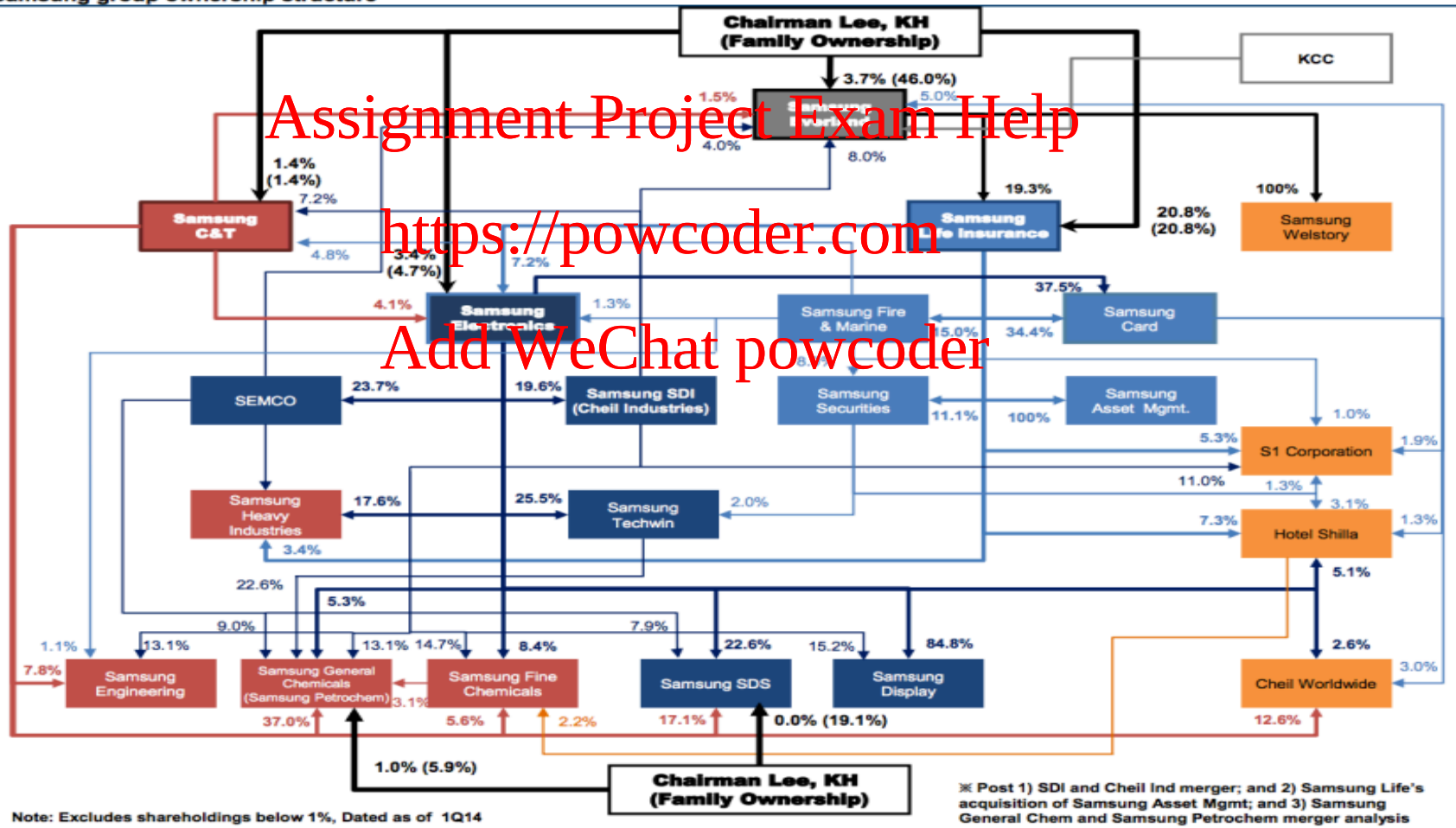
- Represent cross-department data about products and processes as a graph and store it as one
  - Aggregate product hierarchies and connections into a single source of truth
  - Easily edit, or expand your model as your need evolve
  - Saving time by searching and exploring your data
  - Visually track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



Anti Money Laundering

Samsung Group

Figure 8: **Samsung group ownership structure**

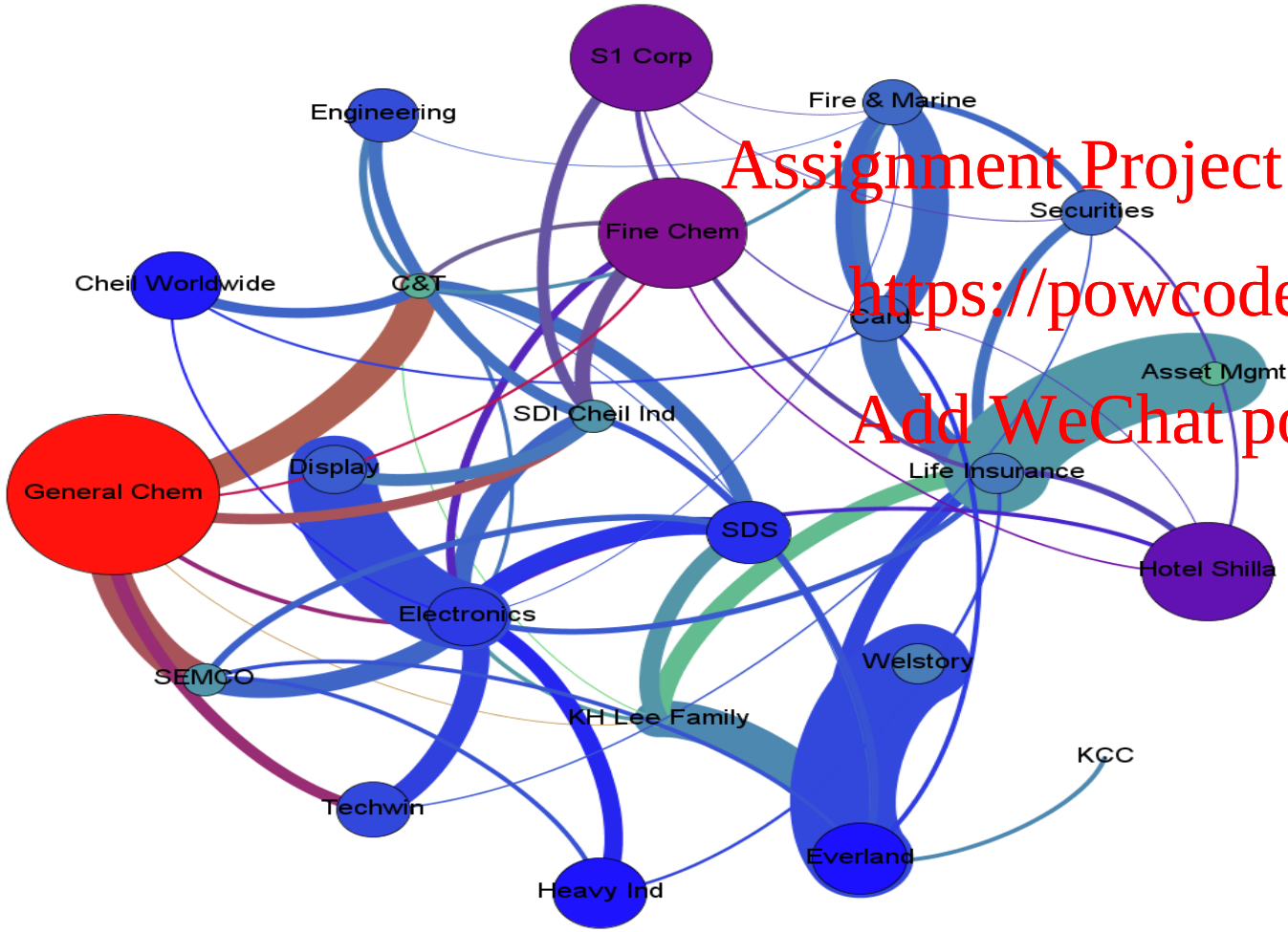


Note: Excludes shareholdings below 1%, Dated as of 1Q14

※ 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

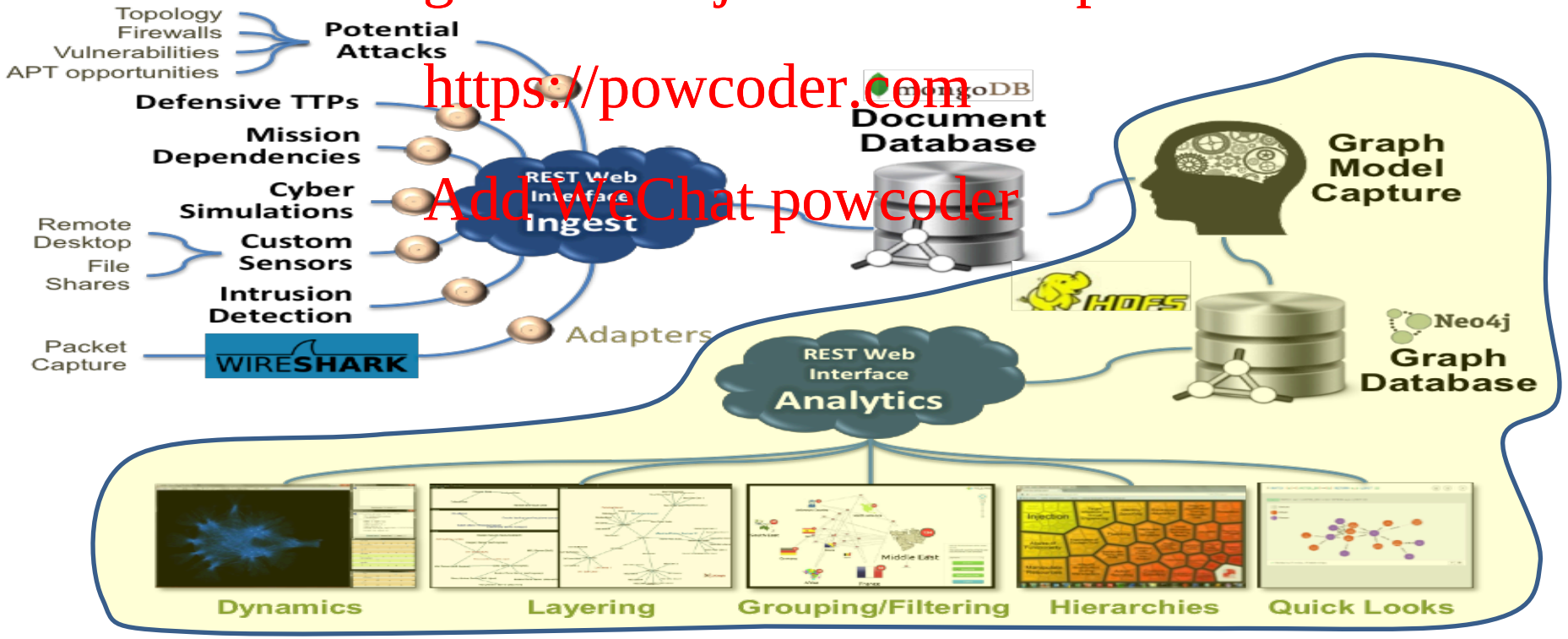
- Global-scale organisation fosters complex transaction flows
  - can be circular, making it impossible to investigate ML using traditional method.
  - Graph not only can visual, but can model the data for deep analysis
    - Graph properties such as **eccentricity** and **betweenness** are useful factors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

Cybersecurity

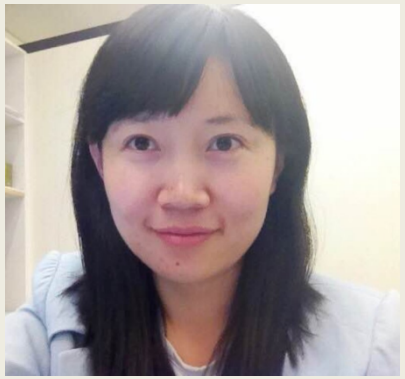
- Goal
  - A unified web framework to manage cyber attack relationships in the network
  - Understand how cyber attackers can leverage initial footholds to extend their reach through the network
- Challenge
  - correlate data from numerous sources into a common model
  - flexible and easy to extend,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

Cybersecurity: MITRE Case Study

- Solution
  - Graph model



Many thanks to my team



A/Prof. Wenjie Zhang

100+ ERA A* (CCF A) ranked papers
Associate Head of CSE, UNSW
Graph Data, Spatial-Temporal Data, Algorithms

A/Prof. Ying Zhang
80+ ERA A* (CCF A) ranked papers
University of Technology Sydney (UTS)
Social Network, Streaming Data, Algorithm

Dr. Lu Qin
60+ ERA A* (CCF A) ranked papers
Senior Lecturer of UTS
Big Graph Computing, Algorithm

Dr. Xin Cao
20+ ERA A* (CCF A) ranked papers
Senior Lecturer of UNSW
Social Network, Data Mining, Algorithm

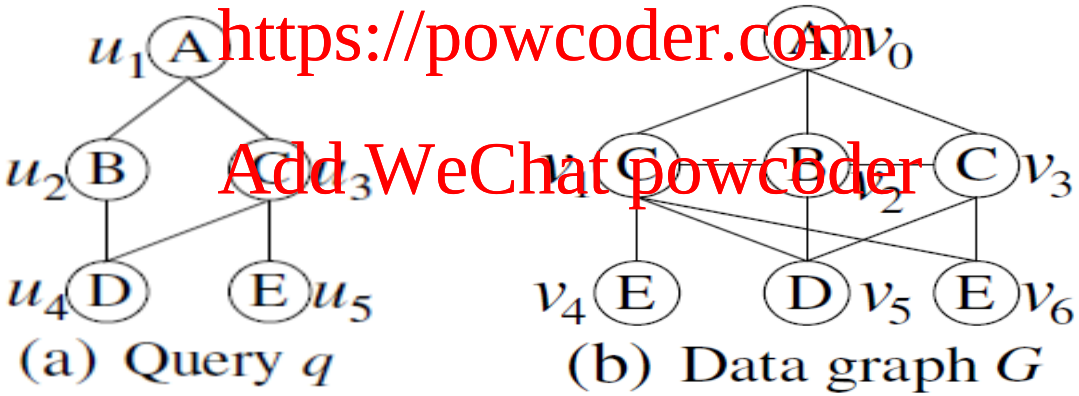
- **Dr. Yixiang Fang: Postdoctoral Fellow** of UNSW，Social Network, Big Graph Computing.
- **Dr. Xing Feng: Postdoctoral Fellow** r of UTS, Big Graph Computing.
- **Dr. Longbin Lai: Postdoctoral Fellow** of UNSW, Big Graph Computing.
- **Dr. Long Yuan: Postdoctoral Fellow** of UNSW, Big Graph Computing.
- **Many past students and current students**

GRAPH PATTERN MATCHING

Introduction

- **Subgraph Matching**

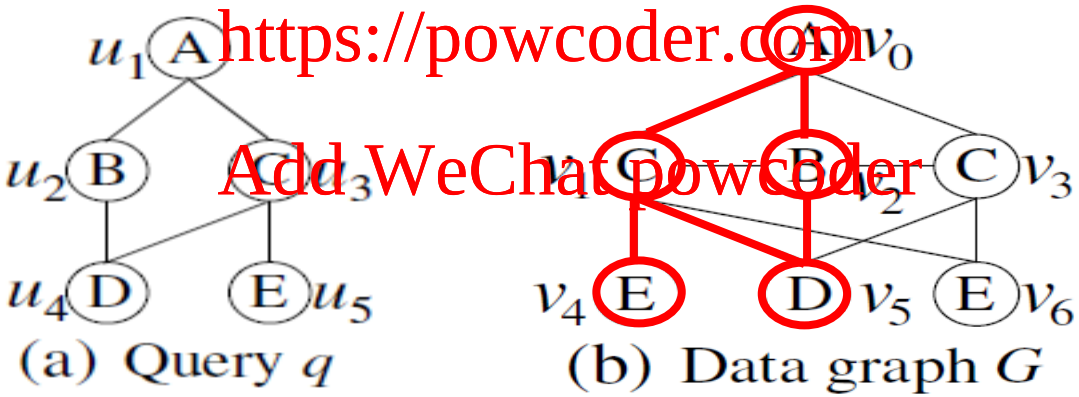
  Given a query ***q*** and a large data graph ***G***, the problem is to extract all subgraph isomorphic embeddings of ***q*** in ***G***.



Introduction

- **Subgraph Matching**

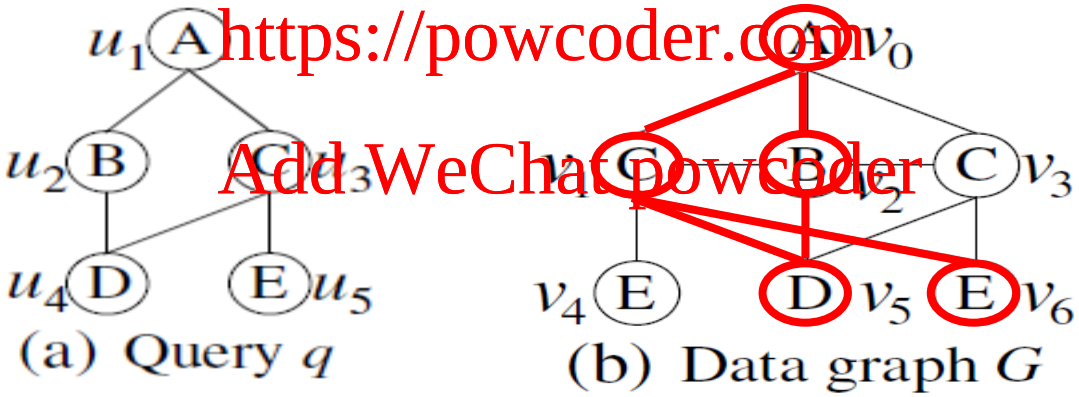
  Given a query ***q*** and a large data graph ***G***, the problem is to extract all subgraph isomorphic embeddings of ***q*** in ***G***.



Introduction

- **Subgraph Matching**

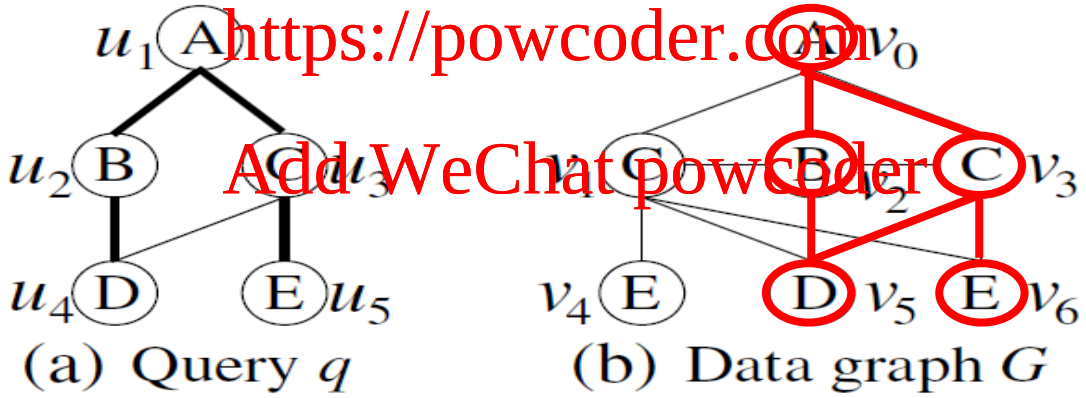
  Given a query ***q*** and a large data graph ***G***, the problem is to extract all subgraph isomorphic embeddings of ***q*** in ***G***.



Introduction

- **Subgraph Matching**

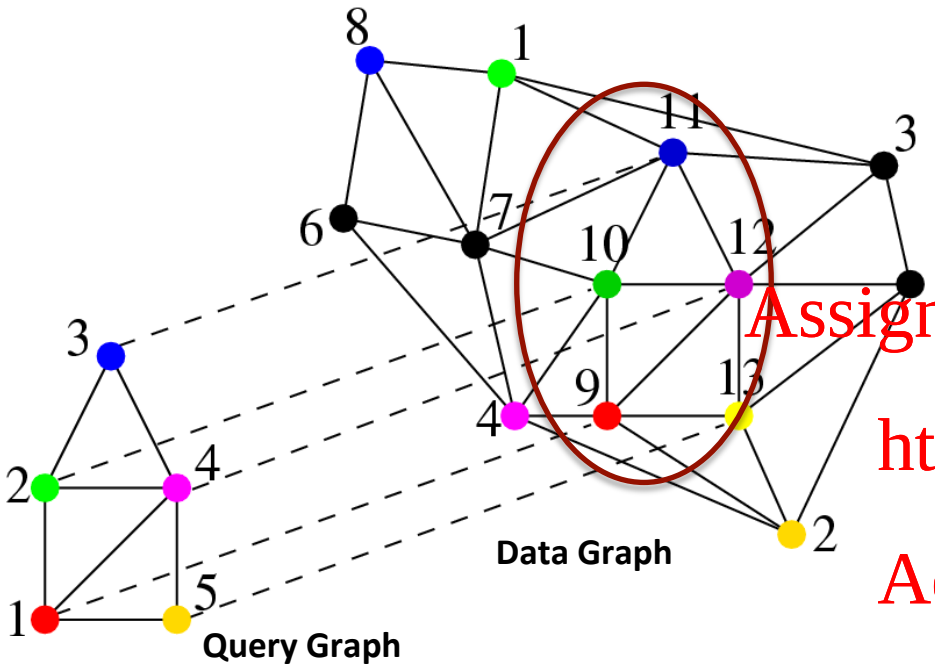
  Given a query ***q*** and a large data graph ***G***, the problem is to extract all subgraph isomorphic embeddings of ***q*** in ***G***.



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

Detecting Designated Communities



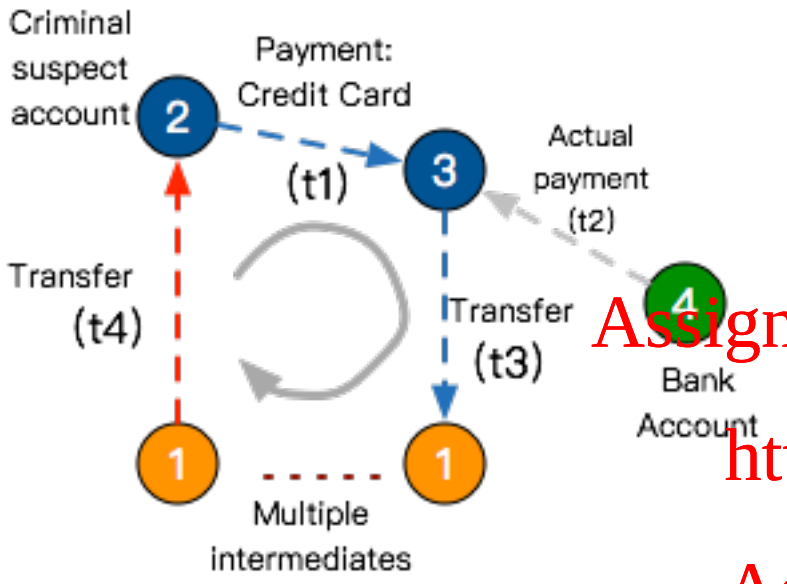
Applications:

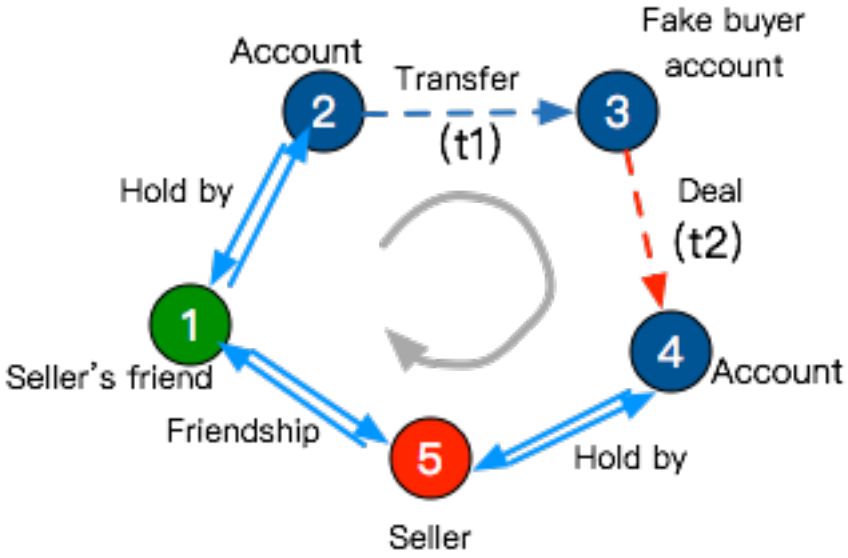
Group of Terrorists, Anomaly detection etc.

Challenges:

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

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:

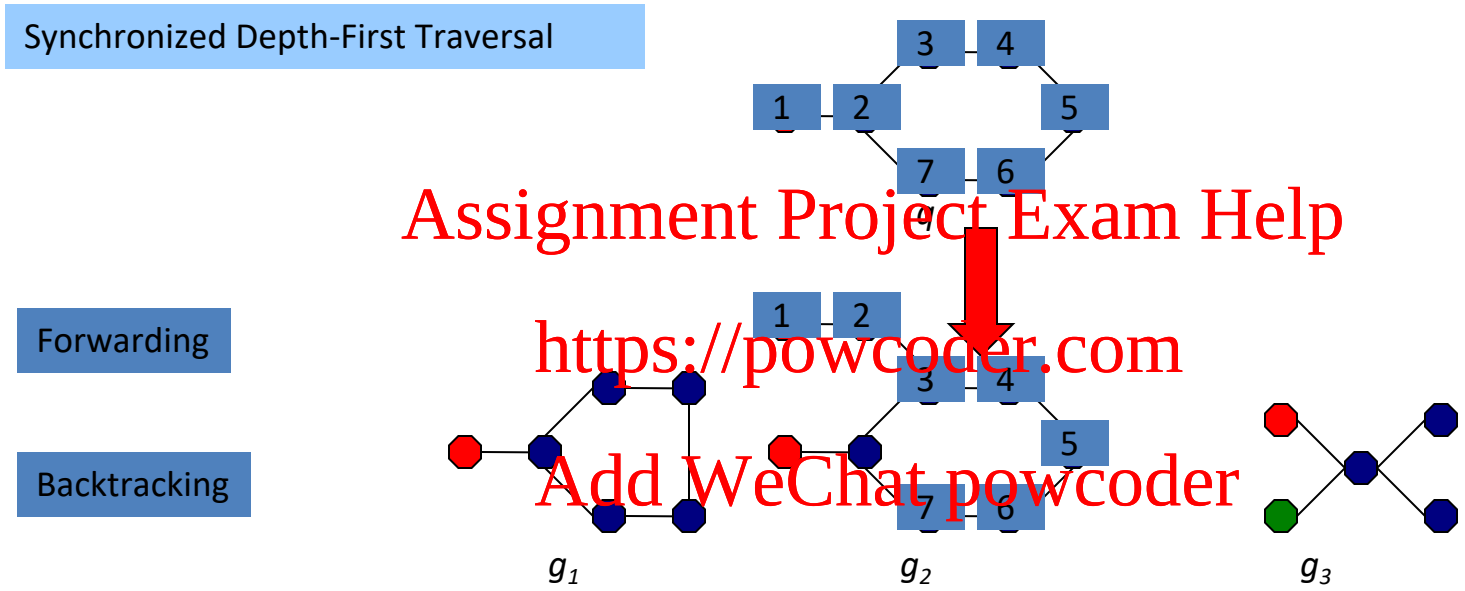
- Real-time cycle detection on each edge
- Light-weighted index structure (easy to maintain and update) and efficient query algorithm

Real-data simulation：

- Detect 6-edge cycles within 10ms

Single CPU

QuickSI (VLDB2008)



1. Determine the access order in *q*
2. Detect corresponding subgraphs in g_1, g_2 which can be mapped to the currently traversed vertices.

TurboISO (SIGMOD2013)

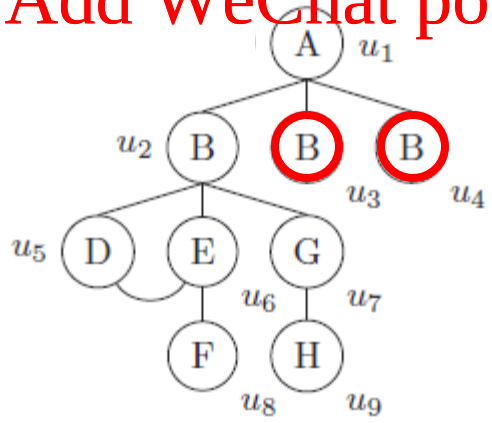
- Overview of TurboISO
  - ➢ Neighborhood Equivalence Class (NEC)
    - Merging vertices with same neighbor to reduce query size
    - Combine / Permute strategy
  - ➢ Candidate Region Exploration
    - Constructed on-the-fly based on q
    - A path-based data structure containing all embeddings of q in G

Neighborhood Equivalence Class(NEC)

- Definition
  - Let $\simeq$ be an *equivalence relation over all query vertices in q such that,* $u_i(\in V(q)) \simeq u_j(\in V(q))$ *if for every embedding m of q contains* (u_i, v_x) *and* $(u_j, v_y)\,(v_x, v_y \in V(g))$*, there exists another embedding* m' *such that*

$$m' = m - \{(u_i, v_x), (u_j, v_y)\} \cup \{(u_i, v_y), (u_j, v_x)\}$$

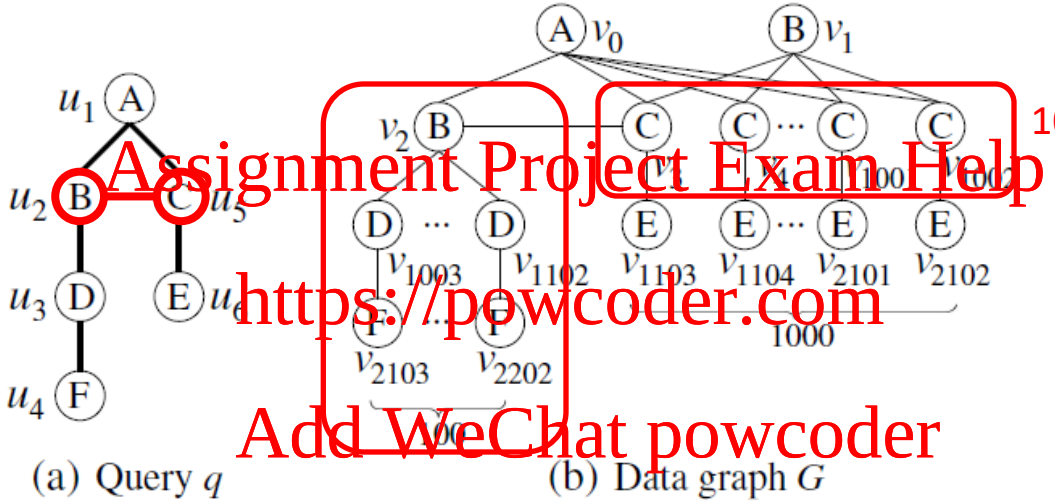
- Example



u_3 and u_4 are NEC nodes .

CFL-Match (SIGMOD2016)

Reduce Candidates



10^5 - 100 partial mappings are redundant.

Matching order of QuickSI and Turbo$_{ISO}$: (u_1, u_2, u_3, u_4, u_5, u_6).

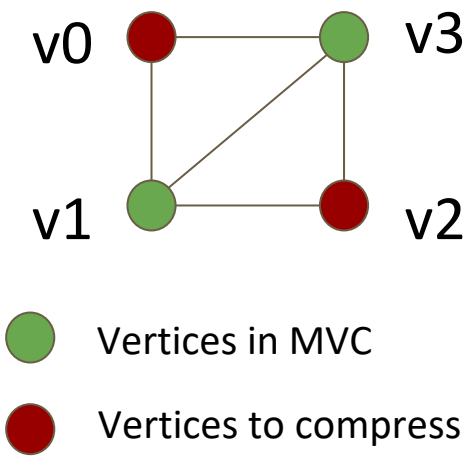
(u_1, u_2, u_5, u_3, u_4, u_6)

Cartesian products:

- 100 mappings (v_0, v_2, v_{1000+i}, v_{2100+i}) ($3 \le i \le 102$) of (u_1, u_2, u_3, u_4)
- 1000 mappings (v_0, v_j) ($3 \le j \le 1002$) of (u_1, u_5)

Compression

- Originally proposed by Qiao et al. [7]
- Intuition
  - Subgraph enumeration can generate enormous (intermediate) results
  - Some vertices can be compressed as they are not needed in future computation
    - Heuristics by [7]: the vertices that do **not** belong to the minimum vertex cover (MVC)



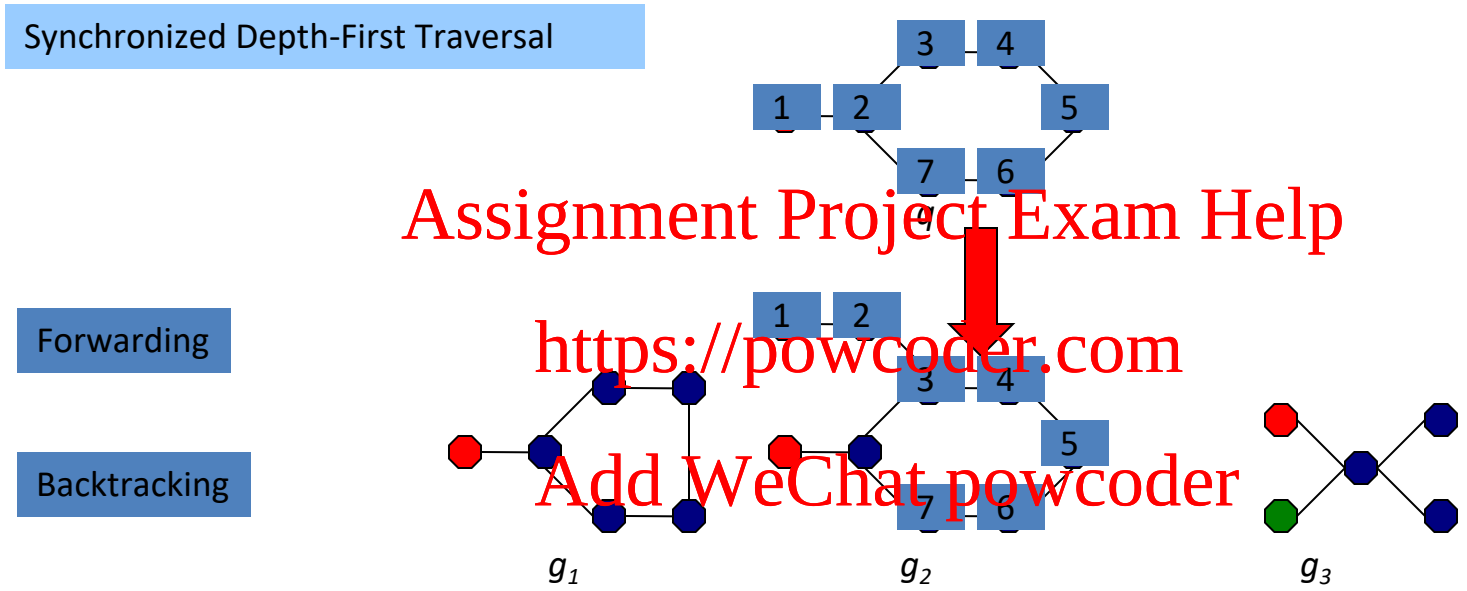
$$v_1 \rightarrow u_1, v_3 \rightarrow u_2$$
$$N(u_1) = \{u_2, u_3, \ldots, u_{1003}\}$$
$$N(u_2) = \{u_1, u_3, \ldots, u_{1003}\}$$
$$Match(v_0, v_2) = N(u_1) \cap N(u_2) = \{u_3, u_4, \ldots, u_{1003}\}$$

Distributed Subgraph Enumeration

(VLDB2015 & VLDB2017)

Single CPU

QuickSI (VLDB2008)



1. Determine the access order in *q*
2. Detect corresponding subgraphs in g_1, g_2 which can be mapped to the currently traversed vertices

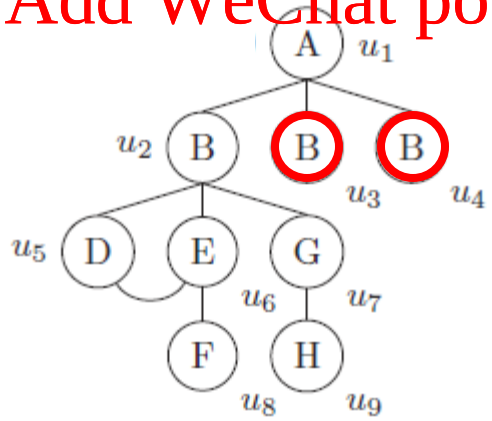
TurboISO (SIGMOD2013)

- Overview of TurboISO
  - ➢ Neighborhood Equivalence Class (NEC)
    - Merging vertices with same neighbor to reduce query size
    - Combine / Permute strategy
  - ➢ Candidate Region Exploration
    - Constructed on-the-fly based on q
    - A path-based data structure containing all embeddings of q in G

Neighborhood Equivalence Class(NEC)

- Definition $u_i(\in V(q)) \simeq u_j(\in V(q))$
  - Let $\simeq$ be an *equivalence relation over all query vertices in q such that, if for every embedding m that contains* (u_i, v_x) *and* (u_j, v_y) $(v_x, v_y \in V(g))$, *there exists an embedding m' such that* $m' = m - \{(u_i, v_x), (u_j, v_y)\} \cup \{(u_i, v_y), (u_j, v_x)\}$

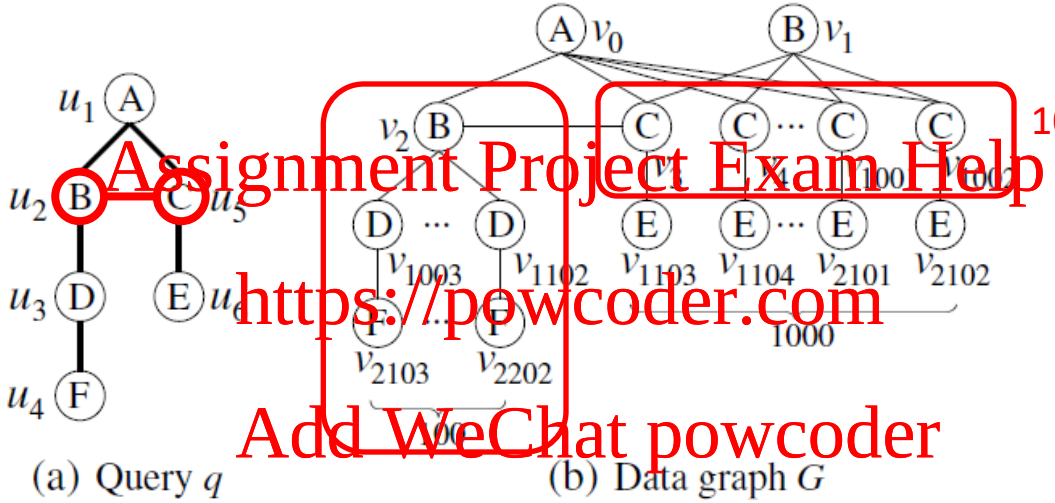
- Example



u_3 and u_4 are NEC nodes .

CFL-Match (SIGMOD2016)

Reduce Candidates



10^5 - 100 partial mappings are redundant.

Matching order of QuickSI and Turbo$_{ISO}$: (u_1, u_2, u_3, u_4, u_5, u_6).

