Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Efficient Subgraph Matching by Postponing Cartesian Products

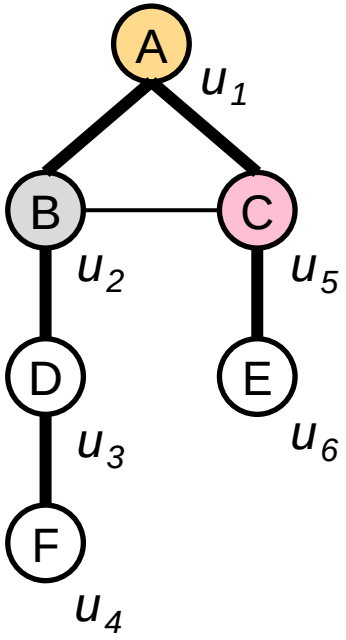
Matthias Barkowsky

Subgraph Isomorphism - Approaches

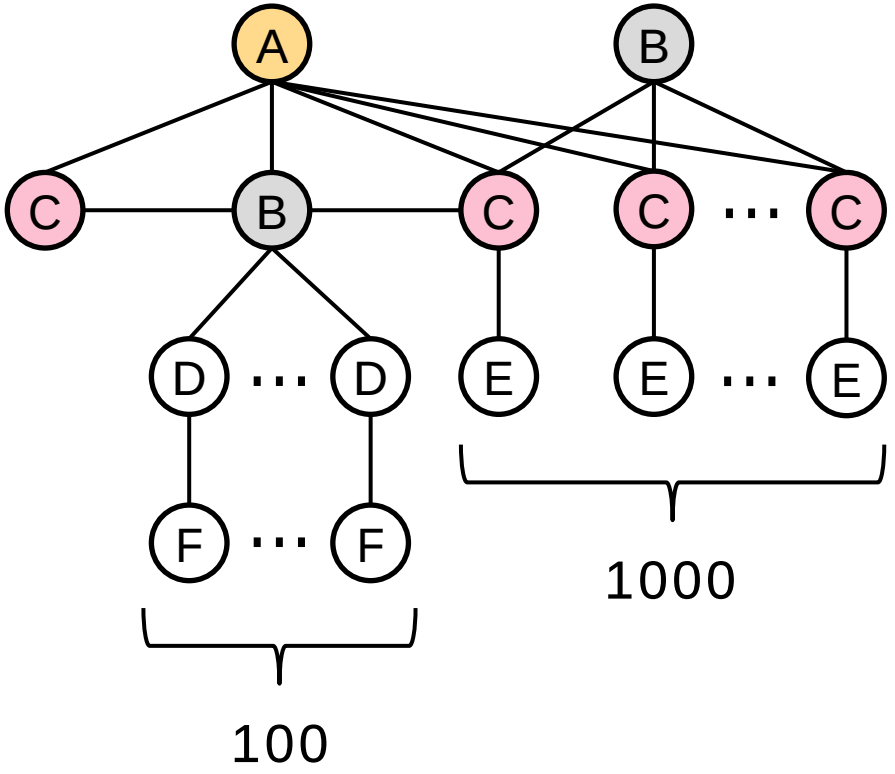
- search problem: find all embeddings of a query graph in a data graph
- NP-hard, but can often be solved efficiently in practice
- general approach: iteratively map query vertices to data vertices -> ordering very important
- but finding optimal order is NP-hard as well…

Query Vertex Ordering

Query Graph:



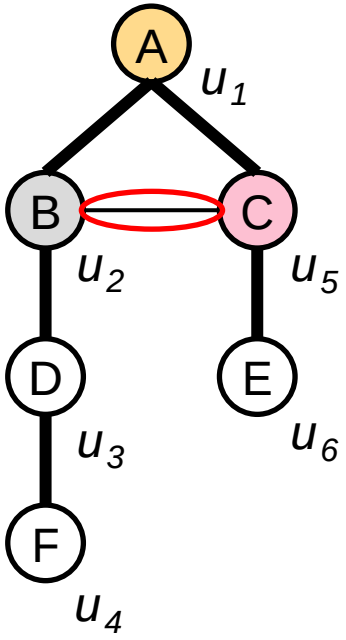
Data Graph:



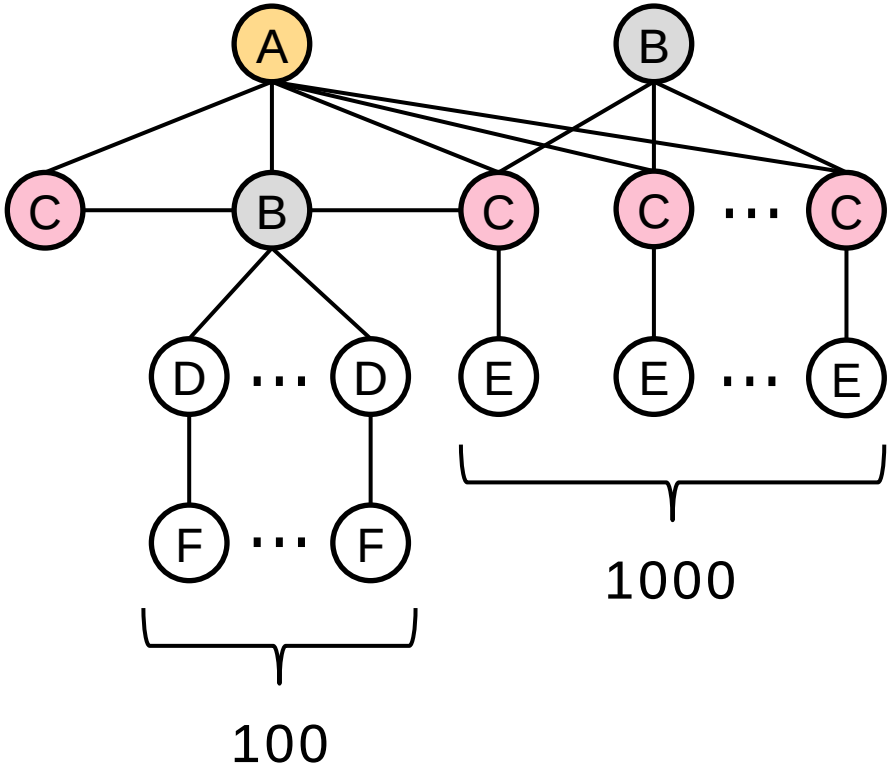
QuickSI: $(u_1, u_2, u_3, u_4, u_5, u_6)$

Query Vertex Ordering

Query Graph:



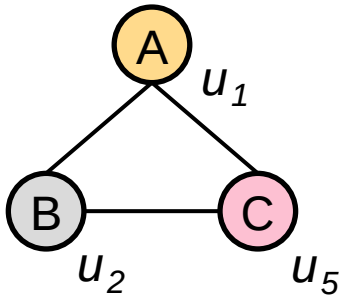
Data Graph:



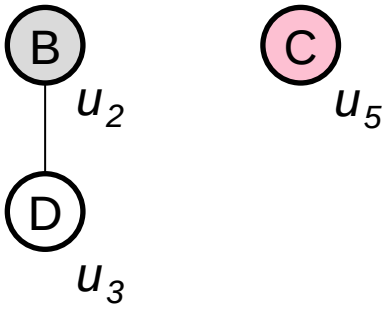
QuickSI: $(u_1, u_2, u_3, u_4, u_5, u_6)$

Core-Forest-Leaf Decomposition

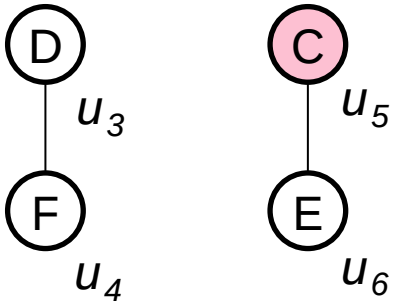
- **Core:** dense subgraph of Query Graph



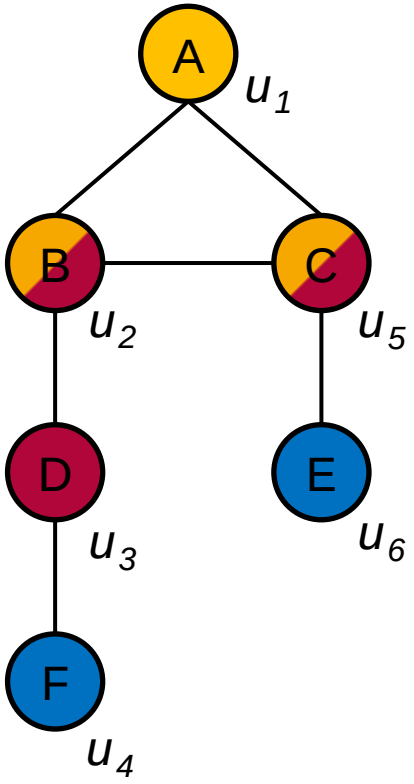
- **Forest:** number of trees connected to Core via one vertex



- **Leafs:** leafs of trees in the Forest structure



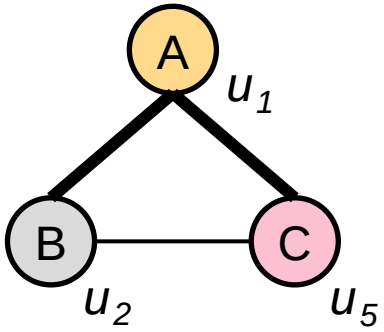
Core-Forest-Leaf Decomposition



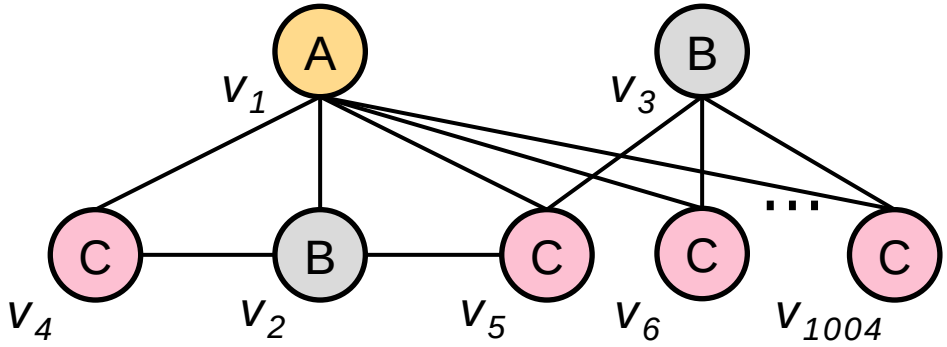
matching order: **Core** → **Forest** → **Leafs**

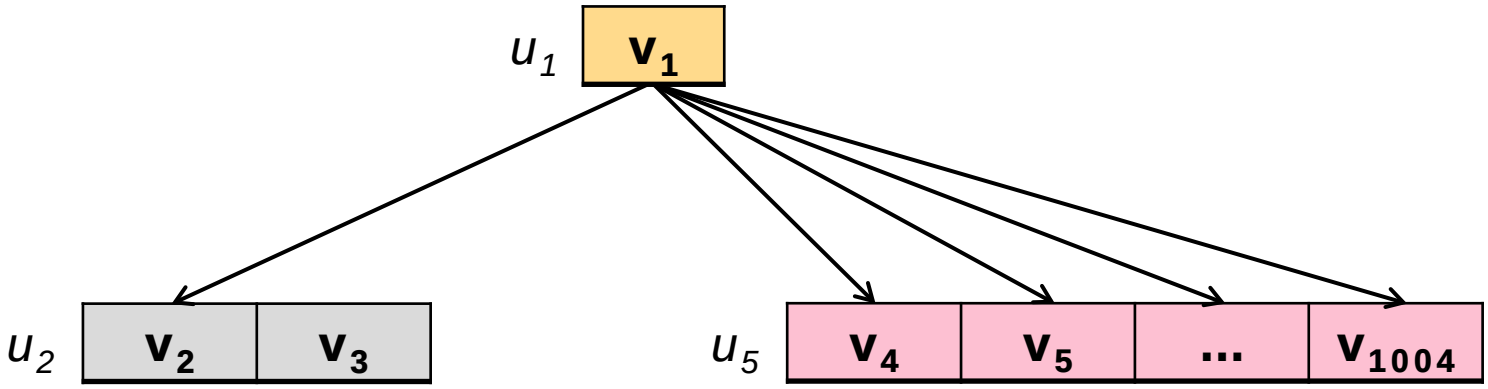
Compact Path Index – Idea

Query Graph:



Data Graph:



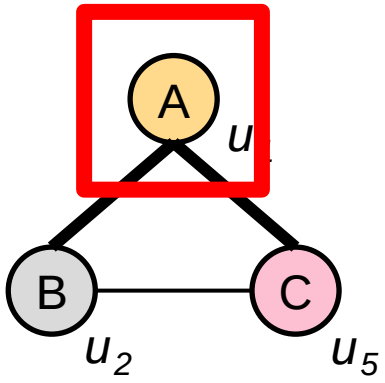


Compact Path Index - Construction

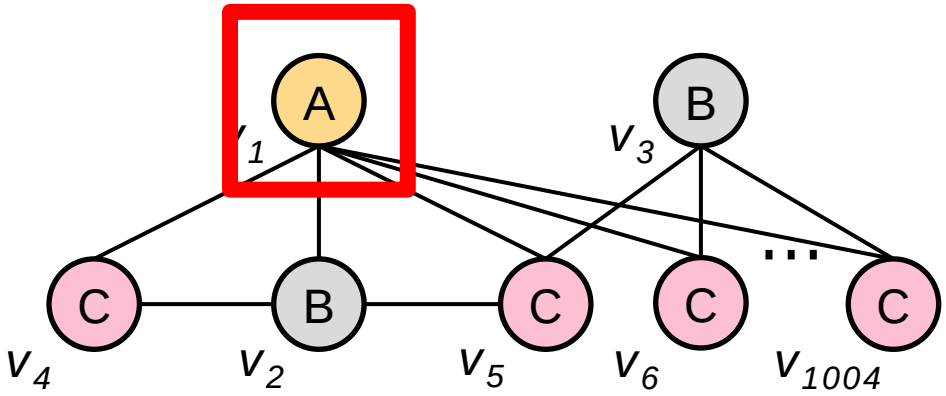
- criteria for data vertices in candidate set of a query vertex:
  1. matching label
  2. neighbor in candidate set of each already processed neighbor query vertex

Compact Path Index – Construction

Query Graph:

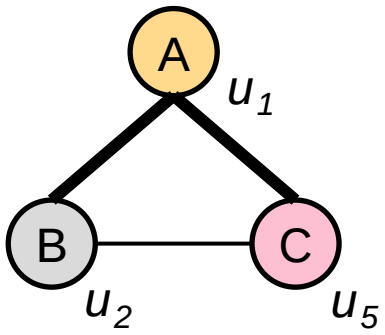


Data Graph:

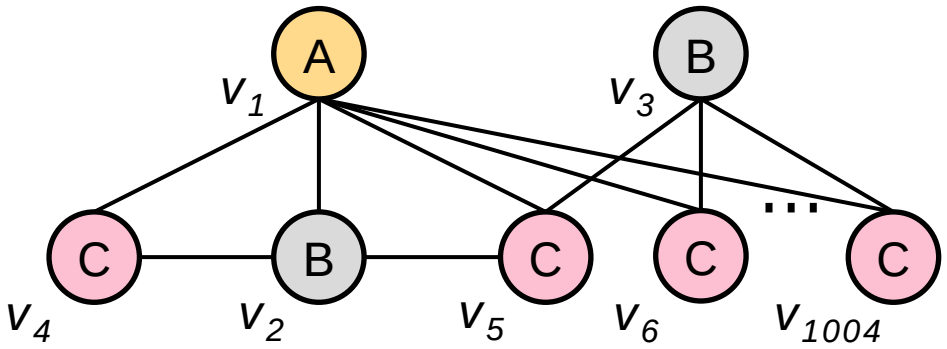


Compact Path Index – Construction

Query Graph:

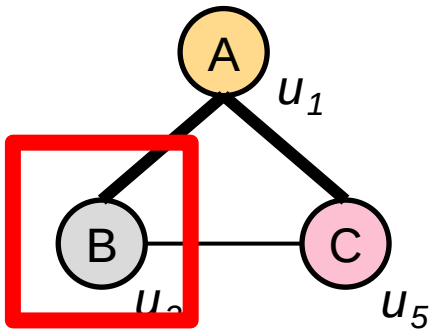


Data Graph:

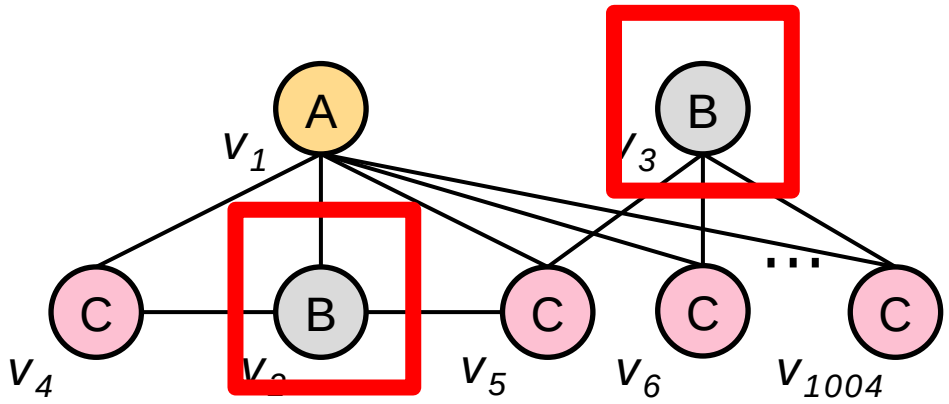


Compact Path Index – Construction

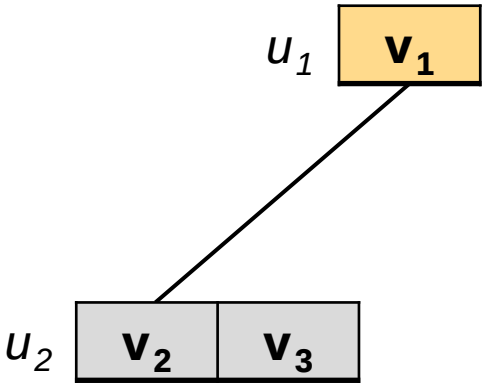
Query Graph:



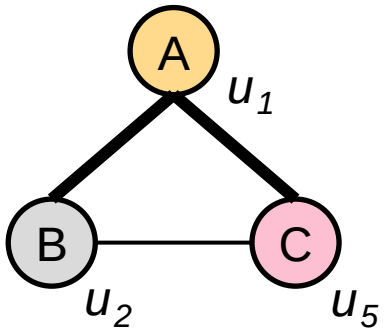
Data Graph:



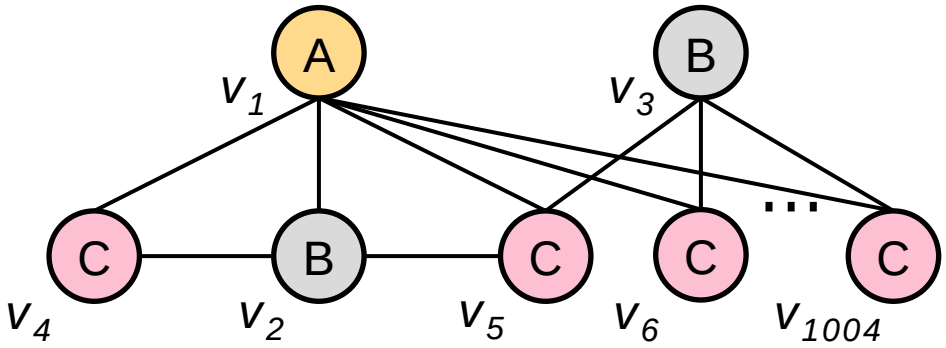
Compact Path Index – Construction



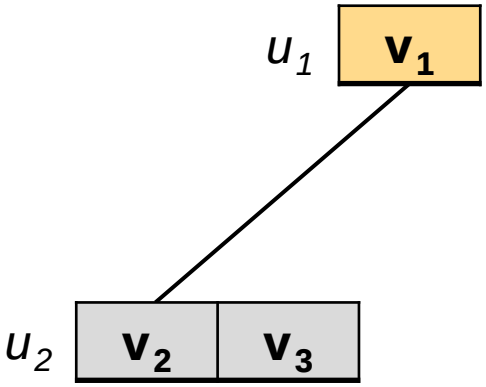
Query Graph:



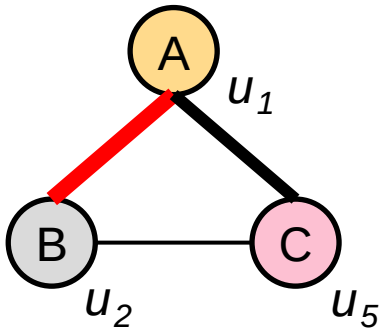
Data Graph:



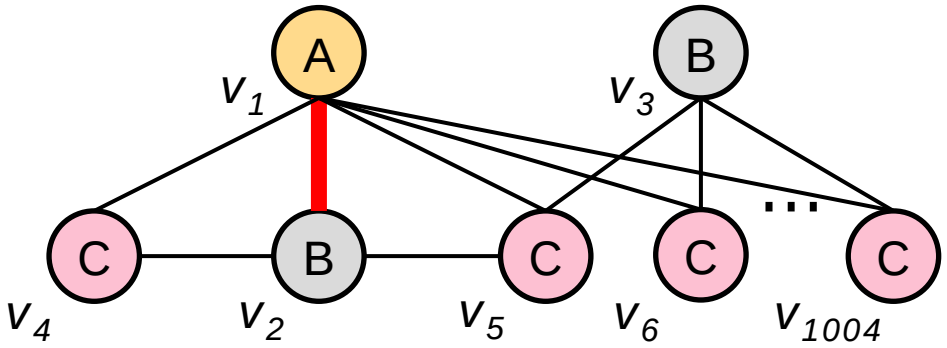
Compact Path Index – Construction



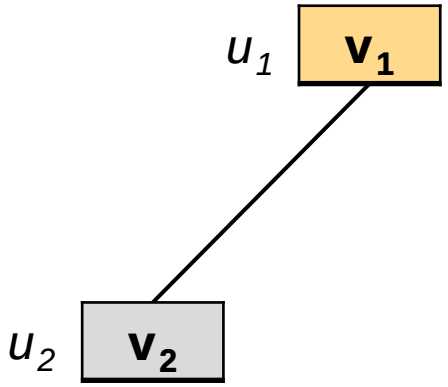
Query Graph:



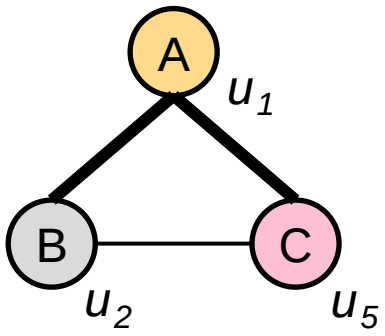
Data Graph:



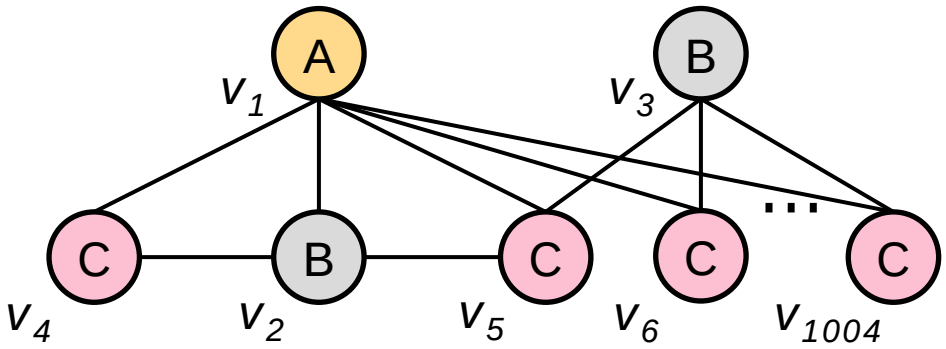
Compact Path Index – Construction



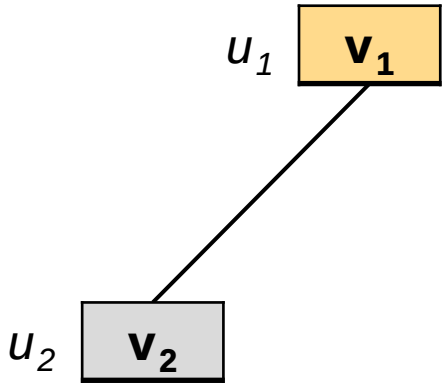
Query Graph:



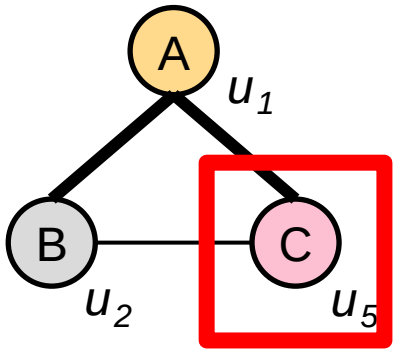
Data Graph:



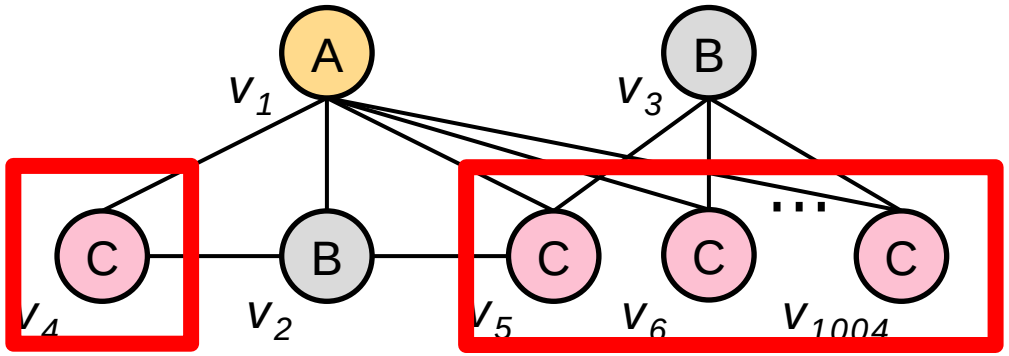
Compact Path Index – Construction



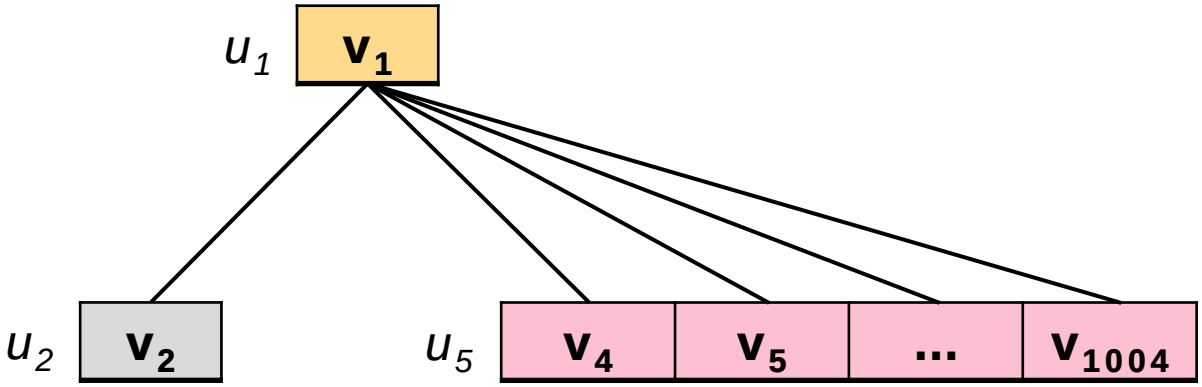
Query Graph:



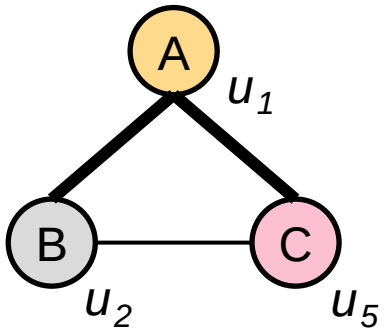
Data Graph:



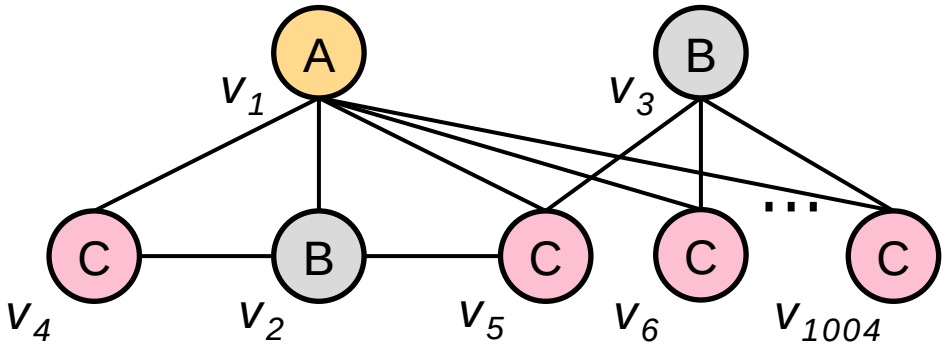
Compact Path Index – Construction



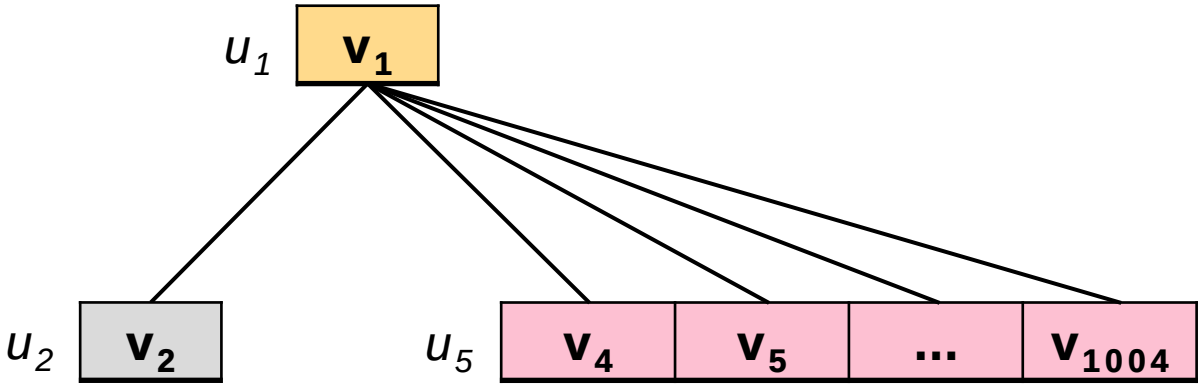
Query Graph:



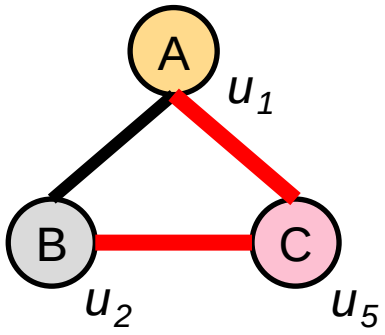
Data Graph:



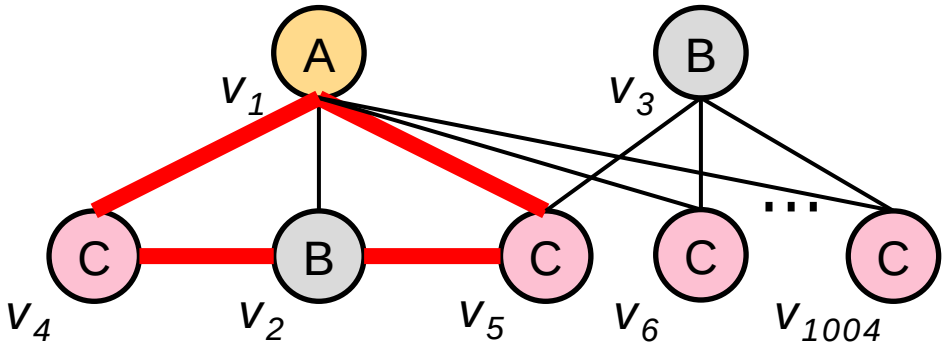
Compact Path Index – Construction



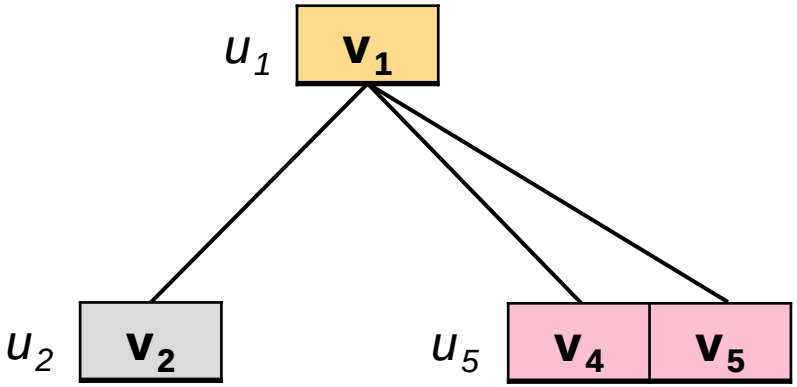
Query Graph:



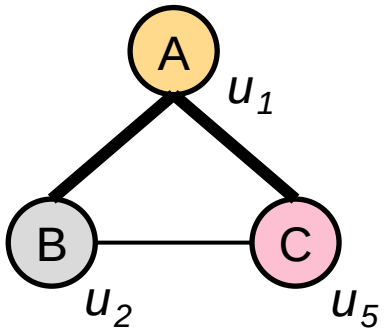
Data Graph:



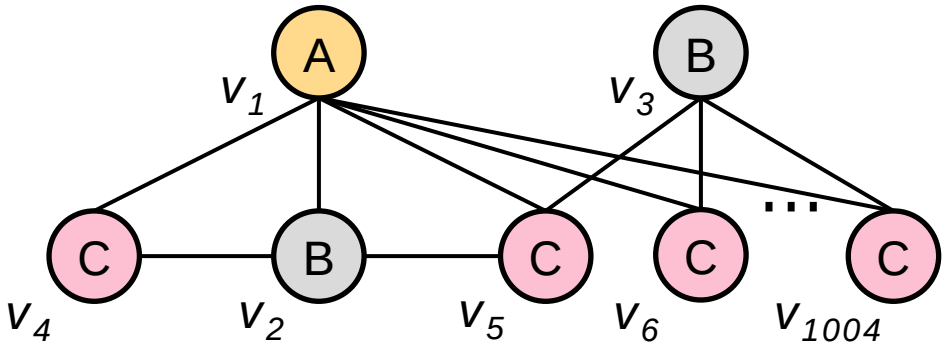
Compact Path Index – Construction



Query Graph:

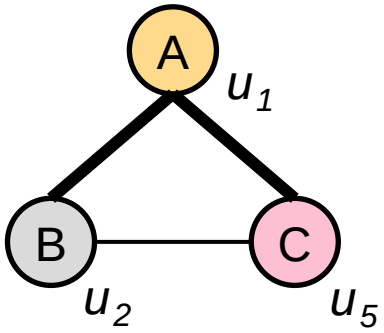


Data Graph:

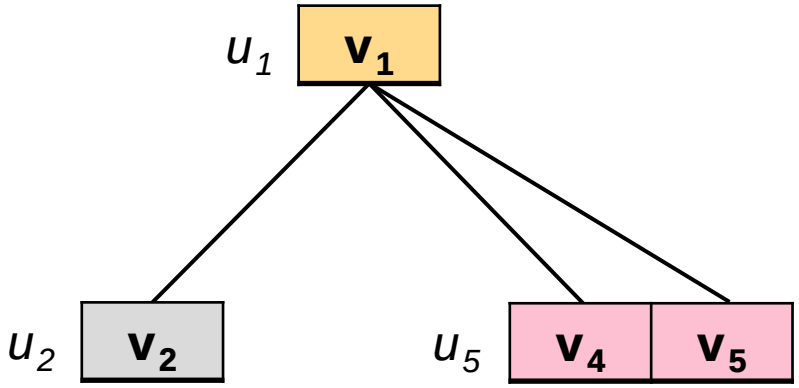


Compact Path Index - Application

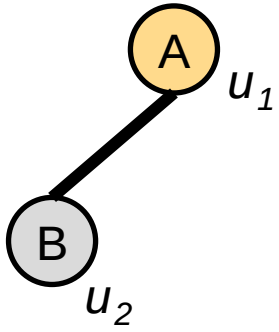
Query Graph:



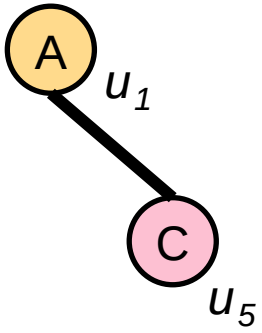
Index:



P_1: 1 embedding



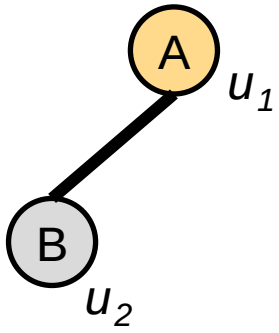
P_2: 2 embeddings



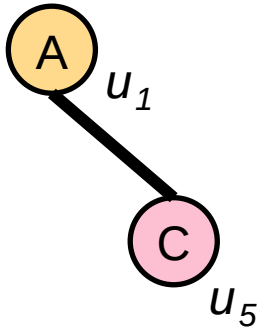
Compact Path Index - Application

- matching sequence: start with $P_1 = (u_1, u_2)$, add $P_2 = (u_1, u_5)$
- $\rightarrow$ **$S = (u_1, u_2, u_5)$**
- iteratively map query vertices to candidates in the index, check data graph for non-tree edges

P_1: 1 embedding

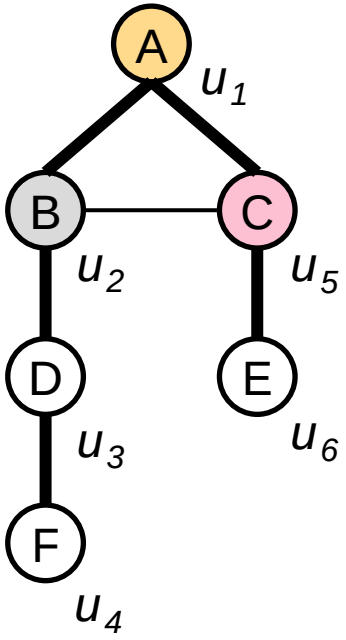


P_2: 2 embeddings

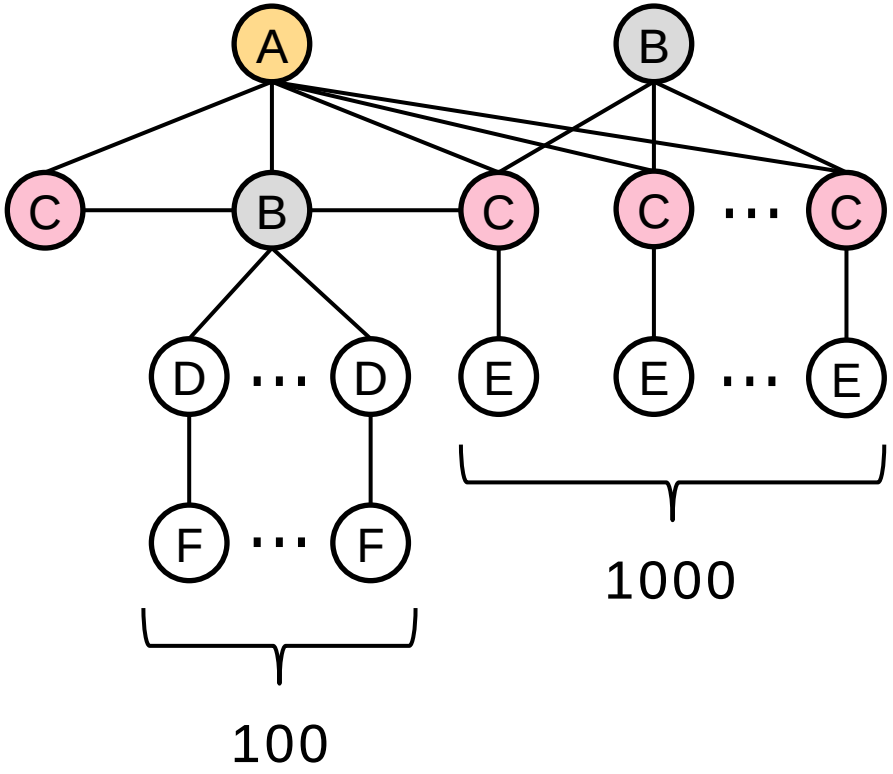


Improved Query Vertex Ordering

Query Graph:



Data Graph:

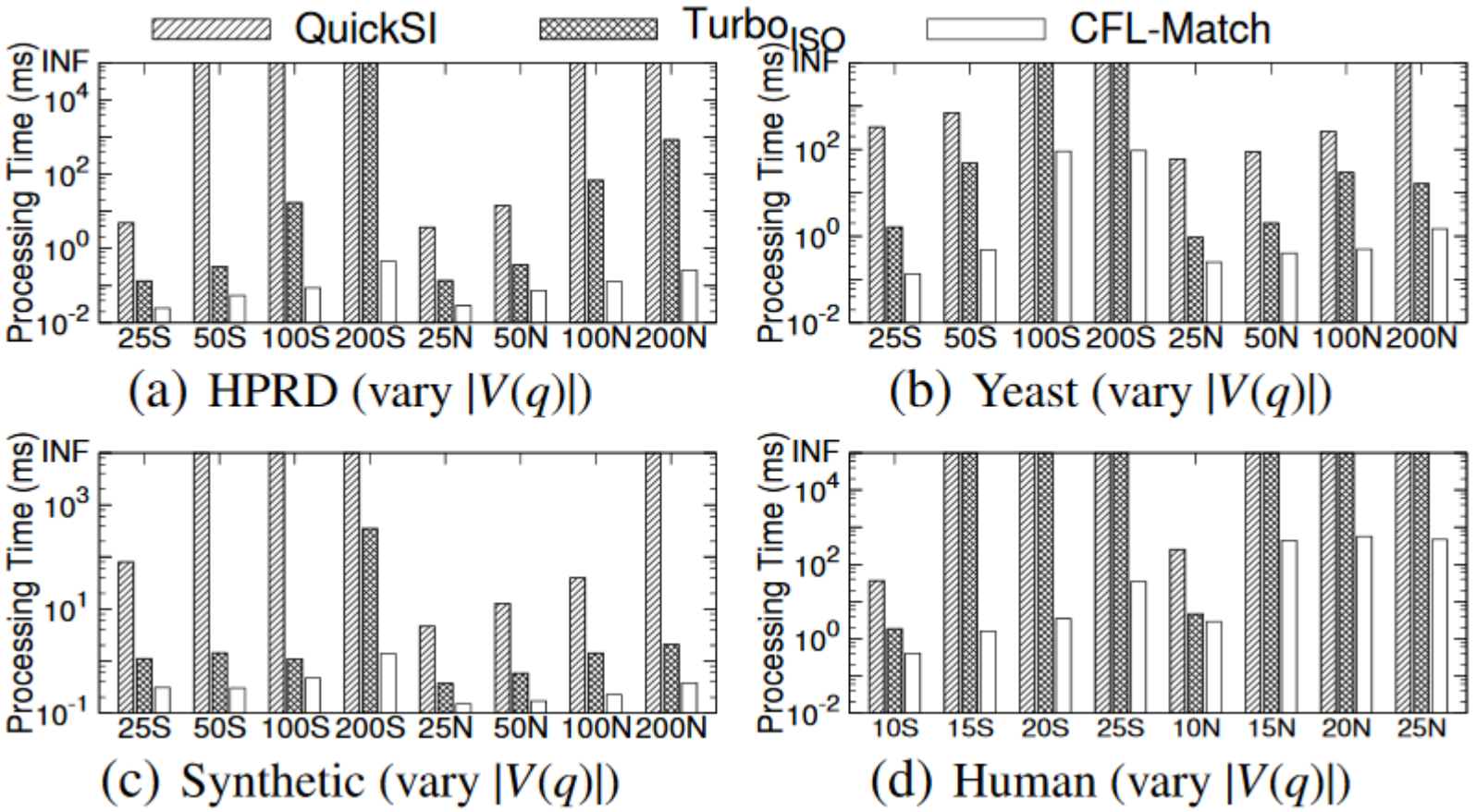


CFL: $(u_1, u_2, u_5, u_3, u_4, u_6)$

Evaluation

- compared to QuickSI and TurboISO
- several datasets of varying size (3 000 – 1 000 000 nodes)
  - real Protein Interaction Networks
  - synthetic graphs
- several generated queries of varying size (25 – 200 nodes)

Evaluation



- $\rightarrow$ faster by multiple orders of magnitude

References

- Bi, Fei, et al. "Efficient subgraph matching by postponing cartesian products." *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2016.