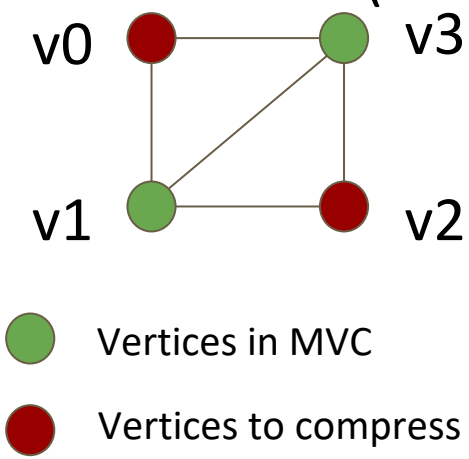
(u_1, u_2, u_5, u_3, u_4, u_6)

Cartesian products:

- 100 mappings (v_0, v_2, v_{1000+i}, v_{2100+i}) ($3 \le i \le 102$) of (u_1, u_2, u_3, u_4)
- 1000 mappings (v_0, v_j) ($3 \le j \le 1002$) of (u_1, u_5)

Compression

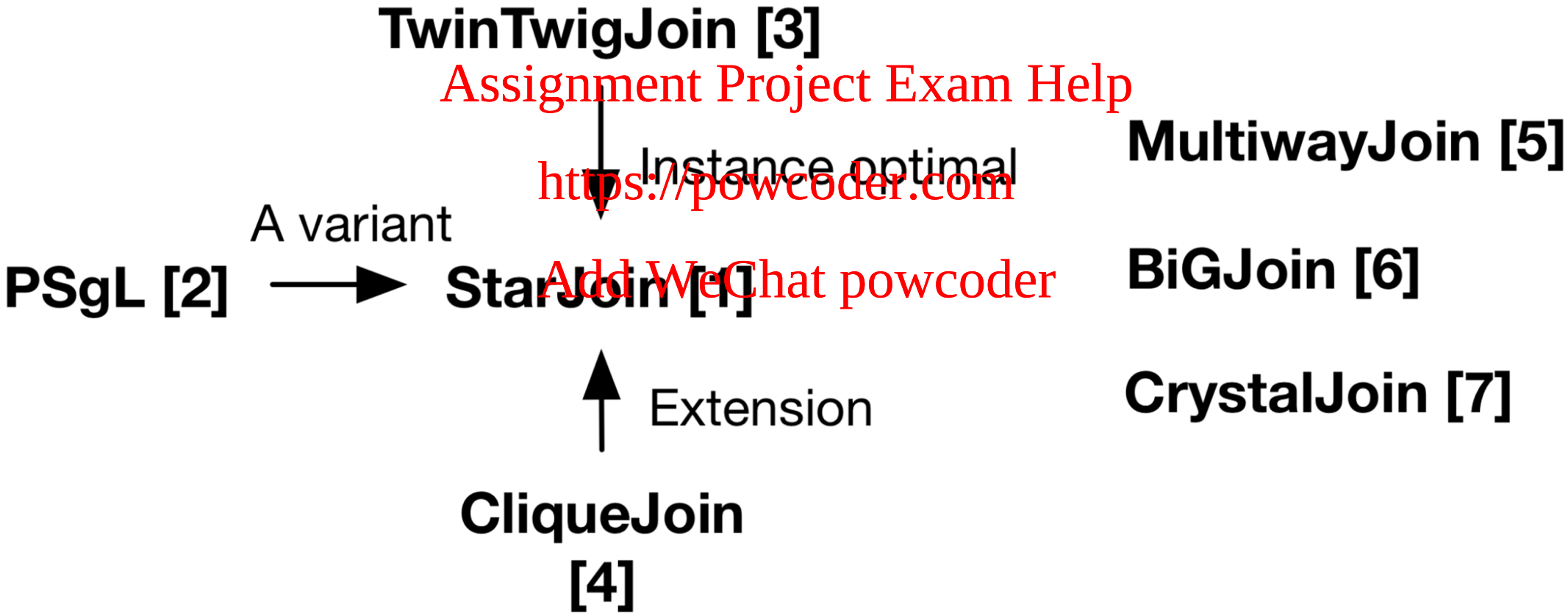
- Originally proposed by Qiao et al. [7]
- Intuition
  - Subgraph enumeration can generate enormous (intermediate) results
  - Some vertices can be compressed as they are not needed in future computation
    - Heuristics by [7]: the vertices that do **not** belong to the minimum vertex cover (MVC)



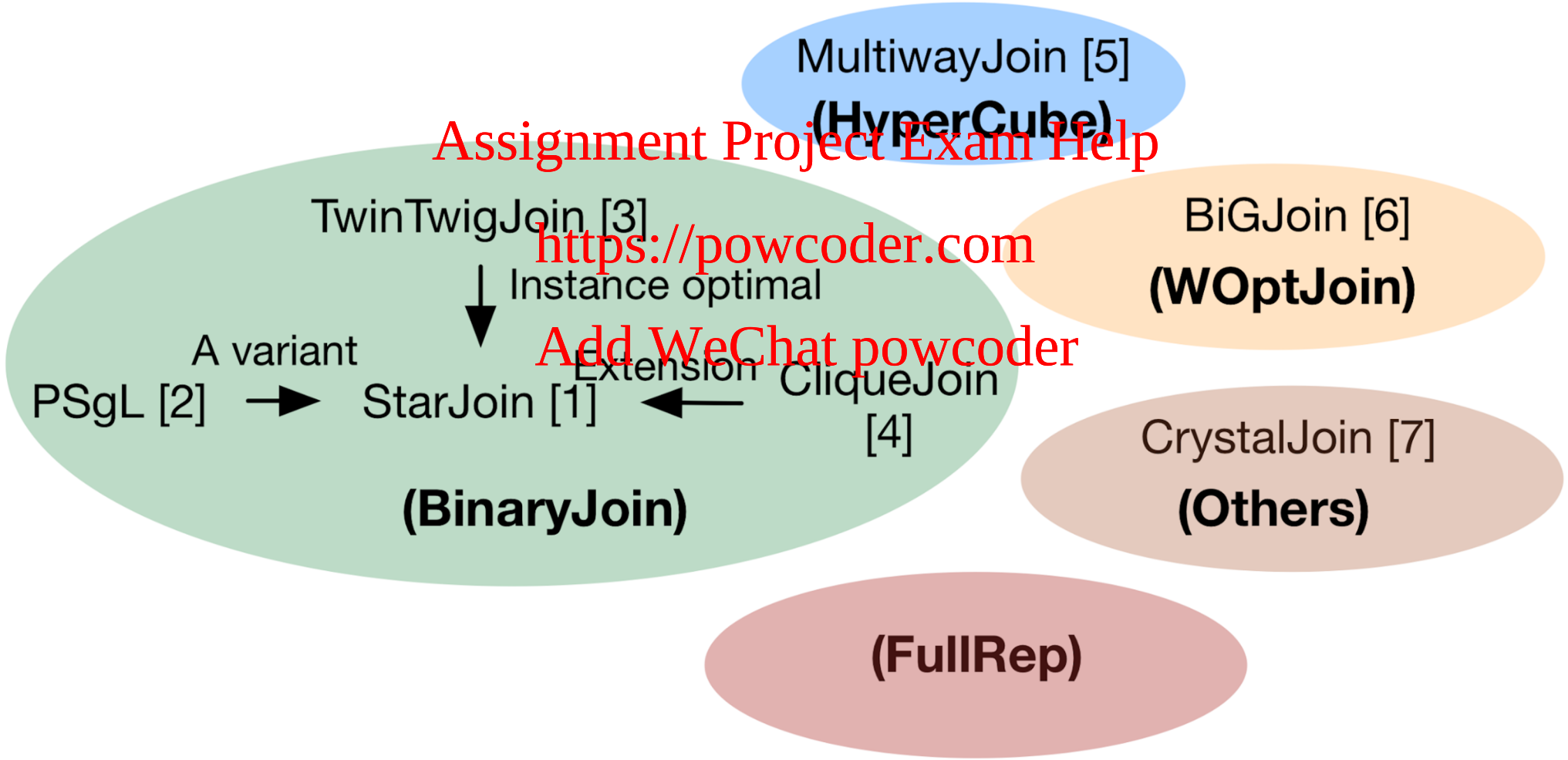
$$v_1 \to u_1, v_3 \to u_2$$
$$N(u_1) = \{u_2, u_3, \ldots, u_{1003}\}$$
$$N(u_2) = \{u_1, u_3, \ldots, u_{1003}\}$$
$$Match(v_0, v_2) = N(u_1) \cap N(u_2) = \{u_3, u_4, \ldots, u_{1003}\}$$

Distributed Techniques

A Thriving Literature



Categorizing by Strategies



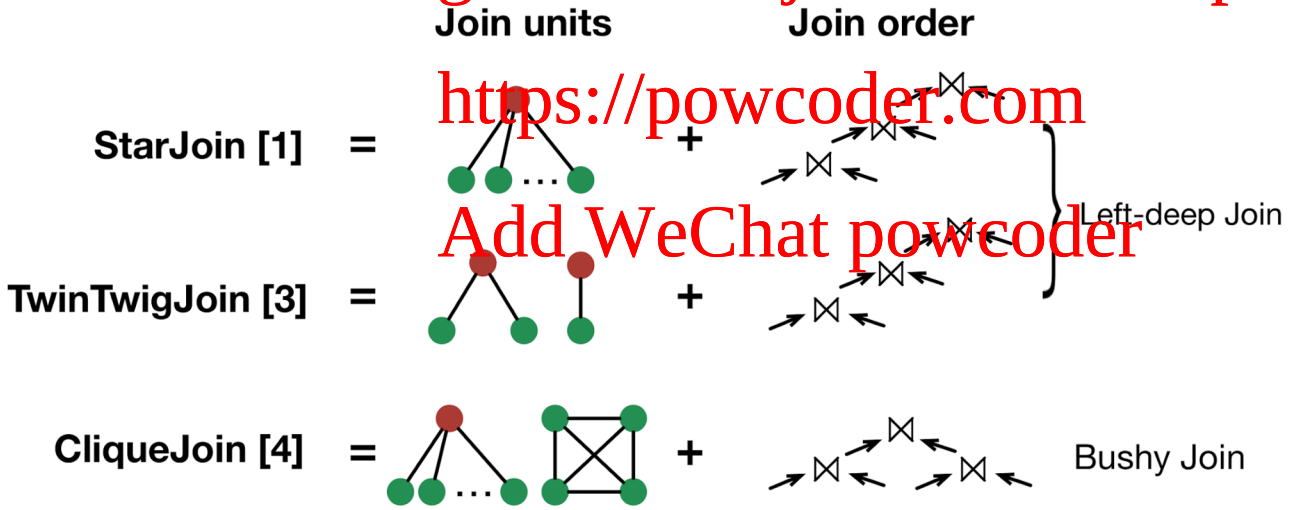
Related Works

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

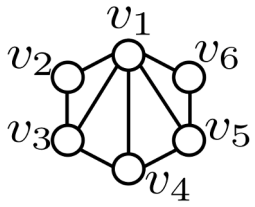
BinaryJoin Strategy

- Divide the pattern graph into a set of **join units** { p1, p2, …, pk }
- Process k-1 join following specific **join order**



- We prove that CliqueJoin is *worst-case optimal* by showing that it can be expressed as **GenericJoin** proposed by Ngo et al. [8]

StarJoin Algorithm

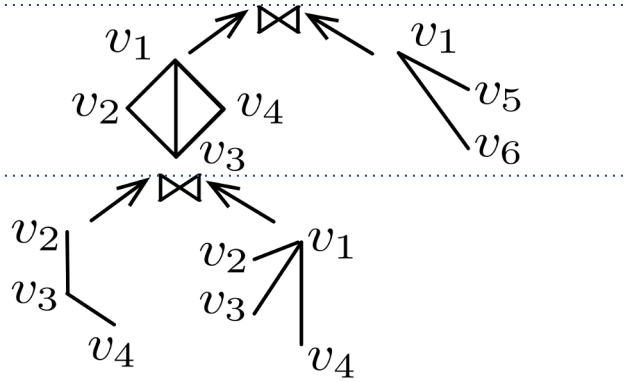


Round 4

Round 3

Round 2

Round 1



StarJoin

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 => matches of a 3-star 10^{18}

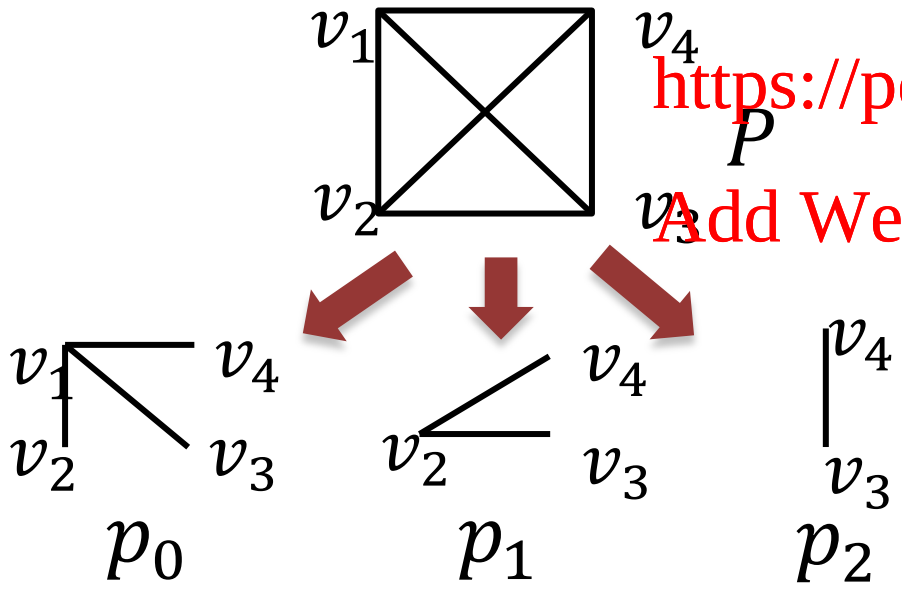
Not unusual: a node contains many neighbors in a large social networks or web graphs

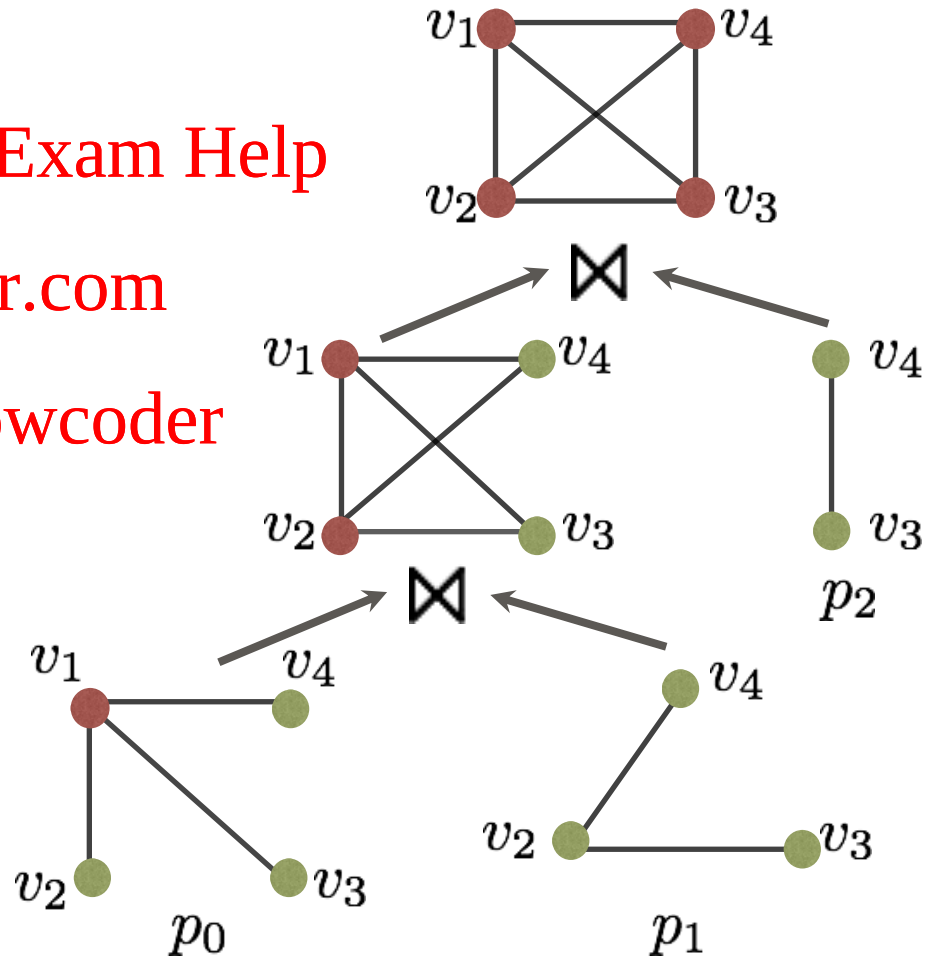
- If star with many edges, a join followed has more pruning power; i.e., less # intermediate results of a join.

Star-based Join

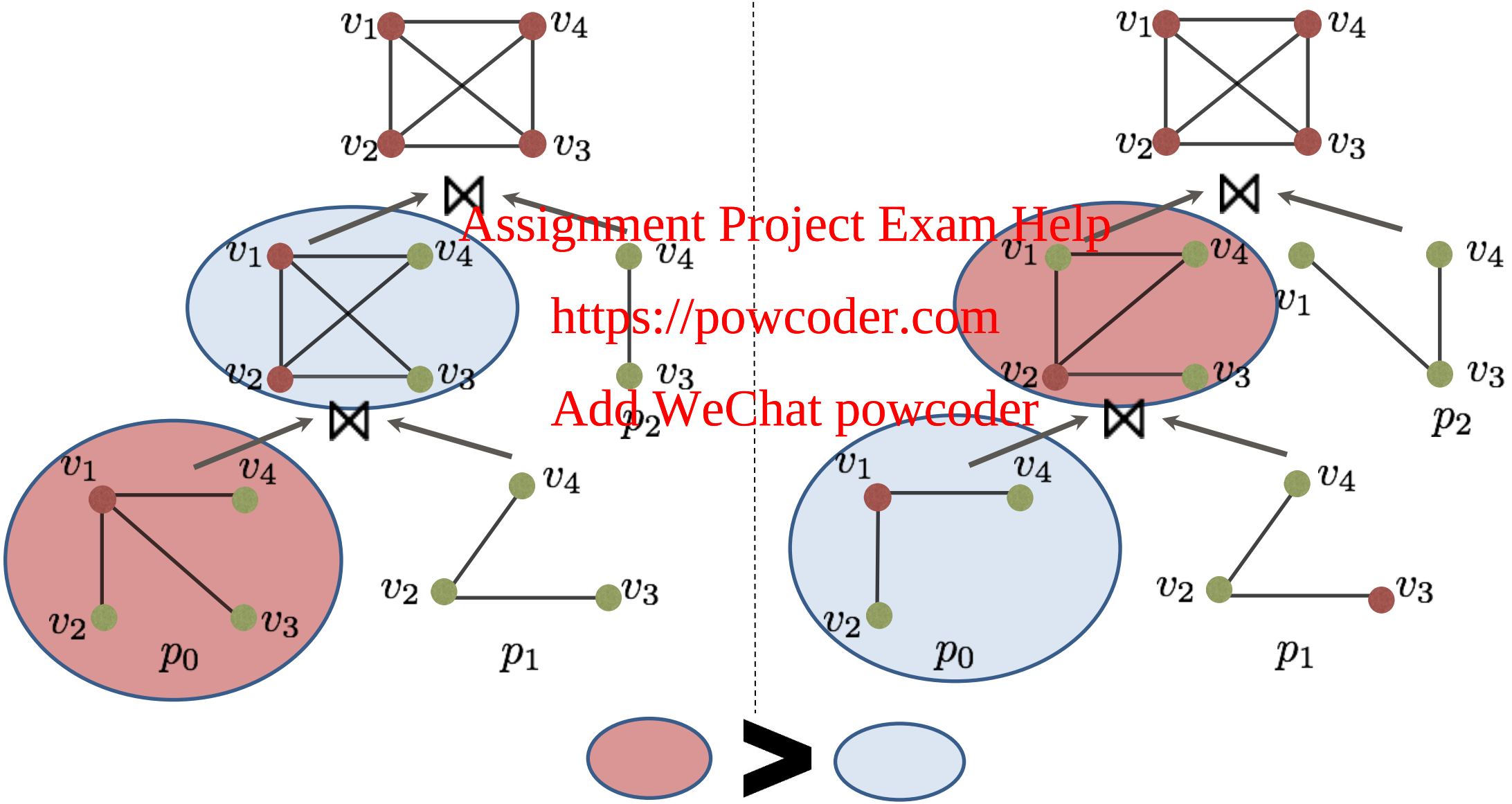
Star decomposition

Left-deep Join

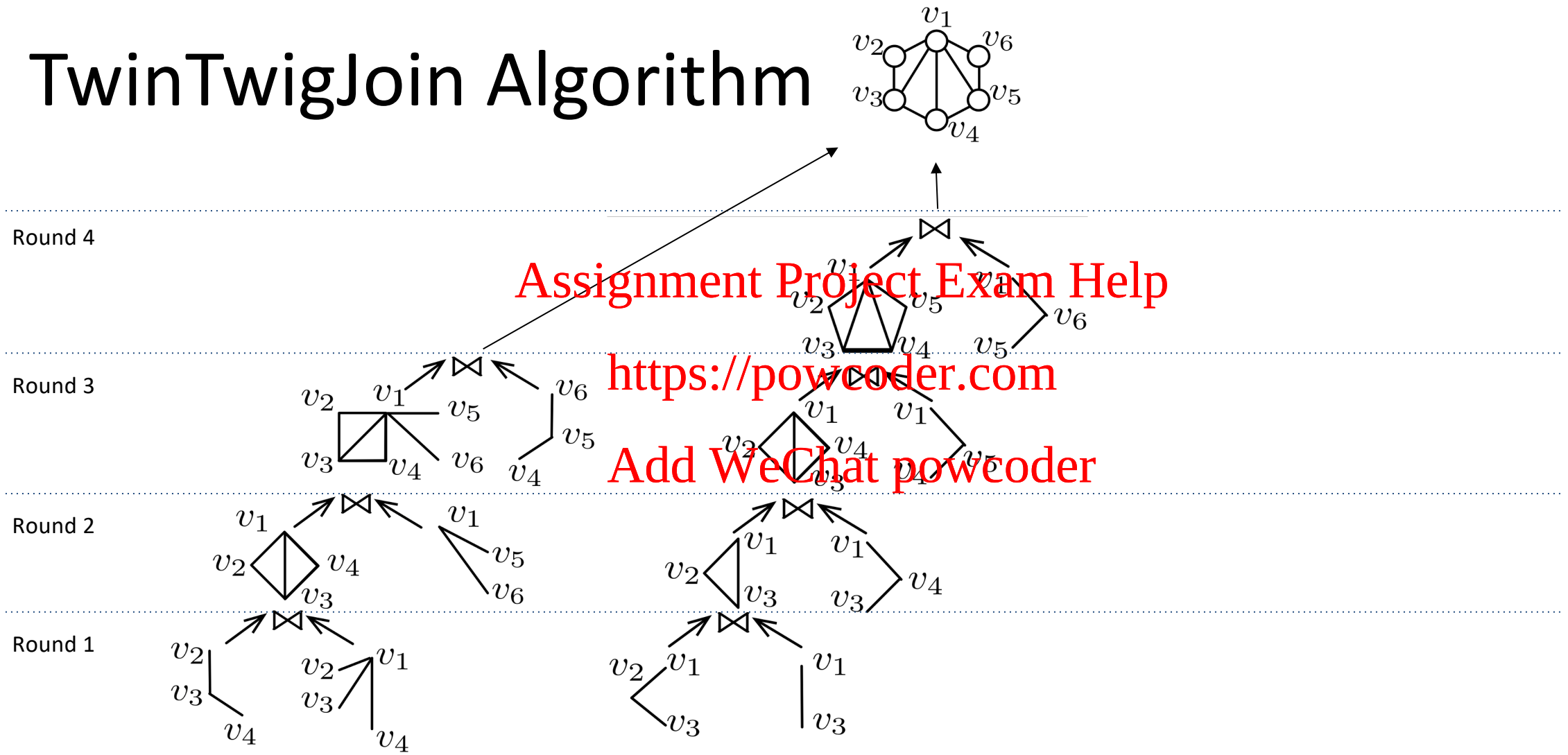




Star-based Join

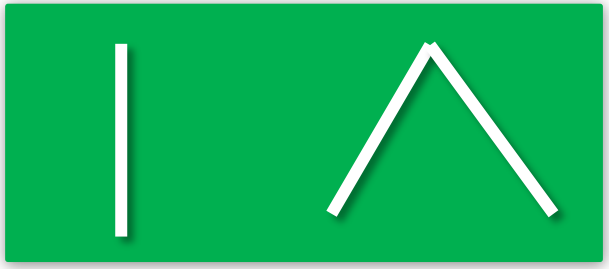


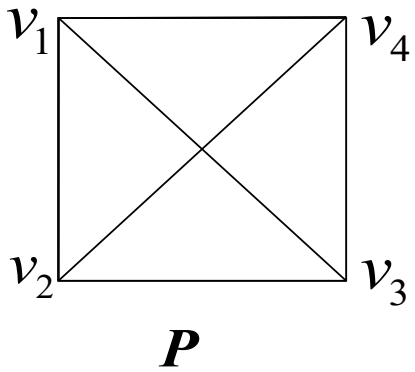
TwinTwigJoin Algorithm



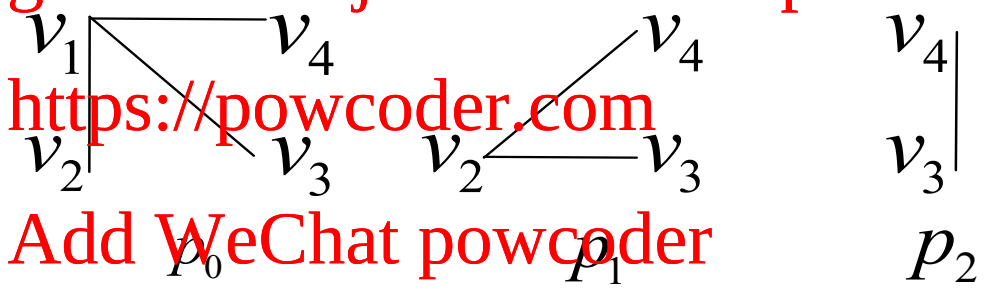
TwinTwig Join VLDB15'

TwinTwig: A star of at most two edges

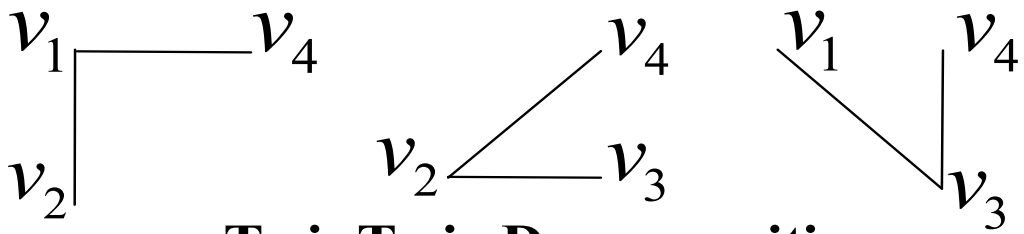




P



Star Decomposition

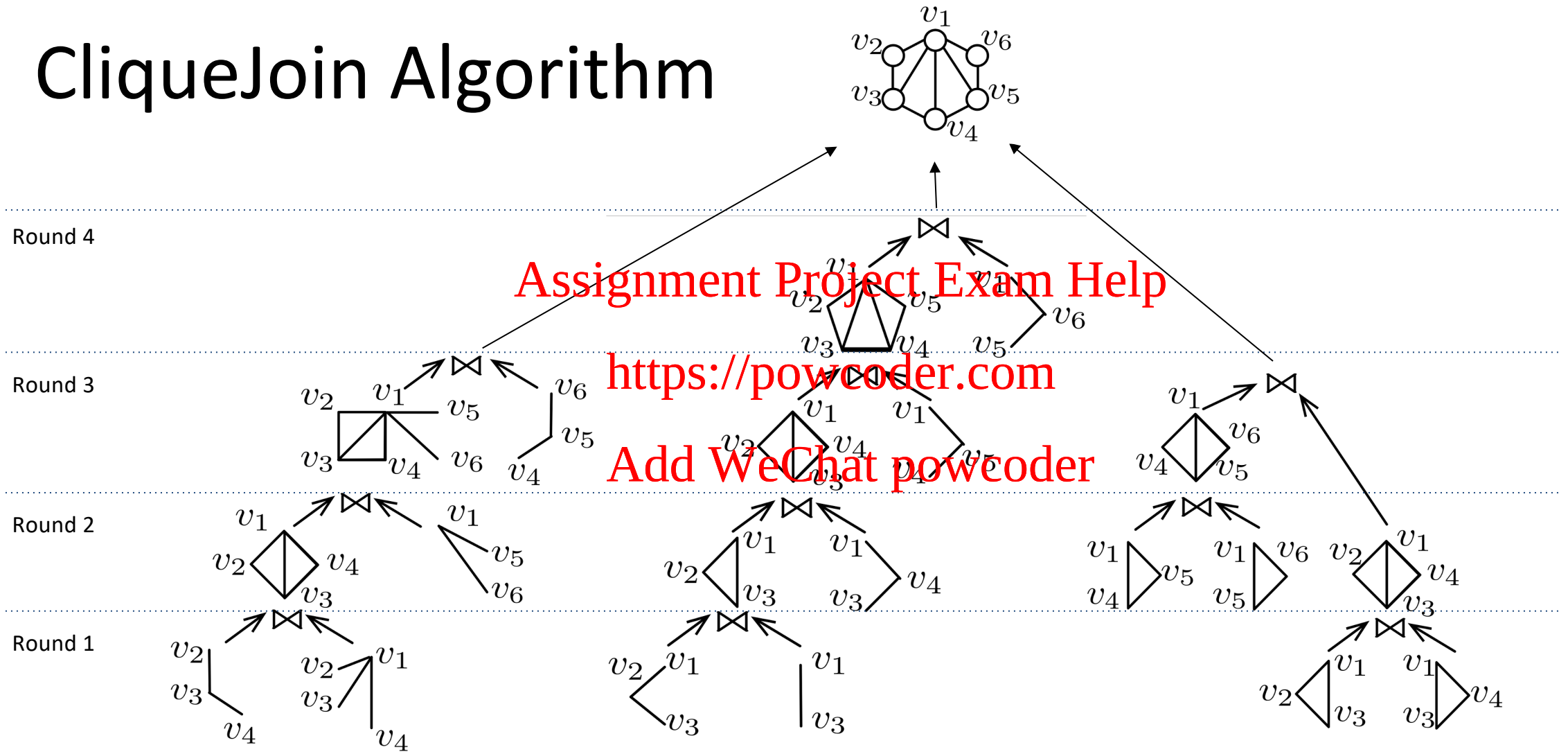


TwinTwig Decomposition

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.

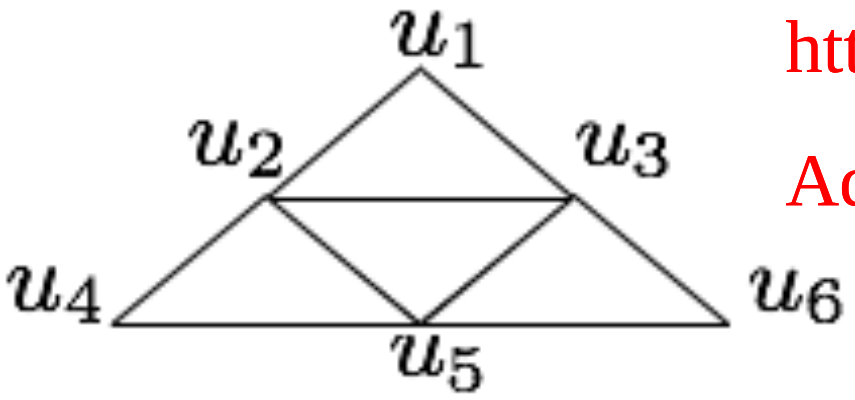
CliqueJoin Algorithm

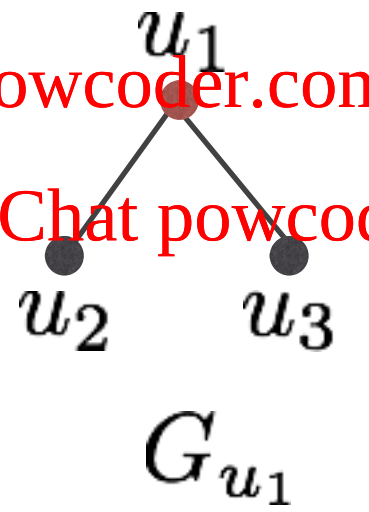


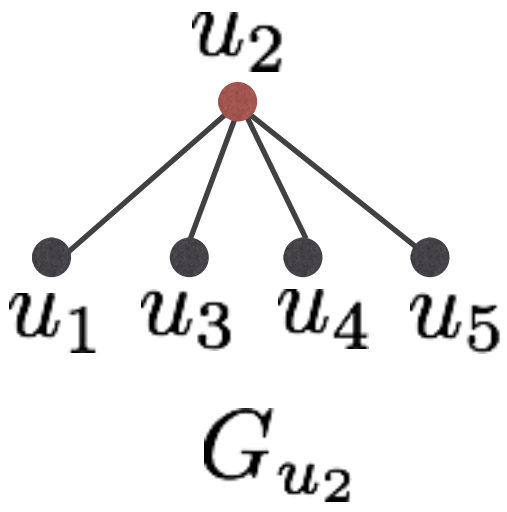
The drawbacks of TwinTwig Join

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*)







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***

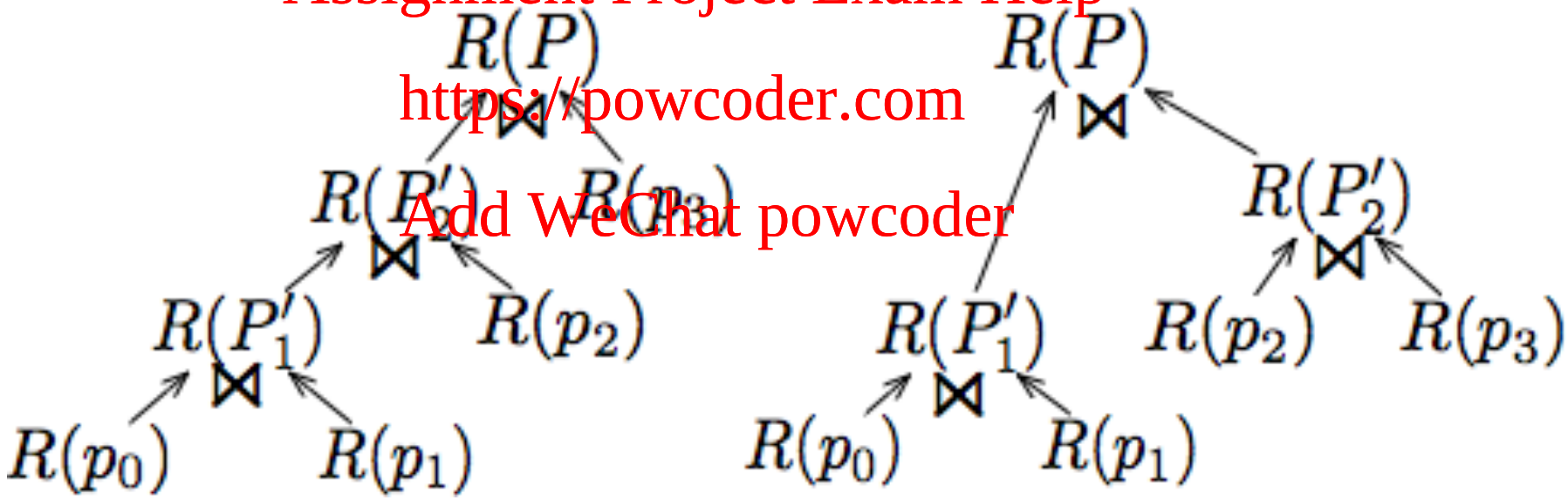
- *e.g.* ***Star*** *in Star-based join,* ***TwinTwig*** *in TwinTwig Join*
- A structure p can be the ***join unit*** iff.

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

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

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

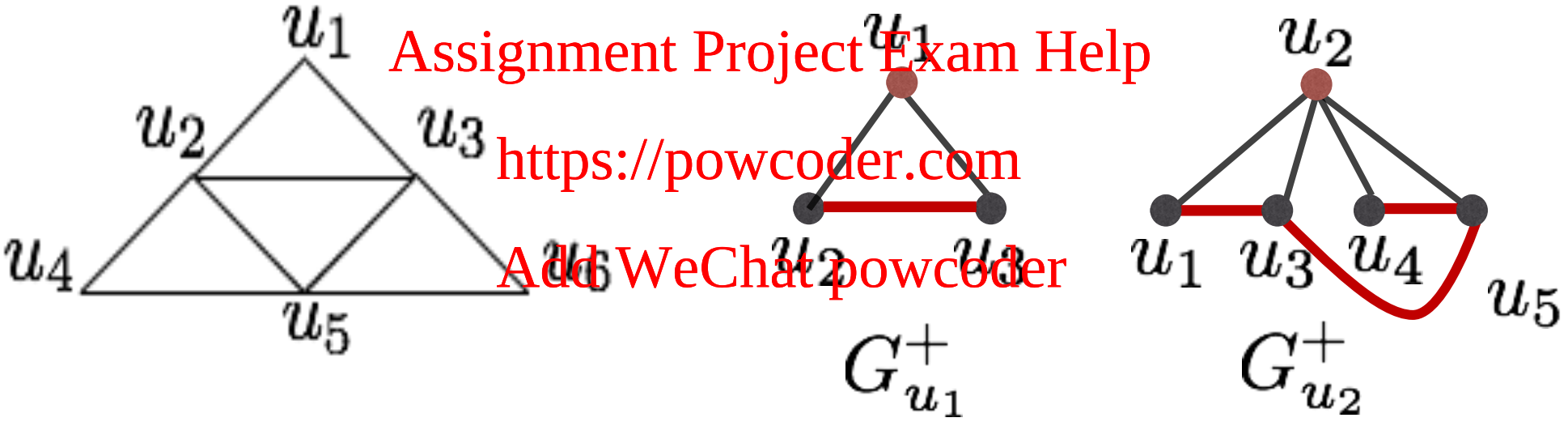
- We can avoid using star if clique is an alternative
- Shorter execution. The 6-clique can now be processed in one single round, instead of 7 rounds in TwinTwig Join

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^+



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

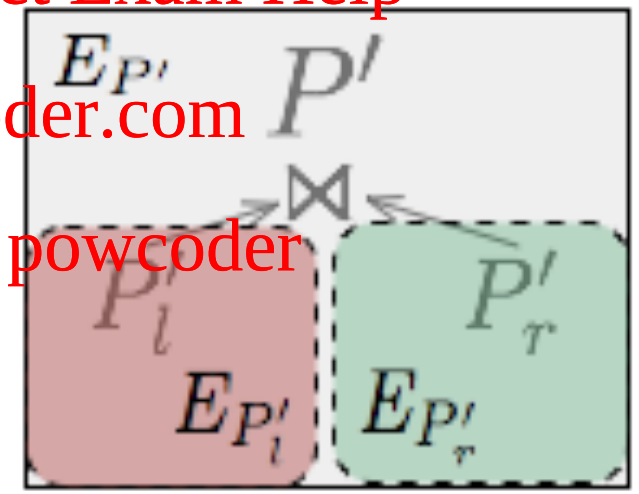
$C(P)$: estimated # matches of P in G (we apply powe-law random graph model for the estimation, while estimation can vary with applications)

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

Optimal Bushy Join

A dynamic programming transform function:

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



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

References

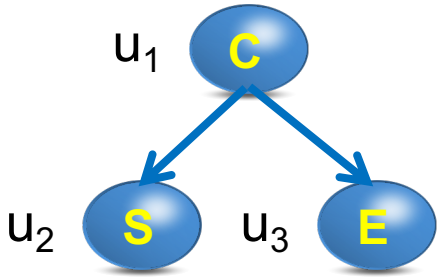
1. Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. PVLDB, 5(9), 2012.
2. Y. Shao, B. Cui, L. Chen, L. Ma, J. Yao, and N. Xu. Parallel subgraph listing in a large-scale graph. In SIGMOD'14, pages 625-636.
3. L. Lai, L. Qin, X. Lin, and L. Chang. Scalable subgraph enumeration in mapreduce. PVLDB, 8(10), 2015.
4. L. Lai, L. Qin, X. Lin, Y. Zhang, L. Chang, and S. Yang. Scalable distributed subgraph enumeration. PVLDB, 10(3), 2016.
5. F. N. Afrati, D. Fotakis, and J. D. Ullman. Enumerating subgraph instances using map-reduce. In Proc. of ICDE, 2013.
6. K. Ammar, F. McSherry, S. Salihoglu, and M. Joglekar. Distributed evaluation of subgraph queries using worst-case optimal low-memory dataflows. PVLDB, 11(6), 2018.
7. M. Qiao, H. Zhang, and H. Cheng. Subgraph matching: On compression and computation. PVLDB, 11(2), 2017.
8. H. Q. Ngo, E. Porat, C. Re, and A. Rudra. Worst-case optimal join algorithms. J. ACM, 65(3), 2018.
9. H. Kim, J. Lee, S. S. Bhowmick, W.-S. Han, J. Lee, S. Ko, and M. H. Jarrah. Dualsim: Parallel subgraph enumeration in a massive graph on a single machine. SIGMOD '16, pages 1231{1245, 2016.

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

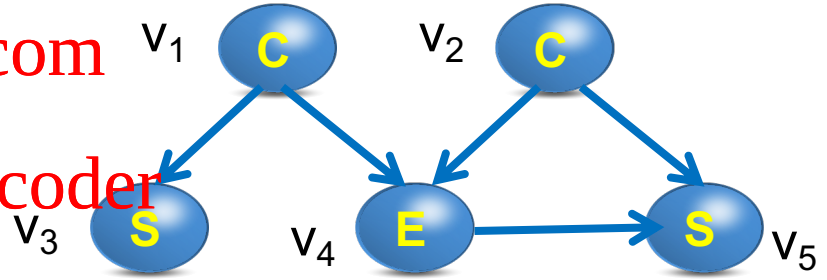
Introduction

- 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.

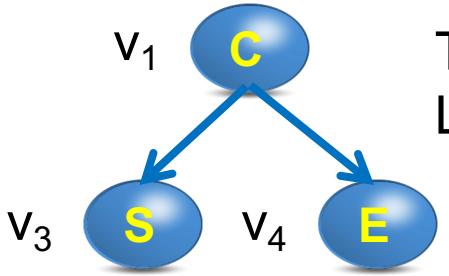
Query Tree *T*:



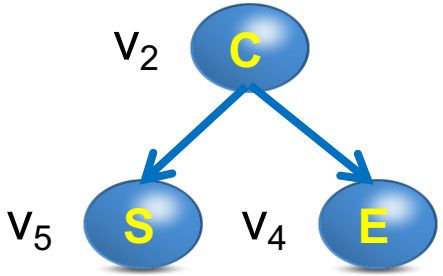
Data Graph *G*:



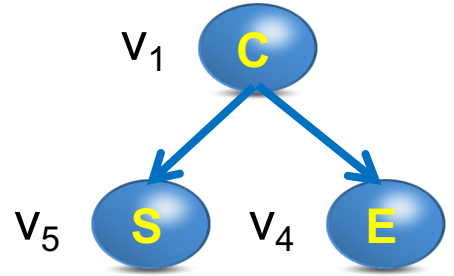
Top-1 Match
Length=2



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

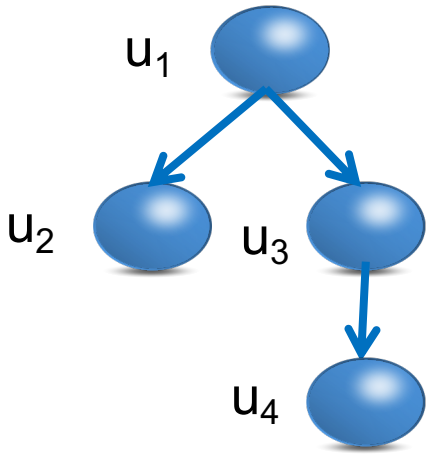
A New Enumeration Strategy

- 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: $\{v_1\} \times \{v_2\} \times \ldots \times \{v_i\} \times \ldots \times (V_j - U_j - v_j) \times \ldots \times V_n$
    - » $(n - i)$ 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).

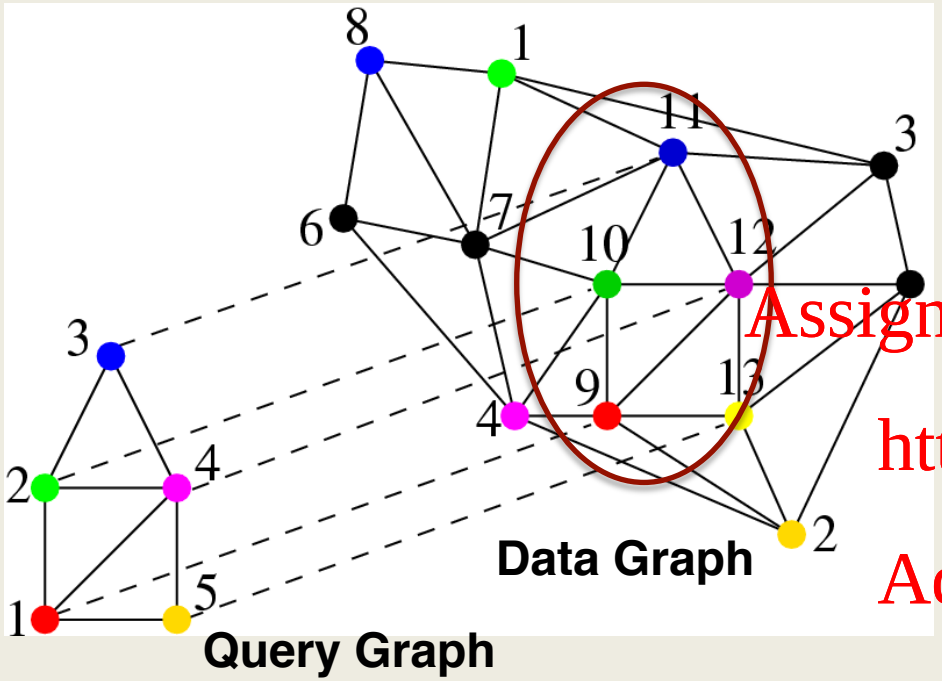


An Optimal Approach

- 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*

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).

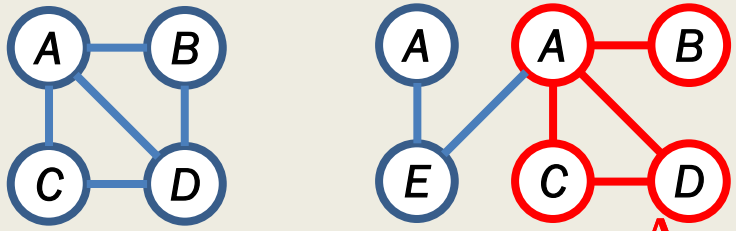
Algorithms and publications:

- 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

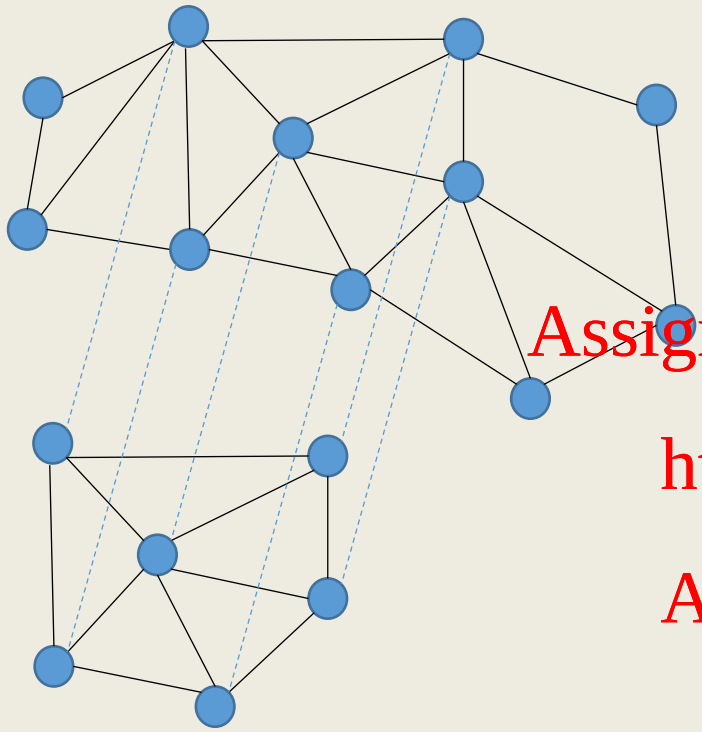
Contributions:

- 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

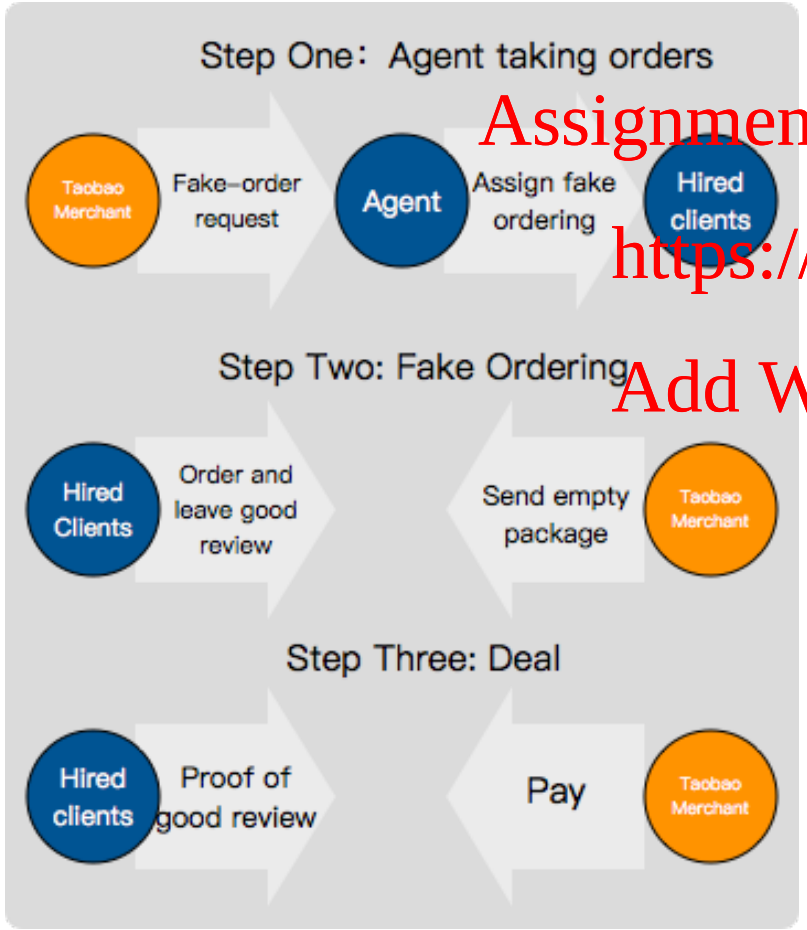
Contributions:

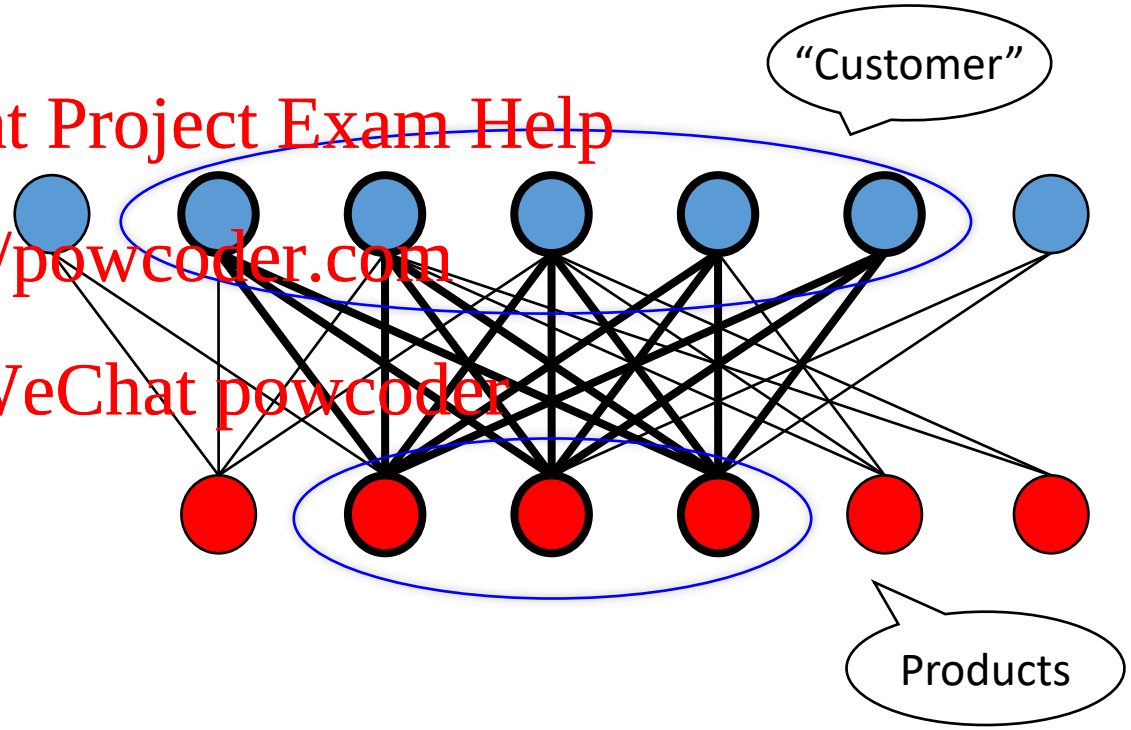
- Propose the index called DGTree that is free from the costly subgraph mining procedure.
- Based on DGTree, propose the algorithm that is free from verification
- 8 times faster than the state-of-the-art

COHESIVE SUBGRAPHS

Applications

Fraud detection



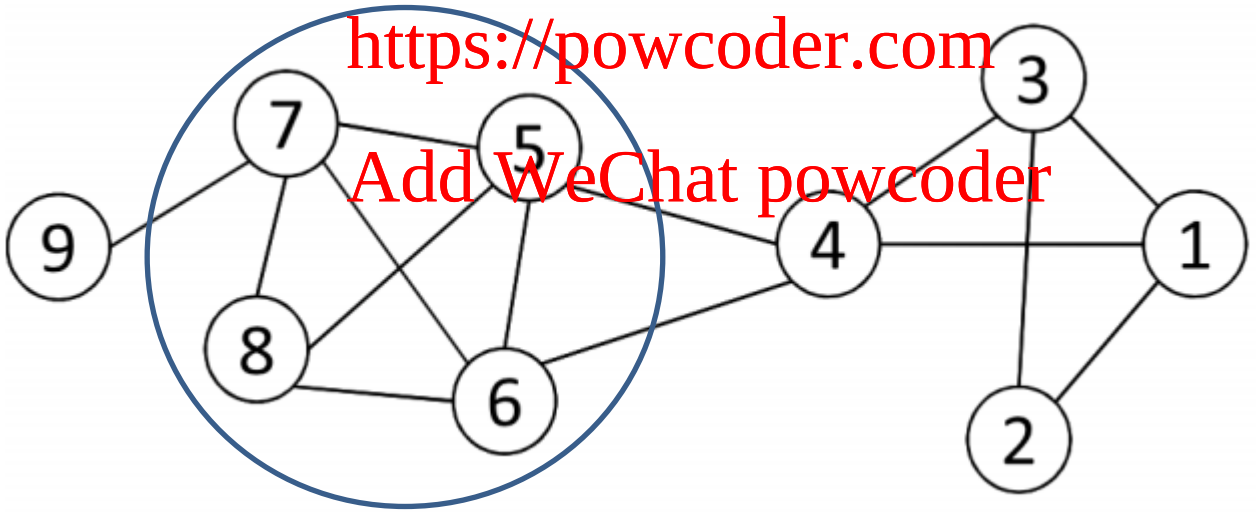


Cohesive Subgraph Models

- Global cohesiveness: *cliques and variants*
- Node and edge constraints: *k-core, k-truss, k-fortress*
- Connectivity: *k-ECC, k-VCC*

Cliques

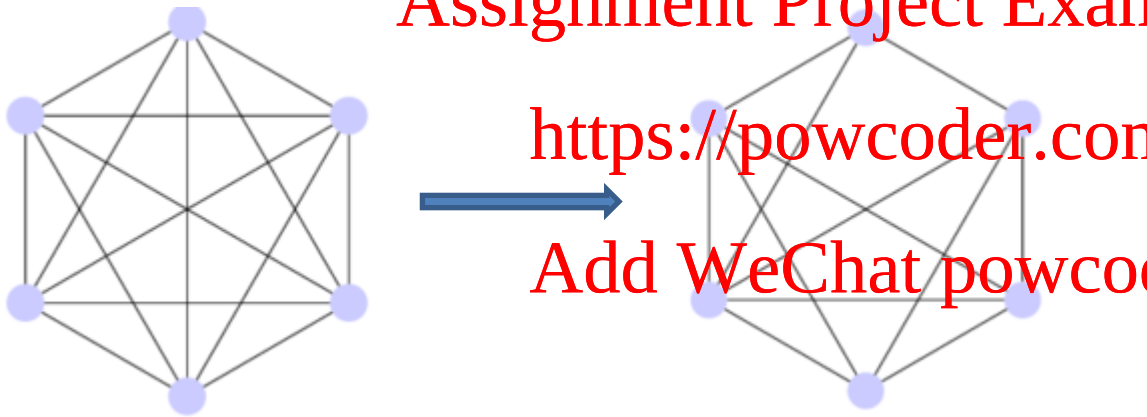
- Every pair of vertices pair is connected
- A clique is called maximal clique if there exist no other bigger cliques that contain it
- Also called complete graph



R. D. Luce and A. D. Perry, "A method of matrix analysis of group structure," *Psychometrika*, vol. 14, no. 2, pp. 95–116, 1949

Variations of cliques: quasi-clique

- Quasi-clique: relax on density or degree.



total # of edges: $\frac{|V|(|V|-1)}{2}$ to $\gamma \frac{|V|(|V|-1)}{2}$

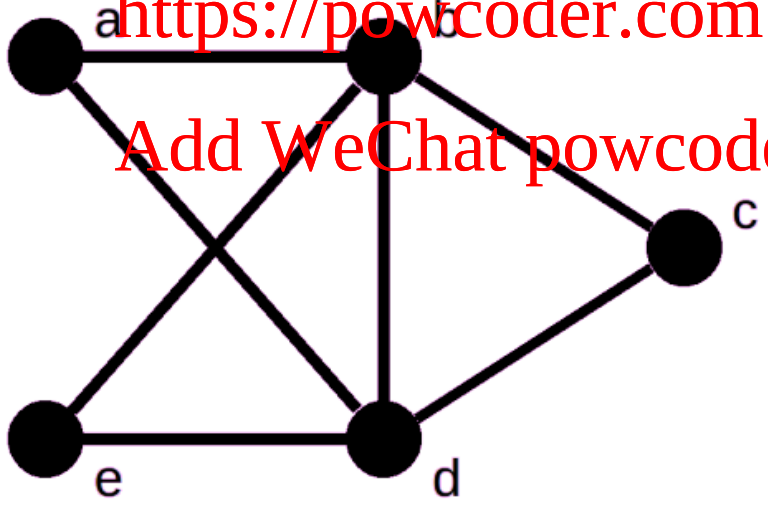
$\gamma = 0.8$

H. Matsuda, T. Ishihara, A. Hashimoto. Classifying molecular sequences using a linkage graph with their pairwise similarities. Theor. Comput. Sci., 1999

Variations of cliques: k-plex

- K-plex: relax vertex degree from *n-1* to *n-k*, n is number of vertices in the subgraph

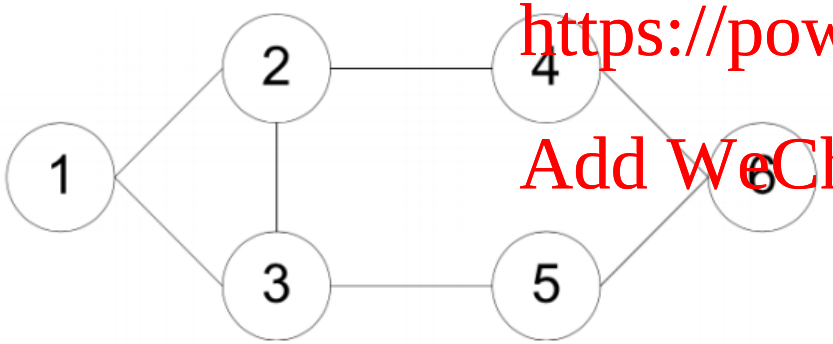
{a, b, e, d} <-- 2-plex



S. Seidman and B. Foster. “A graph-theoretic generalization of the clique concept. Journal of Mathematical Sociology”, 139-154, 1978.

Variations of cliques: k-clique and k-club

- k-clique: the subgraph such that the distance between any two vertices in original graph is within k
- k-club: a subgroup with diameter within k



Clique: (1, 2, 3)

2-clique: (1,2,3,4,5),(2,3,4,5,6,)

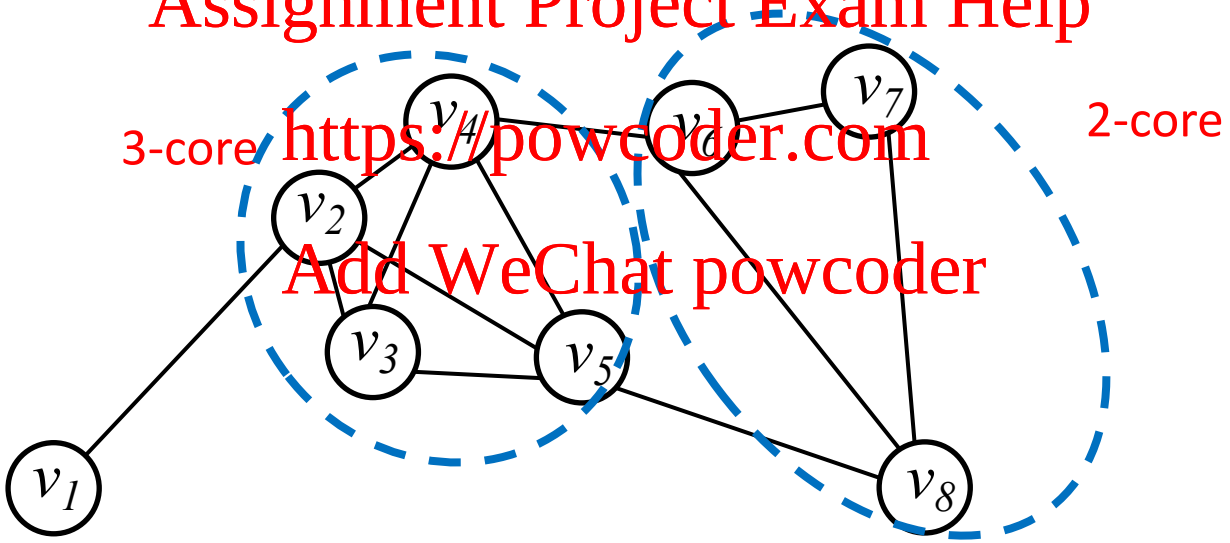
2-club: (1,2,3,4),(1,2,3,5),(2,3,4,5,6)

R.D. Luce. Connectivity and generalized cliques in sociometric group strucuture. Psychometrika, 15(2): 169-190, 1950.

R.J. Mokken. Clique, clubs and clans. Quality and Quantity, 13(2): 161-173, 1979.

K-core

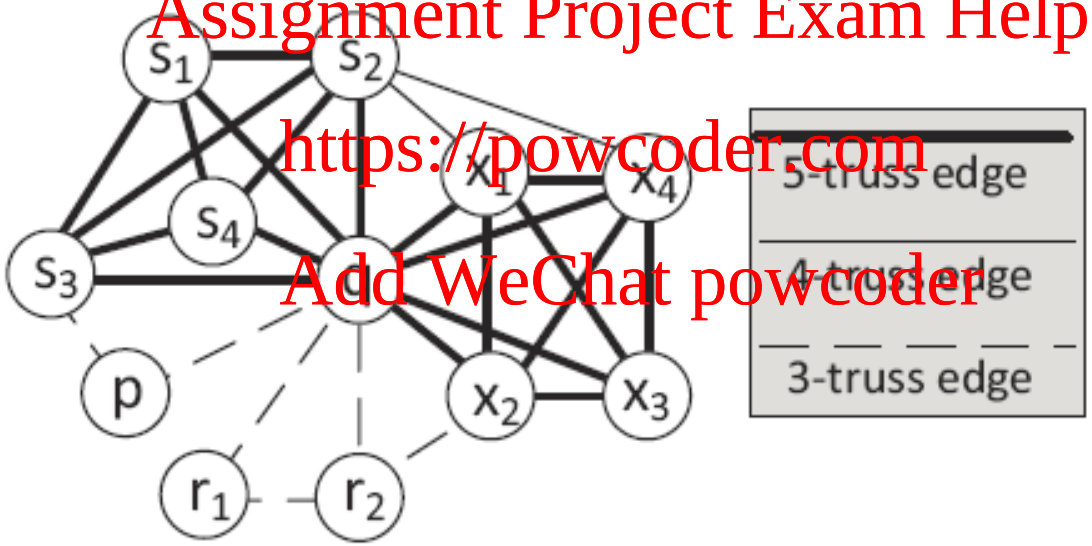
Given a graph *G*, the *k*-core of *G* is a maximal subgraph where each vertex has at least *k* neighbors.



S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

K- truss

- A maximal subgraph where each edge has a support of at least *k-2.* Edge support is the number of triangles which contains the edge.

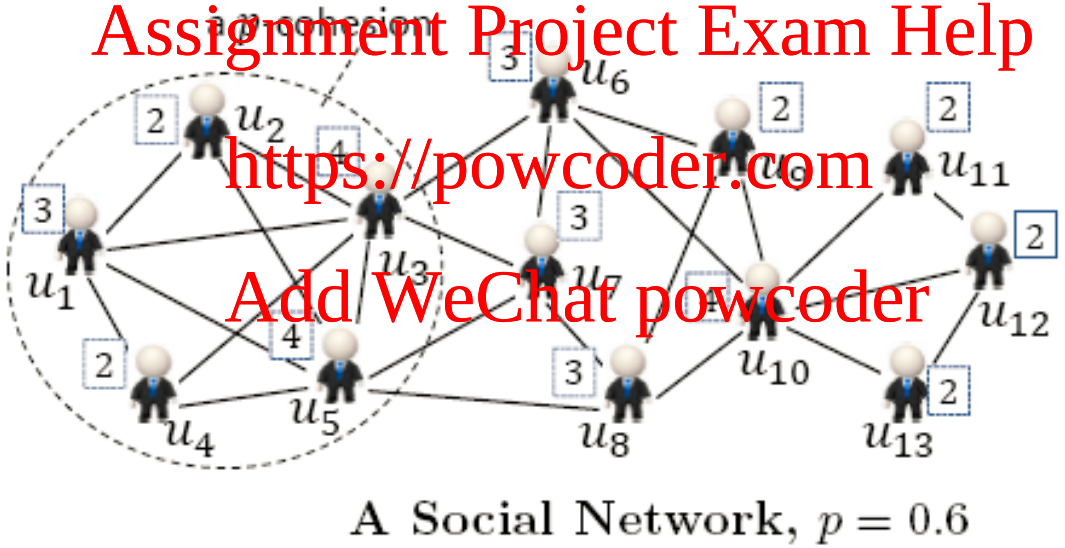


M. Granovetter. The strength of weak ties. American Journal of Sociology, 1360-1380, 1973.

M. Wang, C. Wang, J. X. Yu, and J. Zhang, Community detection in social networks: An in-depth benchmarking study with a procedure-oriented framework. *PVLDB*, 8(10):998–1009, 2015.

p-cohesion

- A subgraph S where the degree of each vertex u in S is at least p fraction of the whole graph. deg(u, S) ≥ p× deg(u, G)



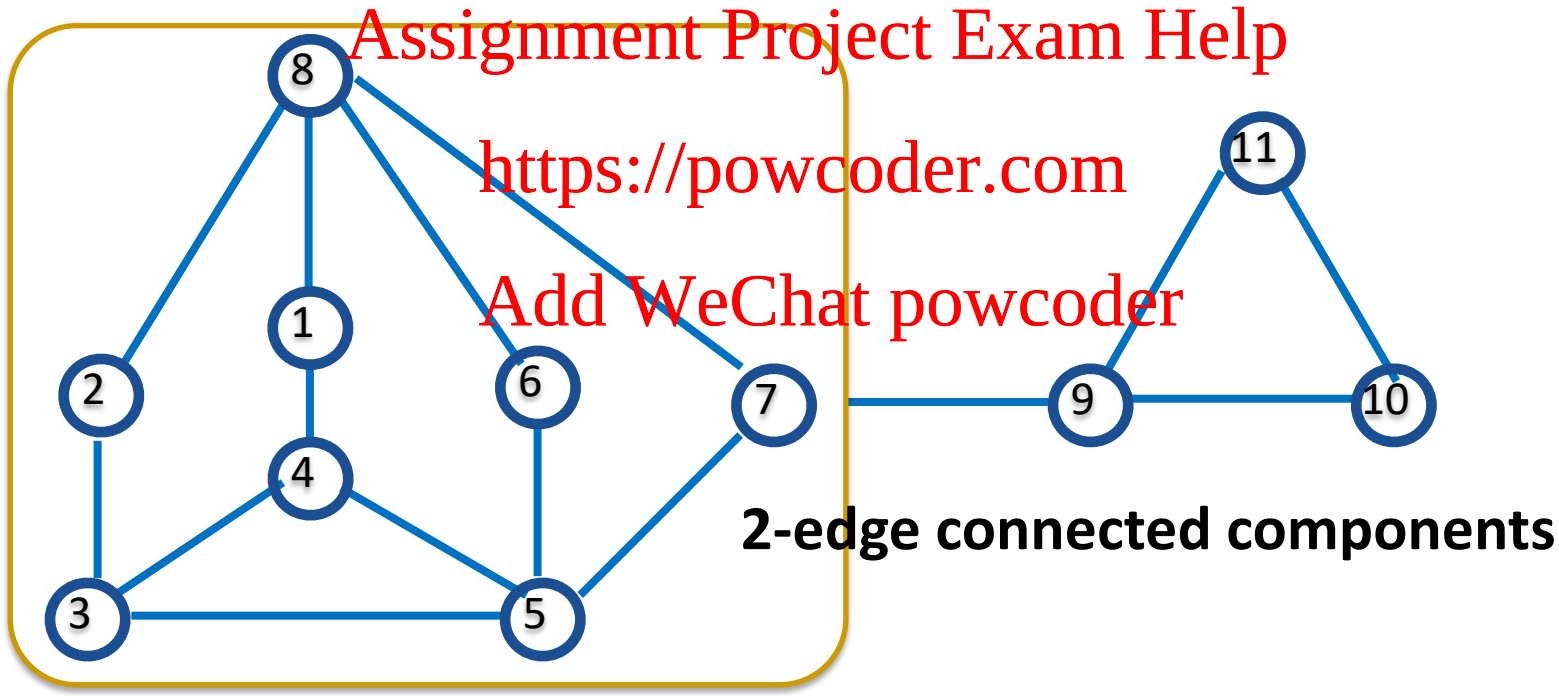
A Social Network, $p = 0.6$

S. Morris, Contagion. The Review of Economic Studies 67(1): 57-78, 2000.

C. Li, F. Zhang, Y. Zhang, L. Qin, W. Zhang, X. Lin, Discovering Fortress-like Cohesive Subgraphs, under review.

K-edge connected components

Given a graph G, a subgraph is k-edge connected if it is still connected after removing any set of (k-1) edges from it

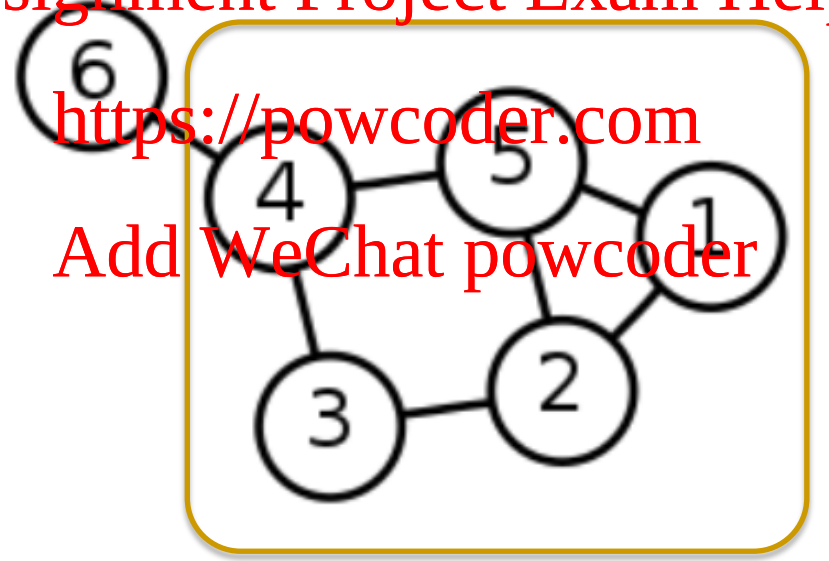


2-edge connected components

J., Camille . "Sur les assemblages de lignes". *Journal für die reine und angewandte Mathematik* (in French). 70 (2): 185–190, 1869.

K-vertex connected components

- Given a graph G, a subgraph is k-vertex connected if it is still connected after removing any set of (k-1) vertices from it



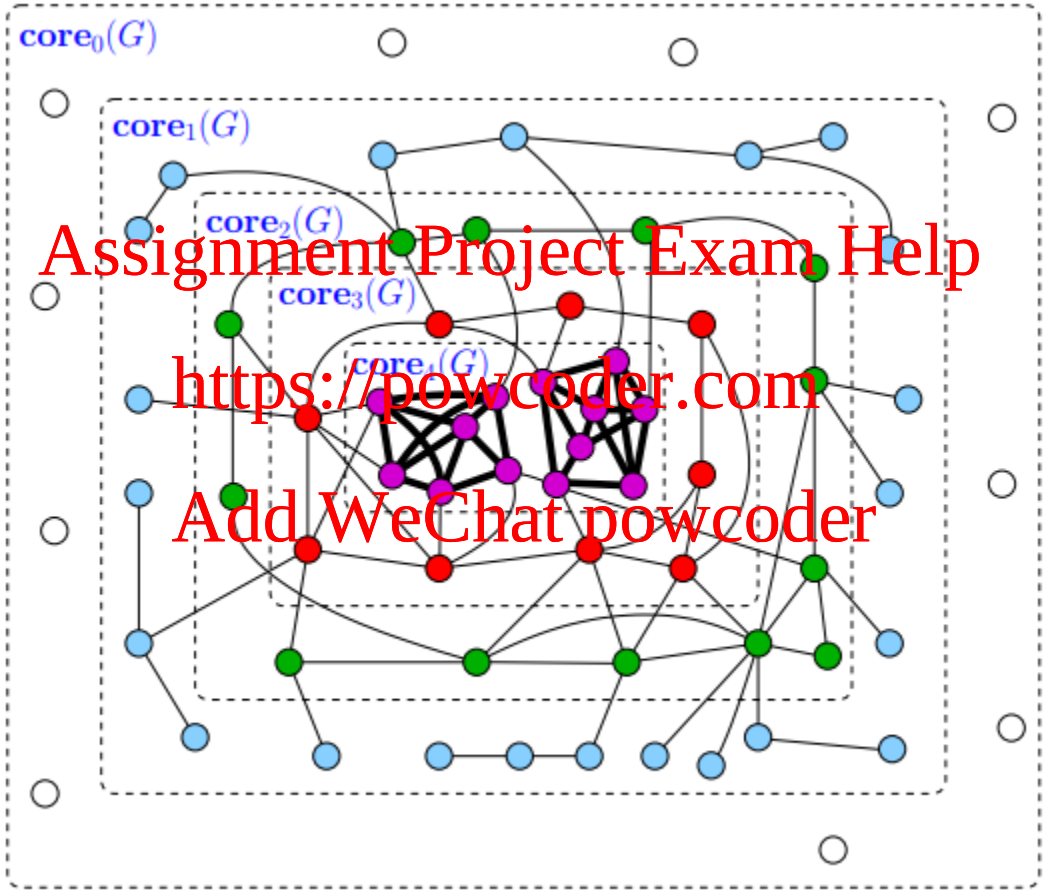
2-vertex connected components

S. Even and R. E. Tarjan. Network flow and testing graph connectivity. SIAM J. Comput., 4, 1975.

Hierarchical decomposition

- Compute cohesive subgraphs w.r.t all possible parameters, e.g., *k*-core for all possible *k* values

Hierarchical decomposition



C. Giatsidis *et al,* Journal of Machine Learning Research, 2016.

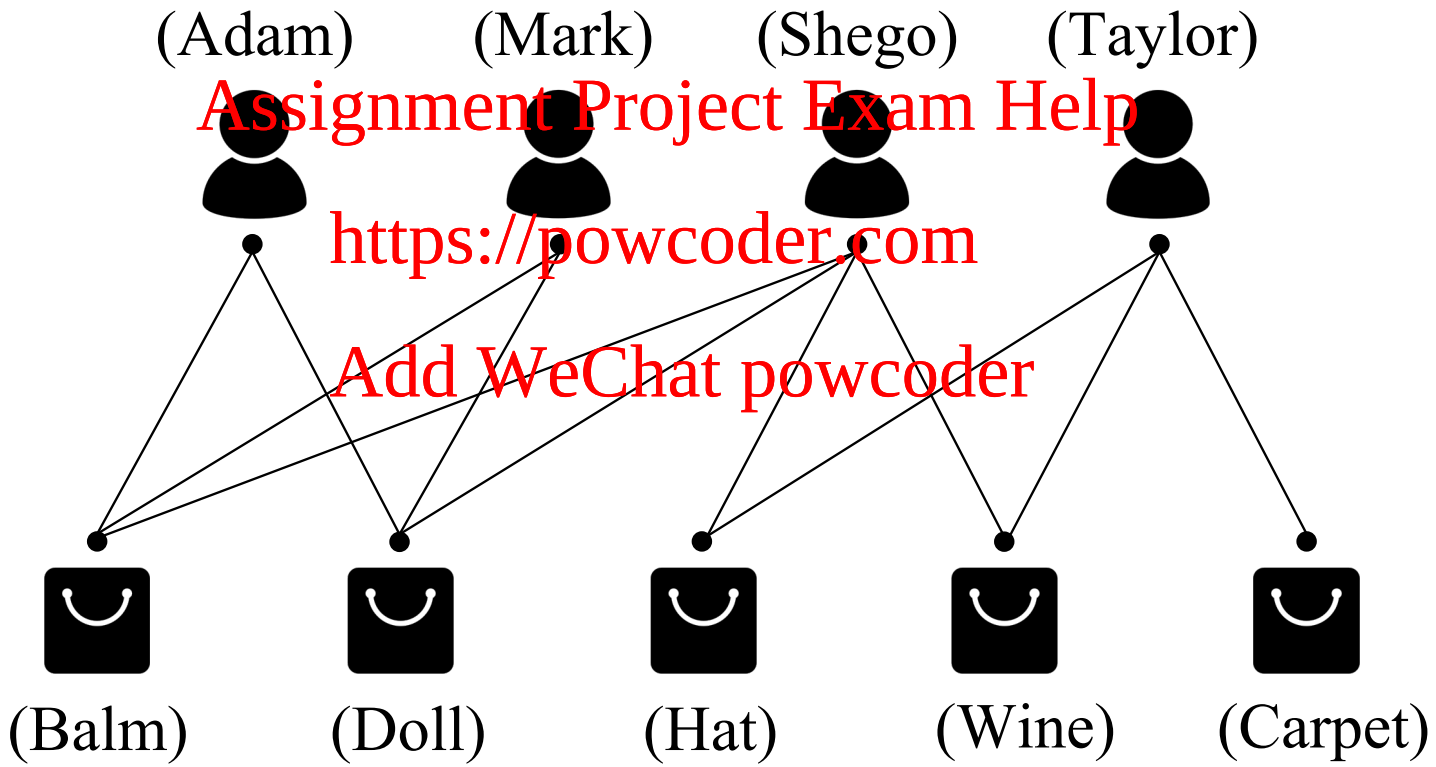
Different Cohesive Models

- Level of cohesiveness
- Computational cost
- Different applications

Butterfly Counting for Large-scale Bipartite Networks

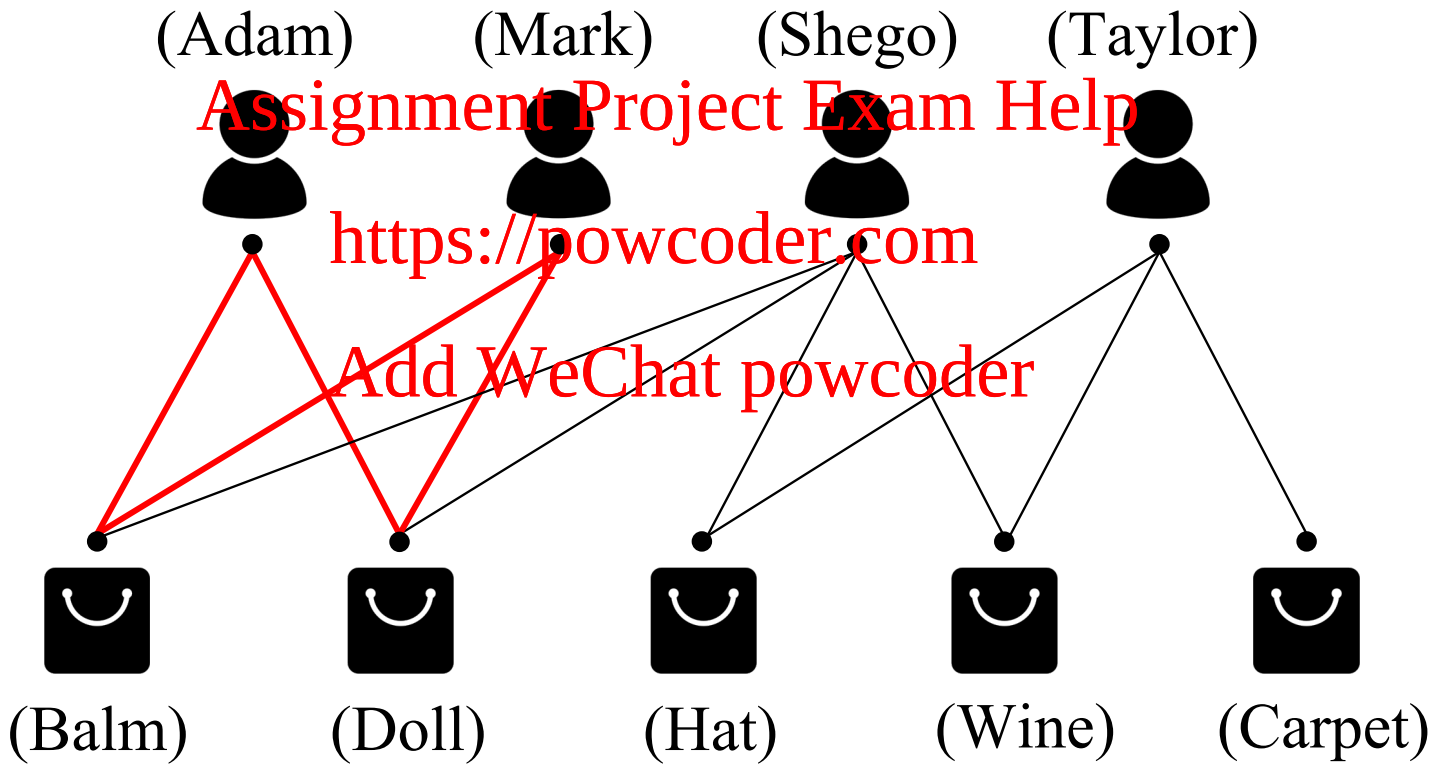
Introduction

Bipartite network:



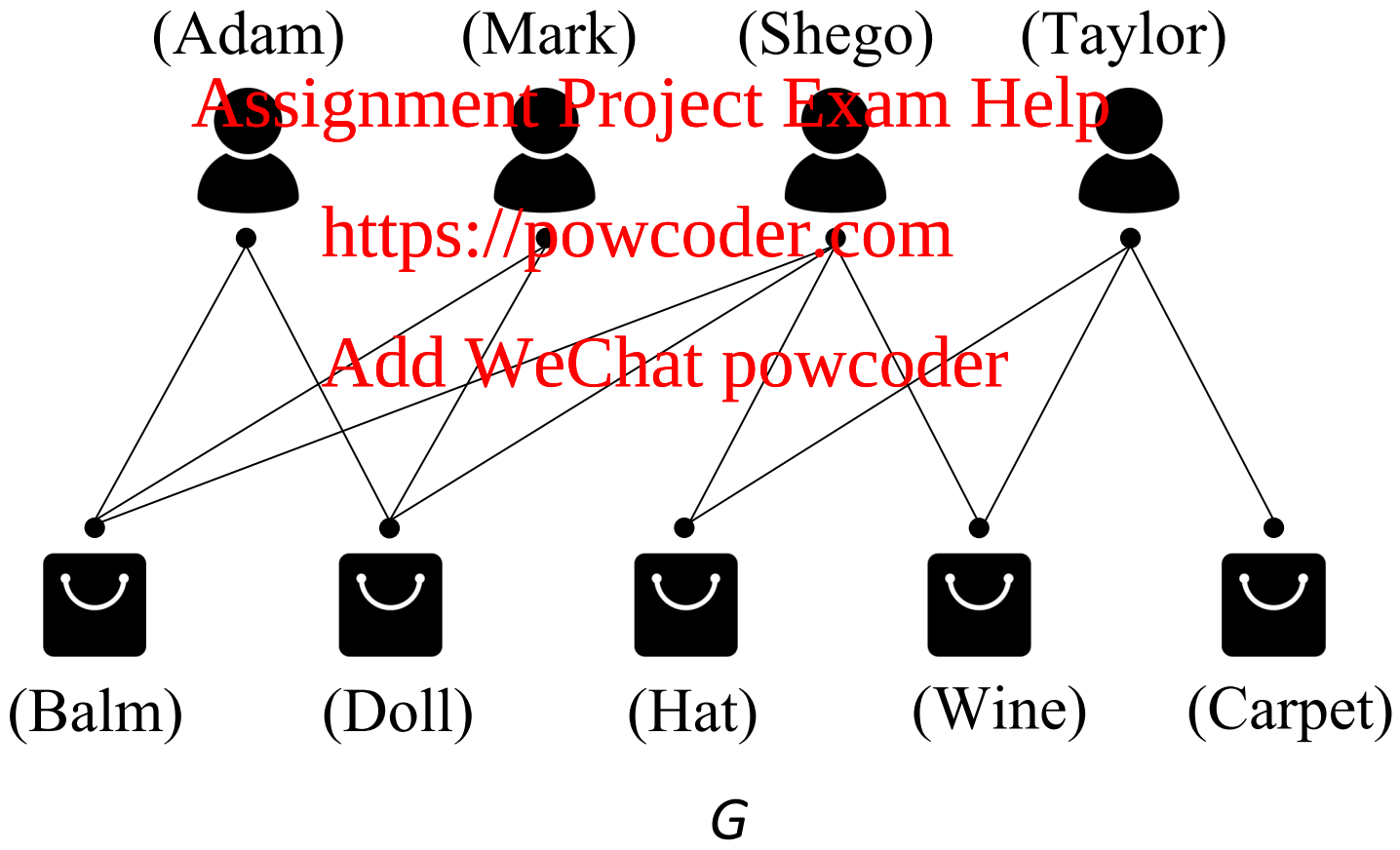
Introduction

The smallest non-trivial biclique: butterfly



Introduction

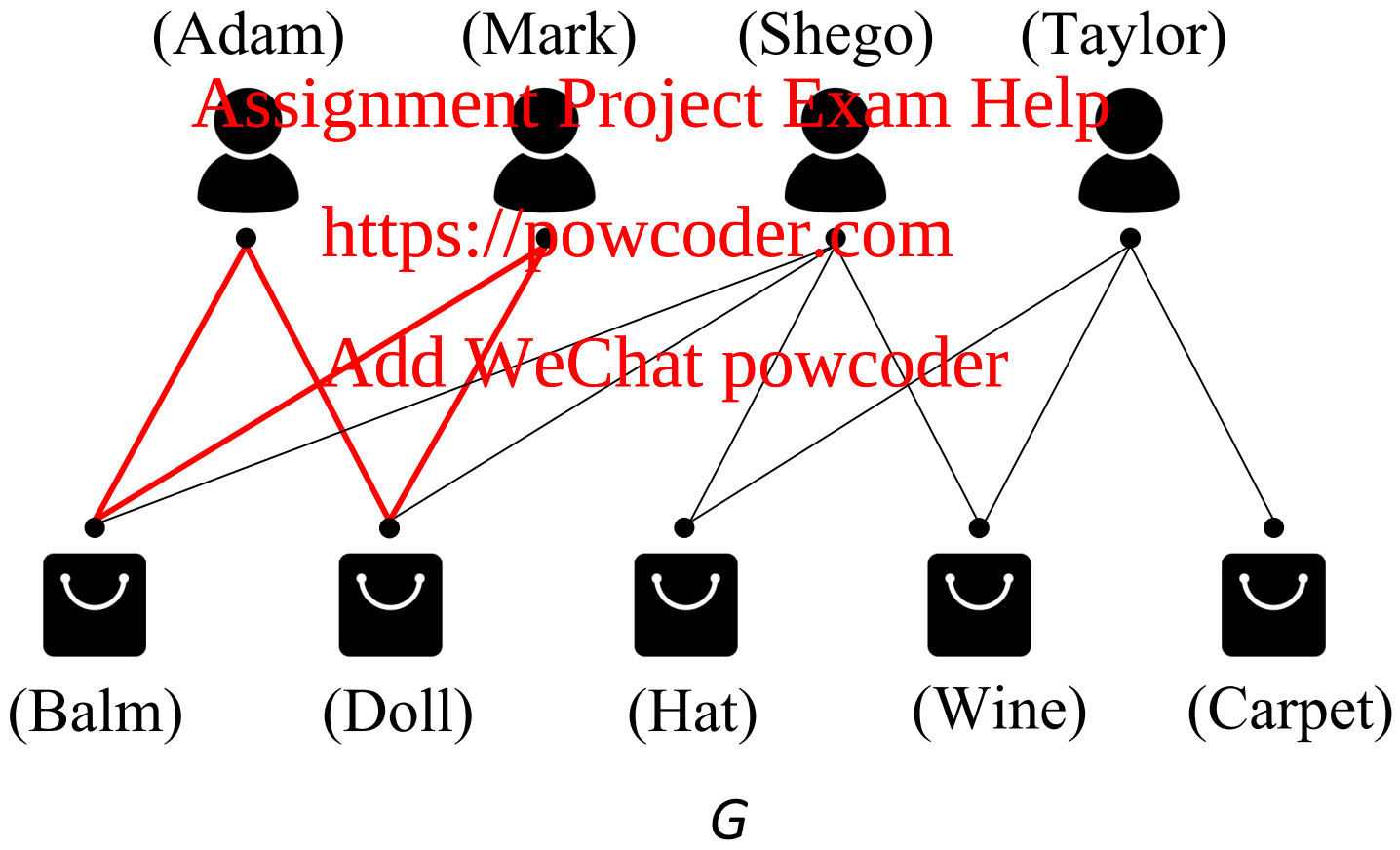
Butterfly counting: compute the number of butteries in a bipartite graph G, denoted by $\bowtie_G$



$\bowtie_G = 4$

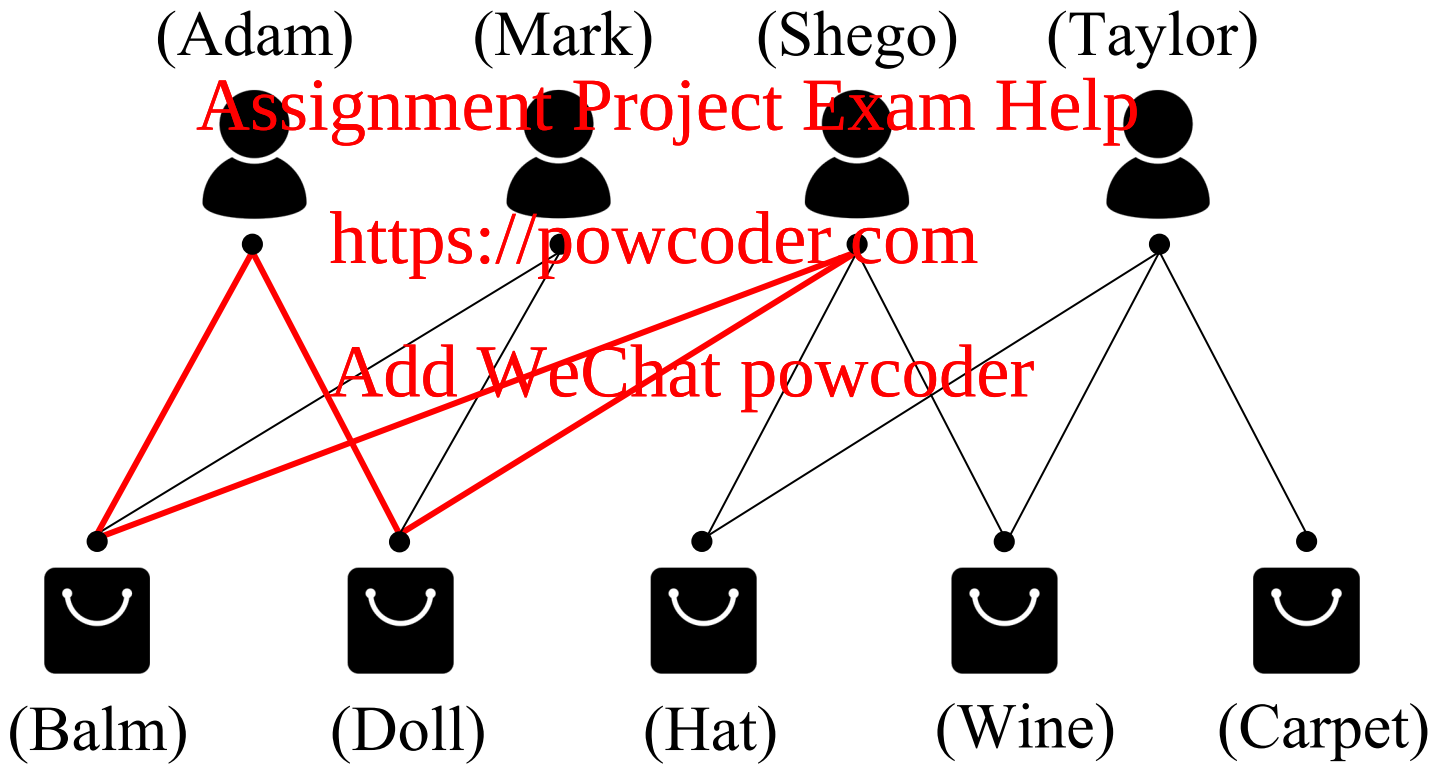
Introduction

Butterfly counting: compute the number of butteries in a bipartite graph G, denoted by $\bowtie_G$



Introduction

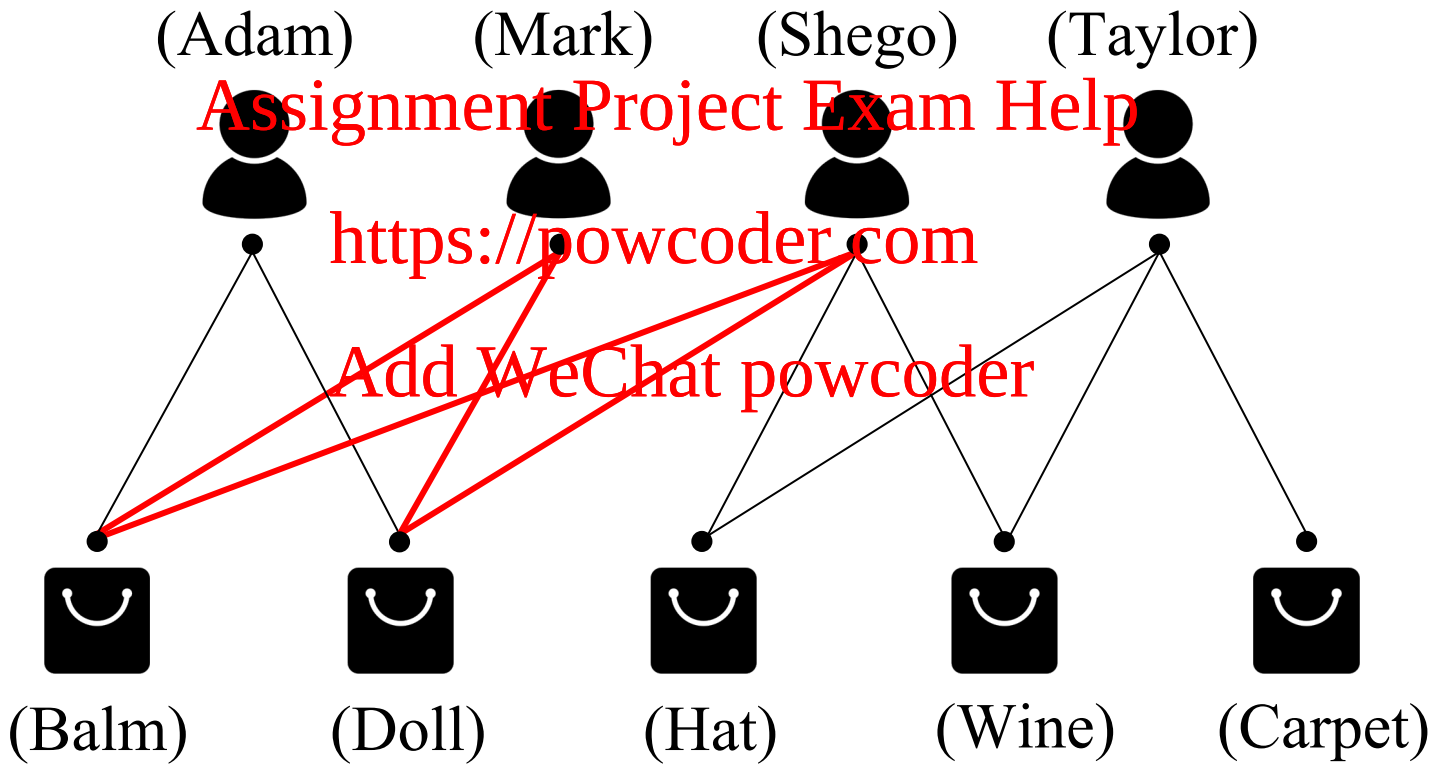
Butterfly counting: compute the number of butteries in a bipartite graph G, denoted by $\bowtie_G$



G

Introduction

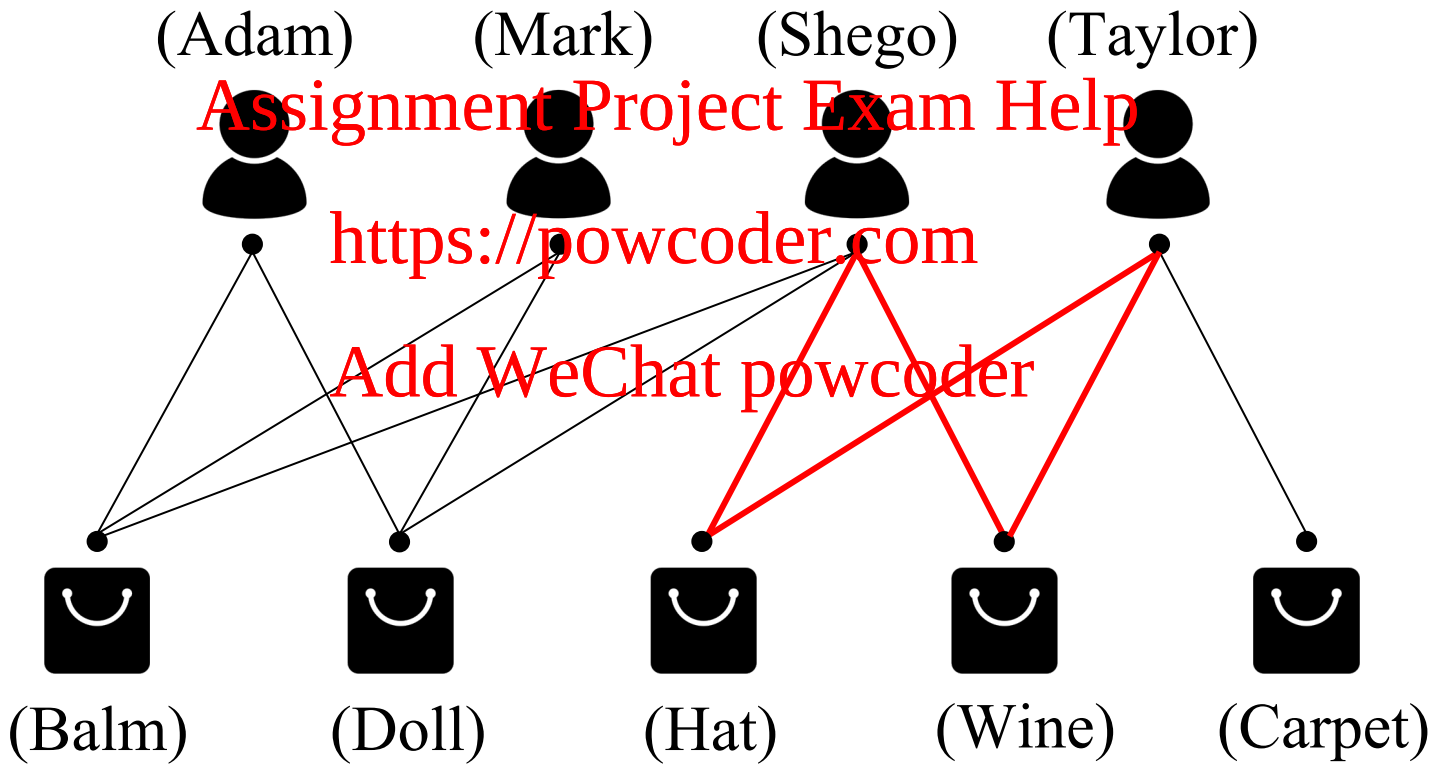
Butterfly counting: compute the number of butteries in a bipartite graph *G*, denoted by $\bowtie_G$



G

Introduction

Butterfly counting: compute the number of butteries in a bipartite graph G, denoted by $\bowtie_G$



G

Applications

- Bipartite clustering coefficient (c) – $c = \frac{4 \boxtimes_G}{\Sigma}$ - the number of three-paths (computed in

Network measurement: high bipartite clustering coefficient indicates localized closeness; a building block for creating community structure [Aksoy et. al, 2017].

- *K*-wing – each edge in *k*-wing has at least *k* number of butterflies [Sarıyüce et. al, 2018].

Community detection; word-document clustering; viral marketing ...

Related Works

- Unipartite networks
  - *triangle counting:* [1], [2], [3], [4], …
  - other motifs: 4-vertices [5], 5-vertices [6], cycles of length k [7], …
- Bipartite networks
  - *butterfly counting:* BFC-BS [8], BFC-IBS [9], …
  - other motifs: 3x3 biclique [10], 4-path [11], …

State-of-the-art

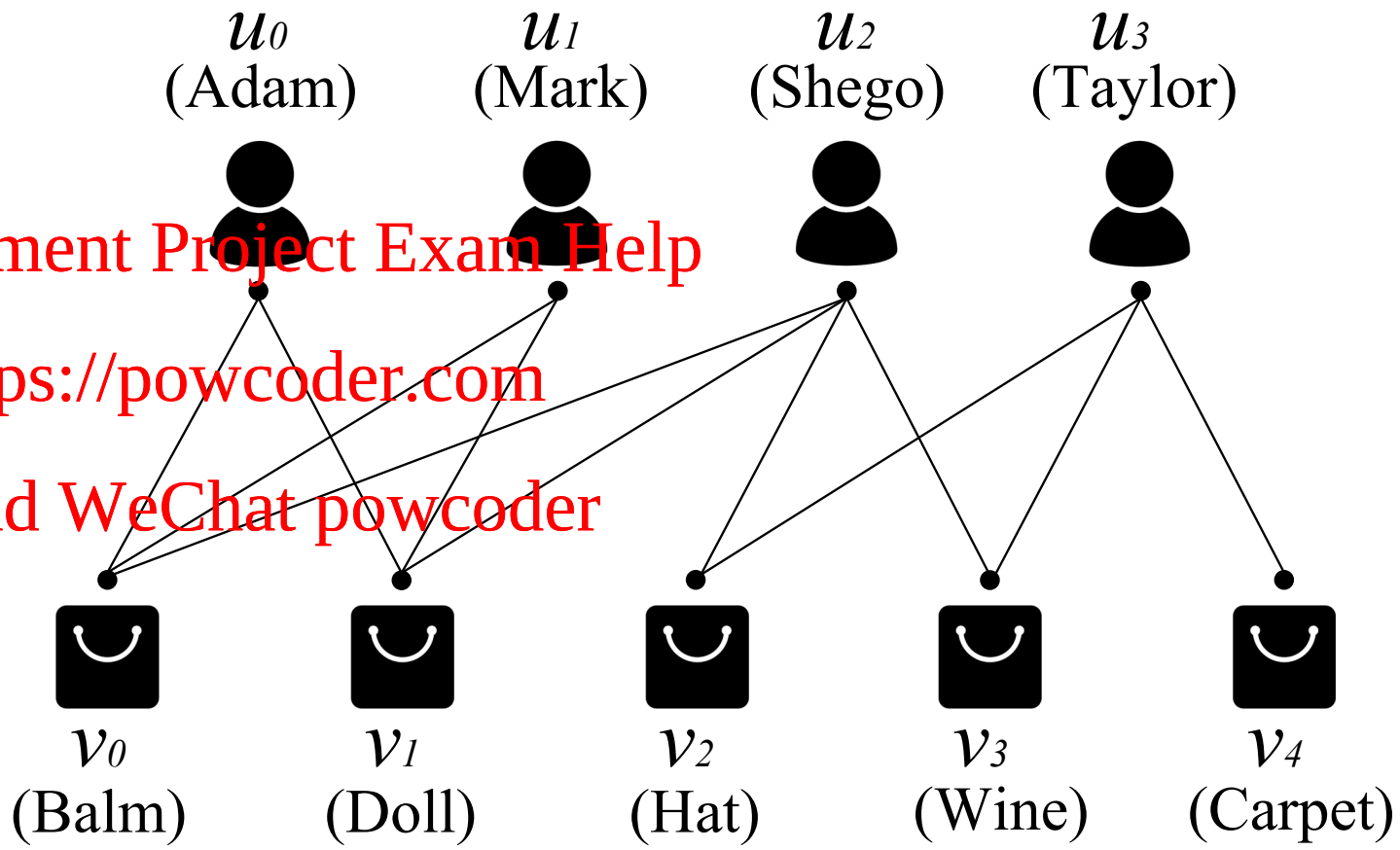
[1] M. Al Hasan and V. S. Dave. Triangle counting in large networks: a review. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2018.
[2] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In KDD,2008.
[3] L. Chang, C. Zhang, X. Lin, and L. Qin. Scalable top-k structural diversity search. In ICDE, 2017.
[4] X. Hu, Y. Tao, and C.-W. Chung. Massive graph triangulation. In SIGMOD, 2013.
[5] M. Jha, C. Seshadhri, and A. Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In WWW, 2015.
[6] A. Pinar, C. Seshadhri, and V. Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In WWW, 2017.
[7] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. Algorithmica,1997.
[8] J. Wang, A. W.-C. Fu, and J. Cheng. Rectangle counting in large bipartite graphs. In BigData Congress, 2014.
[9] S.-V. Sanei-Mehri, A. E. Sariyuce, and S. Tirthapura. Butterfly counting in bipartite networks. In KDD, 2018.
[10] S. P. Borgatti and M. G. Everett. Network analysis of 2-mode data. Social networks, 1997.
[11] T. Opsahl. Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. Social Networks, 2013.

State-of-the-art

- BFC-BS & BFC-IBS

$$\bowtie_u = \sum_{w \in 2hop_G(u)} \binom{|N_G(u) \cap N_G(w)|}{2}$$

$$\bowtie_G = \frac{1}{2} \sum_{u \in U(G)} \bowtie_u = \frac{1}{2} \sum_{v \in L(G)} \bowtie_v$$



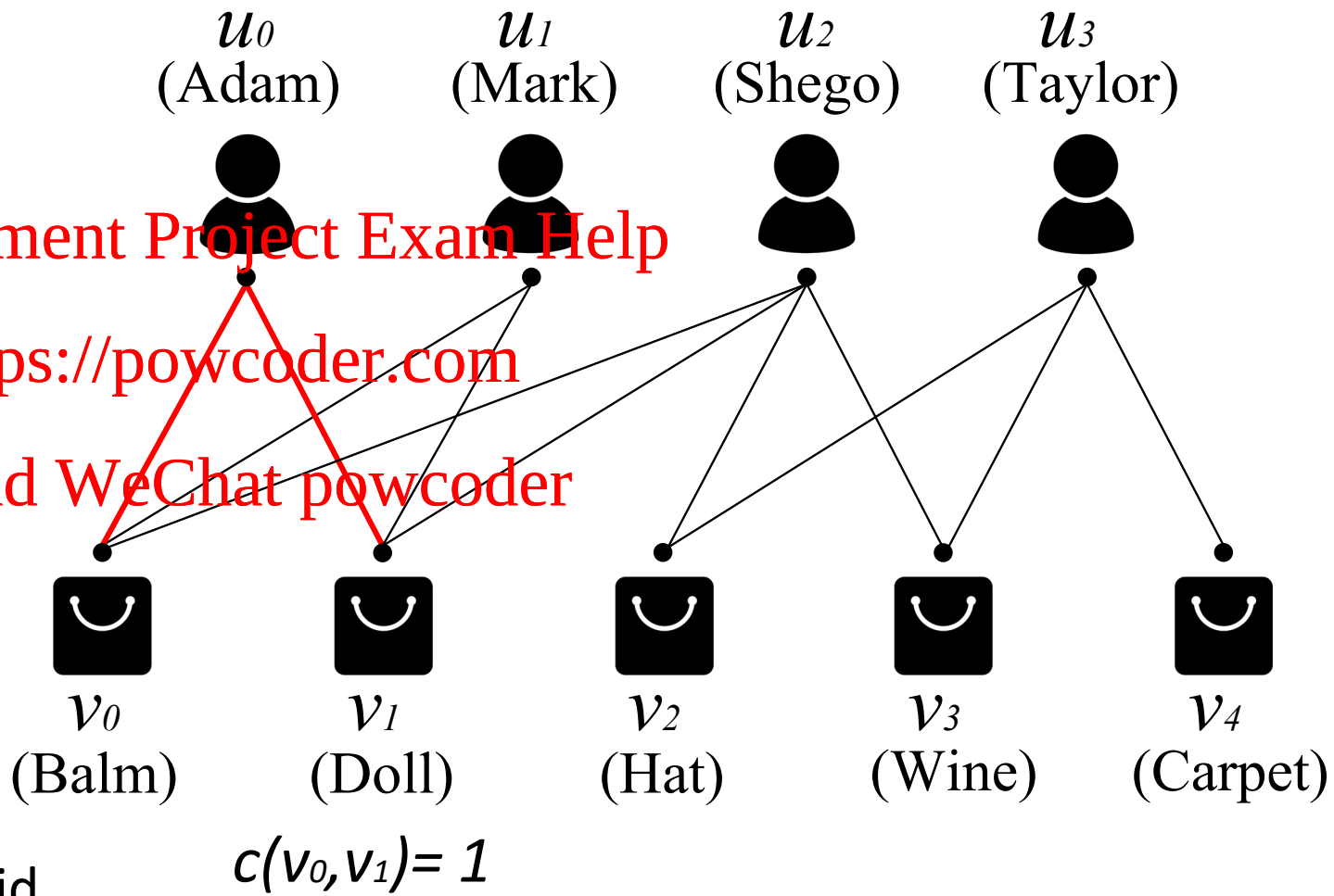
BFC-IBS improves BFC-BS in two aspects: (1) pre choosing the layer of start-vertices to achieve a lower time complexity; (2) using a hash map to speed up the implementation.
(3) avoid counting a butterfly twice (process the wedge (u, v, w) only if w.id > u.id)

State-of-the-art

- BFC-BS & BFC-IBS

$$\bowtie_u = \sum_{w \in 2hop_G(u)} \binom{|N_G(u) \cap N_G(w)|}{2}$$

$$\bowtie_G = \frac{1}{2} \sum_{u \in U(G)} \bowtie_u = \frac{1}{2} \sum_{v \in L(G)} \bowtie_v$$



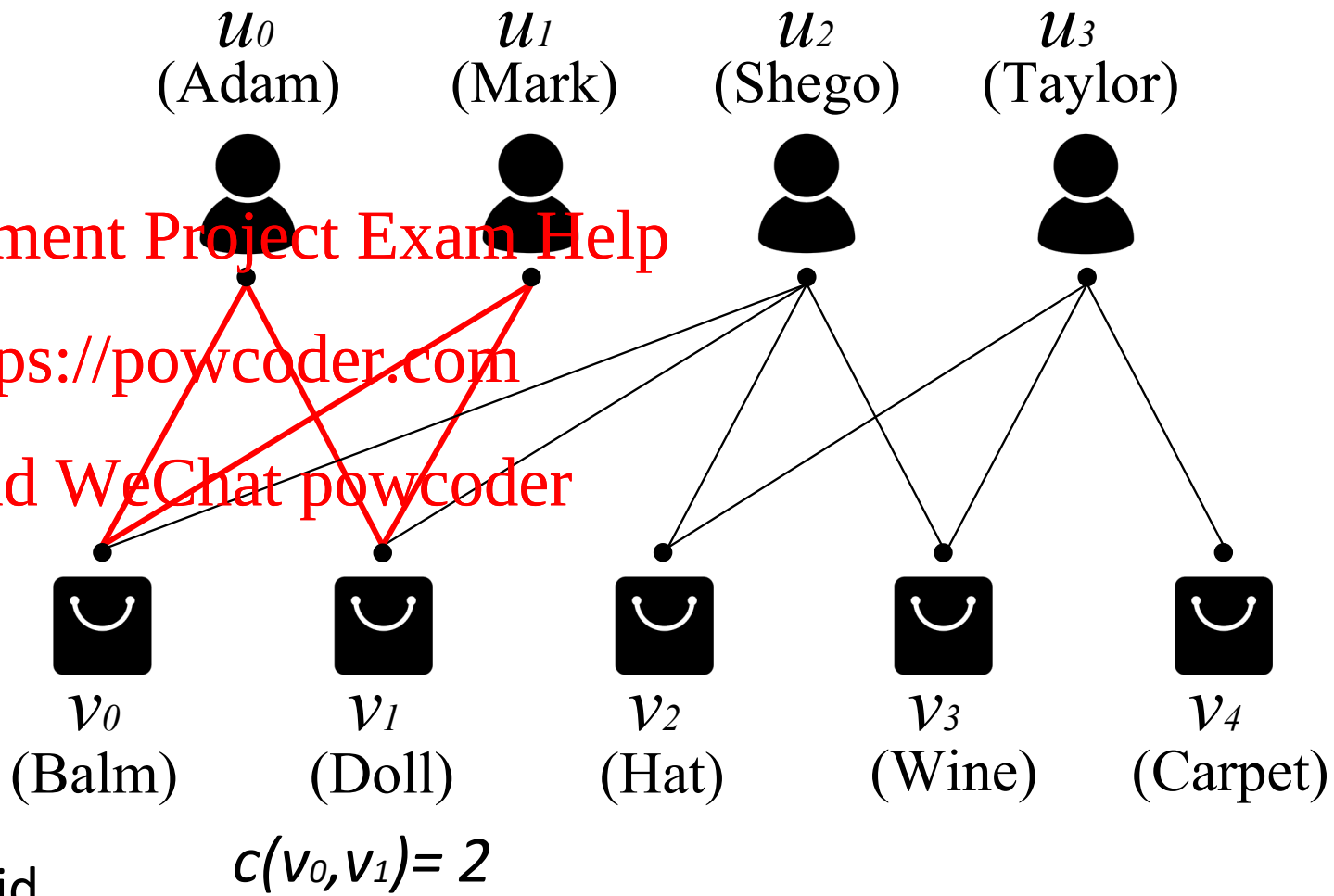
BFC-IBS improves BFC-BS in two aspects: (1) pre choosing the layer of start-vertices to achieve a lower time complexity; (2) using a hash map to speed up the implementation.
(3) avoid counting a butterfly twice (process the wedge (u, v, w) only if w.id > u.id)

State-of-the-art

- BFC-BS & BFC-IBS

$$\bowtie_u = \sum_{w \in 2hop_G(u)} \binom{|N_G(u) \cap N_G(w)|}{2}$$

$$\bowtie_G = \frac{1}{2} \sum_{u \in U(G)} \bowtie_u = \frac{1}{2} \sum_{v \in L(G)} \bowtie_v$$



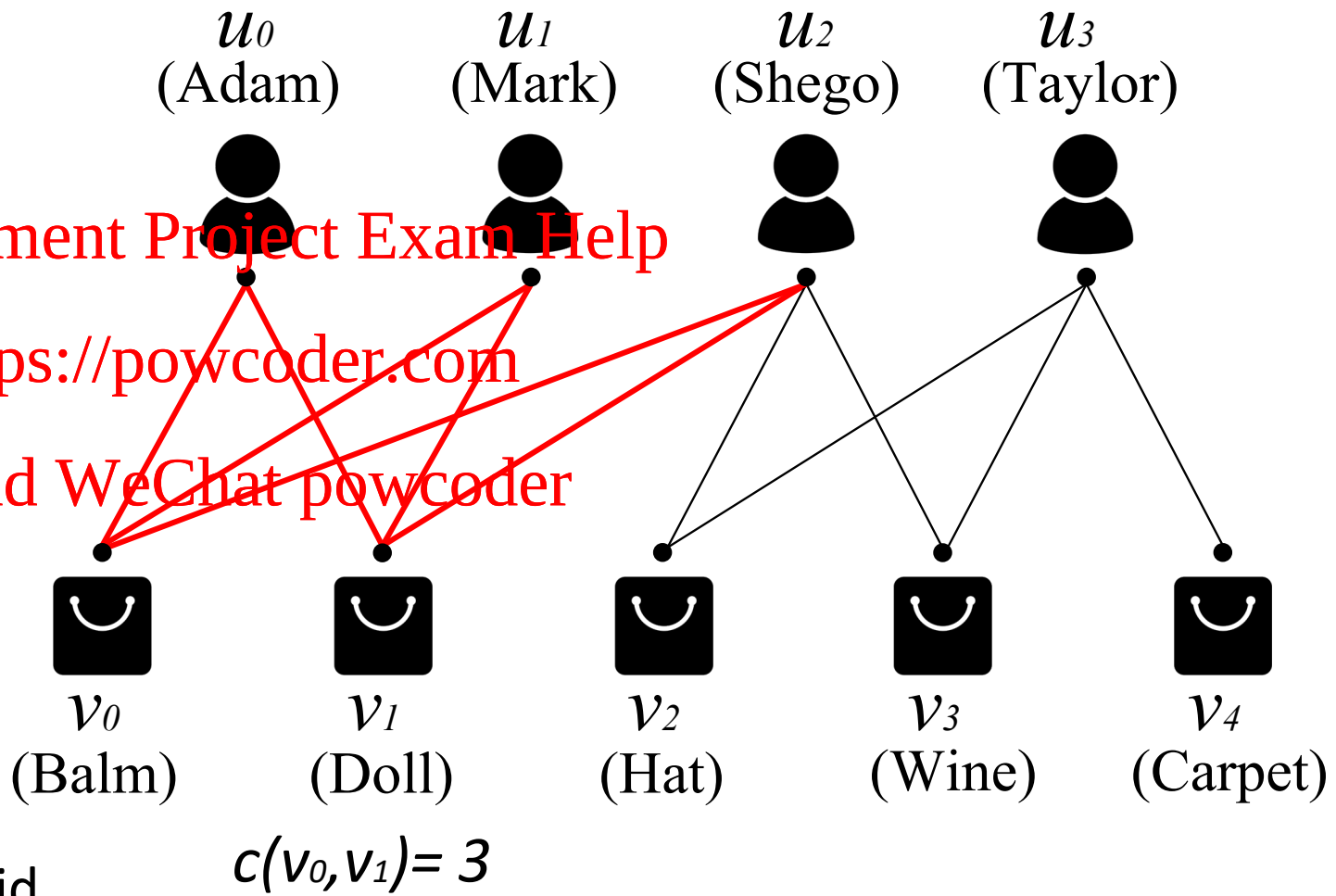
BFC-IBS improves BFC-BS in two aspects: (1) pre choosing the layer of start-vertices to achieve a lower time complexity; (2) using a hash map to speed up the implementation.
(3) avoid counting a butterfly twice (process the wedge (u, v, w) only if w.id > u.id)

State-of-the-art

- BFC-BS & BFC-IBS

$$\bowtie_u = \sum_{w \in 2hop_G(u)} \binom{|N_G(u) \cap N_G(w)|}{2}$$

$$\bowtie_G = \frac{1}{2} \sum_{u \in U(G)} \bowtie_u = \frac{1}{2} \sum_{v \in L(G)} \bowtie_v$$