Supervised Random Walks
Predicting and Recommending Links in Social Networks

Ronny Pachel – Uni Potsdam

Feb 07th 2017

Contents

Link Prediction

- Network consisting of nodes and edges
- Between which two nodes will new edges (links) be created? [In a specific timeframe]

Challenges

Motivation

- Author working at Facebook - motivation obvious
- Every new Link (Friendships, etc.) is increasing Network value

Challenges of real Networks

- Extremely sparse
  - Facebook has nearly 2 billion users, how many friends do you have?
  - To predict that no new Links will be create is nearly 100% accurate!
- What network features have an influence on the creation of new links?

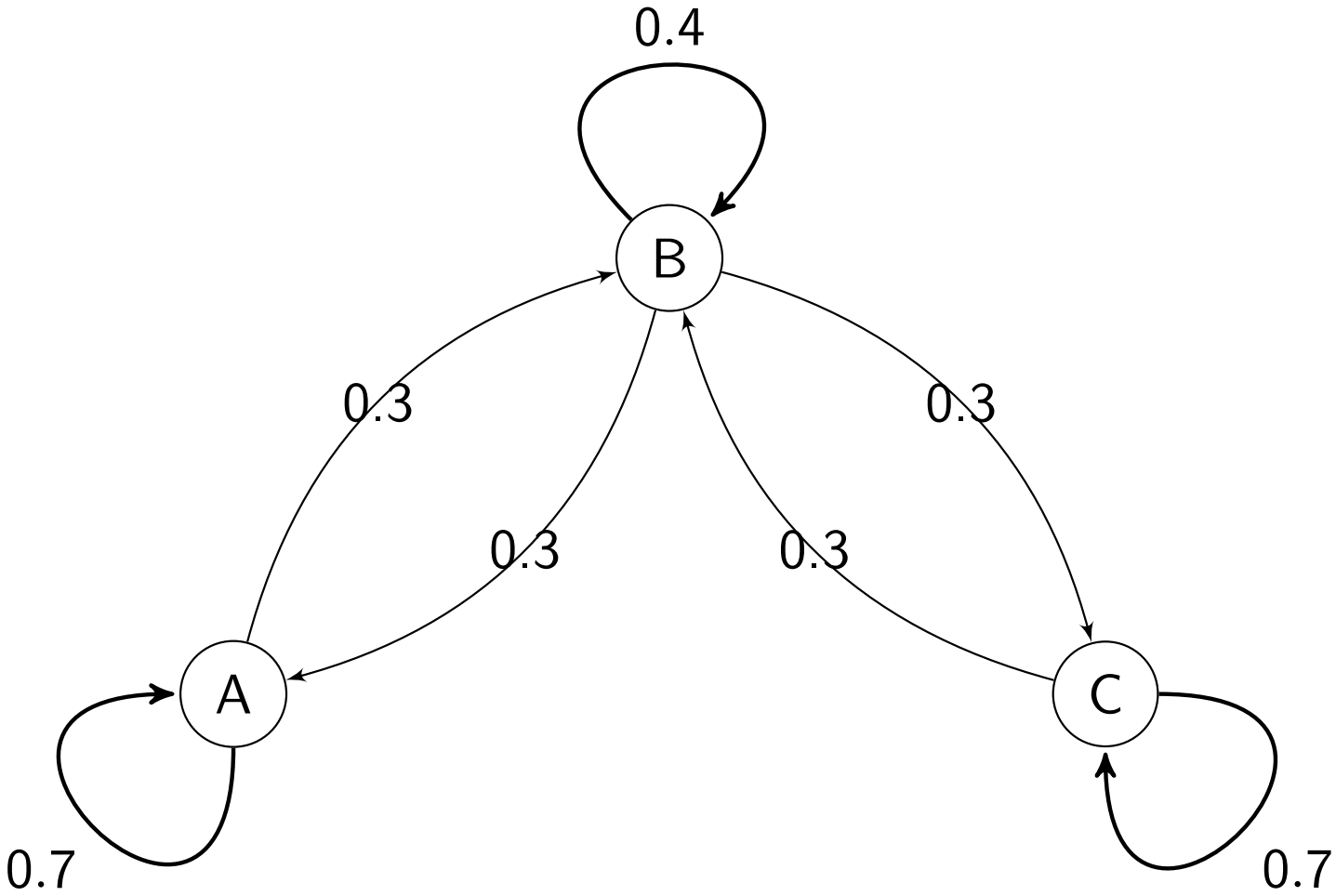
Present Work

- Method for link prediction and recommendation with supervised random walks
- Combines network structure and network properties
- PageRank-like random walk that visits given nodes (positive training examples) more often
- LEARN edge strength (transition probabilities)

Contents

Random Walks

Random Walks!



Transition Matrix

Probability after one step:

$$\begin{bmatrix} 0.7 & 0.3 & 0.0 \\ 0.3 & 0.4 & 0.3 \\ 0.0 & 0.3 & 0.7 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.3 \\ 0.0 \end{bmatrix}$$

Stationary Probability:

$$\begin{bmatrix} 0.7 & 0.3 & 0.0 \\ 0.3 & 0.4 & 0.3 \\ 0.0 & 0.3 & 0.7 \end{bmatrix} \cdot \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

Overview

- Given is a graph G
- Starting from a single node s with already existing Links and for which new links should be predicted
- Also required is a set of "positive" and "negative" training Nodes
- General approach - can also be used for link prediction, recommendation, link anomaly detection, missing link prediction, etc.

Approaches

We have two general approaches for this problem:

- As Classification task:
  - Each candidate node is either a created Link in the training data (positive example), or not (negative example)
  - But: Has high class imbalance, which makes learning very hard!
  - How to take the structure of the Network into account?
  - And: The choice and extraction of useful features is very hard! There are countless ways to describe node proximity alone!
- As Ranking Task:
  - Give Nodes with new created Links a higher Score
  - For Example PageRank and Random Walks with Restarts (RWR) are used
  - Stationary Distribution of a Random Walk gives a measure of "closeness"
  - Takes the structure into account, but not the properties

This Approach

- Combination of Both Approaches - Uses Network Structure and Node and Edge Properties
- Idea: Control a RWR via its transisiton properties to visit positive training nodes more often than the negative training nodes
- Trivial Approach by setting the transition properties manually would massively overfit
- Solution: Find transition properties depending on node and edge properties via a learning algorithm!

Problem Formulation

- Given
  - Directed Graph: G(V, E)
  - Start node: s
  - Set of candidate nodes: $C = c_i = D \cup L$
  - Set of destination nodes (positive training examples): $D = d_i$
  - Set of no-link nodes (negative training examples): $L = l_i$
  - Feature Vector for each Edge (u, v): ψ_{uv}
    Includes the Properties of both, the Nodes and the Edge!
- Find
  - Edge Strength: a_{uv}
    Used as transition probability
- Use
  - Learned Function in Training phase $f_w(\psi_{uv})$
  - w: vector of weights

Predicting Process

- We take a source node for which we want to predict future links
- For each Edge (u, v) in G we compute the strength $a_{uv} = f_w(\psi_u v)$
- Then a RWR is run and stationary distribution assigns every node u a probability value p_u
- Nodes are ordered by p_u values - highest values are predicted as destinations for future links
- $\rightarrow$ Weights must be learned!

Optimization Problem

- $\min_w F(w) = ||w||^2 + \lambda \sum_{d \in D, l \in L} h(p_l - p_d)$
- Minimization Problem!
- p_d should be higher than p_l
- h(): loss function - measures how "bad" the current prediction is in form of a penalty
- $||w||^2$: second norm of the weight vector - smaller values are wanted!
- λ: trade-off between the loss and the weights vector
  A high λ means that the loss is more important than small weights, used to prevent overfitting to the data
- But how do we come from the input (w) to the measurable output (p)?

Optimization Problem

- Set a start value of w
- For each edge (u,v) calculate the edge transition $a_{uv} = f(\psi_{uv})$
- Build the random walk transition probability matrix Q
- Incorporate the restart probabability α into Q
- Find the vector p of the node probabilities (also known as Personalized PageRank) via a eigenvalue problem (or stationary distribution)

- This process can be written as a single formula
- The derivation of this formula is used for a iterative calculation to converge to the p vector (gradient descent method)
- "If I change the weight values like this, is the final result (F(w)) better (smaller)?"
- Note: This problem is not generally convex - no global minimum is guaranteed!

Contents

Network

- 4 physics co-authorship networks,
  complete Facebook network of Iceland
  - Iceland has a very high Facebook penetration
  - Relatively few Edges link to users from other countries
  - Timeframe from Nov 1st 2009 (t_1) to Jun 13th 2010 (t_2)
  - More than 174.000 users, about 55% of Icelands population
  - On average 168 friends at t_1 and 26 new friends until t_2
- Focus on predicting Links to Nodes that are 2 hops away
  - Empirically most new Links close a triangle

Preprocessing

- Only "active" Nodes with more than 20 Links
- Candidate Nodes that have less than 4 common friends have been pruned
  Empirically shown that it is very unlikely that they would become friends
- Nodes with too many links have been pruned
  Even after just 2 hops some nodes can have several millions neighbours
- These measure speed up the computation but do not influence the prediction performance much

Learner Setup

- From these users, 200 have been randomly selected, 100 as training set to learn the algorithm, and 100 as test set to measure the performance
- These Properties have been used to generate 7 Features:
  - Edge Age
  - Edge Initiator
  - Communication and Observation Features
  - Number of Common Friends
- Two performance metrics have been used:
  - AUC - Area under ROC curve
  - Prec@20 - Precision at the Top 20

Contents

Algorithm

We need to choose 4 aspects of the algorithm:

- Loss Function
- Edge Strength Function
- Choice of RWR Restart Parameter α
- Choice of Regularization Parameter λ

Loss Function

- Must be continuous and differentiable to optimize over them
- candidates:
  - Squared loss with margin b:
    $h(x) = max\{x + b, 0\}^2$
  - Huber Loss with margin b and window z ¿ b:
    $$h(x) = \begin{cases} 0 & \text{if } x \leq -b, \\ (x+b)^2/(2z) & \text{if } -b < x \leq z - b, \\ (x+b) - z/2 & \text{if } x > z - b \end{cases}$$
  - Wilcoxon-Mann-Whitney (WMW) loss with width b:
    $h(x) = \frac{1}{1+exp(-x/b)}$
- Squared loss and Huber loss can be computed easier
- WMW has a slightly better performance
- But: Only WMW increases the AUC and Prec@20!

Strength Function

- $f_w(\psi_u v)$ must be non-negative and differentiable
- Two candidates, both are basing on the inner product of the weight vector and the feature vector
- candidates:
  - Exponential edge strength: $a_{uv} = exp(\psi_{uv} \cdot w)$
  - Logistic edge strength: $a_{uv} = (1 + exp(-\psi \cdot w))^{-1}$
- Experiments show no significant difference in performance
- But the exponential function has a potential problem with underflowing or overflowing double precision numbers

Choice of α

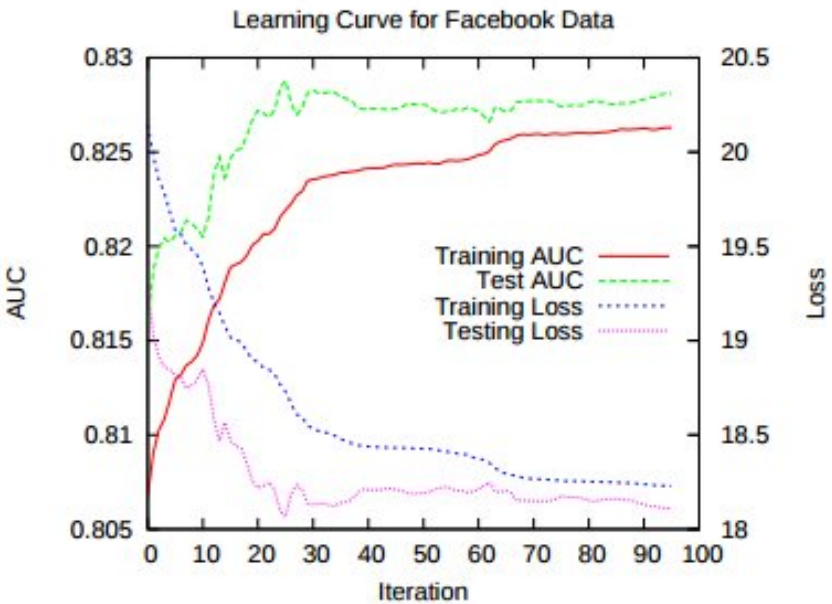
- α controls "how far a walk wanders before it's reset"
- $\alpha = 0$: PageRank of a node is it's degree
- $\alpha = 1$: After one hop the walk is reset
- Big values are short and local walks, low values are long
- Have a influence in unweighted graphs, but with edge strengths assigned the influence diminishes
- Empirically $0.3 < \alpha < 0.7$ has same performance

Choice of λ

- Overfitting is not an issue
- And number of features is small
- We chose a λ of 1

Results

- SRW with all weights $w = 0$ as starting point
- Compared to two unsupervised methods (RWR, node degree) and two supervised machine learning methods (decision trees and logistic regression)



Learning Method	AUC	Prec@20
Random Walk with Restart	0.81725	6.80
Degree	0.58535	3.25
DT: Node features	0.59248	2.38
DT: Path features	0.62836	2.46
DT: All features	0.72986	5.34
LR: Node features	0.54134	1.38
LR: Path features	0.51418	0.74
LR: All features	0.81681	7.52
SRW: one edge type	**0.82502**	6.87
SRW: multiple edge types	**0.82799**	**7.57**

Table 3: Results for the Facebook dataset.

Contents

Conclusions

- Supervised Random Walks - a new learning algorithm for link prediction and recommendation
- Combines two different approaches that see the problem either as classification task or as ranking task
- Demonstrated on real Facebook data
- Predictions show large improvements over RWR and easier than classical machine learning approaches
- SRW can be adopted to many other problems

Social Influence Computation and Maximization in Signed Networks with Competing Cascades

David Schumann

Hasso Plattner Institute

Graph Mining
Tuesday 7th February, 2017

What is this paper about?