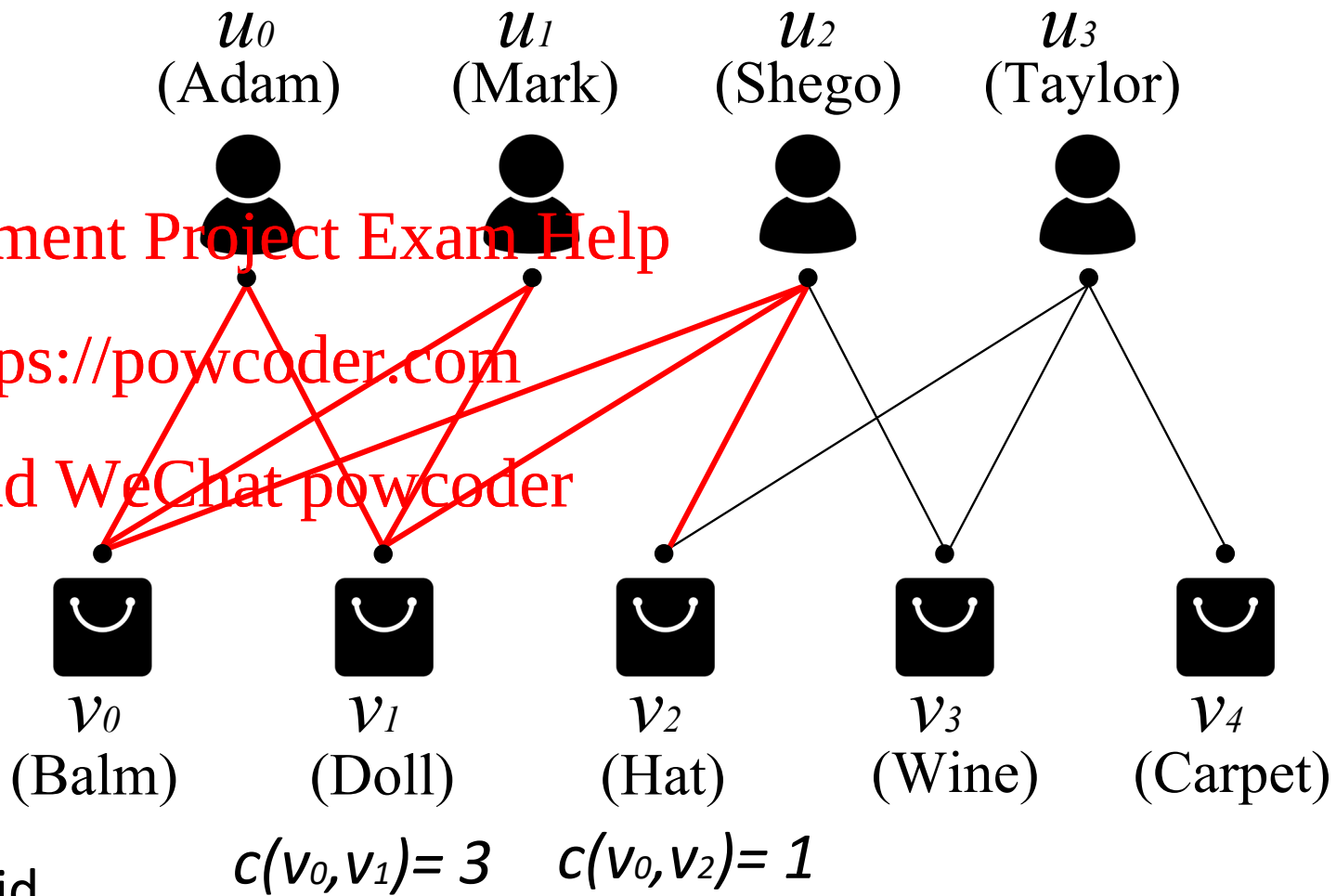
BFC-IBS improves BFC-BS in two aspects: (1) pre choosing the layer of start-vertices to achieve a lower time complexity; (2) using a hash map to speed up the implementation.
(3) avoid counting a butterfly twice (process the wedge (u, v, w) only if w.id > u.id)

State-of-the-art

- BFC-BS & BFC-IBS

$$\bowtie_u = \sum_{w \in 2hop_G(u)} \binom{|N_G(u) \cap N_G(w)|}{2}$$

$$\bowtie_G = \frac{1}{2} \sum_{u \in U(G)} \bowtie_u = \frac{1}{2} \sum_{v \in L(G)} \bowtie_v$$



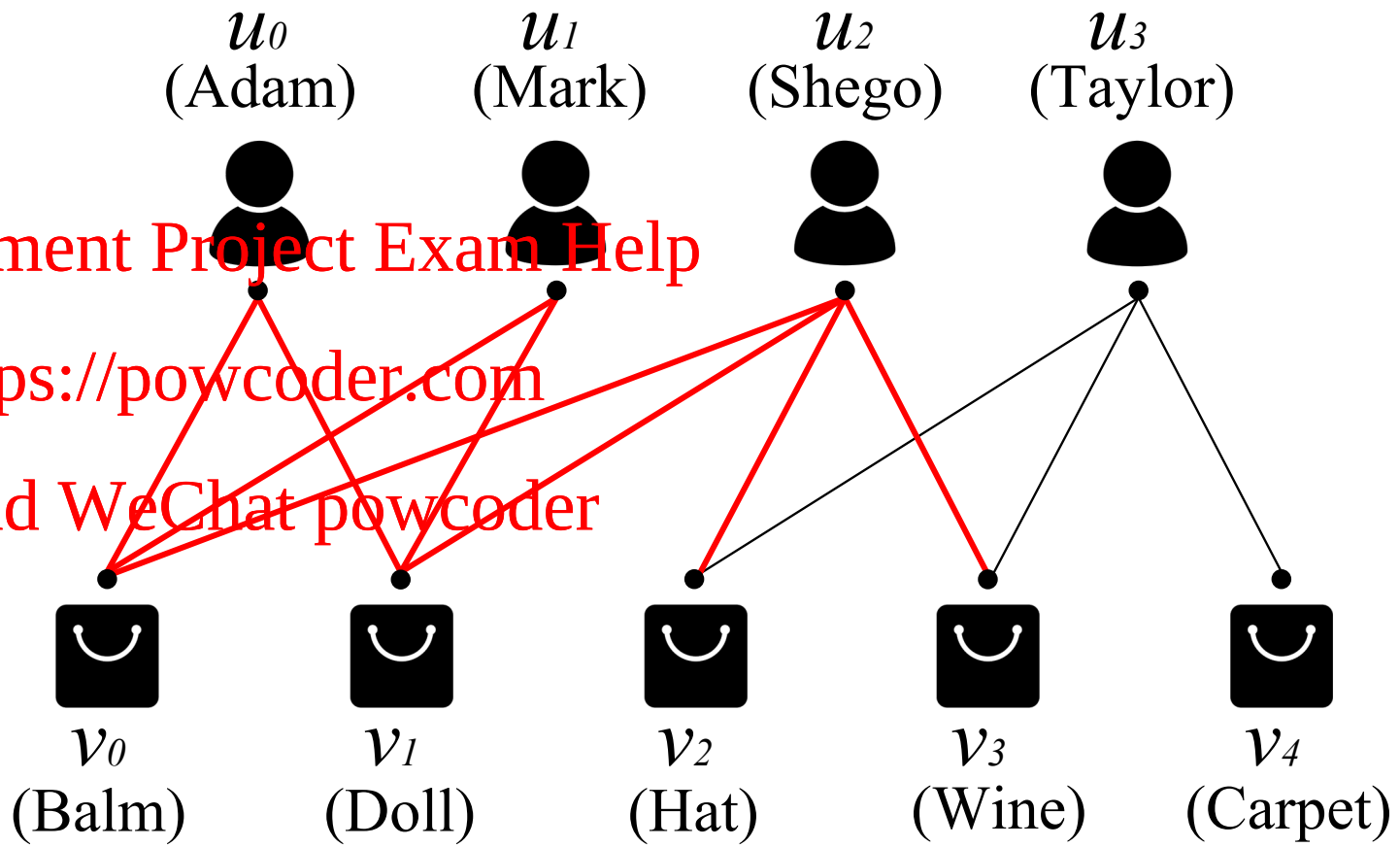
BFC-IBS improves BFC-BS in two aspects: (1) pre choosing the layer of start-vertices to achieve a lower time complexity; (2) using a hash map to speed up the implementation.
(3) avoid counting a butterfly twice (process the wedge (u, v, w) only if w.id > u.id)

State-of-the-art

- BFC-BS & BFC-IBS

$$\bowtie_u = \sum_{w \in 2hop_G(u)} \binom{|N_G(u) \cap N_G(w)|}{2}$$

$$\bowtie_G = \frac{1}{2} \sum_{u \in U(G)} \bowtie_u = \frac{1}{2} \sum_{v \in L(G)} \bowtie_v$$



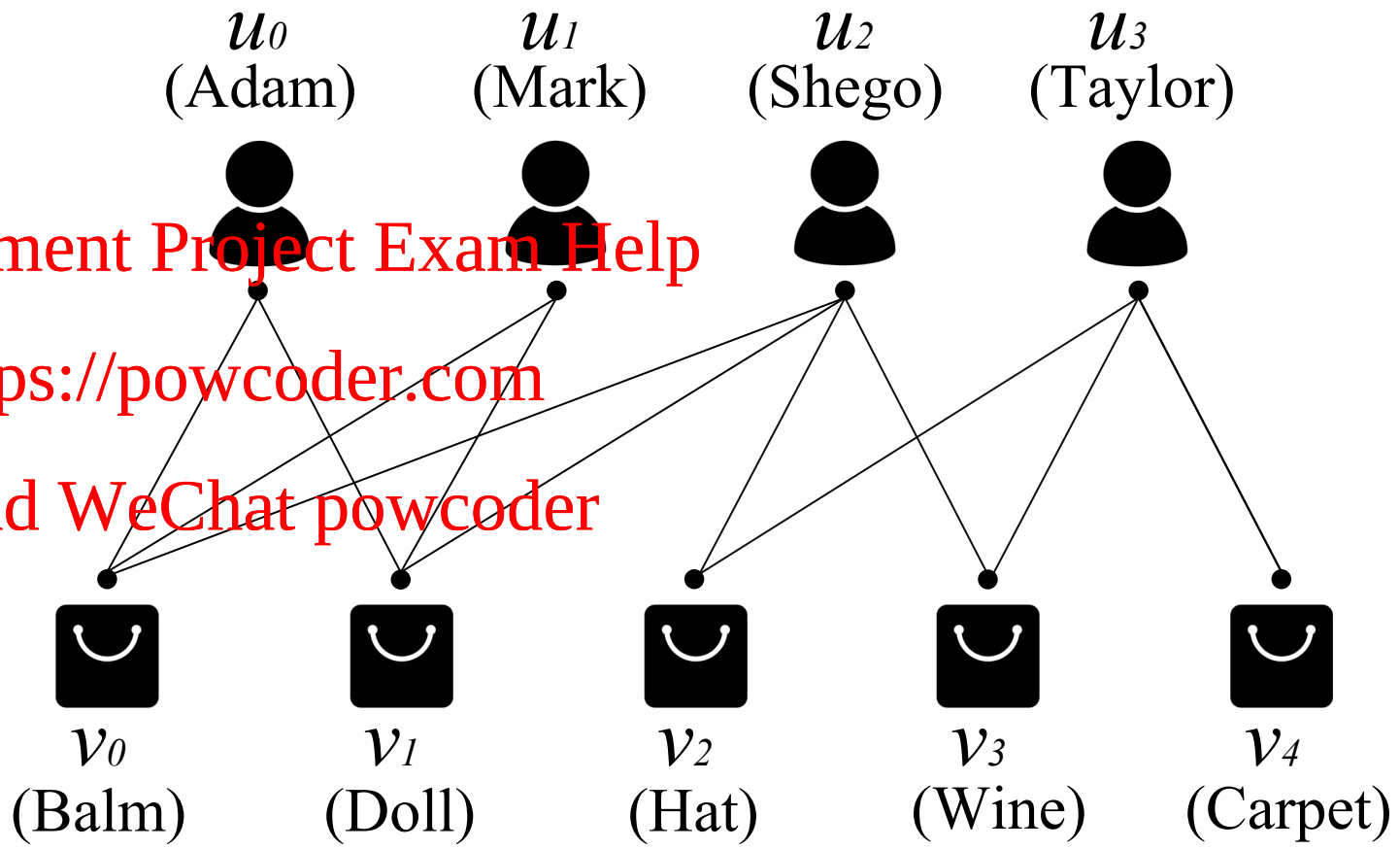
BFC-IBS improves BFC-BS in two aspects: (1) pre choosing the layer of start-vertices to achieve a lower time complexity; (2) using a hash map to speed up the implementation.
(3) avoid counting a butterfly twice (process the wedge (u, v, w) only if w.id > u.id)

State-of-the-art

- BFC-BS & BFC-IBS

$$\bowtie_u = \sum_{w \in 2hop_G(u)} \binom{|N_G(u) \cap N_G(w)|}{2}$$

$$\bowtie_G = \frac{1}{2} \sum_{u \in U(G)} \bowtie_u = \frac{1}{2} \sum_{v \in L(G)} \bowtie_v$$



BFC-IBS improves BFC-BS in two aspects: (1) pre choosing the layer of start-vertices to achieve a lower time complexity; (2) using a hash map to speed up the implementation.
(3) avoid counting a butterfly twice (process the wedge (u, v, w) only if w.id > u.id)

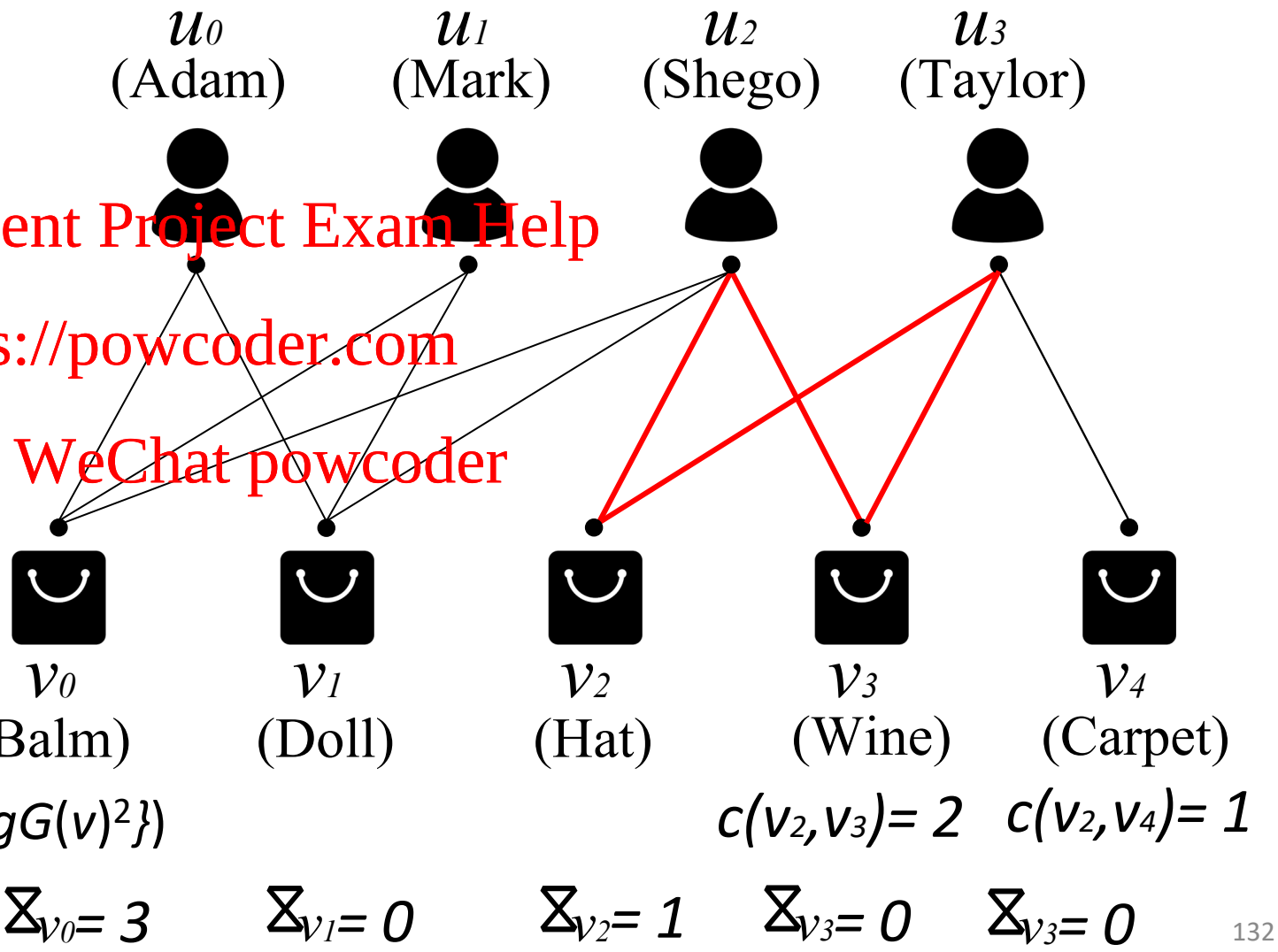
$\bowtie_{v_0} = 3$ $\bowtie_{v_1} = 0$

State-of-the-art

- BFC-BS & BFC-IBS

$$\bowtie_u = \sum_{w \in 2hop_G(u)} \binom{|N_G(u) \cap N_G(w)|}{2}$$

$$\bowtie_G = \frac{1}{2} \sum_{u \in U(G)} \bowtie_u = \frac{1}{2} \sum_{v \in L(G)} \bowtie_v$$



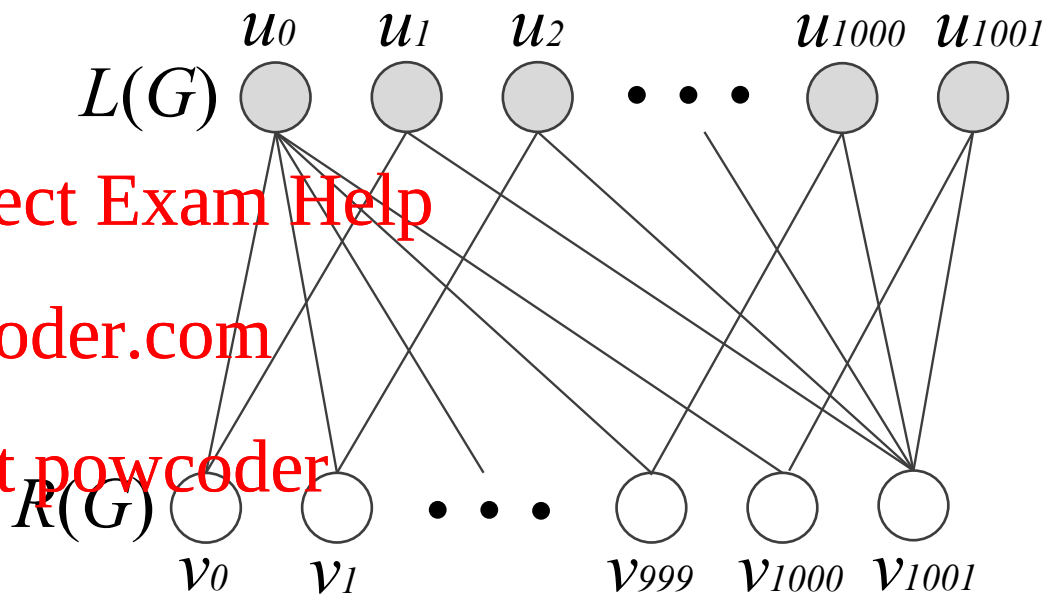
BFC-IBS improves BFC-BS in two aspects: (1) pre choosing the layer of start-vertices to achieve a lower time complexity; (2) using a hash map to speed up the implementation. (3) avoid counting a butterfly twice (process the wedge (u, v, w) only if w.id > u.id)

Time Complexity

$O(min\{\sum_{u \in U(G)} degG(u)^2, \sum_{v \in L(G)} degG(v)^2\})$

State-of-the-art

Issues in handling hub vertices:

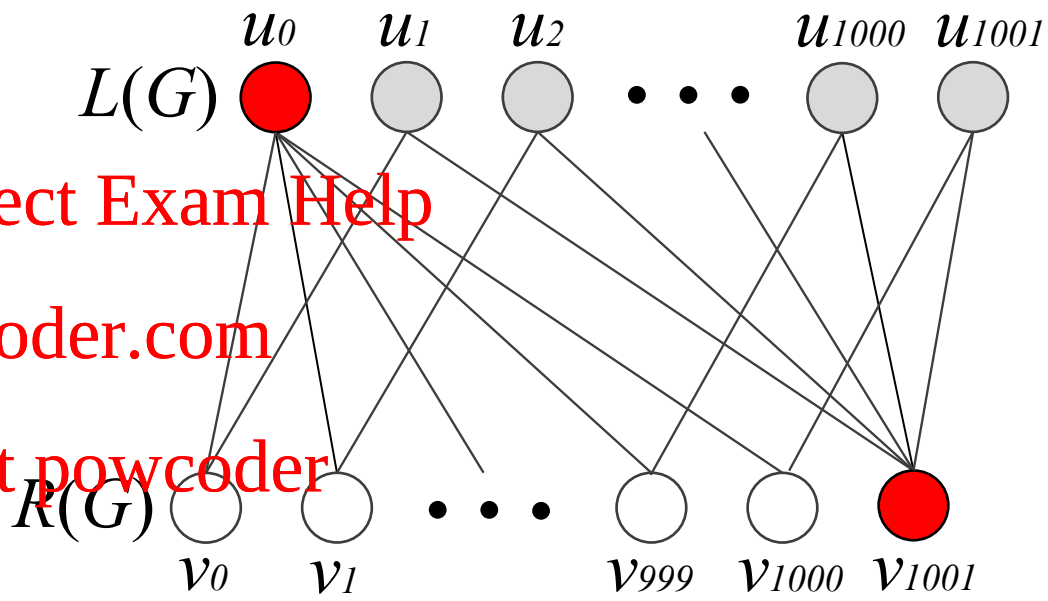


State-of-the-art

Issues in handling hub vertices:

The existing algorithms need to go through u_0 (or v_{1000}) as the middle vertex if choosing $L(G)$ (or $U(G)$) to start.

Regardless of whether the upper or the lower layer is selected to start, we must traverse totally $C_{2,1000}$ (= 499,500) plus 1,000 different wedges by the existing algorithms.

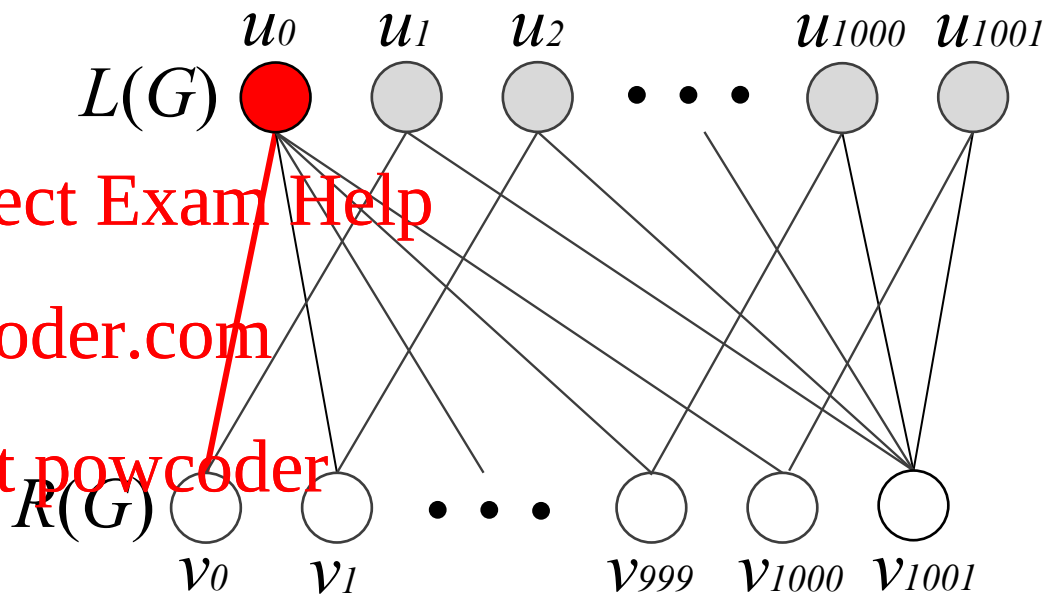


State-of-the-art

Issues in handling hub vertices:

The existing algorithms need to go through u_0 (or v_{1000}) as the middle vertex if choosing $L(G)$ (or $U(G)$) to start.

Regardless of whether the upper or the lower layer is selected to start, we must traverse totally $C_{2,1000}$ (= 499,500) plus 1,000 different wedges by the existing algorithms.

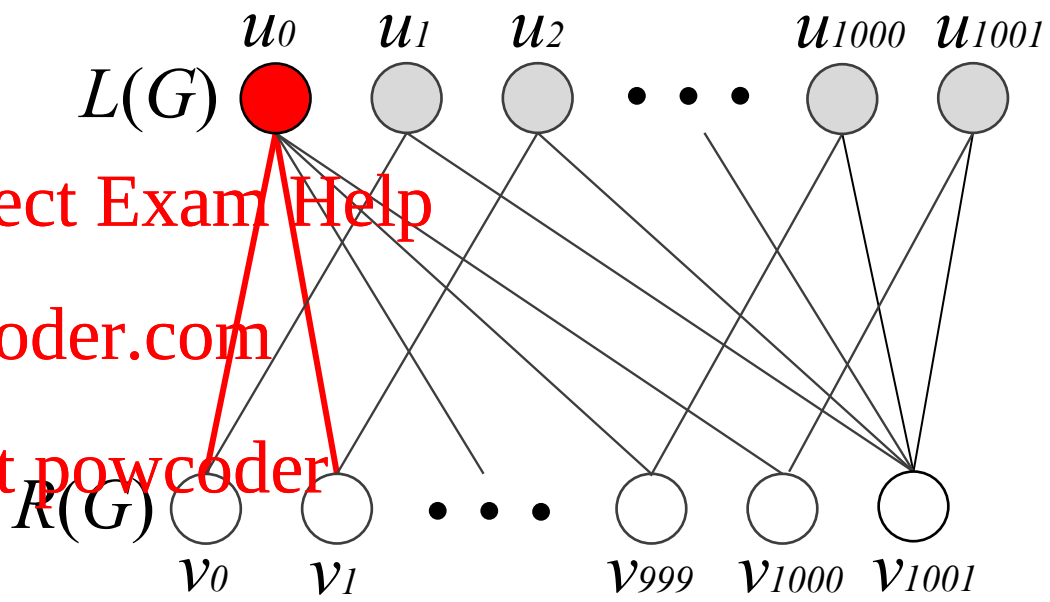


State-of-the-art

Issues in handling hub vertices:

The existing algorithms need to go through u_0 (or v_{1000}) as the middle vertex if choosing $L(G)$ (or $U(G)$) to start.

Regardless of whether the upper or the lower layer is selected to start, we must traverse totally $C_{2,1000}$ (= 499,500) plus 1,000 different wedges by the existing algorithms.

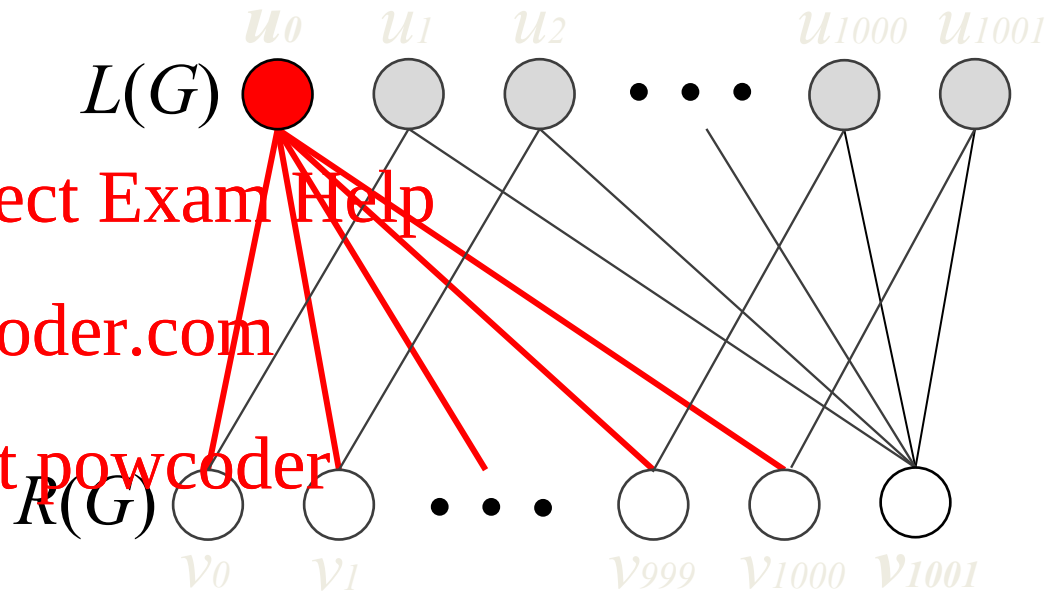


State-of-the-art

Issues in handling hub vertices:

The existing algorithms need to go through u_0 (or v_{1000}) as the middle vertex if choosing $L(G)$ (or $U(G)$) to start.

Regardless of whether the upper or the lower layer is selected to start, we must traverse totally $C_{2,1000}$ (= 499,500) plus 1,000 different wedges by the existing algorithms.



Challenges

- It is a challenge to effectively handle high-degree vertices.
- It is also a challenge to utilize CPU cache to speed up butterfly counting.

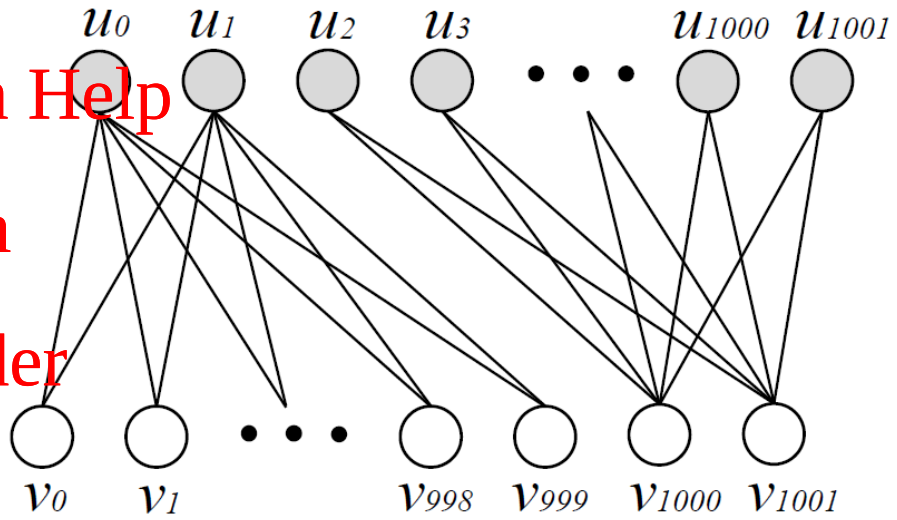
Solutions

- Solution 1: The vertex-priority-based algorithm
  - **BFC-VP**, to conquer challenge 1
- Solution 2: BFC-VP + cache aware techniques

  BFC-VP++, to conquer challenge 1 & 2

The vertex-priority-based algorithm

Motivation: a hub vertex may not always necessary to become a middle-vertex in a wedge for the construction of a butterfly.



$[u_0, v_0, u_1, v_1]$ in the Figure can be constructed in two ways: 1) by the wedges (u_0, v_0, u_1) and (u_0, v_1, u_1), or 2) by the wedges (v_0, u_0, v_1) and (v_0, u_1, v_1).

The vertex-priority-based algorithm

- Assign a priority to each vertex.

DEFINITION 3 (PRIORITY). *Given a bipartite graph $G(V, E)$, for a vertex $\in V(G)$, the priority $p(u)$ is an integer where $p(u) \in [1, |V(G)|]$. For two vertices $u, v \in V(G)$, $p(u) > p(v)$ if*

- $deg_G(u) > deg_G(v)$, *or*
- $deg_G(u) = deg_G(v)$, $u.id > v.id$.

Input: $G(V = (U, L), E)$: the input bipartite graph
Output: $\bowtie_G$

```
1  Compute p(u) for each u ∈ V(G) // Definition 3
2  Sort N(u) for each u ∈ V(G) according to their priorities
3  ⋈_G ← 0
4  foreach u ∈ V(G) do
5      initialize hashmap count_wedge with zero
6      foreach v ∈ N_G(u) : p(v) < p(u) do
7          foreach w ∈ N_G(v) : p(w) < p(u) do
8              count_wedge(w) ← count_wedge(w) + 1
9      foreach w ∈ count_wedge do
10         if count_wedge(w) > 1 then
11             ⋈_G ← ⋈_G + C(count_wedge(w), 2)
12 return ⋈_G
```


- Traversal necessary wedges.
  - Only process the wedges (u, v, w) with

    p(u) > p(v) & p(u) > p(w).

- Count butterflies.

Time Complexity

$O(\sum_{(u,v) \in E(G)} min\{degG(u), degG(v)\})$

Theorem: $\sum_{(u,v)\in E(G)} min\{degG(u), degG(v)\} \leq$

$min\{\sum_{u\in U(G)} degG(u)^2, \sum_{v\in L(G)} degG(v)^2\}$

Proof: $\sum_{u\in U(G)} degG(u)^2 = \sum_{(u,v)\in E(G),u\in U(G)} degG(u)$

$\geq \sum_{(u,v)\in E(G)} min\{degG(u), degG(v)\}$

$\sum_{v\in L(G)} degG(v)^2 = \sum_{(u,v)\in E(G),v\in L(G)} degG(v)$

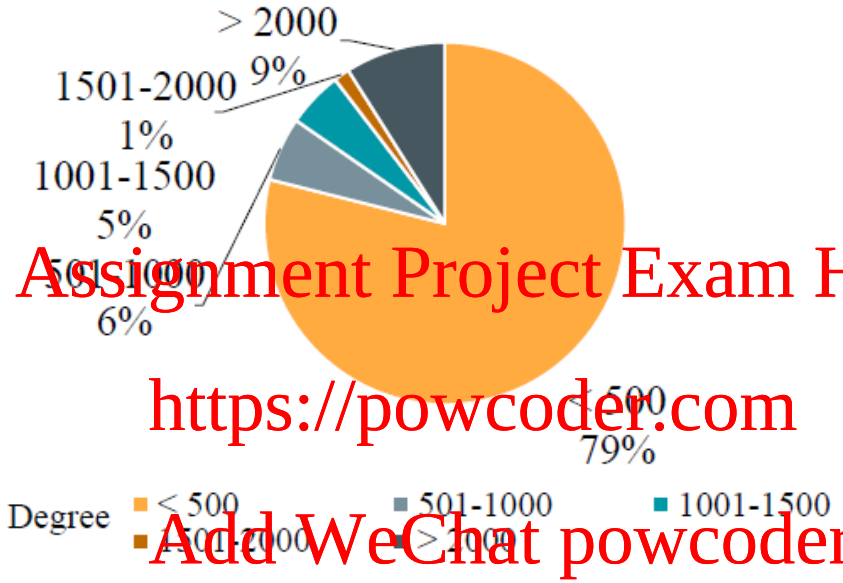
$\geq \sum_{(u,v)\in E(G)} min\{degG(u), degG(v)\}$

Cache aware techniques

- Cache aware wedge processing
- Cache aware graph projection

Improve CPU cache performance, keep time complexity and space complexity unchanged.

Cache aware wedge processing



The degree distribution of the end-vertex-accesses on ***Tracker*** by BFC-VP

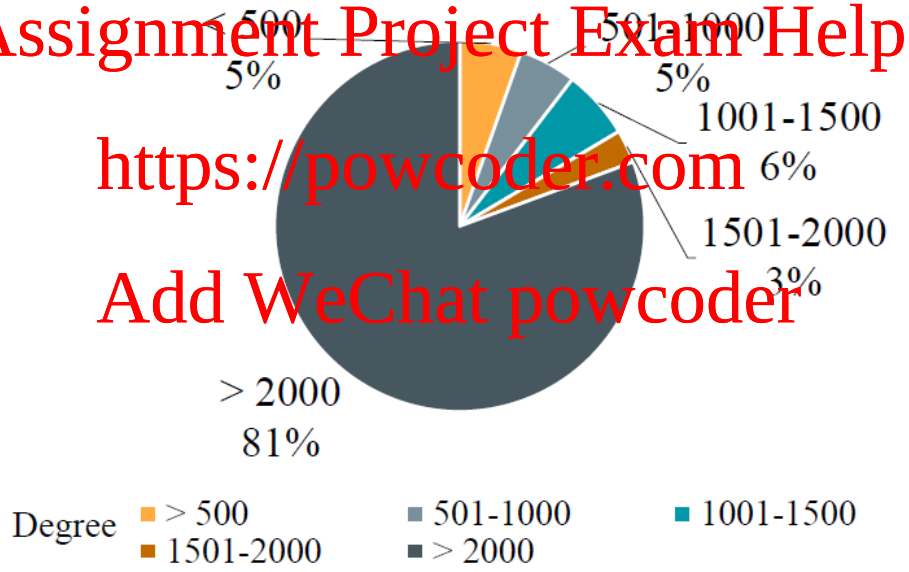
79% of total accesses are accesses of low-degree vertices (i.e., degree < 500)

Since the locality of accesses is a key aspect of improving the CPU cache performance, we hope the algorithm can access more high-degree vertices as end-vertices.

Cache aware wedge processing

New wedge processing strategy: processing the wedges where the priorities of end-vertices are higher than the priorities of middle-vertices and start-vertices.

We proved that the total access of end-vertices remaining unchanged - the time complexity unchanged.

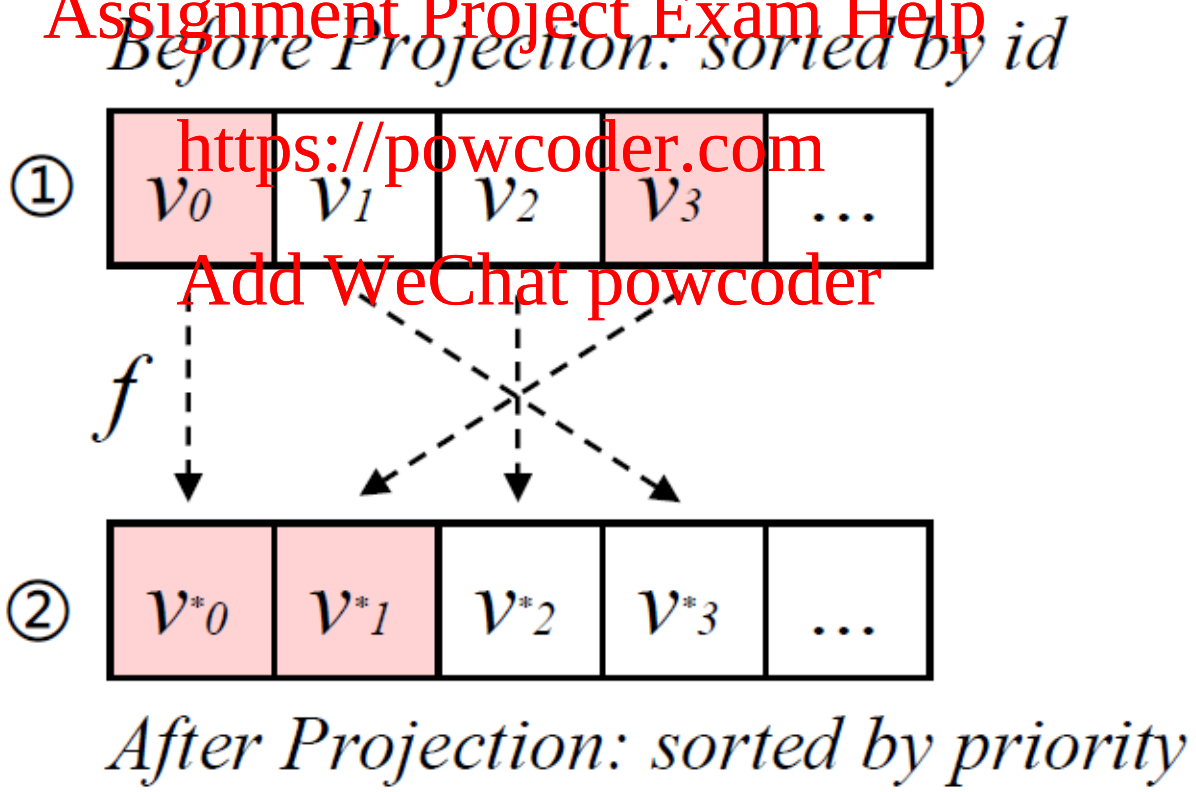


The degree distribution of the end-vertex-accesses on ***Tracker*** using new wedge processing strategy

The percentage of accesses of high-degree vertices (i.e., degree > 2000) increases from 9% to 81%.

Cache aware graph projection

Generally, vertices are sorted by their ids when storing in the buffer. Although end-vertices are mostly high-priority vertices, the distance between two end-vertices (e.g., v_0 and v_3) can be very long.



Extensions

- Counting the butterflies for each edge

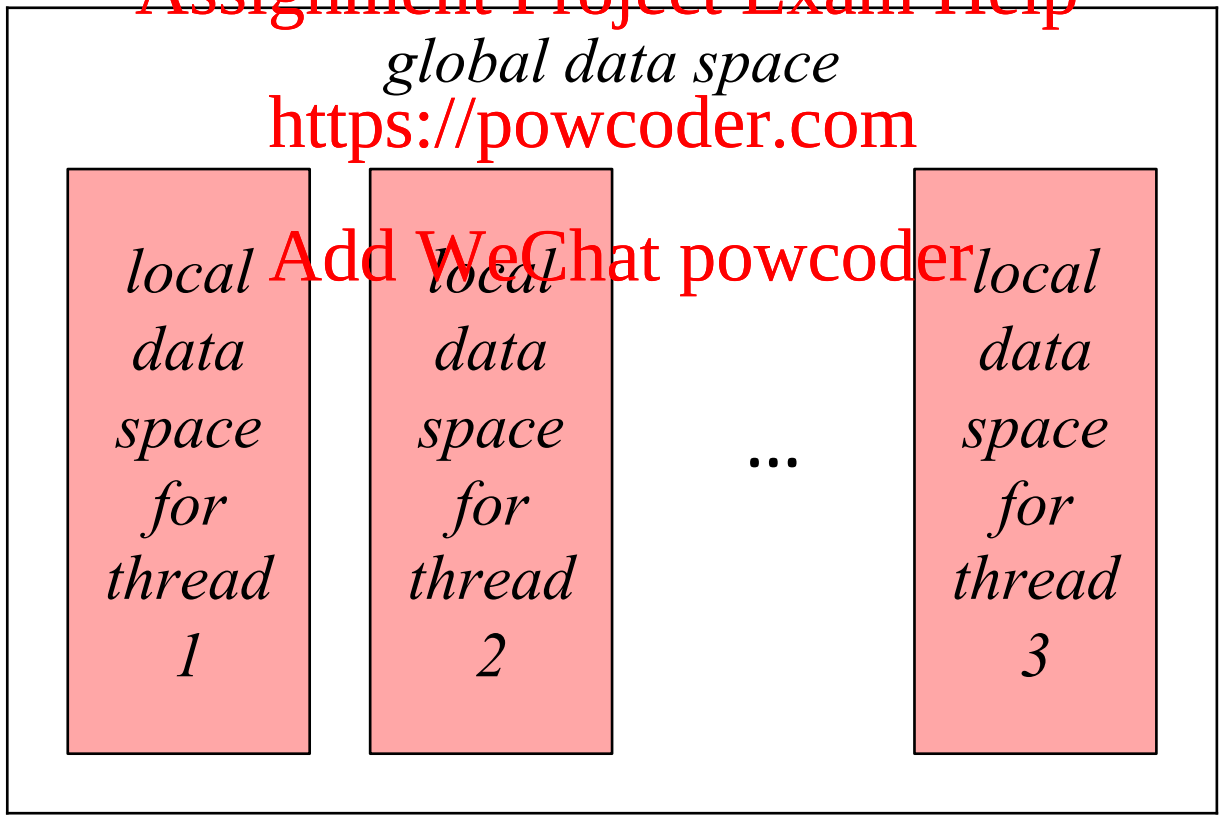
$$\bowtie_{e=(u,v)} = \sum_{w \in 2hop_G(u), w \in N_G(v)} (|N_G(u) \cap N_G(w)| - 1)$$

$$= \sum_{x \in 2hop_G(v), x \in N_G(u)} (|N_G(v) \cap N_G(x)| - 1)$$

Extensions

- Parallel butterfly counting – shared memory parallelization

  Easily extern our algorithms into a parallel version using O(n*t+m) space.



Extensions

- External memory butterfly counting

1. Process wedges based on our strategy.

2. For each wedge (u, v, w), store the vertex-pair (u, w) on disk.

3. Sequentially scan these vertex-pairs and for the vertex-pair (u, w), we count the occurrence of it and compute $\bowtie_G$ using the following lemma.

$$\bowtie_u = \sum_{w \in 2hop_G(u)} \binom{|N_G(u) \cap N_G(w)|}{2}$$

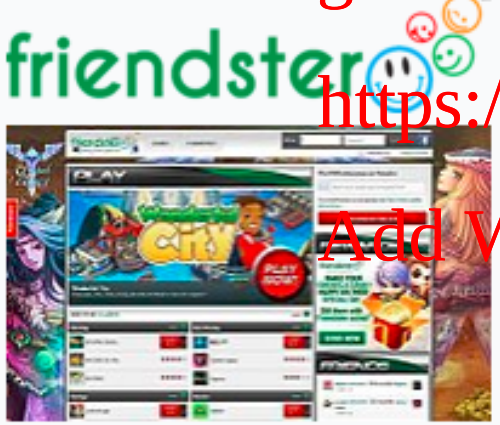
$$\bowtie_G = \frac{1}{2} \sum_{u \in U(G)} \bowtie_u = \frac{1}{2} \sum_{v \in L(G)} \bowtie_v$$

OLAK: An Efficient Algorithm to Prevent Unravelling in Social Networks

VLDB 2017

The collapse of *Friendster, Inc.*

- Founded in 2002.
- Popular at early 21st century with over 115 million users.
- Suspended in 2015 for lack of user engagement.



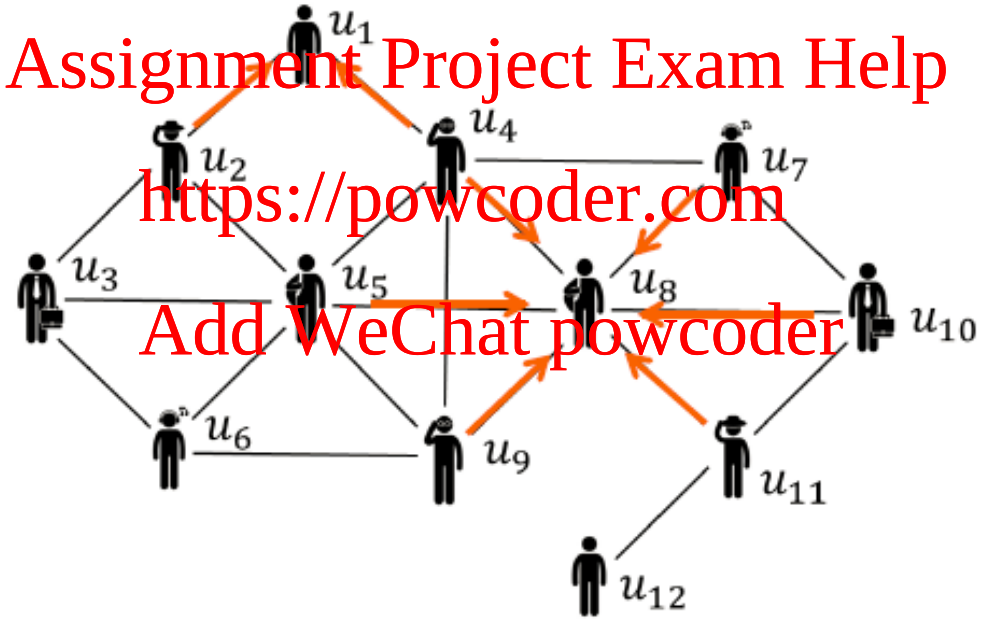
Screenshot of Friendster's main page, updated in July 2011

D. Garcia, P. Mavrodiev, and F. Schweitzer. Social resilience in online communities: the autopsy of friendster. In COSN, pages 39–50, 2013.

K. Seki and M. Nakamura. The collapse of the friendster network started from the center of the core. In ASONAM, pages 477–484, 2016.

Network Unraveling

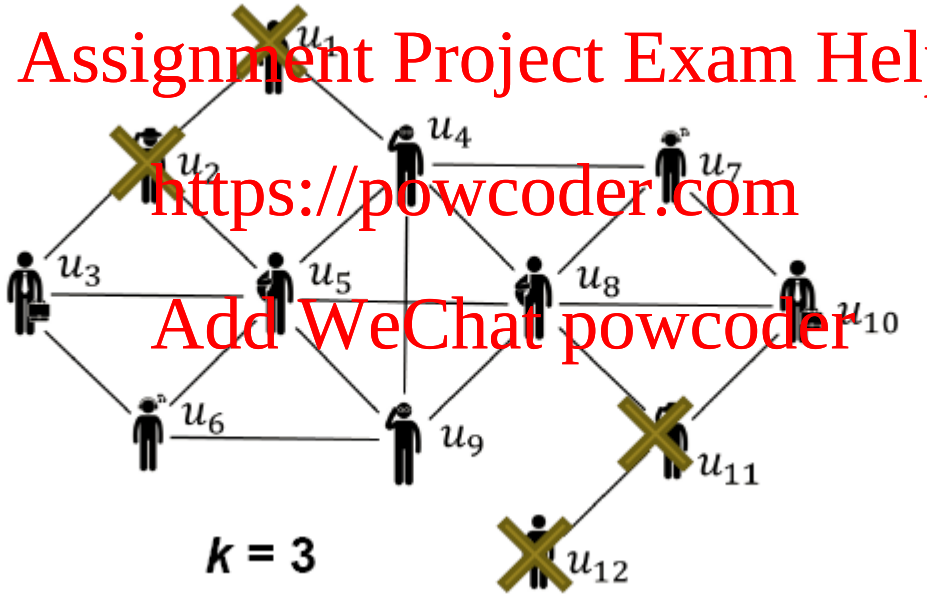
The engagement of a user is influenced by the number of her engaged friends.



K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics*, 29(3):1452–1475, 2015.

Network Unraveling

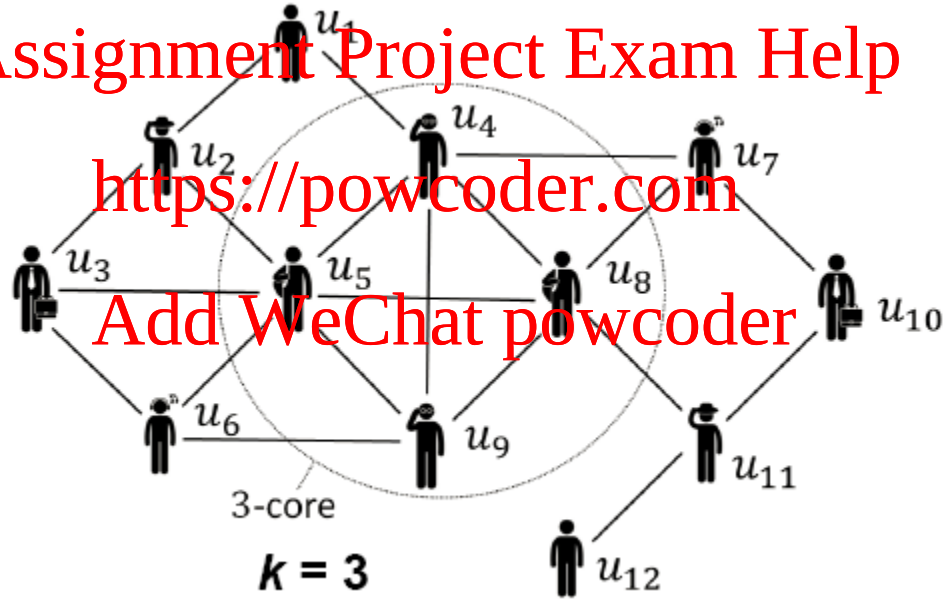
An equilibrium: a group has the minimum degree of *k*



K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics*, 29(3):1452–1475, 2015.

Network Unraveling

A Social group tends to be a k-core in the network



K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics*, 29(3):1452–1475, 2015.

Network Unraveling

- Founded in 2002.
- Popular at early 21st century with over 115 million users.
- Suspended in 2015 for lack of user engagement.

D. Garcia, P. Mavrodiev, and F. Schweitzer. Social resilience in online communities: the autopsy of friendster. In COSN, pages 39–50, 2013.

The core number steadily increased.

Network Unraveling

- Founded in 2002.
- Popular at early 21st century with over 115 million users.
- Suspended in 2015 for lack of user engagement.

K. Seki and M. Nakamura. The collapse of the friendster network started from the center of the core. In ASONAM, pages 477–484, 2016.

The collapse started from the center of the core.

Network Unraveling

- Founded in 2002.
- Popular at early 21st century with over 115 million users.
- Suspended in 2015 for lack of user engagement.

J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg. Structural diversity in social contagion. PNAS, 109(16):5962–5966, 2012.

Social influence is tightly controlled by the number of friends in current subgraph, like *k*-core.

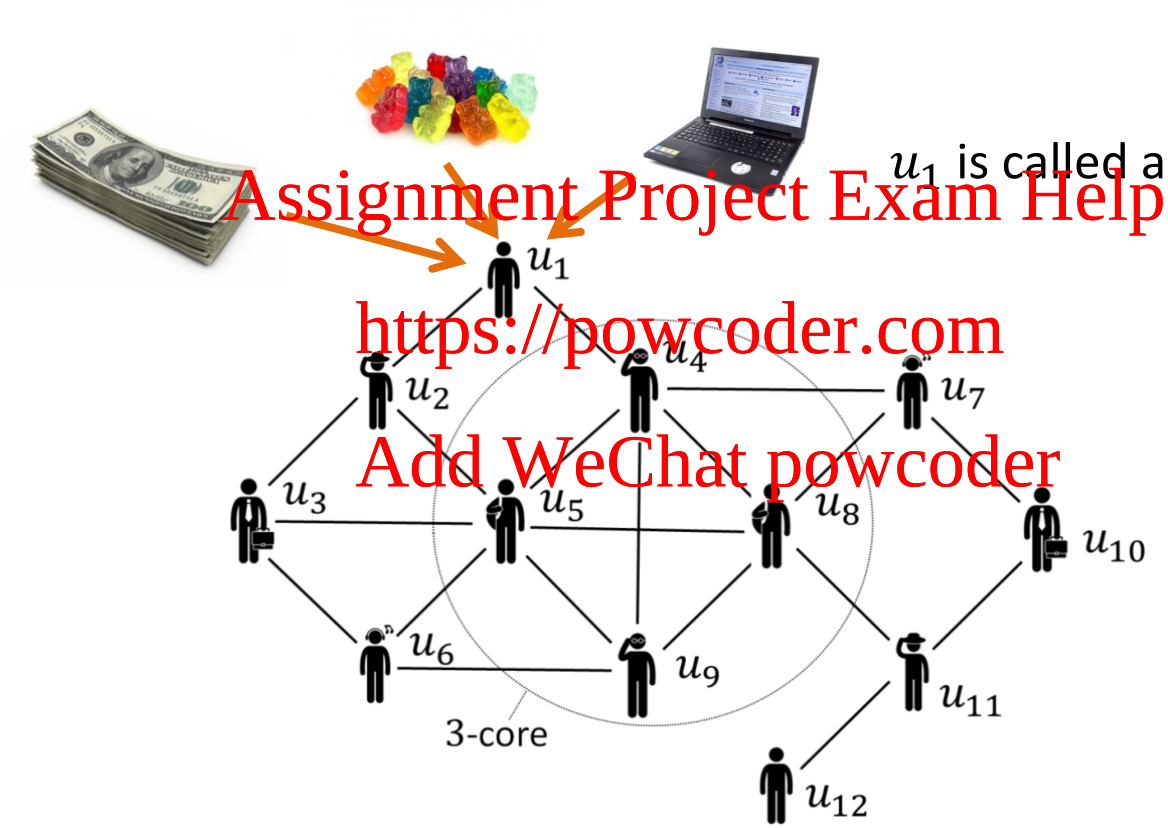
Network Unraveling

- Founded in 2002.
- Popular at early 21st century with over 115 million users.
- Suspended in 2015 for lack of user engagement.

F. D. Malliaros and M. Vazirgiannis. To stay or not to stay: modeling engagement dynamics in social graphs. In CIKM, pages 469–478, 2013.

The degeneration property of k-core can be used to quantify engagement dynamics.

Prevent Network Unraveling



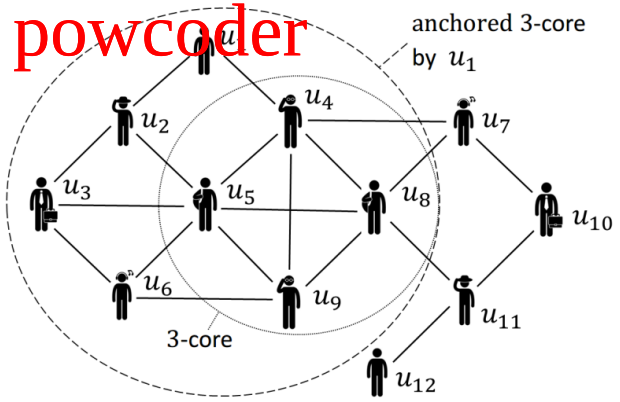
u_1 is called an anchor.

Prevent Network Unraveling

Follower: a node *v* is a follower of an anchor *u,* if *v* is not in k-core but belongs to anchored k-core by anchoring *u.*

Anchored k-Core Problem: Given two integers *k* and *b*, find *b* anchors to maximize the number of followers (i.e., maximize the number of nodes in anchored *k*-core).

NP-Hard



Performance Evaluation

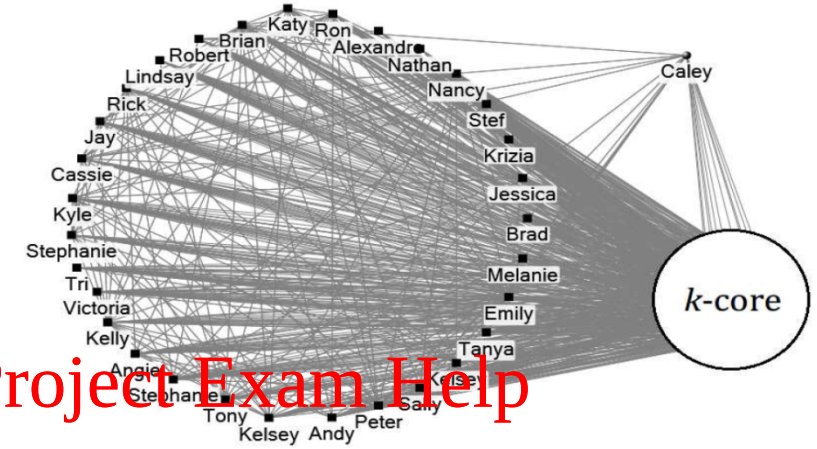
- Datasets:

Dataset	Nodes	Edges	d_{avg}	d_{max}
Facebook	4,039	88,234	43.7	1045
Brightkite	58,228	194,090	6.7	1098
Gowalla	196,591	456,830	4.7	9967
Yelp	552,339	1,781,998	6.5	3812
Flickr	105,938	2,316,948	43.7	5465
YouTube	1,134,890	2,987,624	5.3	28754
DBLP	1,566,919	6,461,300	8.3	2023
Pokec	1,632,803	8,320,605	10.2	7266
LiveJournal	3,997,962	34,681,189	17.4	14815
Orkut	3,072,441	117,185,083	76.3	33313

- Environments:
  - Intel Xeon 2.3GHz CPU and Redhat Linux System.
  - All algorithms are implemented in C++.

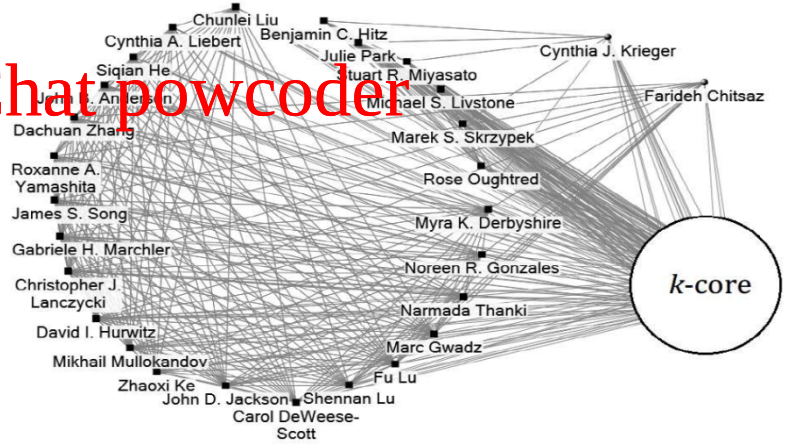
Case Studies

Yelp is a crowd-sourced local business review and social networking site.



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

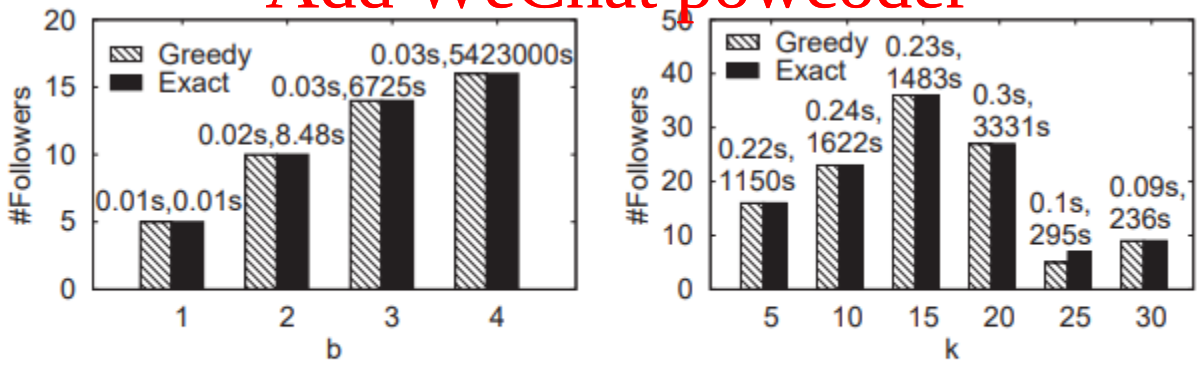
DBLP is a computer science bibliography website.



(b) DBLP, k=20, b=2

Effectiveness: Number of Followers

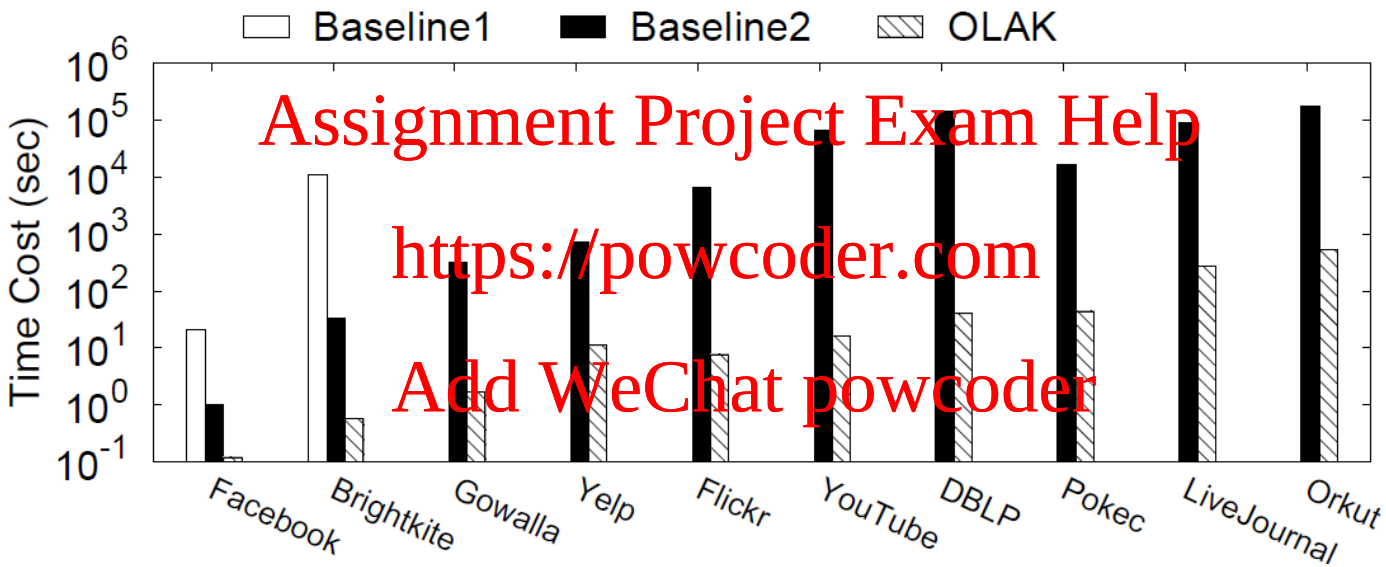
(a) 10 Datasets, k=20, b=20



(a) Facebook, k=30 (b) Brightkite, b=2

Figure 7: Greedy vs Exact

Efficiency

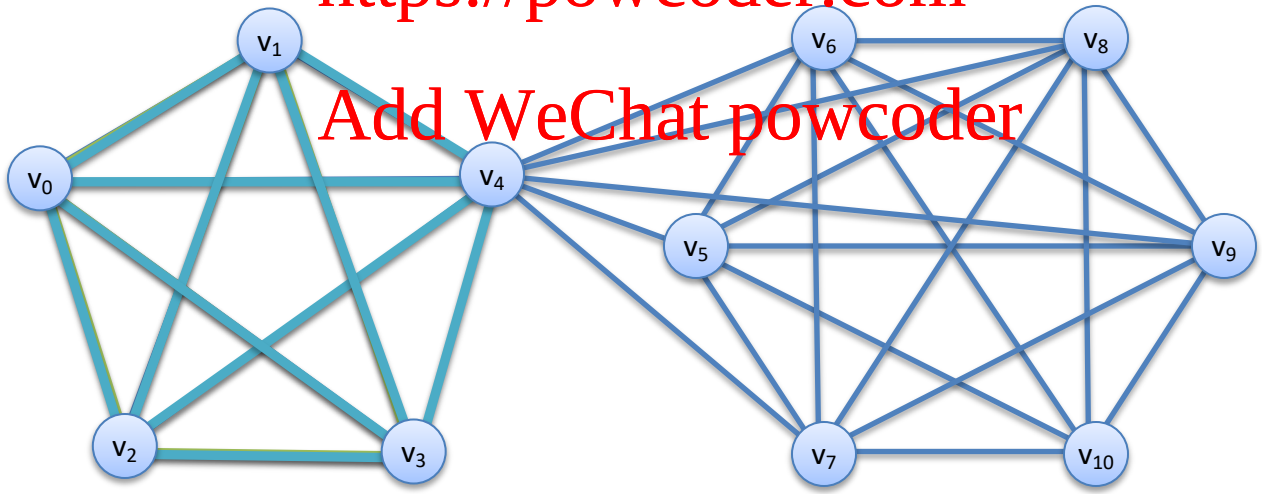


Diversified Top-k Clique Search

ICDE 2015

Clique and Maximal Clique

- Given a graph G, a clique is a set of nodes such that for any pair of them have an edge
- A clique is called maximal clique if there exist no other bigger cliques that contain it

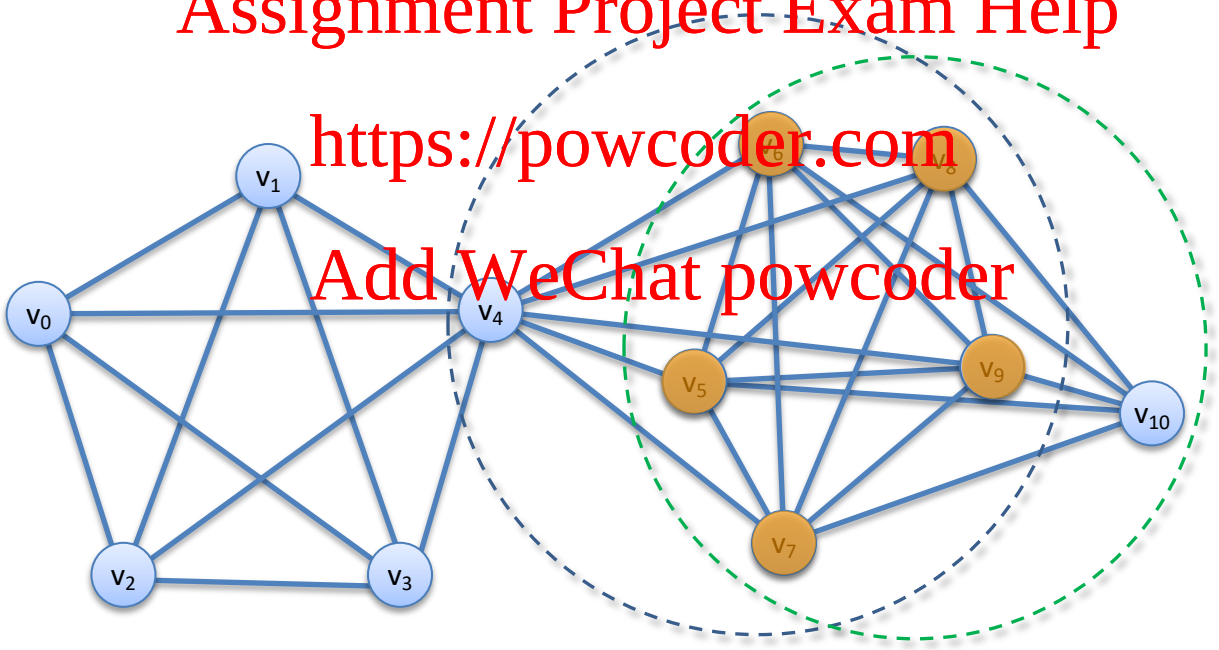


Traditional models for Maximal Clique

- Maximal Clique Enumeration:
  - Enumerate all the maximal cliques in the graph
    - Exponential number of maximal cliques
    - Large redundant and useless information
- Top-K Maximal Clique:
  - Return top-k maximal cliques with largest size
    - Still contain large redundancy and hold little information as a whole.

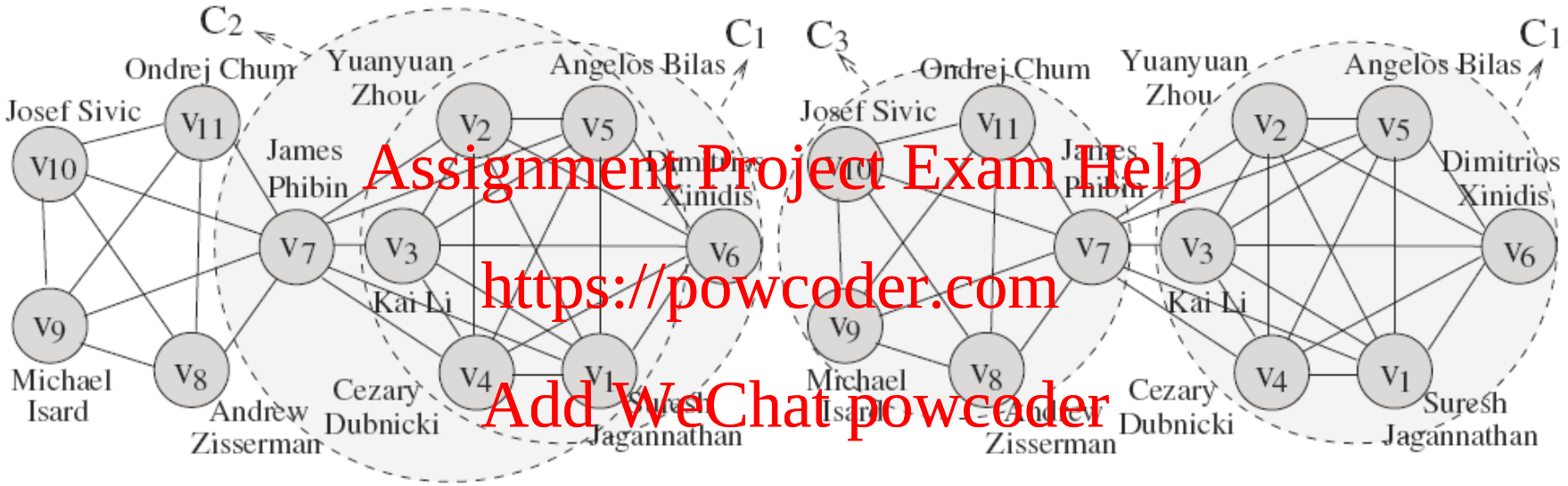
Diversified Top-K Clique

Traditional top-k clique high overlap



Diversified Top-K Clique

Diversified top-k clique: cover more vertices



(a) Top-2 Maximal Cliques (b) Diversified Top-2 Cliques

Fig. 1: Part of the Collaboration Network in *DBLP*

Diversified Top-K Clique Search

- Diversified Top-K Clique:
  - Given: a graph *G* and an integer *k*,
    - Maximize $|C_1 \cup C_2 \ldots C_k|$
    - s.t $C_i (1 \leq i \leq k)$ is a maximal clique
  - NP-hard Problem
- Advantage:
  - Consider both size and diversity, provide a better query result for users.

Baseline Solutions

- EnumAll

  Phase 1: Enumerate all the maximal cliques in the graph(D. Eppstein et al., SEA'11)

  Phase 2: Greedily select k cliques from all the maximal cliques that cover most nodes in the graph

- Problem
  - Clique enumeration is a costly operation
  - Keeping all maximal cliques in memory is infeasible
    - The number of cliques is exponential to the number of nodes

Baseline Solutions

- EnumSub(sample-based enumeration)

  Phase 1: Sample a subset of the maximal cliques in the graph(J. Wang et al., KDD'13)

  Phase 2: Greedily select k cliques from the sampling that cover most nodes in the graph

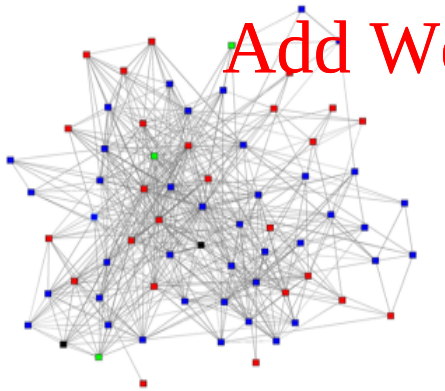
- Problem
  - It still outputs exponential number of maximal cliques without a bound

Challenge

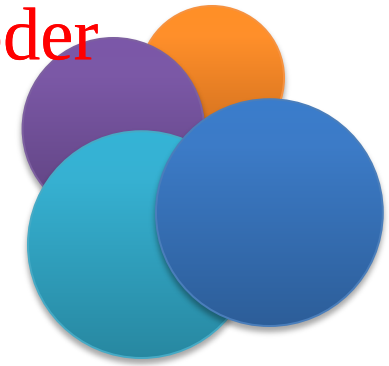
- Retain the result quality, while avoid:
  - generating all maximal cliques, and
  - keeping all generated maximal cliques in memory

Our Approach: extending online k coverage problem

- Main Idea
  1. online model storing k maximal cliques in memory
  2. update top-k candidate set when enumerate cliques
     - replace small existing cliques with big new cliques



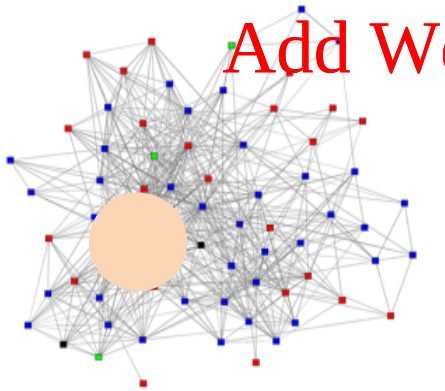
Input Graph



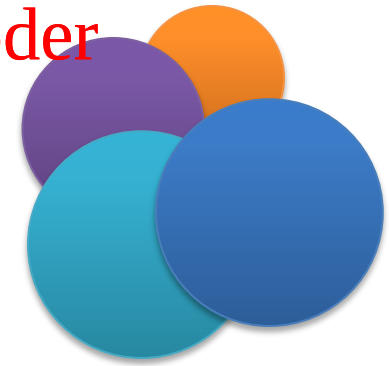
Top-K Candidate Set

Our Approach: extending online k coverage problem

- Main Idea
  1. online model storing k maximal cliques in memory
  2. update top-k candidate set when enumerate cliques
     - replace small existing cliques with big new cliques



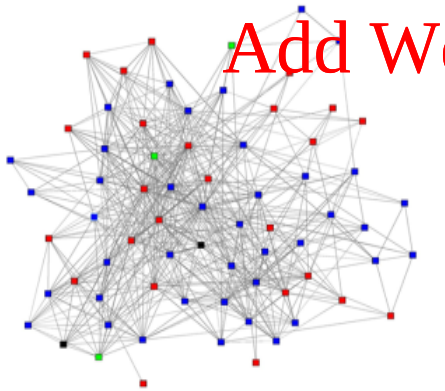
Input Graph



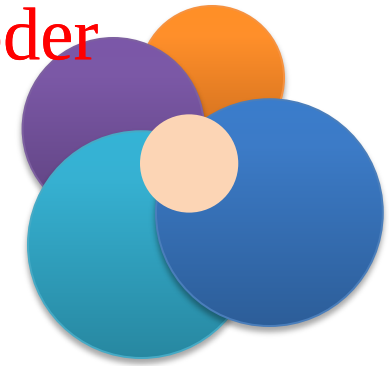
Top-K Candidate Set

Our Approach: extending online k coverage problem

- Main Idea
  1. online model storing k maximal cliques in memory
  2. update top-k candidate set when enumerate cliques
     - replace small existing cliques with big new cliques



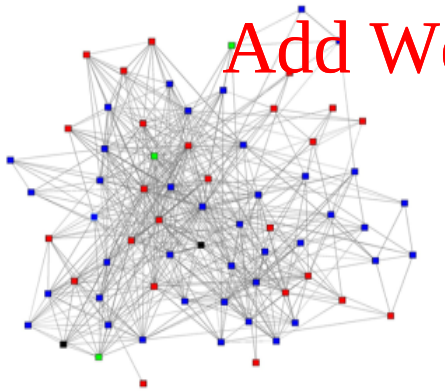
Input Graph



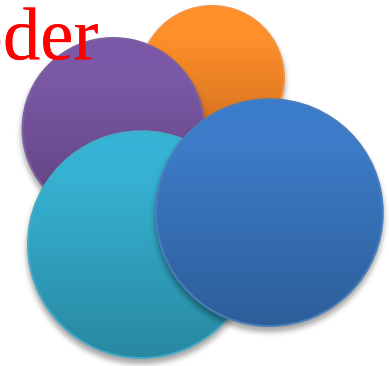
Top-K Candidate Set

Our Approach: extending online k coverage problem

- Main Idea
  1. online model storing k maximal cliques in memory
  2. update top-k candidate set when enumerate cliques
     - replace small existing cliques with big new cliques



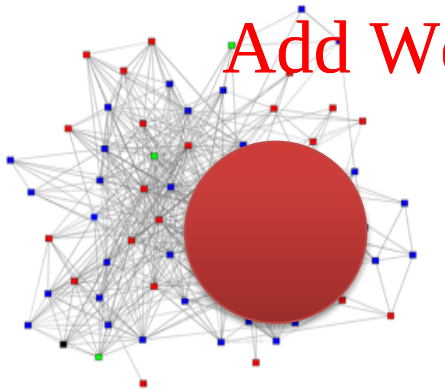
Input Graph



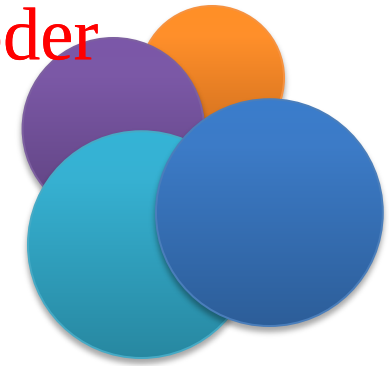
Top-K Candidate Set

Our Approach: extending online k coverage problem

- Main Idea
  1. online model storing k maximal cliques in memory
  2. update top-k candidate set when enumerate cliques
     - replace small existing cliques with big new cliques



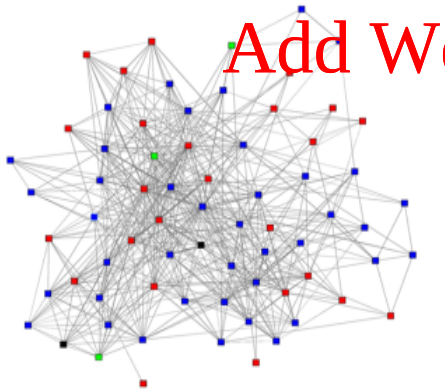
Input Graph



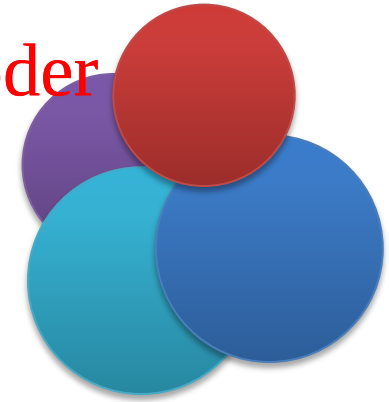
Top-K Candidate Set

Our Approach: extending online k coverage problem

- Main Idea
  1. online model storing k maximal cliques in memory
  2. update top-k candidate set when enumerate cliques
     - replace small existing cliques with big new cliques

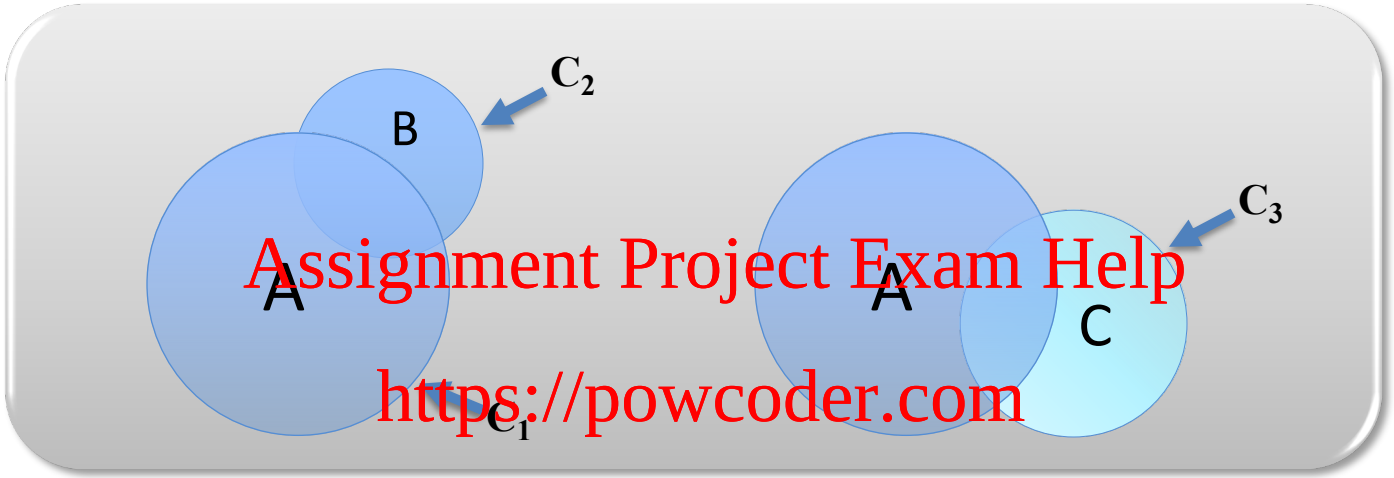


Input Graph



Top-K Candidate Set

Replacement Strategy



- which one to be replaced
  - private set
  - C_{min}: clique with smallest private set
- what is the replacement condition
  - $|C| > |B| + \alpha \cdot (|A| + |B|)/k$
  - α is a parameter $(0 < \alpha \leq 1)$

Advantages of Our Approach

- Guaranteed result quality
  - Achieve a guaranteed approximation ratio of **0.25**, and much better in practice.
- Low memory consumption
  - Instead of all maximal cliques, just **K** most promising candidates are kept in memory
- Efficiency and Scalability ?