Domain:

- Network of susceptible nodes.
- Trust and Distrust Relationships
- Diffusion Model: Independent Cascade

Problem:

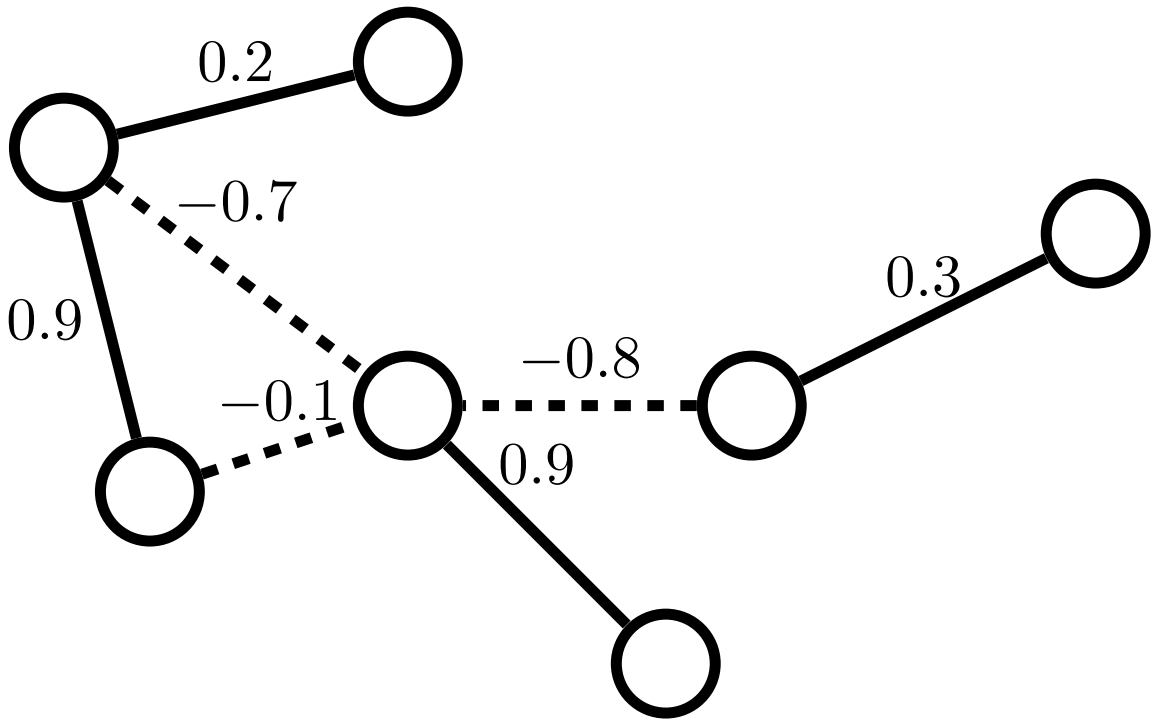
- Starting seed-set of size m with the highest expectation of nodes colored red

Signed **N**etwork **I**nfluence **Max**imization **(SiNiMax)** Problem $\rightarrow$ NP-hard

What is this paper about?

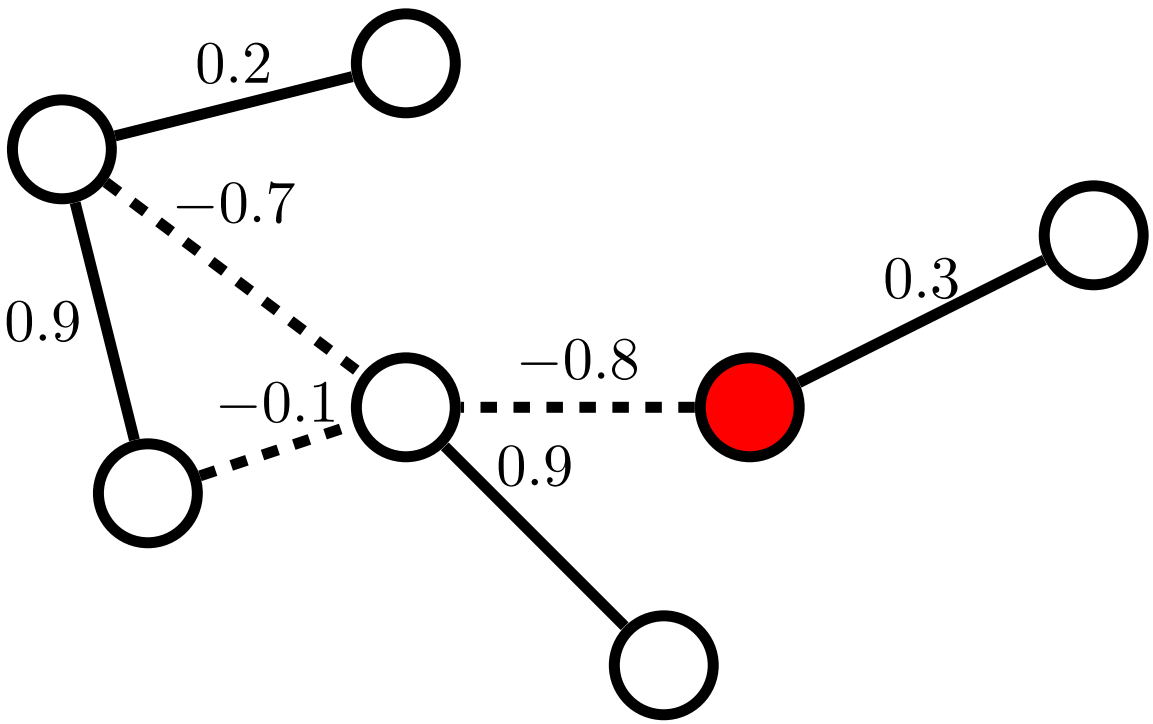
Signed Networks with competing Cascades

- Weighted, directed and signed graph $G = (V, E, W)$
- Weight Matrix W:
  - sign represents **trust** or **distrust** relationships.
  - Absolute value denotes strength of influence.



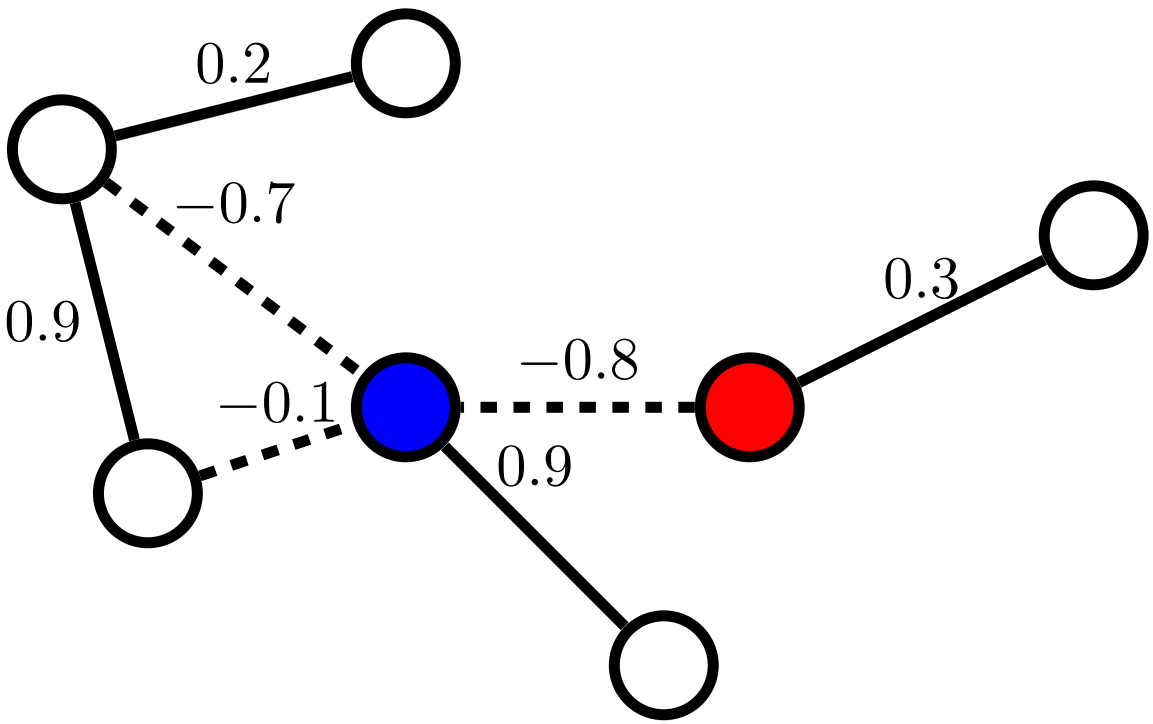
Signed Networks with competing Cascades

- Two influnce diffusion processes spreading in discrete time steps
- Using the standard Independent Cascade Model (ICM)
  - Infected node v gets one chance to infect neighbour w with probability $p_{v,w}$
  - On success w becomes infected at step $t + 1$.
  - Infected nodes never return to susceptible state.



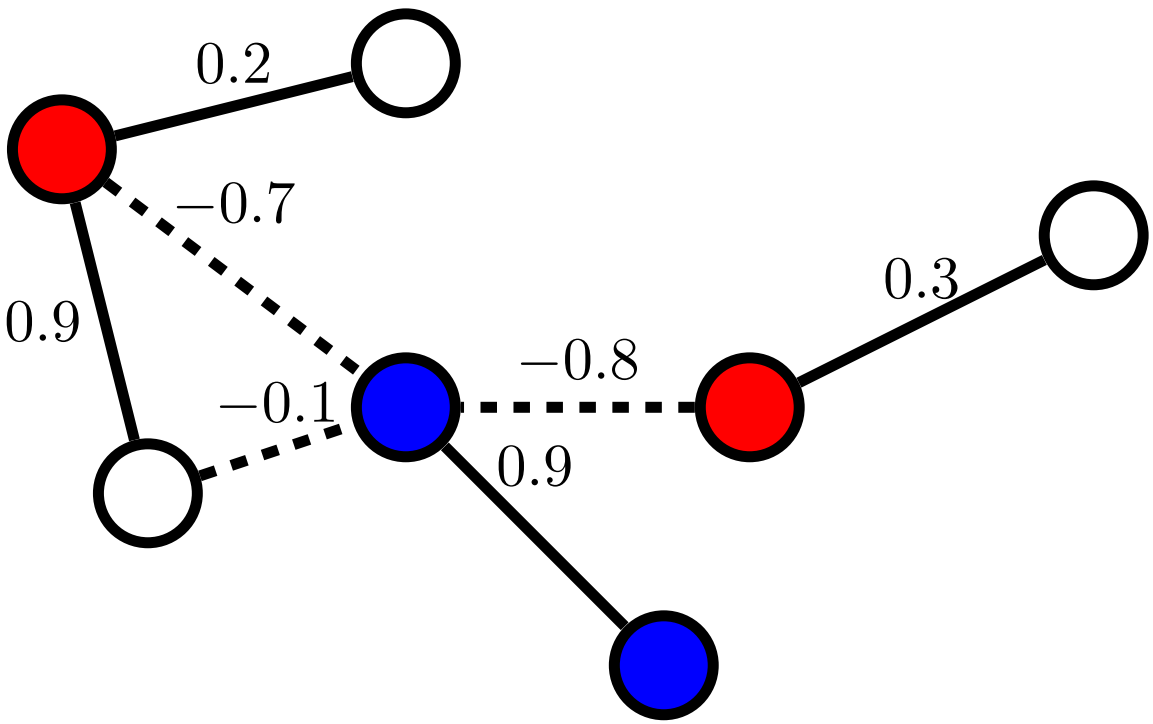
Signed Networks with competing Cascades

- Two influnce diffusion processes spreading in discrete time steps
- Using the standard Independent Cascade Model (ICM)
  - Infected node v gets one chance to infect neighbour w with probability $p_{v,w}$
  - On success w becomes infected at step $t + 1$.
  - Infected nodes never return to susceptible state.



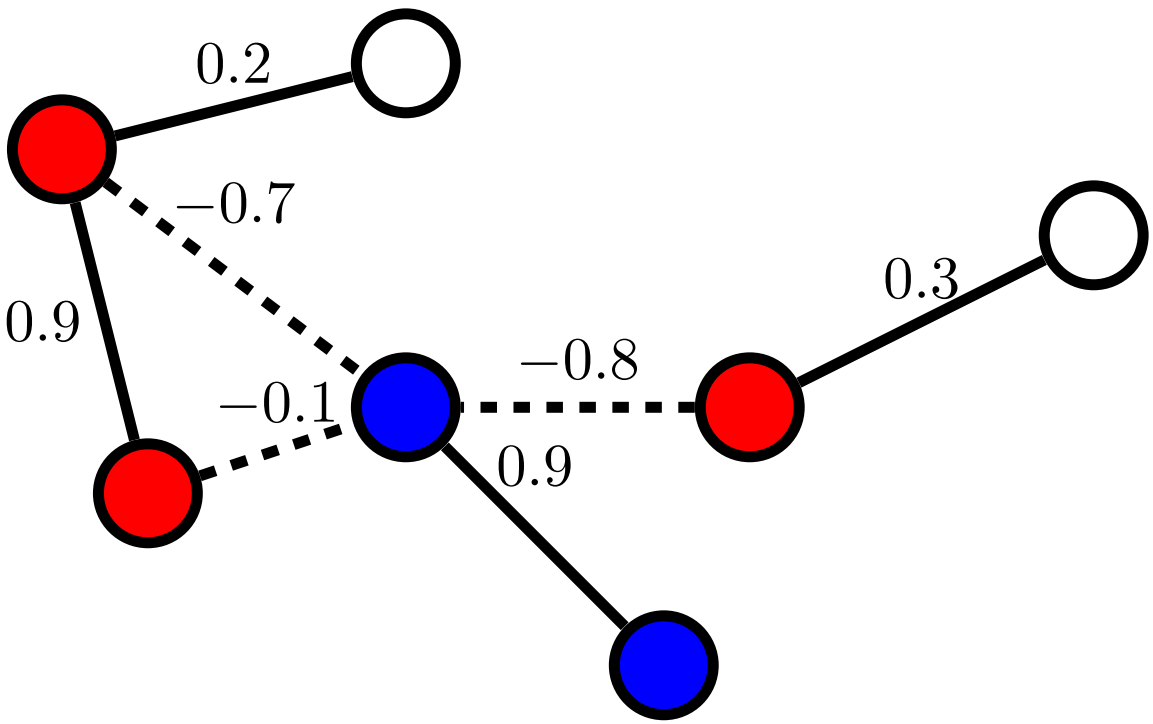
Signed Networks with competing Cascades

- Two influnce diffusion processes spreading in discrete time steps
- Using the standard Independent Cascade Model (ICM)
  - Infected node v gets one chance to infect neighbour w with probability $p_{v,w}$
  - On success w becomes infected at step $t + 1$.
  - Infected nodes never return to susceptible state.



Signed Networks with competing Cascades

- Two influnce diffusion processes spreading in discrete time steps
- Using the standard Independent Cascade Model (ICM)
  - Infected node v gets one chance to infect neighbour w with probability $p_{v,w}$
  - On success w becomes infected at step $t + 1$.
  - Infected nodes never return to susceptible state.



Help of the Unified Model

- Reduces computational cost for commonly used diffusion models.
- Using: Individual and Collective Influence.
- For ICM:

$$p_{(v,u)}(t) = 0$$

$$r_{u,t} = 1 - P(\text{no infected neighbor succeeds})$$

$$r_{u,t} = 1 - \prod_{v \in N_i(u)} (1 - p(B_{v,t-1}))$$

$$\dots$$

Unified Model of Competing Cascades

- $r^{+}_{u,t}$ = Collective influence on u to be red.
- $r^{-}_{u,t}$ = Collective influence on u to be blue.
- $B^{+}_{u,t}$ = Probability of u being red at or before time t.
- $B^{-}_{u,t}$ = Probability of u being blue at or before time t.
- $B^{+}_{u,t}(k,+)$ = Probability of node u being colored red at time t if node k is infected with red at time $t-1$.

Online Seed-set Selection using the Unified Model for Signed Networks: $OSSUM\pm$

Algorithm 1 pseudocode for $OSSUM\pm$

Input: G, m

Output: S

1: $S \leftarrow \emptyset$
2: **for** $t = 1$ to m **do**
3: $\quad (k, c) = arg\,max_{(k,c)} \sum_u (B^+_{u,t}(k, c) - B^+_{u,t})$
4: $\quad S \leftarrow S \cup (k, c)$
5: **end for**

Runtime Complexity: $\mathcal{O}(m(|V| + |E|))$

Experiment Results

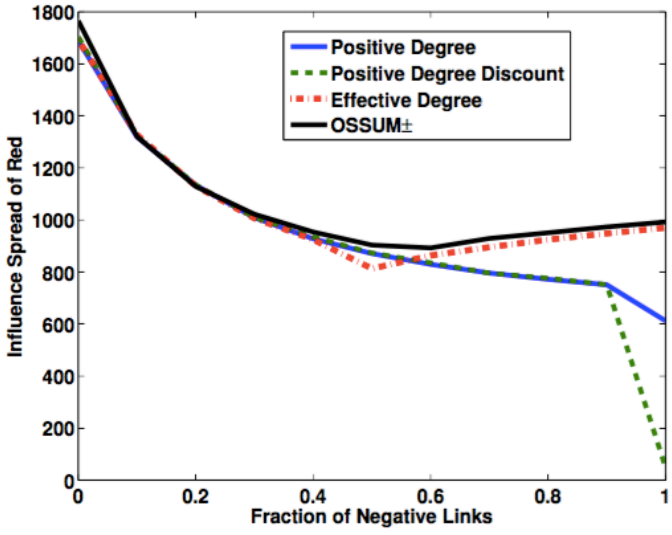
Comparison of $OSSUM\pm$ to three other heuristics:

- Positive Degree: Nodes with highest positive degree are selected
- Positive Degree Discount: Variation of Positive Degree
- Effective Degree: Nodes with highest effective degree are selected

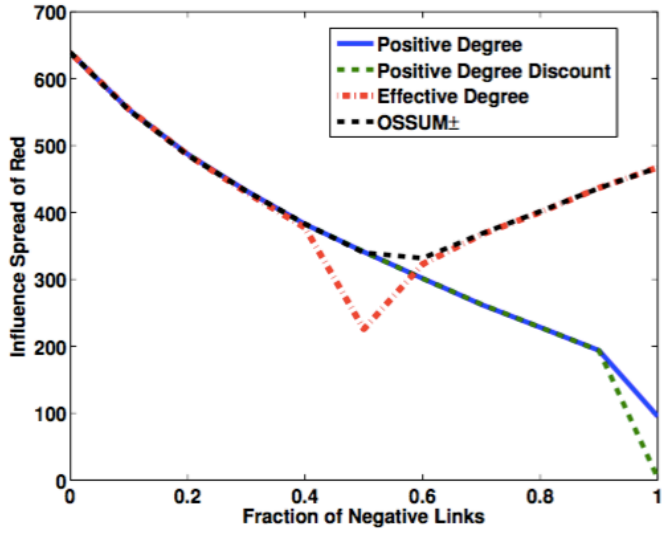
Data Set:

	Epinions	Slashdot
# nodes	131k	82k
# edges	840k	550k
# positive edges	710k	425k
# negative edges	130k	125k

Experiment Results



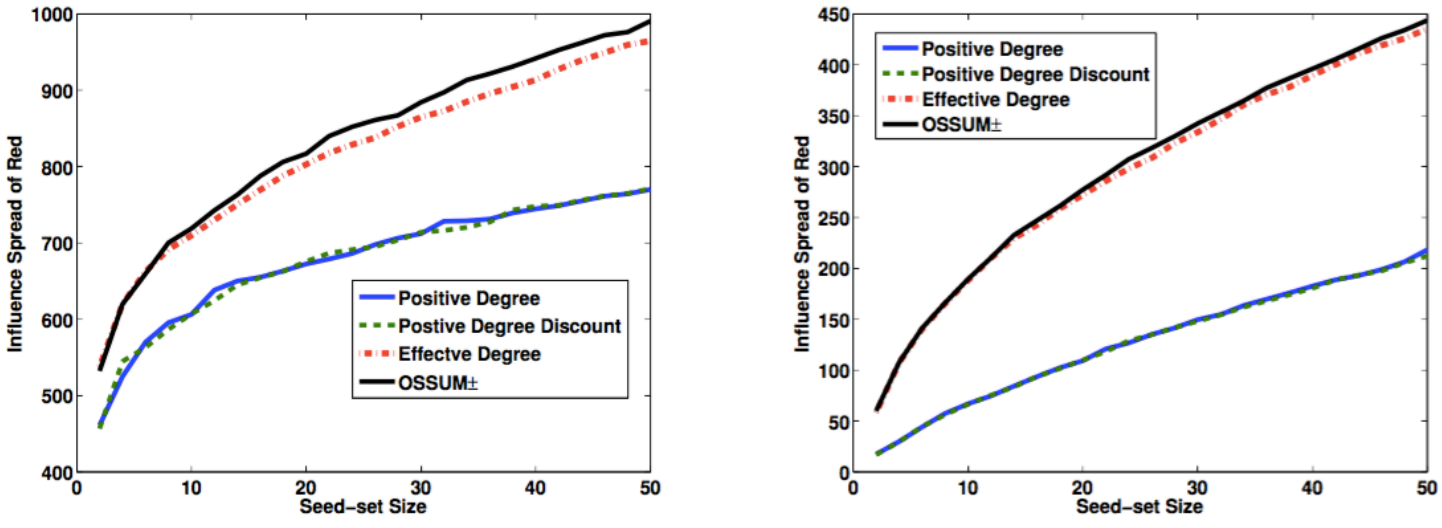
(a) Epinions



(b) Slashdot

Fig. 3. Influence spread achieved by the heuristics on graphs with varying fraction of negative links.

Experiment Results



(a) Epinions (b) Slashdot

Fig. 5. Influence spread achieved by varying size of the seed-set by the heuristics in the datasets after flipping the signs of edges.

Review

Positive:

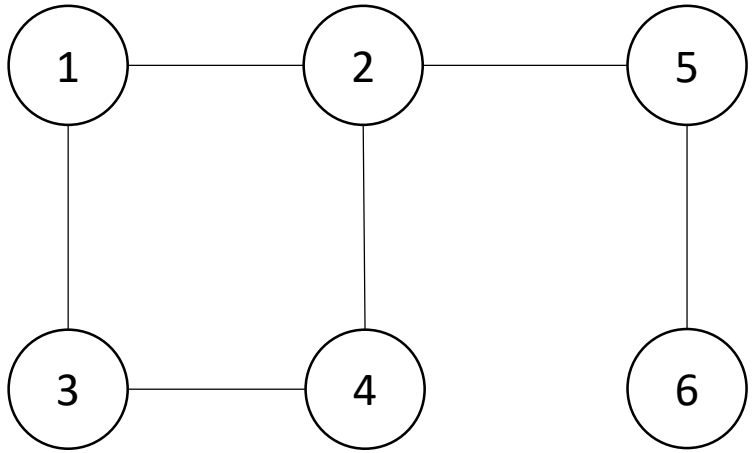
- Introduced novel influence maximization problem SiNiMax
- $OSSUM\pm$ outperforms state-of-the-art heuristics for influence maximization
- Application to real world data

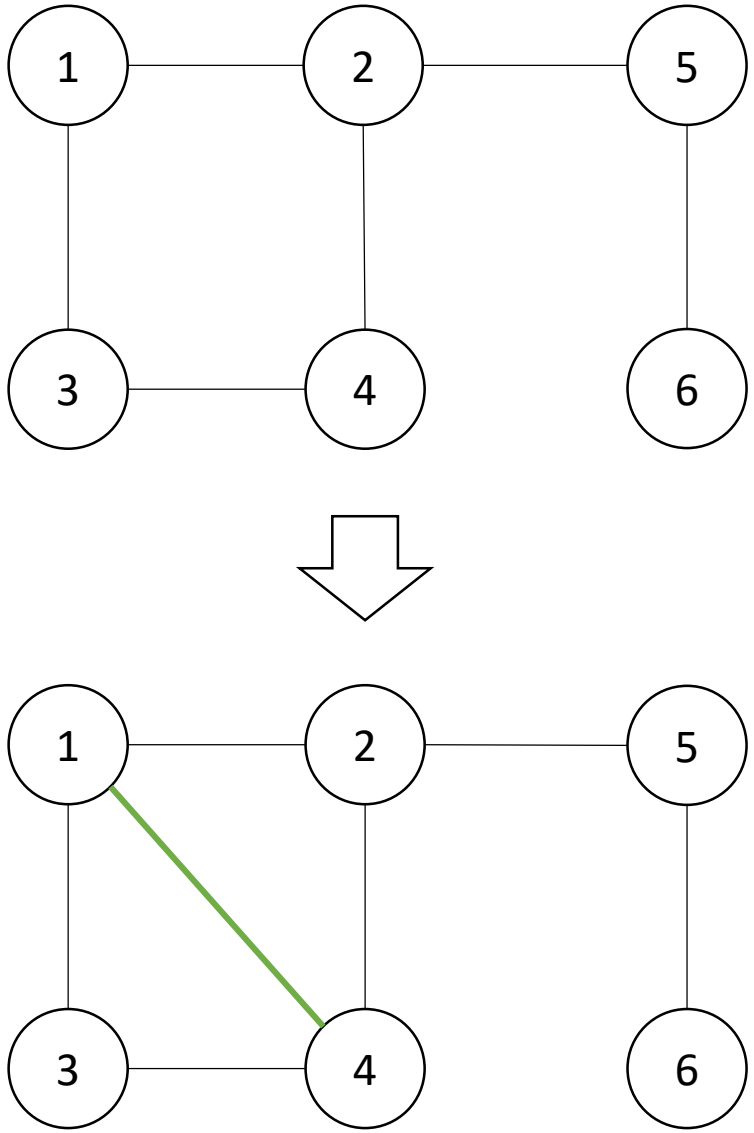
Negative:

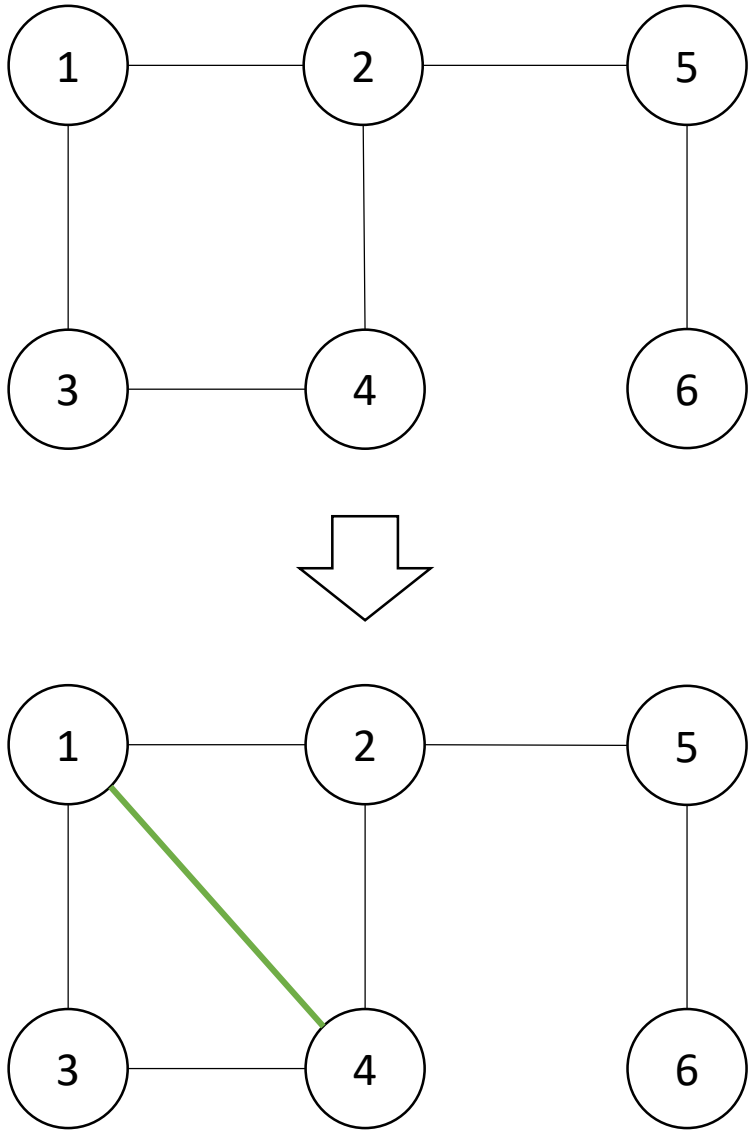
- Inaccuracies in description of method (missing edge cases)
- Unproven claim of extensaibility of this approach
- Real world data had to be modified to show advantage of $OSSUM\pm$

Top-k Link Recommendation in Social Graphs

Sören Tietböhl

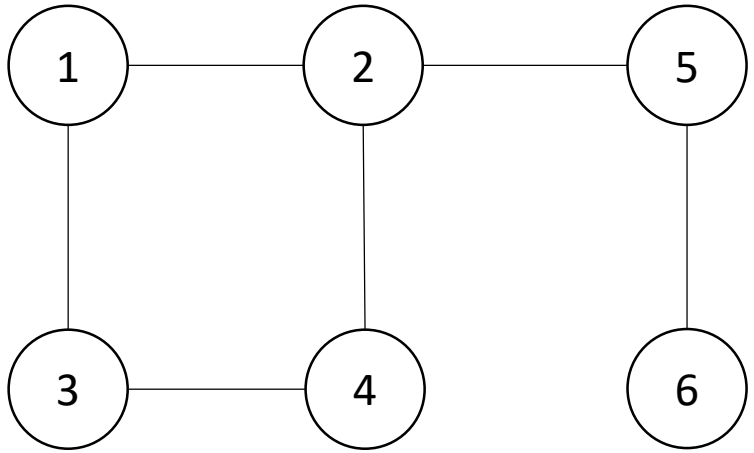

1
2
5
3
4
6

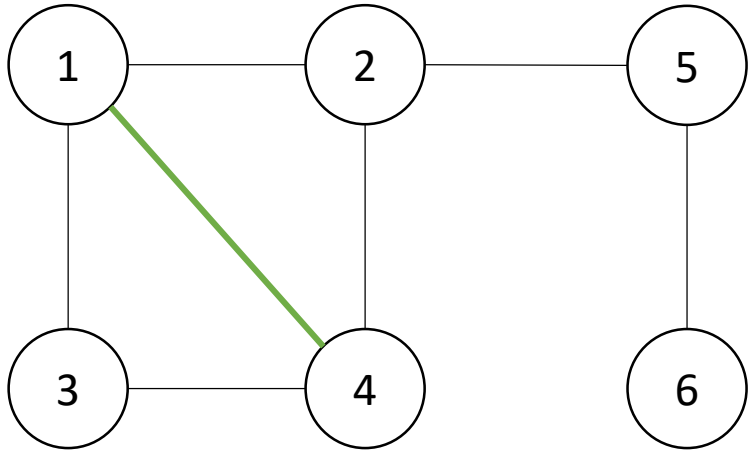

1
2
5
3
4
6
1
2
5
3
4
6



$$score(x, y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$$

$$score(x, y) = \sum_{z \in |\Gamma(x) \cap \Gamma(y)|} \frac{1}{\log |\Gamma(z)|}$$