Cost Analysis

- A naïve implementation of our approach:
  - for each generated maximal clique *C*
  - update *top-k candidate set* by *C* with replacement condition
- The time complexity is :

$$O(\ |\mathcal{A}| * k * |C_{max}| + T_{enum}(G)\ )$$

$|\mathcal{A}|$: number of maximal cliques

$|C_{max}|$: size of maximum clique

k: time for maintaining top-k candidate set

$T_{enum}(G)$: time for enumerating maximal cliques

Cost Analysis

- A naïve implementation of our approach:
  - for each generated maximal clique *C*
  - update *top-k candidate set* by *C* with replacement condition
- The time complexity is :

$$O(\ |\mathcal{A}| * k * |C_{max}| + T_{enum}(G)\ )$$

$|\mathcal{A}|$: number of maximal cliques

$|C_{max}|$: size of maximum clique

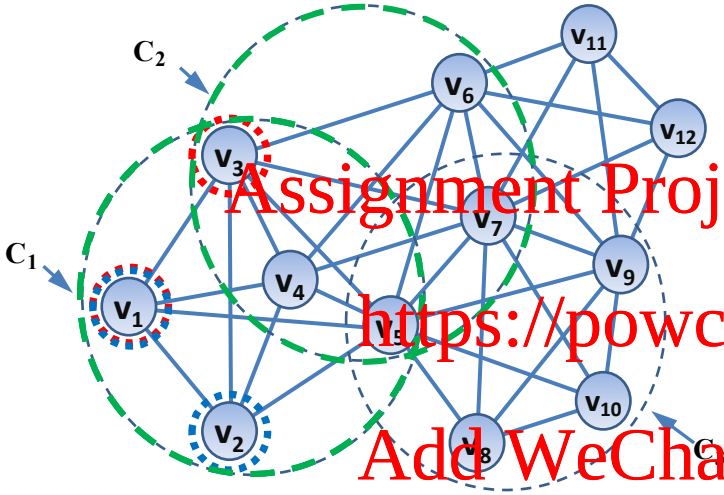
k: time for maintaining top-k candidate set

PNP-Index

$T_{enum}(G)$: time for enumerating maximal cliques

Pruning Rules

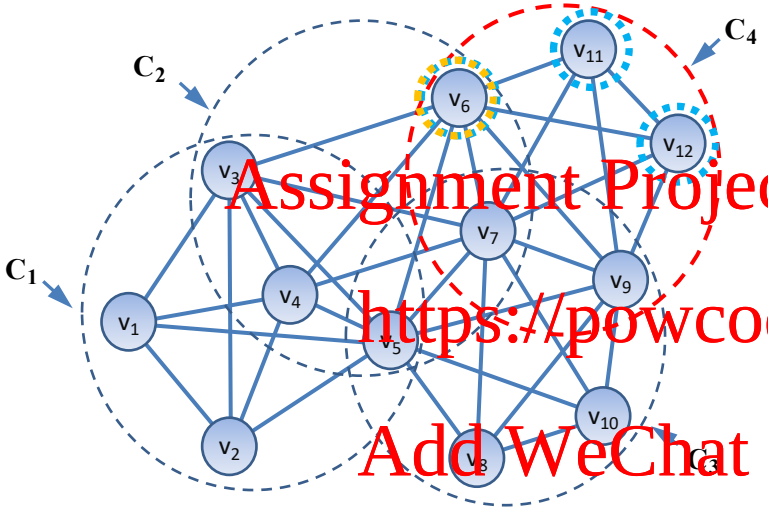
PNP-Index



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	|priv(C)|
c_1	$\bar{1}$	$\bar{1}$	1	1	1								2
c_2			1	1	1	$\bar{1}$	1						1^*
c_3					1		1	$\bar{1}$	$\bar{1}$	$\bar{1}$			3
|rcov(v)|	1	1	2	2	3	1	2	1	1	1			|cov|:10

PNP-Index

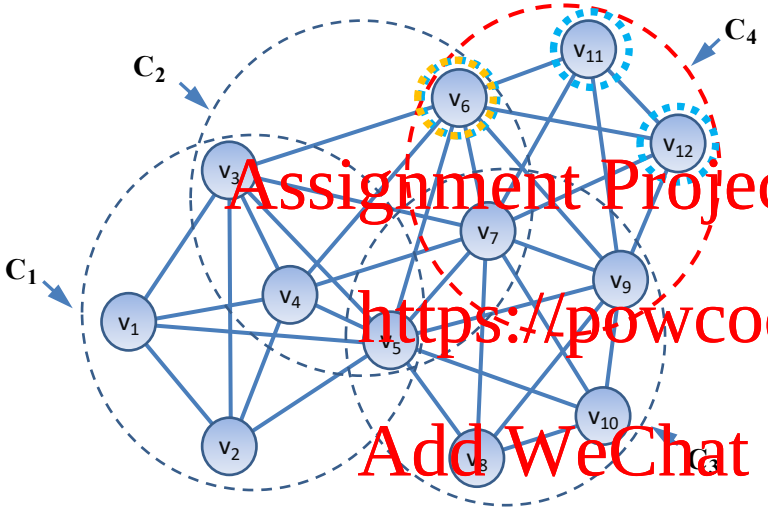
- Step 1: Check the replacement condition



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	|priv(C)|
c_1	$\bar{1}$	$\bar{1}$	1	1	1								2
c_2			1	1	1	$\bar{1}$	1						1^*
c_3					1		1	$\bar{1}$	$\bar{1}$	$\bar{1}$			3
|rcov(v)|	1	1	2	2	3	1	2	1	1	1			|cov|:10
c_4						1	1		1		1	1	

PNP-Index

- Step 1: Check the replacement condition



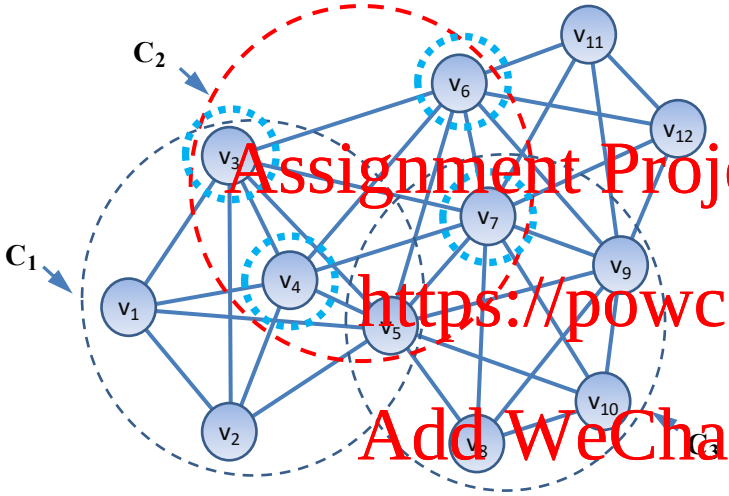
$$3 > 1 + 0.5 \times 10 \div 3 = 2.67$$

α

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	|priv(C)|
c_1	$\bar{1}$	$\bar{1}$	1	1	1								2
c_2			1	1	1	$\bar{1}$	1						1^*
c_3					1		1	$\bar{1}$	$\bar{1}$	$\bar{1}$			3
|rcov(v)|	1	1	2	2	3	1	2	1	1	1			|cov|:10
c_4						1	1		1		1	1	

PNP-Index

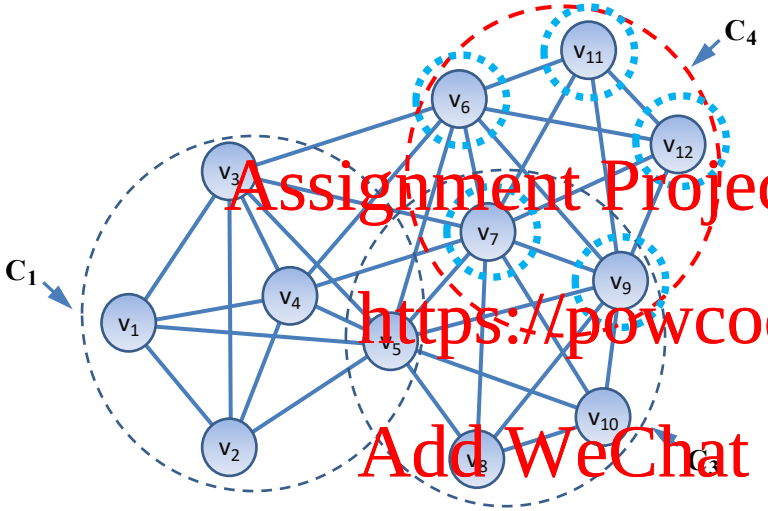
- Step 2: Delete C_2



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	|priv(C)|
c_1	$\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$	1								4
c_3					1		$\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$			4
|rcov(v)|	1	1	1	1	2		1	1	1	1			|cov|:9

PNP-Index

- Step 3: Insert C_4



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	|priv(C)|
c_1	$\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$	1								4
c_4						$\bar{1}$	1		1		$\bar{1}$	$\bar{1}$	3
c_3					1		1	$\bar{1}$	1	$\bar{1}$			2*
|rcov(v)|	1	1	1	1	2	1	2	1	2	1	1	1	|cov|:12

PNP-Index

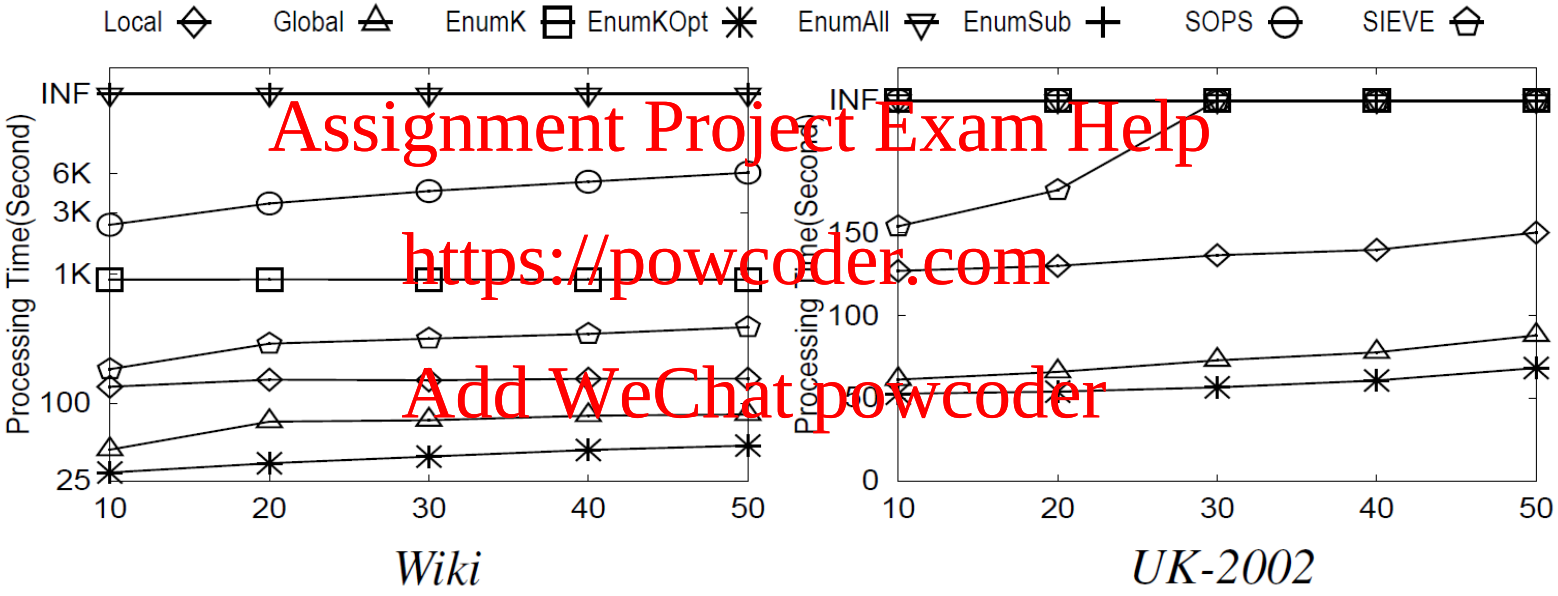
- An naïve implementation for candidate set maintenance needs $O(|\mathcal{A}| * k * |C_{max}|)$
- With the help of PNP-Index, our algorithm can only take $O(\sum_{c \in \mathcal{A}} |C|)$ time

Pruning Rules

- Global Pruning: based on k-core and graph coloring
- Local Pruning: check the current coverage of the neighbourhood
- Initial Candidate Computation: increase the power of the above pruning processes

Performance Evaluation - Efficiency

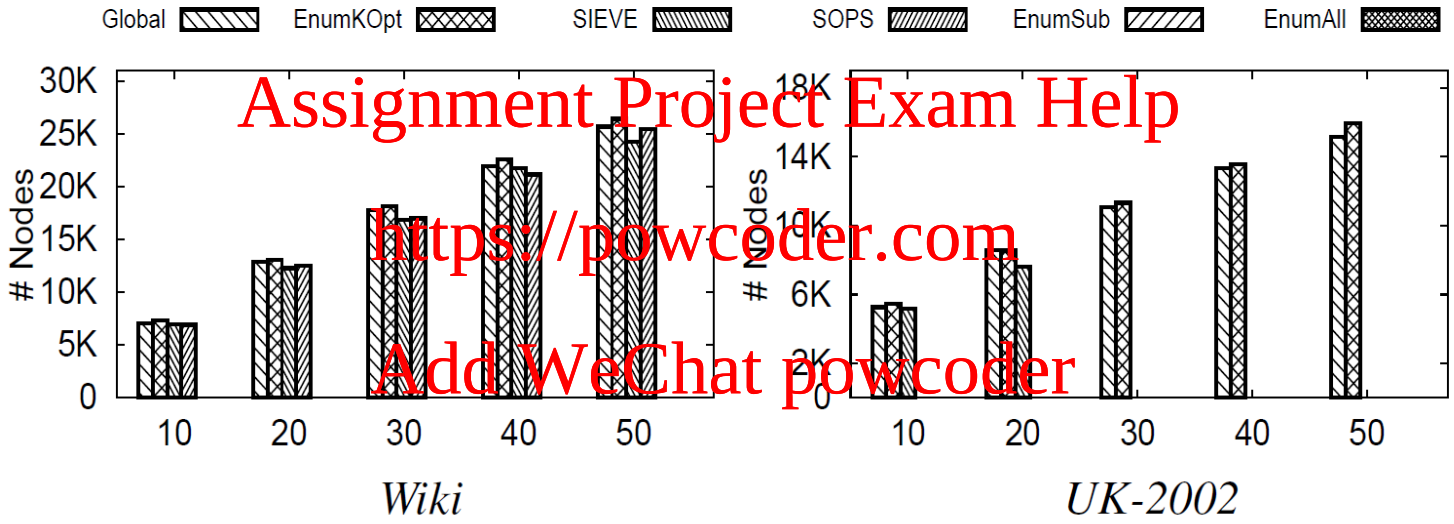
- Vary k (top-k value) and report total processing time



Vary k

Performance Evaluation - Effectiveness

- Vary k(top-k value) and report covered nodes



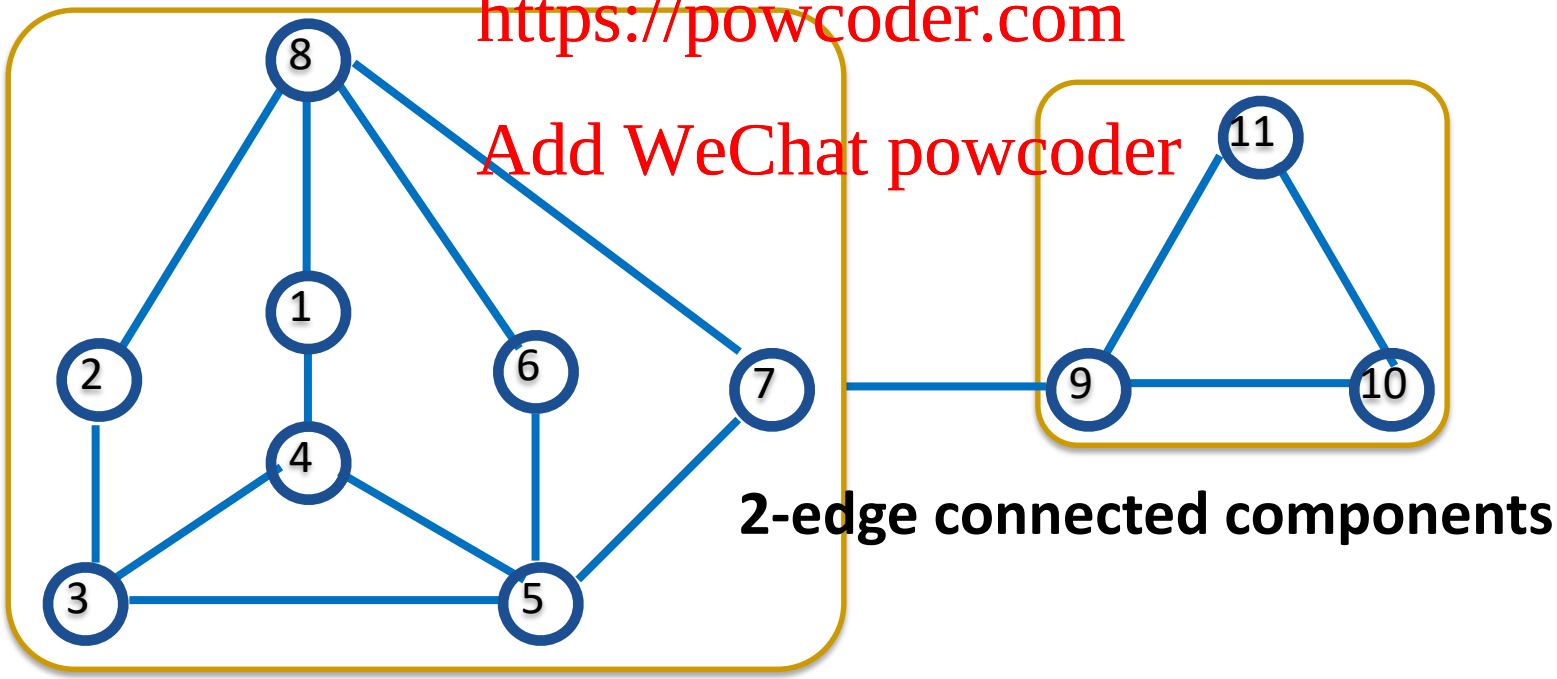
Vary *k*

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

SIGMOD 2015

k-edge connected components

- Computing k-edge connected components plays a vital role in graph-based analysis
  - A graph is k-edge connected if it is still connected after removing any set of (k-1) edges from it



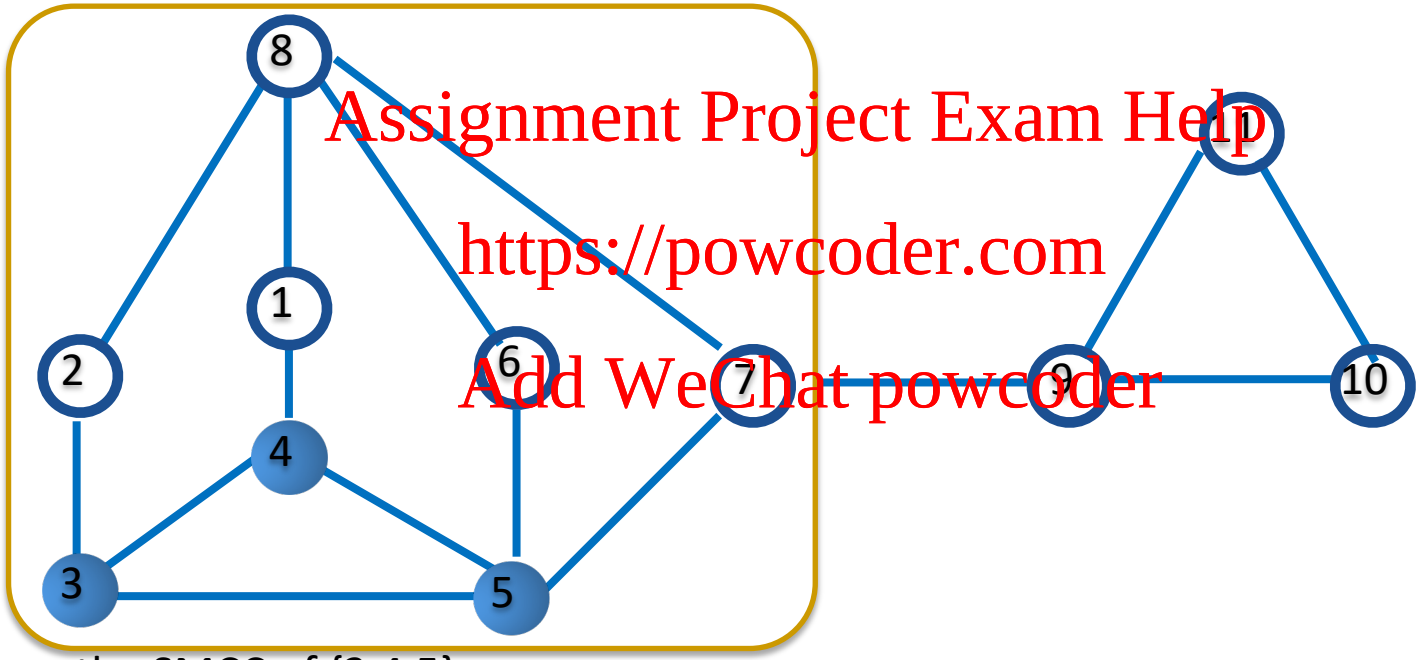
2-edge connected components

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

- *SMCC* of q: maximal subgraph of G containing $q \subseteq V(G)$ with maximum connectivity.

- *Steiner-connectivity* of q: the connectivity of the SMCC of q. Denoted $sc(q)$.

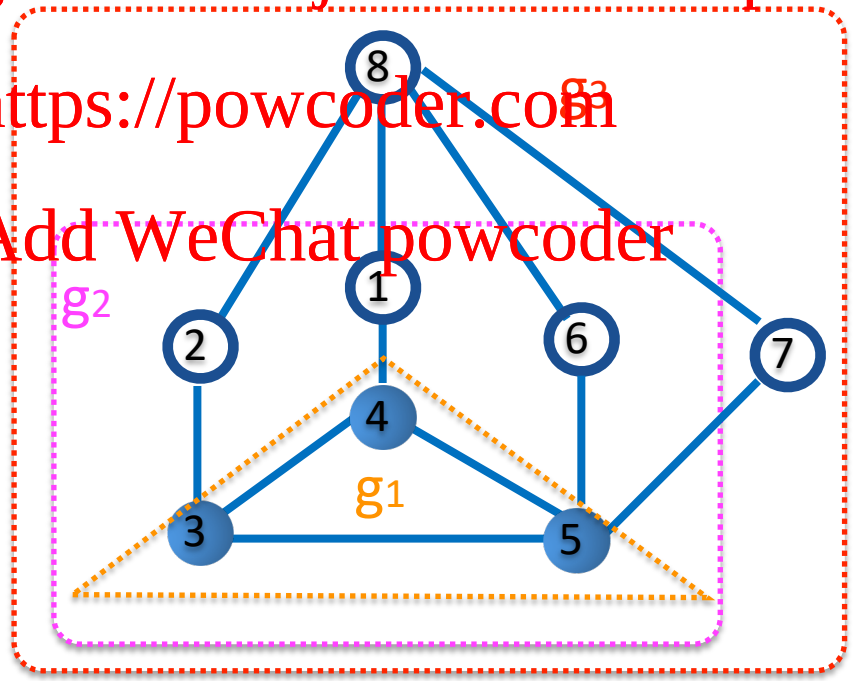
Steiner Maximum-Connected Component (SMCC) and Steiner-Connectivity



the SMCC of {3,4,5}

Challenges

- Challenge-I: connectivities of subgraphs are not monotonic.



Challenges

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

Naive Solution

- SMCC of q is the k-edge connected component of G containing q with the maximum k
- Compute k-edge connected components by enumerating k from $|V|$ to 1.
  - Still expensive, because it needs to traverse the entire graph G multiple times

Overview of Our Approach

- Recall $sc(q)$: steiner-connectivity of q
  - $sc(u,v)$: steiner-connectivity of $\{u,v\}$
- Key observations:
  - 1: for a fixed $u \in q$, $sc(q) = \min_{v \in q} sc(u,v)$.
  - 2: for a fixed $u \in q$, SMCC of q is the set of vertices v satisfying $sc(u,v) \geq sc(q)$.

Framework

- Phase-I: offline index construction
  - Step-1: compute $sc(u,v)$ for all edges (u,v) in G
  - Step-2: compute index 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 $sc(u,v) \geq sc(q)$

Index Construction

- Compute $sc(u,v)$ for every edge (u,v) in G, instead of for all vertex pairs

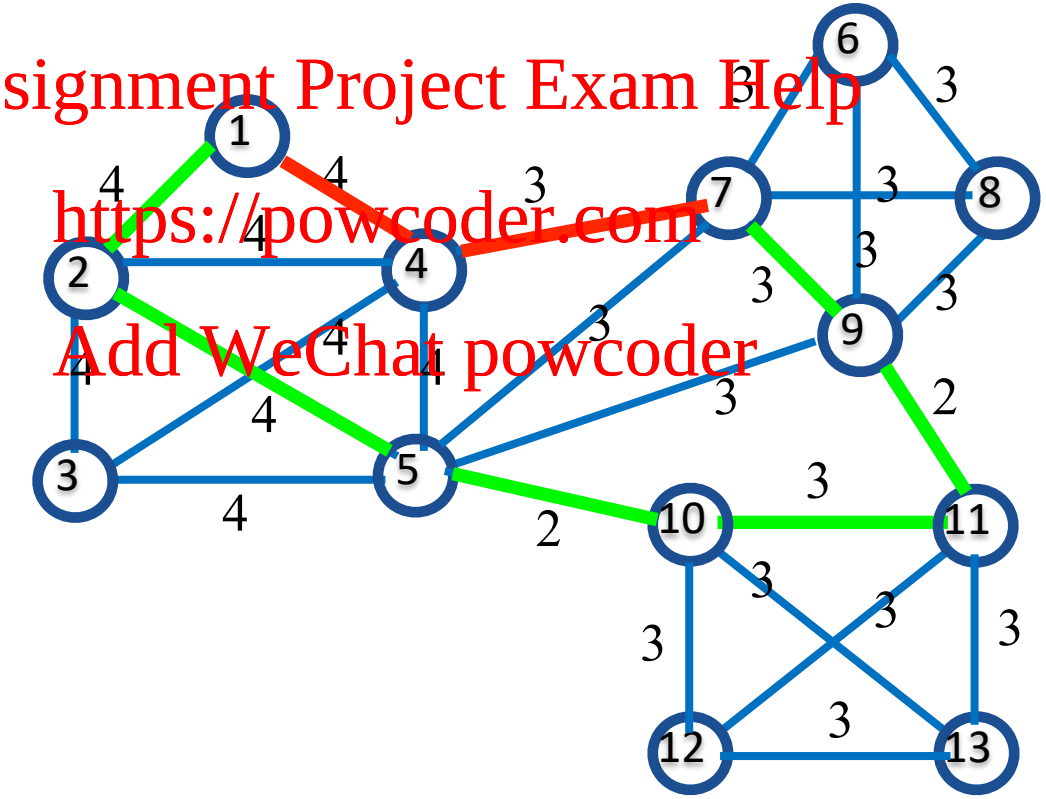
Time complexity: $O(\alpha(G) \times kECC(G))$, where $\alpha(G)$ is the arboricity of a graph G, $kECC(G)$ is the cost of computing kECC of G [SIGMOD 13].

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

Index Construction

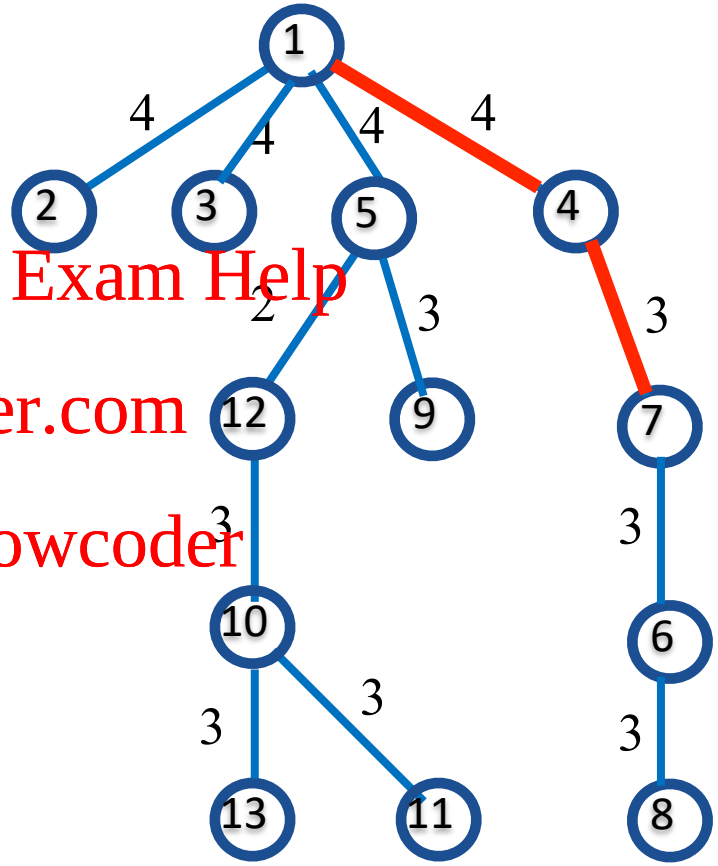
- $sc(u,v)$ equals the maximum among the minimum weights of (edges in) all paths between u and v.

Compute $sc(1,7)$



Index Construction

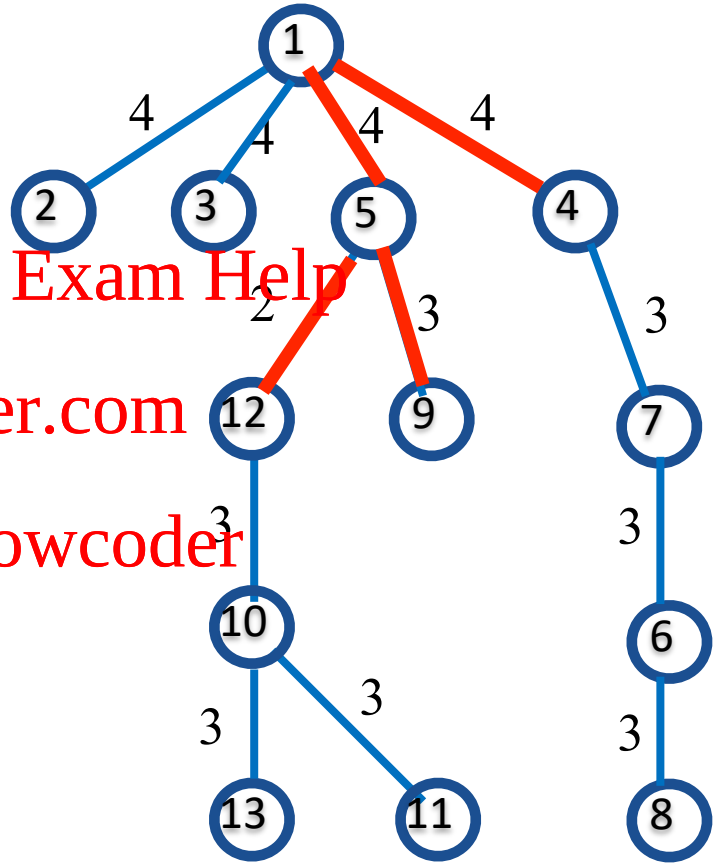
- Index structure: the maximum spanning tree (MST) T of the weighted graph.
  - Time complexity: $O(|E|)$.
  - $sc(u,v)$ equals the minimum weight in the path between u and v in the MST T.



Query Processing: Steiner-Connectivity Query

Algorithm: find the spanning subtree T_q of T connecting vertices in q, and return the minimum weight in T_q as $sc(q)$.

Time complexity: $O(T_q)$



Compute $sc(12, 9, 4)$

Query Processing: SMCC Query

Algorithm: after obtaining $sc(q)$, extend the subgraph from q by visiting edges with weights at least $sc(q)$.

Time complexity: $O(|V_k|)$, where V_k is the result set.

Performance Studies

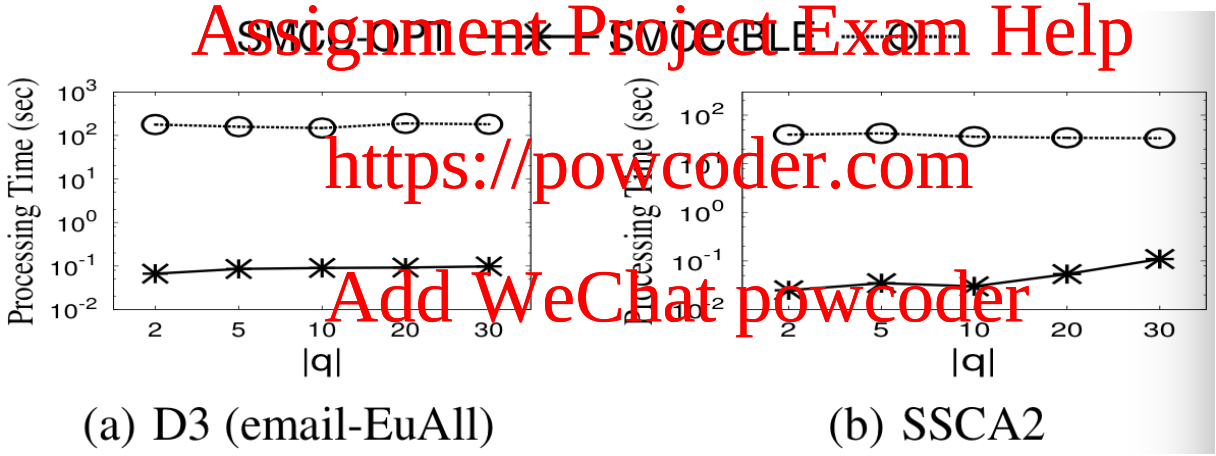
- Statistics of Data

Type	ID	Dataset	#Edges	#Vertices	$\overline{d}$
small	D1	ca-GrQc	13,422	4,158	6.46
	D2	ca-CondMat	91,286	21,363	8.55
	D3	email-EuAll	339,925	224,832	3.02
	D4	soc-Epinions1	405,739	75,877	10.69
large	D5	amazon0601	2,443,311	403,364	12.11
	D6	web-Google	3,074,322	665,957	9.23
	D7	wiki-Talk	4,656,682	2,388,953	3.90
	D8	as-Skitter	11,094,209	1,694,616	13.09
	D9	LiveJournal	42,845,684	4,843,953	17.69
	D10	uk-2002	261,556,721	18,459,128	28.34
	D11	twitter-2010	1,202,513,344	41,652,230	57.7

SMCC Query

SMCC-BLE: baseline algorithm

SMCC-OPT: optimal algorithm



(a) D3 (email-EuAll) (b) SSCA2

Graph	SMCC-OPT	Graph	SMCC-OPT
D5	13	D9	81
D6	6.1	D10	87
D7	2.9	D11	1.5
D8	18		

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.

Algorithm Analysis

Bounded Memory.

- Follow the semi-external model and require only O(n) memory.

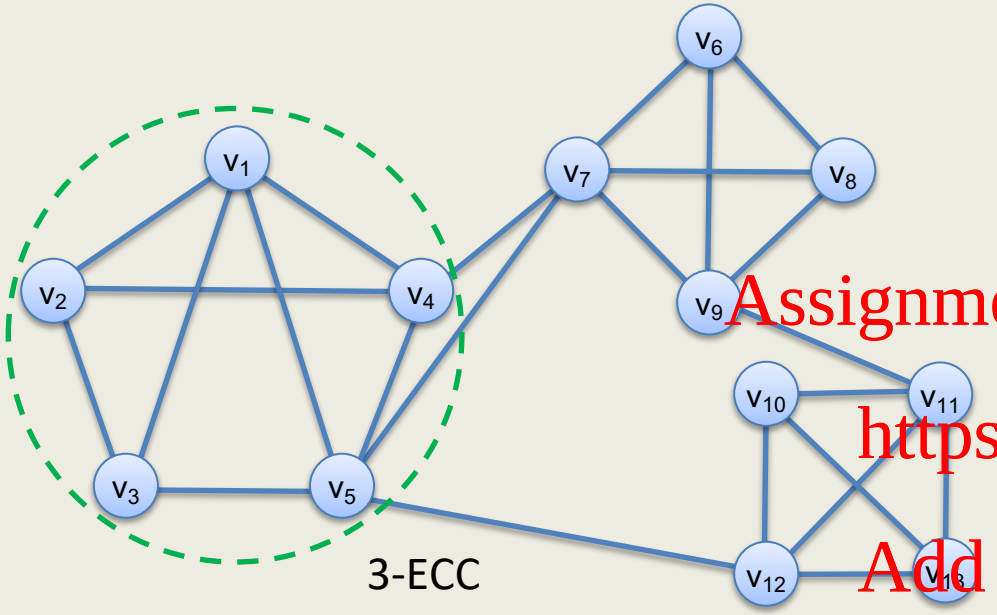
Read I/O Only.

- Only require read I/Os by scanning the node and edge tables sequentially on disk.

Simple In-memory Structure and Data Access.

- Use two vectors to save the core number and count value for each node.

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

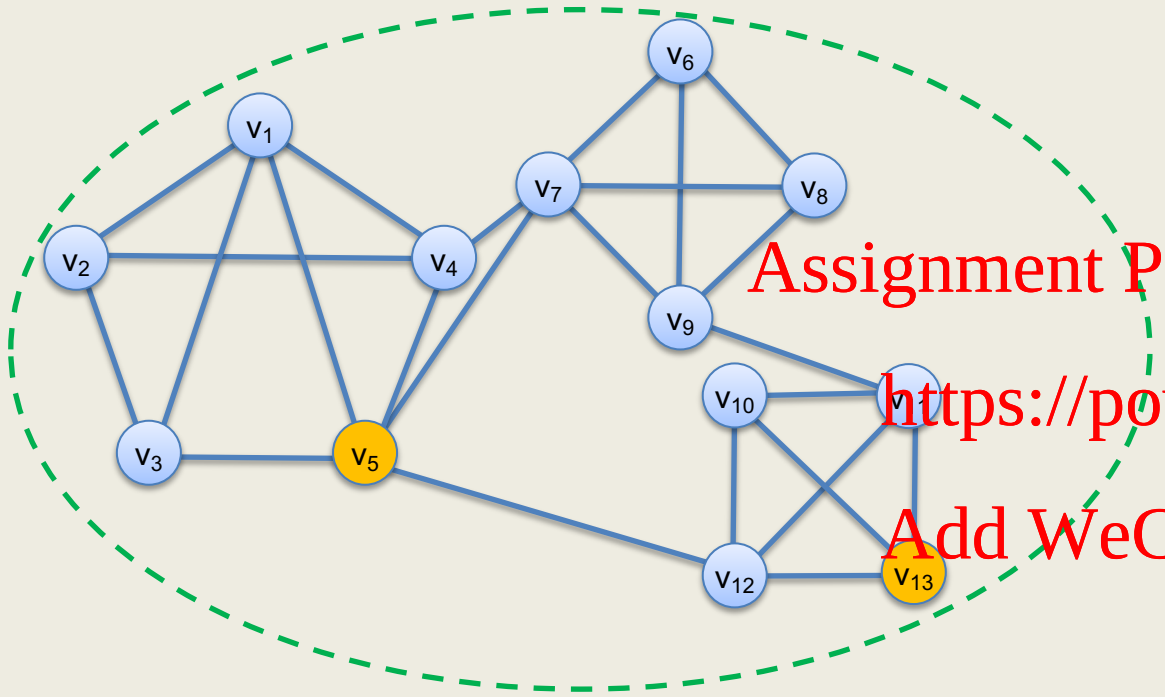
Publications:

- 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



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

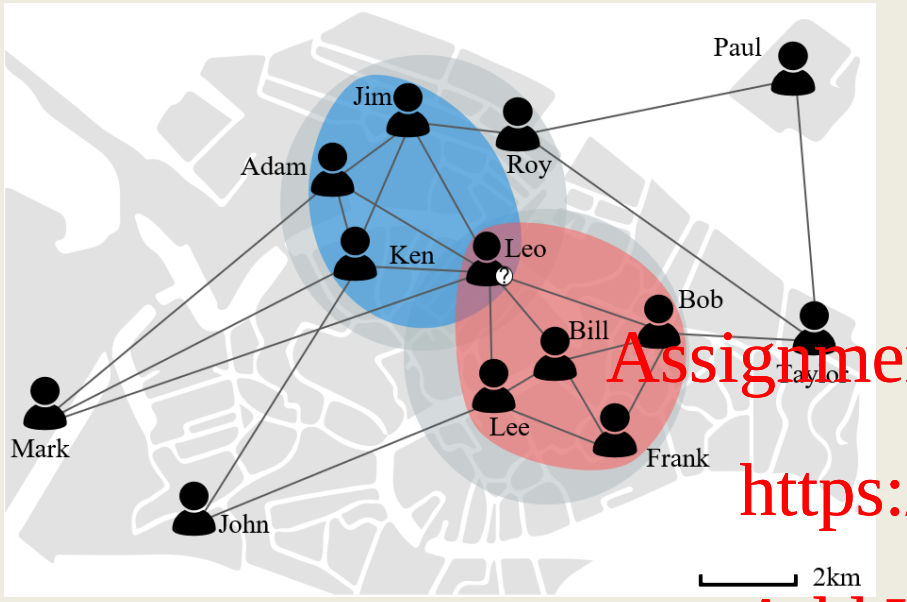
Problem Definition:

- 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)

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.

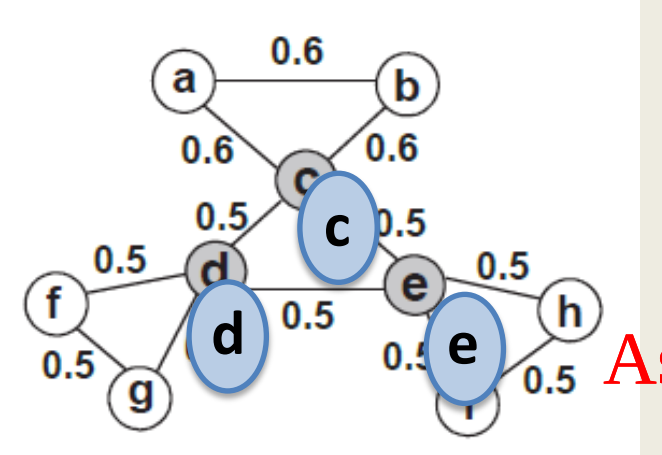
Publications:

- 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

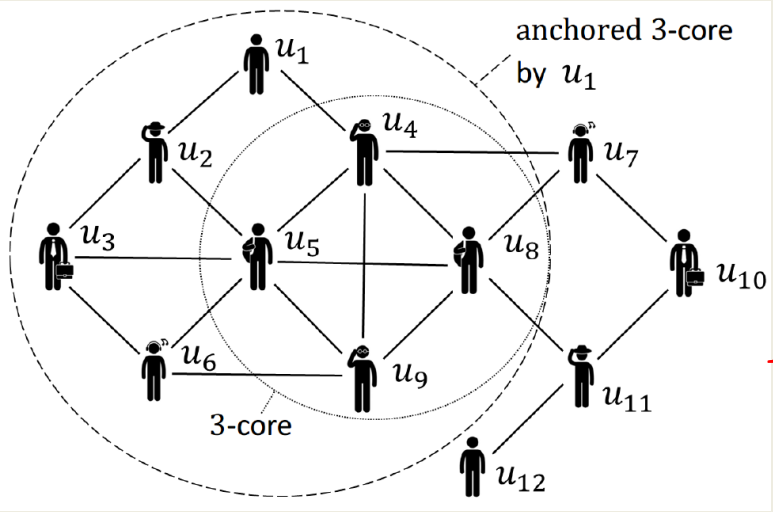
Algorithm and Published Paper:

- 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：

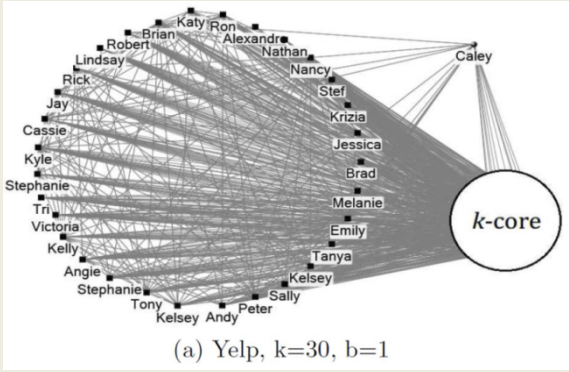
- Iteratively find a best user to anchor based on the heuristics, such that the corresponding subgraph (e.g., *k*-core) is the largest, VLDB 2017, ICDE 2018.

Contributions：

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

Problem:

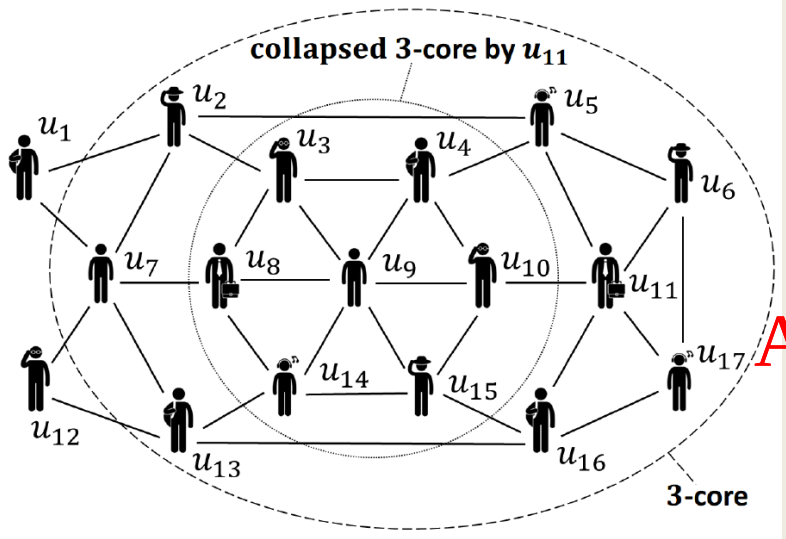
- *k*-core: a maximal subgraph where each vertex is adjacent to at least *k* vertices in the subgraph.
- Given a network *G*, find *b* critical users whose engagement can significantly prevent network unraveling, i.e., find *b* vertices whose persistent existence leads to the *k*-core with the largest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

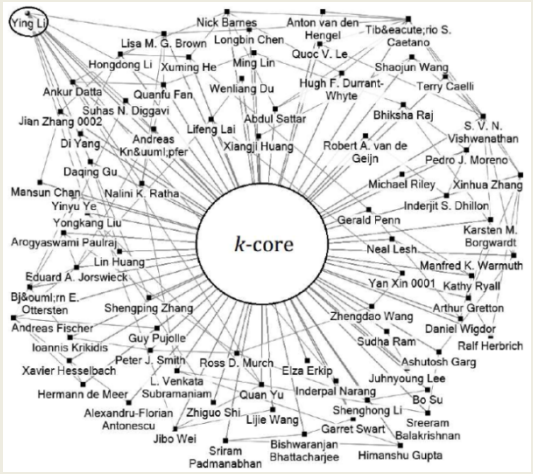
- Iteratively find a best user and delete, such that the corresponding *k*-core is the smallest, AAAI 2017.

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.



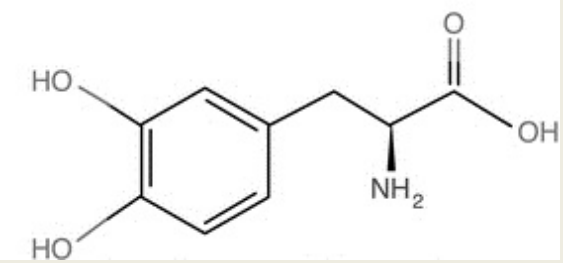
- In DBLP, the co-author network, the leave of Ying Li will break the engagement of other 74 users (who also leave the k-core).

GRAPH SIMILARITY AND CLUSTERING

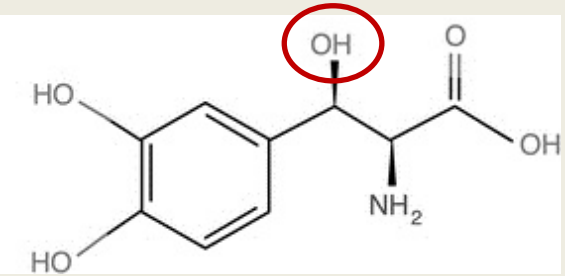
Graph Similarity and Clustering: Summary

- 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

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

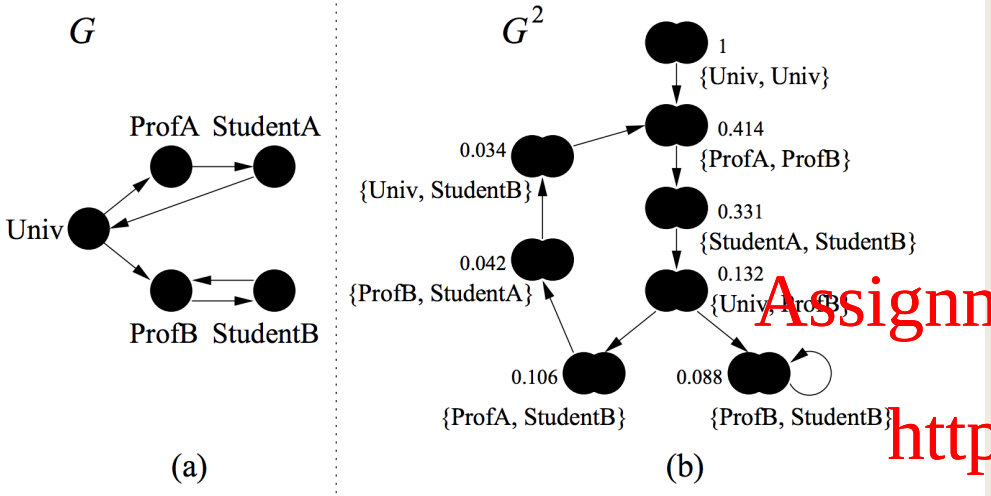
Problem Definition:

- **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.

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

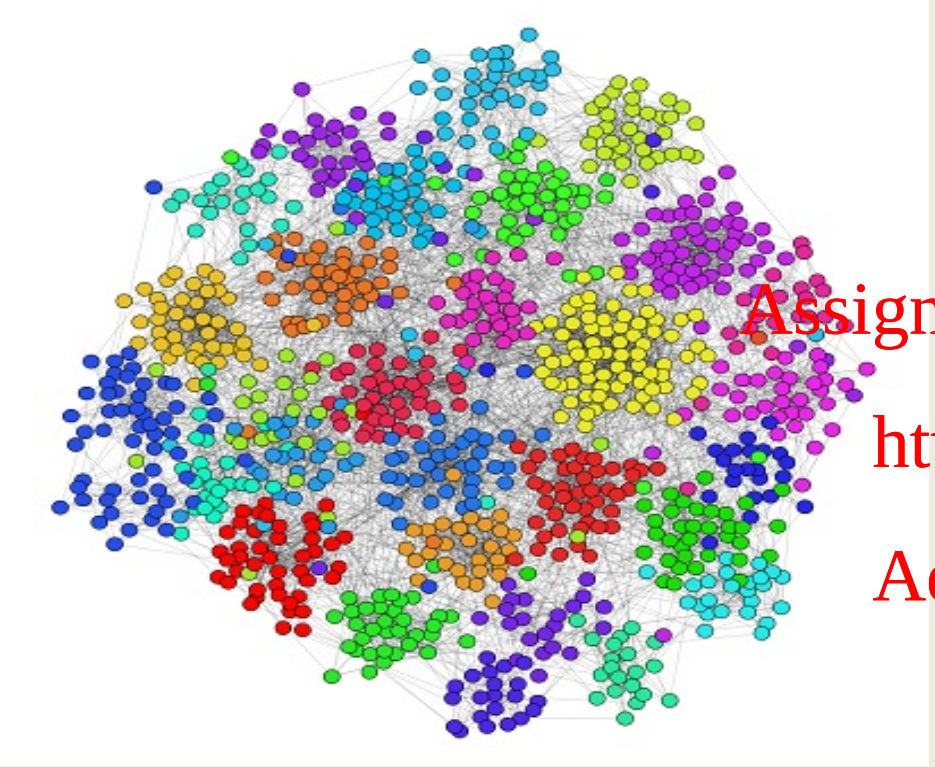
Algorithms and publications

- 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

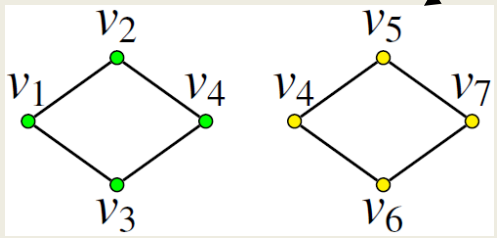
Algorithms and publications:

- 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

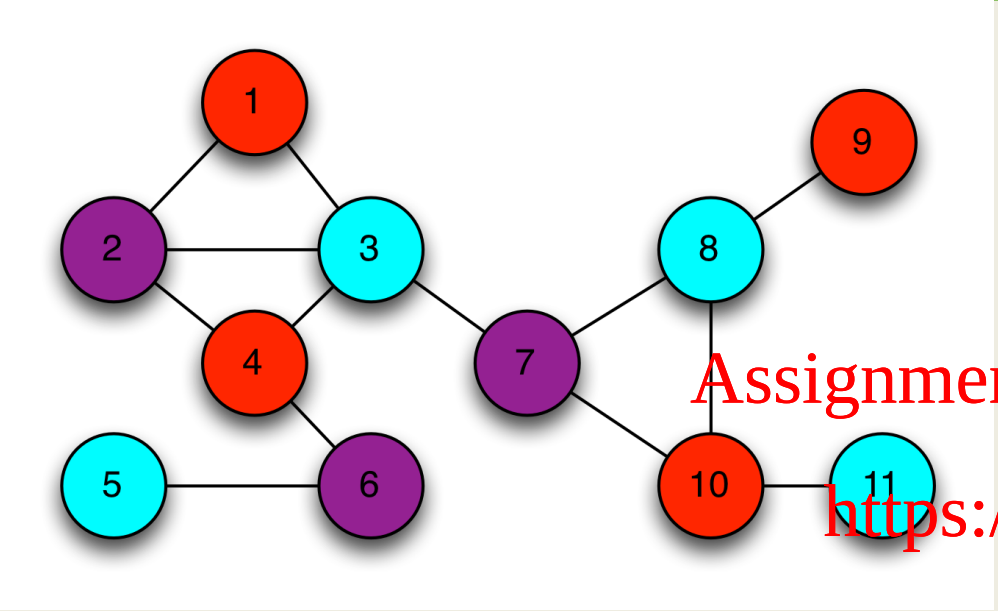
- Algorithms and Recent Publications
  - Distributed Graph Computing Model
    - Scalable Graph Computation Model, SIGMOD’14
  - Distributed Connected Component
    - CC based on graph decomposition and DCC based on label propagation，ICDE’16
  - Distributed Scalable Subgraph Enumeration
    - Optimal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

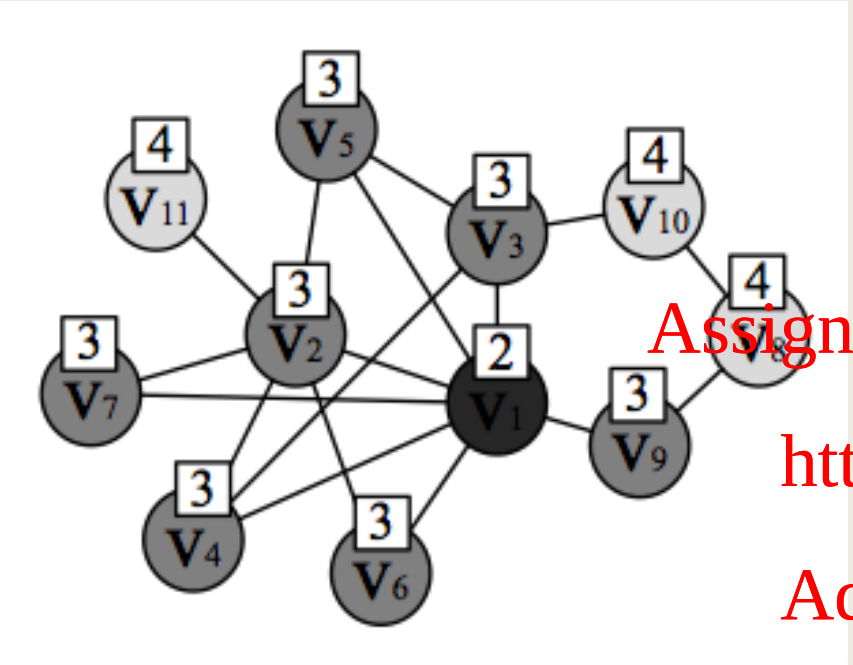
Algorithms and publications:

- 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:

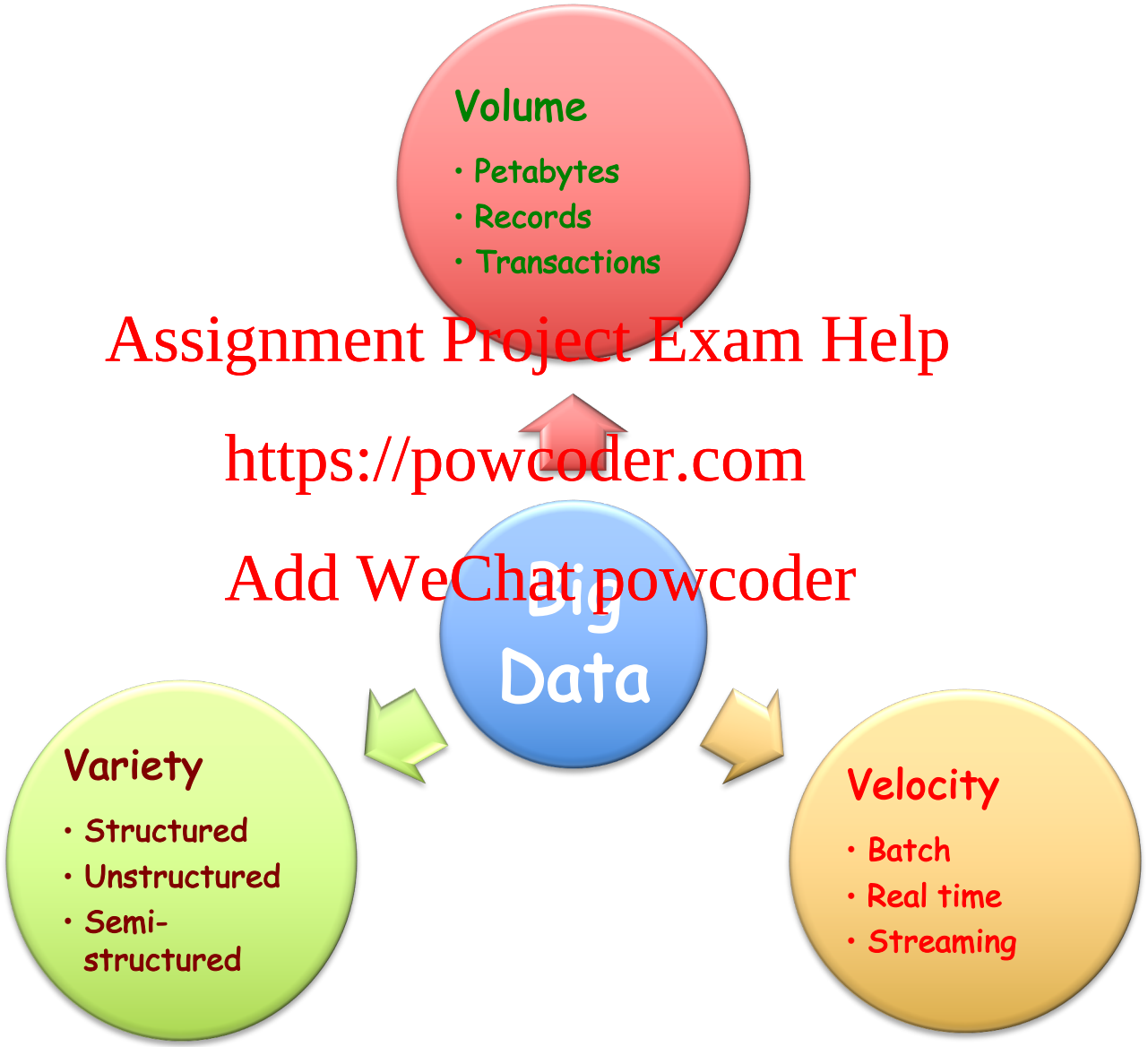
- Vertex ordering (via their importance)
- Compute some other graph features (diameter, radius etc.)

Algorithms and publications:

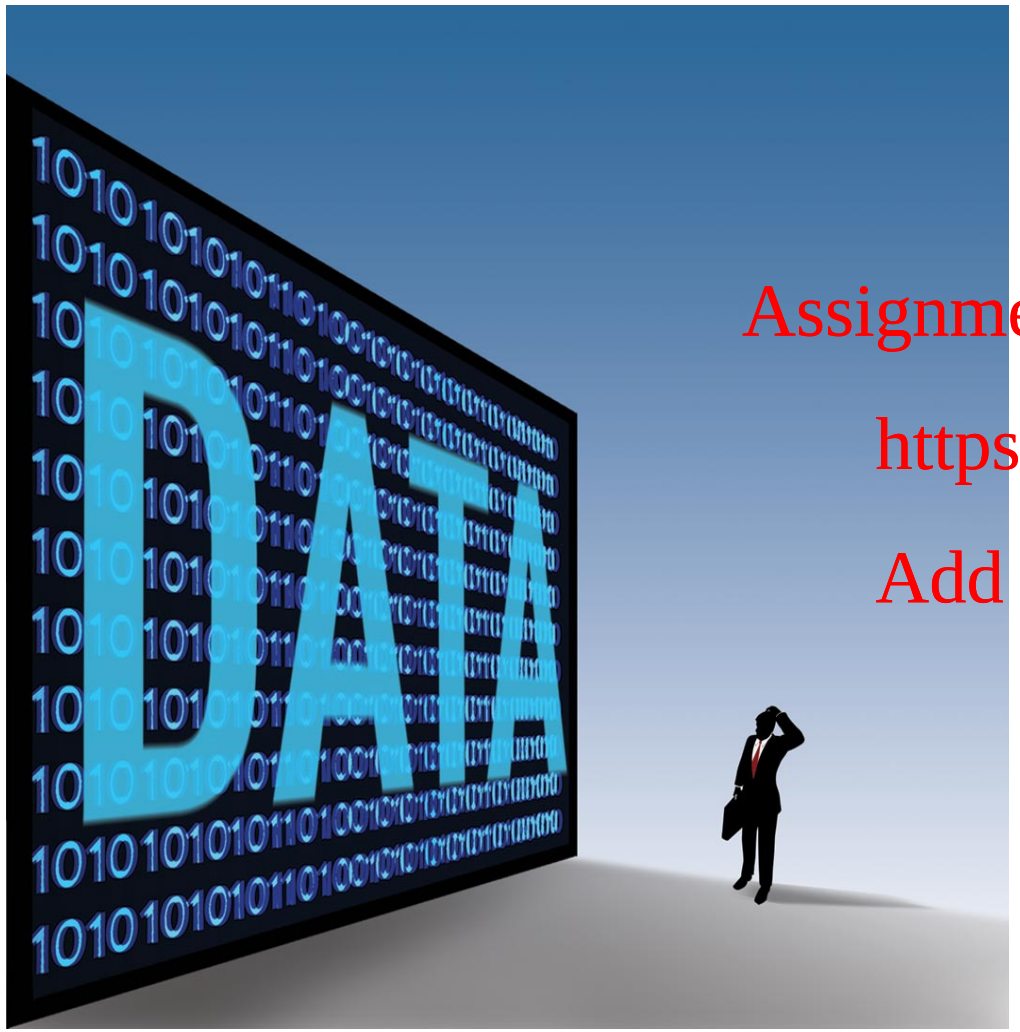
- Local search algorithm based on lower and upper bounds, ICDE 2018

Contributions:

- Three orders of magnitude faster than previous work.



Major Research Issues



New Computing Platform/Architecture

New Graph Analytics Models

New Processing Algorithms & Indexing Techniques

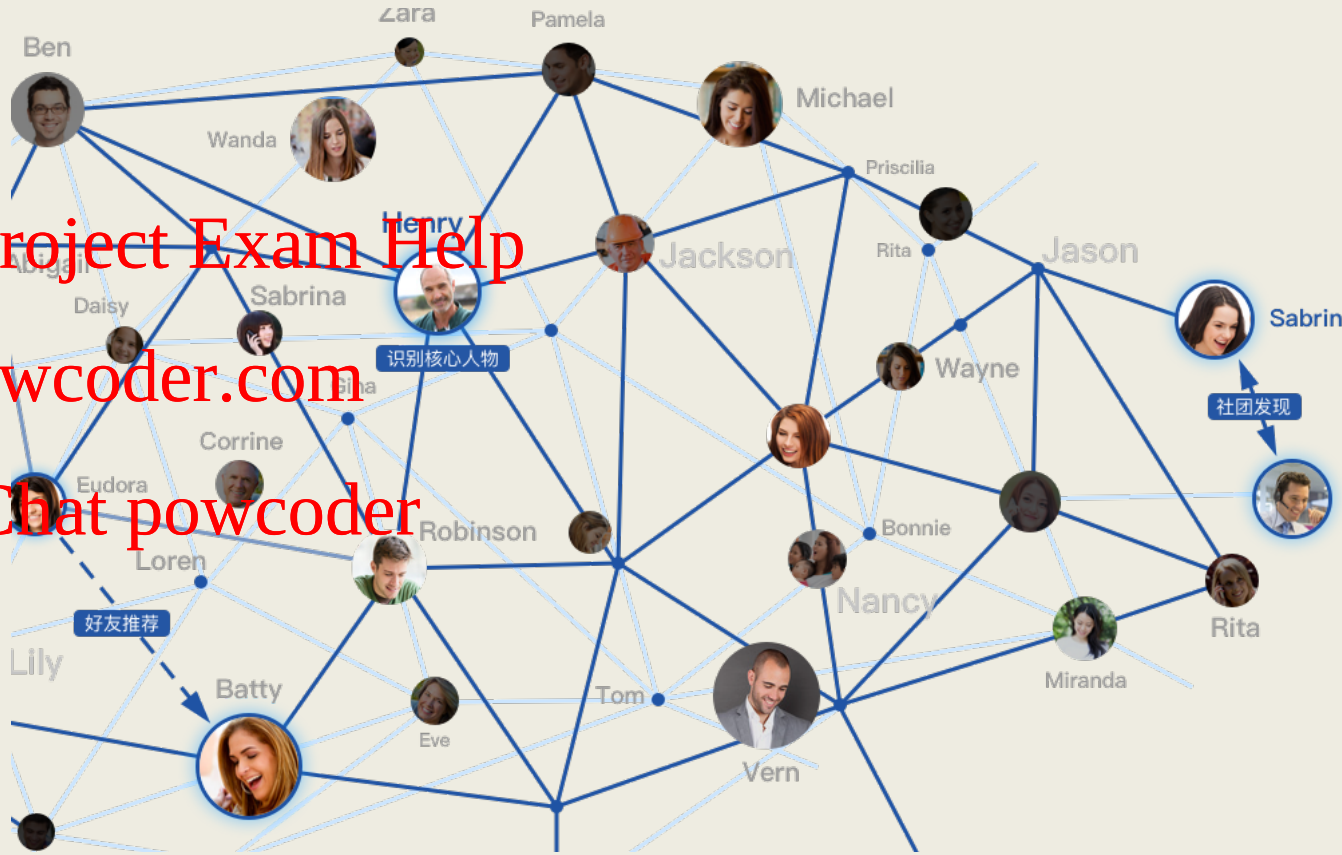
Graph Processing System

- 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



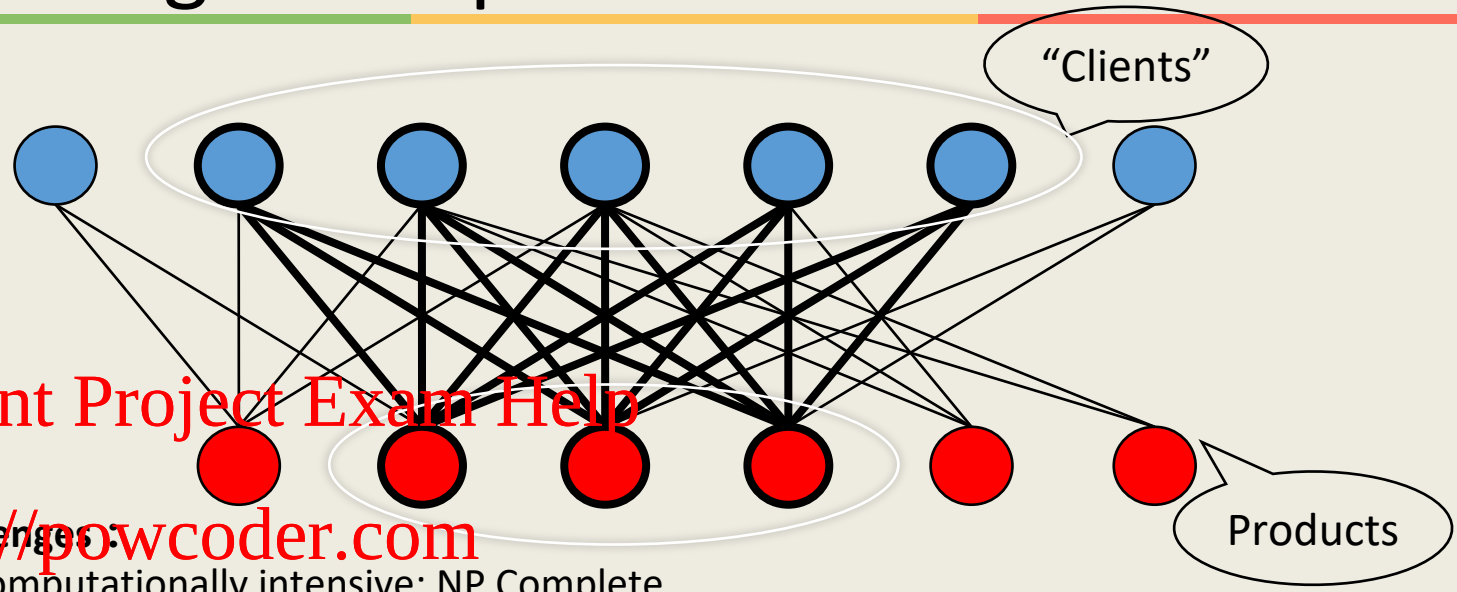
One of the applications on Huawei public cloud

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:

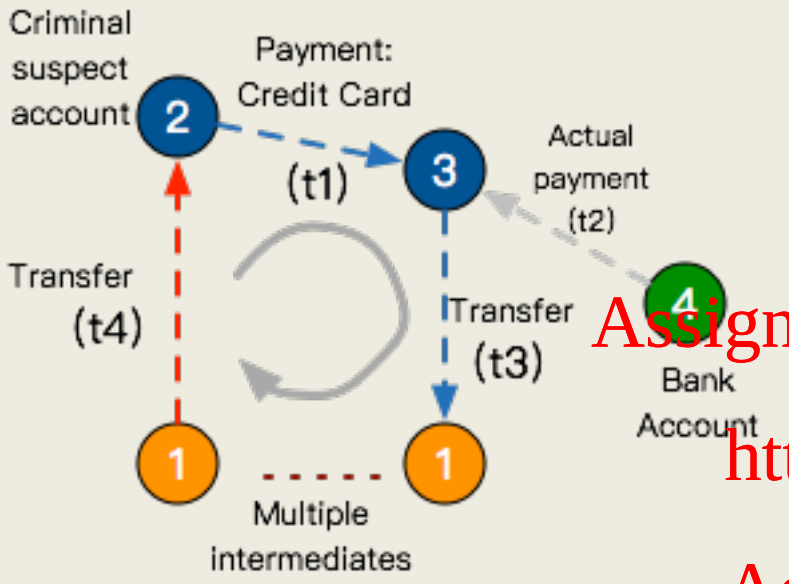
- Computationally intensive: NP Complete
- Big Data：100-million-scale users and products, billion-scale purchasing edge
- Dynamic Data: Frequent buying and selling on Taobao

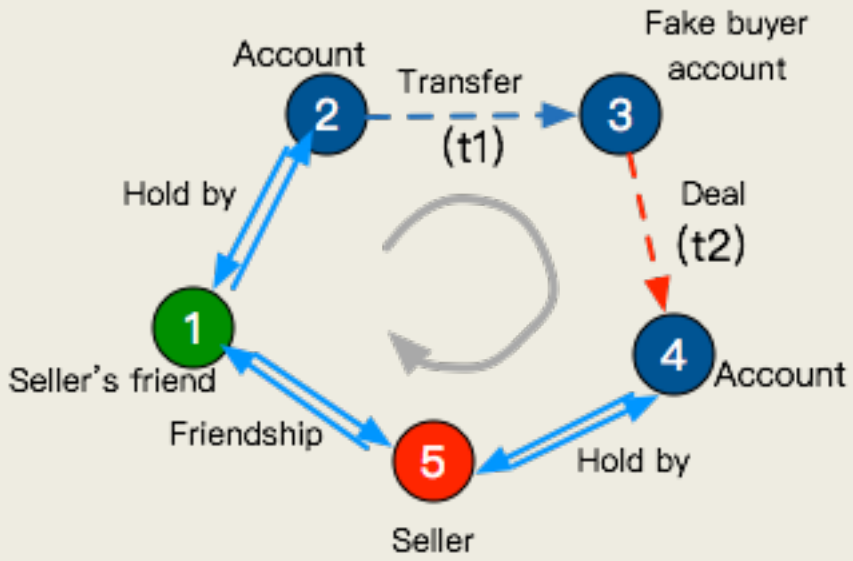
Solutions: In the client-product bi-graph, fake ordering is modelled as a bi-clique, which infers that there exists a potential fake ordering when a group of people simultaneously purchase a set of products.

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:

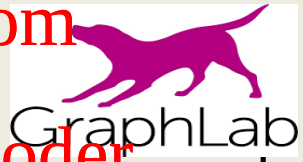
- Real-time cycle detection on each edge
- Light-weighted index structure (easy to maintain and update) and efficient query algorithm

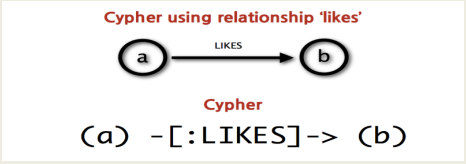
Real-data simulation：

- 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.

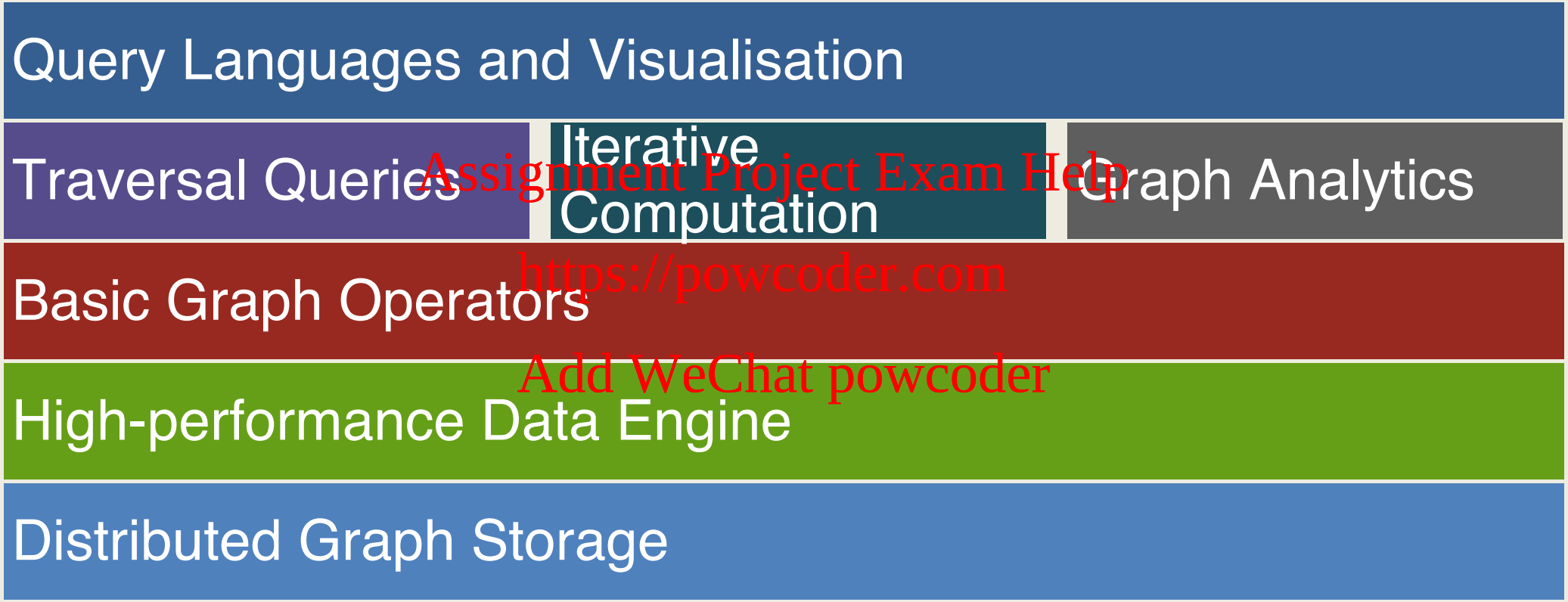




Graph System

- 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?