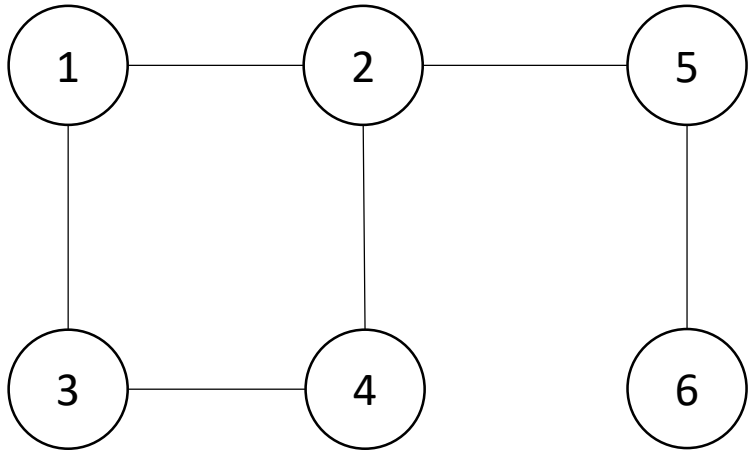


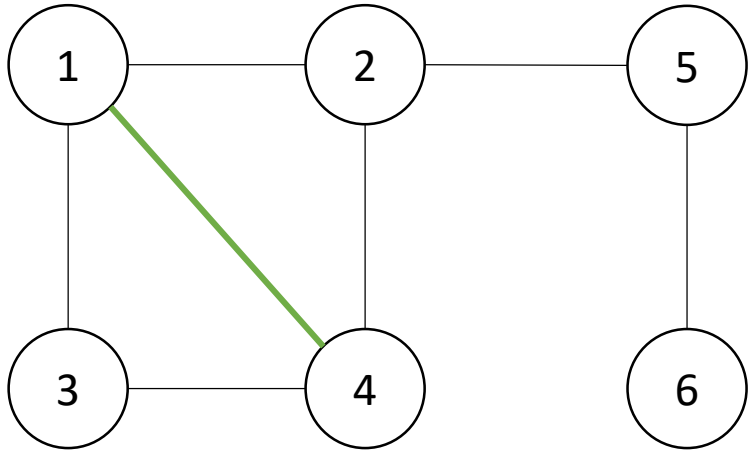


$$score(x, y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$$

$$score(x, y) = \sum_{z \in |\Gamma(x) \cap \Gamma(y)|} \frac{1}{\log |\Gamma(z)|}$$

X = UV

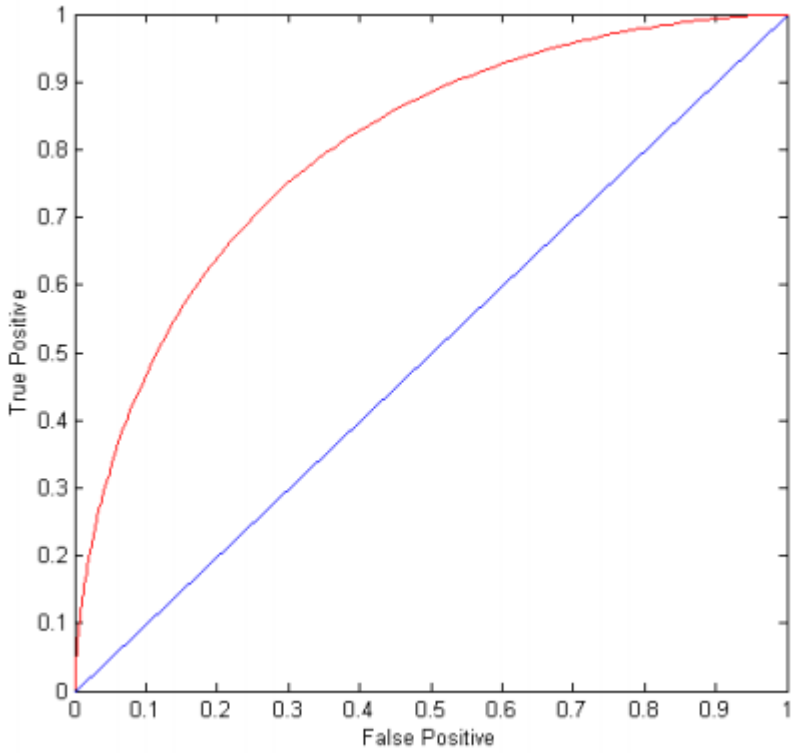


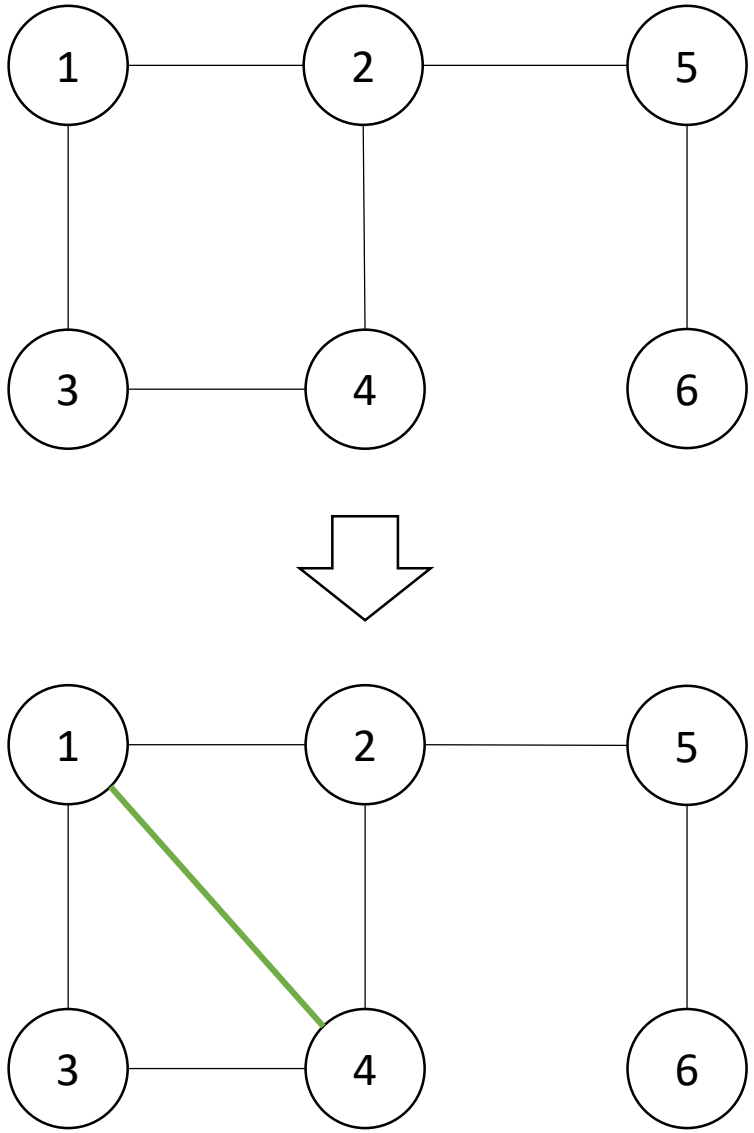


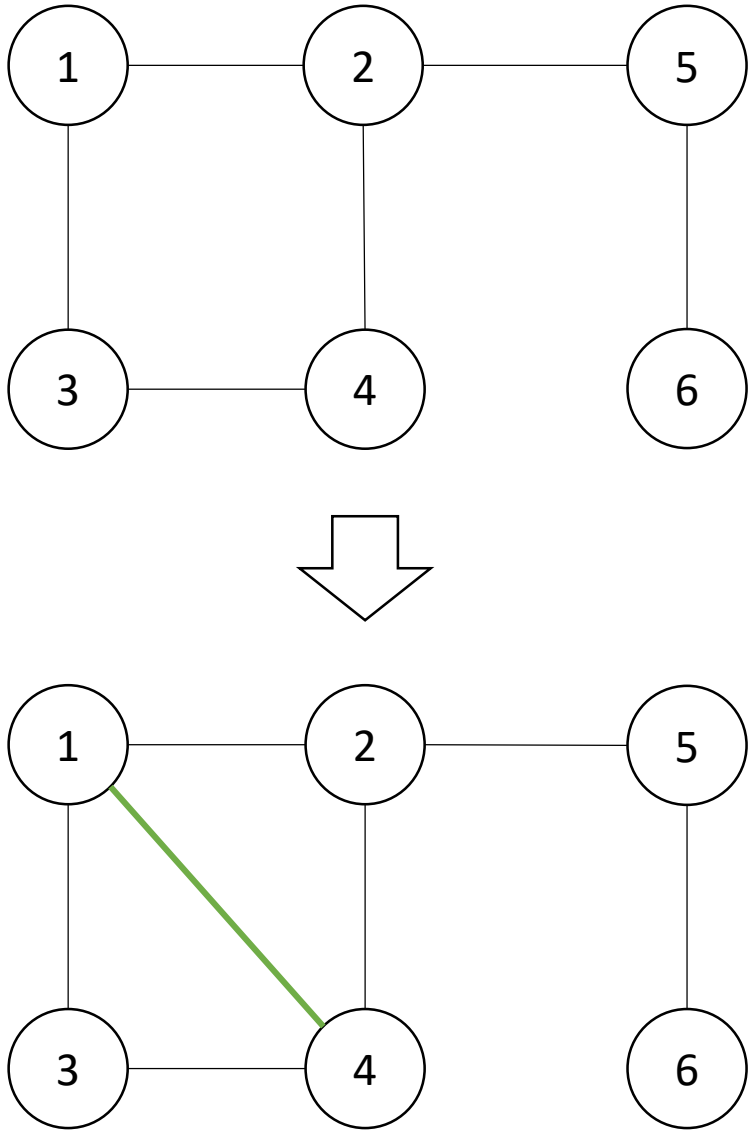
$$score(x,y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$$

$$score(x,y) = \sum_{z \in |\Gamma(x) \cap \Gamma(y)|} \frac{1}{\log |\Gamma(z)|}$$

X = UV




1
2
5
3
4
6
1
2
5
3
4
6

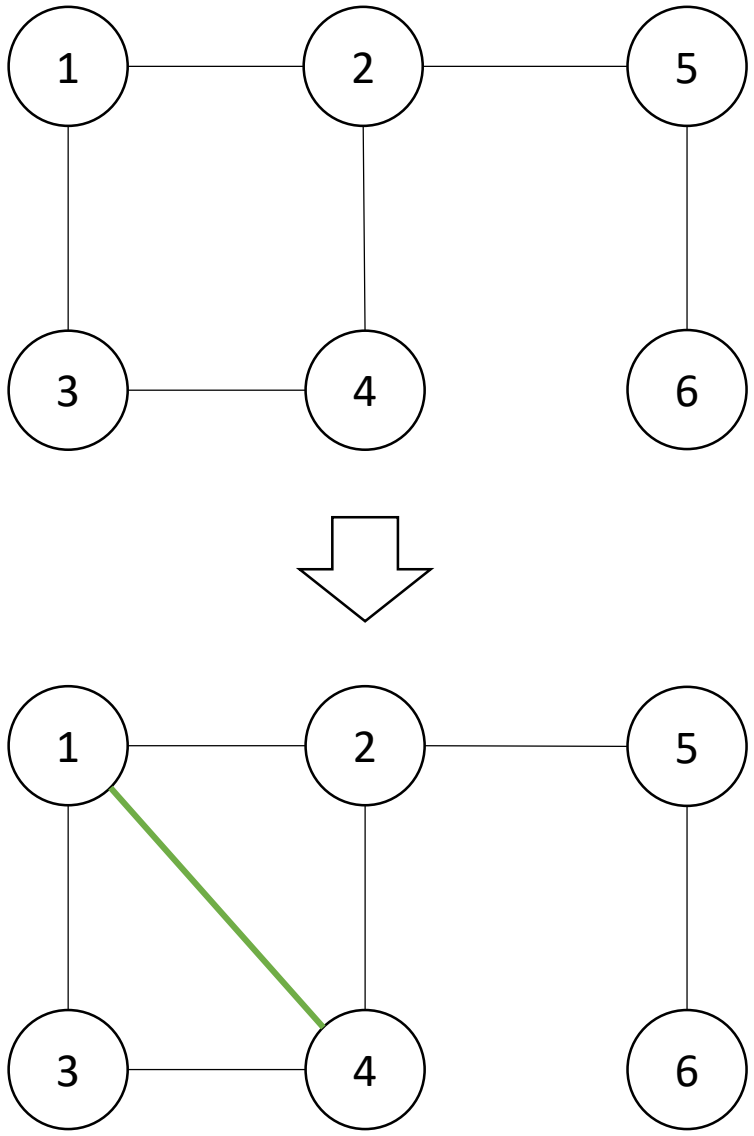


Predictions for user 4:

Ground truth: (1,4),(2,4),(3,4),(4,5),(4,6)

Example prediction 1: (2,4),(3,4),(4,5),(1,4),(4,6)

Example prediction 2: (2,4),(4,5),(3,4),(1,4),(4,6)



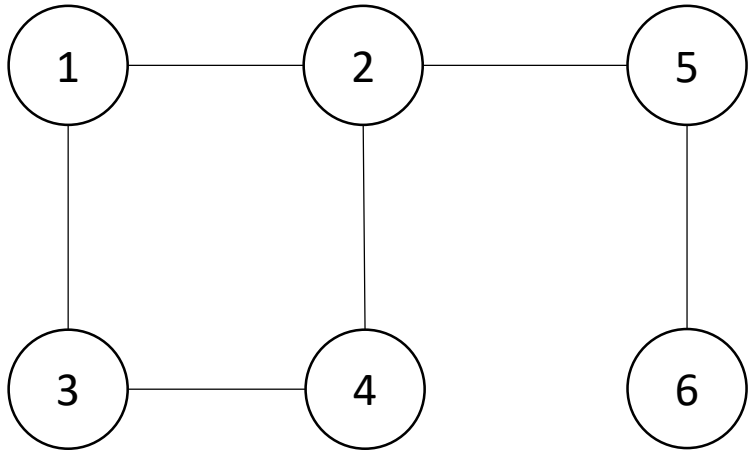
Predictions for user 4:

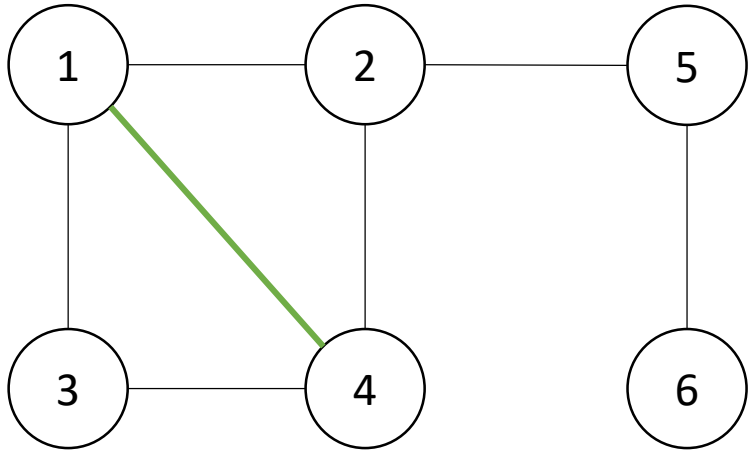
Ground truth: (1,4),(2,4),(3,4),(4,5),(4,6)

Example prediction 1: (2,4),(3,4),(4,5),(1,4),(4,6)

Example prediction 2: (2,4),(4,5),(3,4),(1,4),(4,6)

Loss is 1 in both cases





Predictions for user 4:

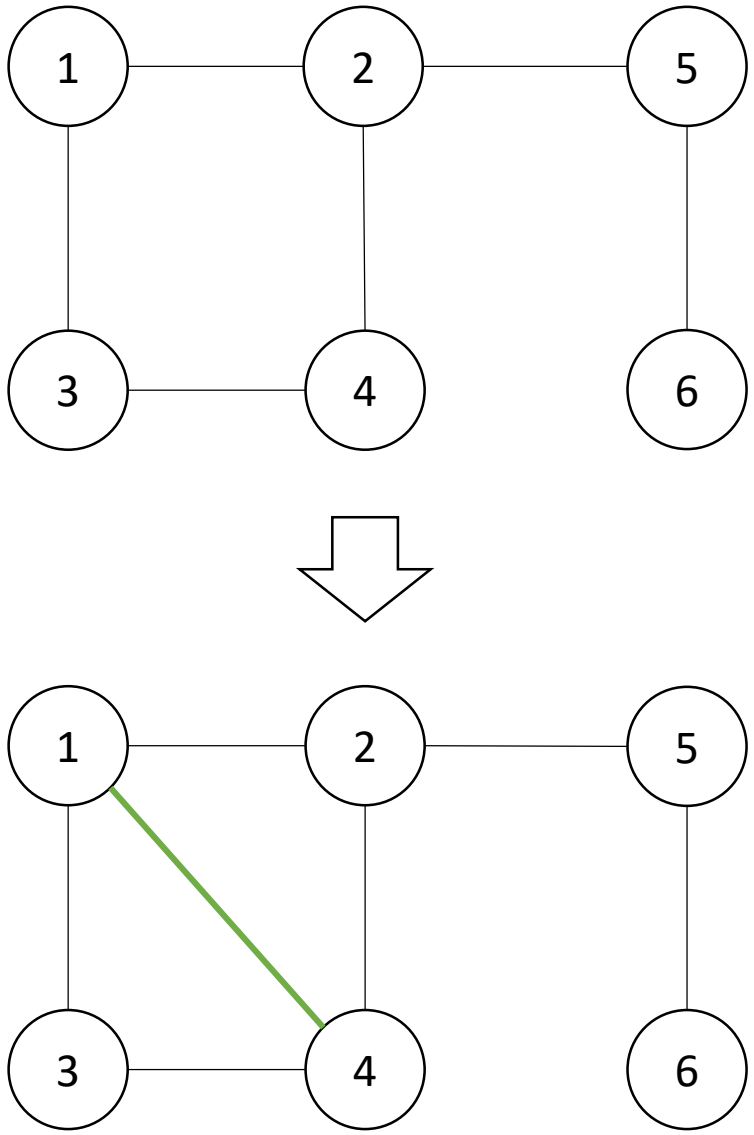
Ground truth: (1,4),(2,4),(3,4),(4,5),(4,6)

Example prediction 1: (2,4),(3,4),(4,5),(1,4),(4,6)

Example prediction 2: (2,4),(4,5),(3,4),(1,4),(4,6)

Loss is 1 in both cases

TODO List:



Predictions for user 4:

Ground truth: (1,4),(2,4),(3,4),(4,5),(4,6)

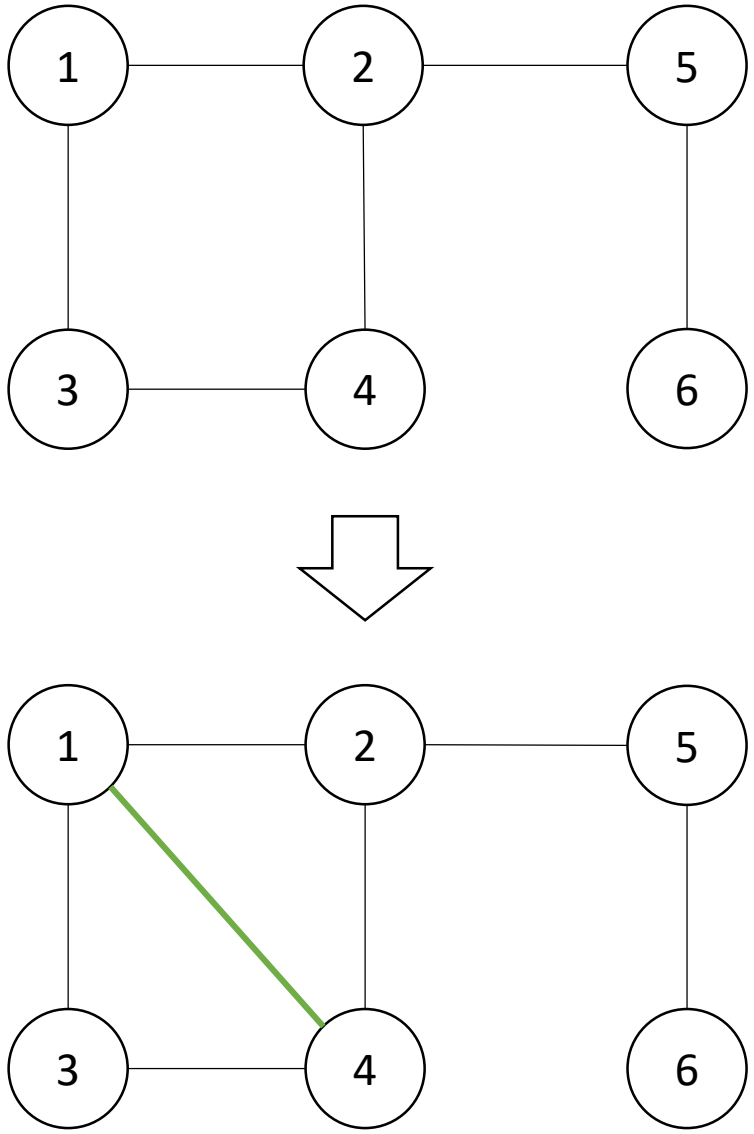
Example prediction 1: (2,4),(3,4),(4,5),(1,4),(4,6)

Example prediction 2: (2,4),(4,5),(3,4),(1,4),(4,6)

Loss is 1 in both cases

TODO List:

1. Define new loss function that penalizes mistakes at the top more than at the bottom



Predictions for user 4:

Ground truth: (1,4),(2,4),(3,4),(4,5),(4,6)

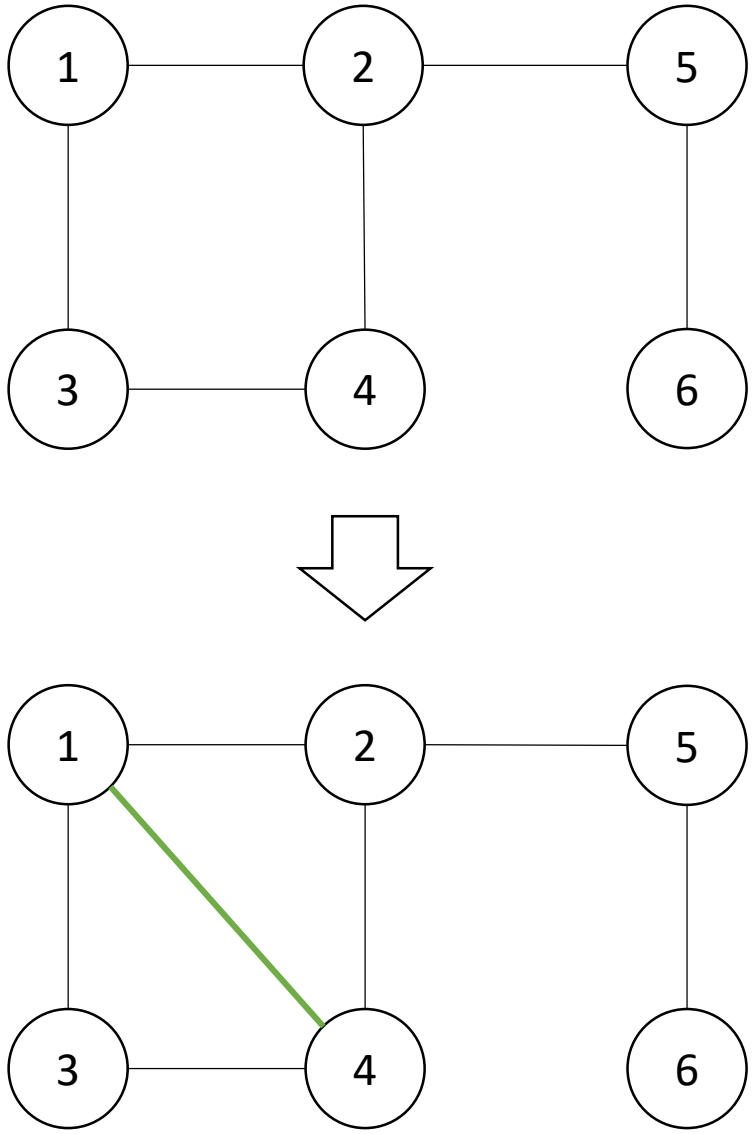
Example prediction 1: (2,4),(3,4),(4,5),(1,4),(4,6)

Example prediction 2: (2,4),(4,5),(3,4),(1,4),(4,6)

Loss is 1 in both cases

TODO List:

1. Define new loss function that penalizes mistakes at the top more than at the bottom
2. Derive an algorithm that minimizes that function



Predictions for user 4:

Ground truth: (1,4),(2,4),(3,4),(4,5),(4,6)

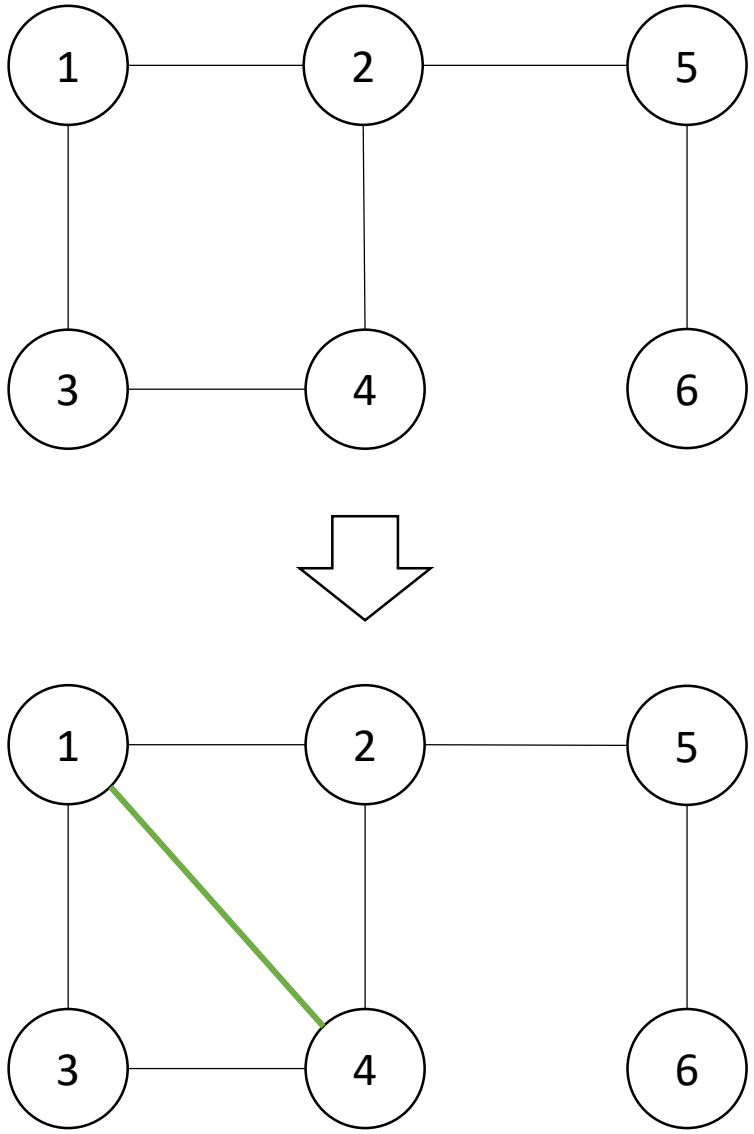
Example prediction 1: (2,4),(3,4),(4,5),(1,4),(4,6)

Example prediction 2: (2,4),(4,5),(3,4),(1,4),(4,6)

Loss is 1 in both cases

TODO List:

1. Define new loss function that penalizes mistakes at the top more than at the bottom
2. Derive an algorithm that minimizes that function
3. Evaluate the algorithm using various datasets and metrics



Predictions for user 4:

Ground truth: (1,4),(2,4),(3,4),(4,5),(4,6)

Example prediction 1: (2,4),(3,4),(4,5),(1,4),(4,6)

Example prediction 2: (2,4),(4,5),(3,4),(1,4),(4,6)

Loss is 1 in both cases

TODO List:

1. Define new loss function that penalizes mistakes at the top more than at the bottom
2. Derive an algorithm that minimizes that function
3. Evaluate the algorithm using various datasets and metrics
4. ???
5. Profit

$$R(X_{ij}, \widehat{X}^i) = I(X_{ij} = 1) \cdot \sum_{s=1}^{n} I(\widehat{X}_{ij} \leq \widehat{X}_{is}) I(X_{is} = 0)$$

X = Training Adjacency Matrix

$\widehat{X}$ = Ranking Score

$$I(a=1) = \begin{cases} 1 & a = 1 \\ 0 & \text{else} \end{cases}$$

$$R(X_{ij}, \widehat{X}^i) = I(X_{ij} = 1) \cdot \sum_{s=1}^{n} I(\widehat{X}_{ij} \leq \widehat{X}_{is}) I(X_{is} = 0)$$

X = Training Adjacency Matrix

$\widehat{X}$ = Ranking Score

$$I(a=1) = \begin{cases} 1 & a = 1 \\ 0 & \text{else} \end{cases}$$

$$R(X_{ij}, \widehat{X}^i) = I(X_{ij} = 1) \cdot \sum_{s=1}^{n} I(\widehat{X}_{ij} \leq \widehat{X}_{is}) I(X_{is} = 0)$$

(2,4),(4,5),(3,4),(1,4),(4,6)

X = Training Adjacency Matrix

$\widehat{X}$ = Ranking Score

$$I(a=1) = \begin{cases} 1 & a = 1 \\ 0 & \text{else} \end{cases}$$

$$R(X_{ij}, \widehat{X}^i) = I(X_{ij} = 1) \cdot \sum_{s=1}^{n} I(\widehat{X}_{ij} \leq \widehat{X}_{is}) I(X_{is} = 0)$$

(2,4),(4,5),(3,4),(1,4),(4,6)

$$L\left(R(X_{ij}, \widehat{X}^i)\right) = \log\left(1 + R(X_{ij}, \widehat{X}^i)\right)$$

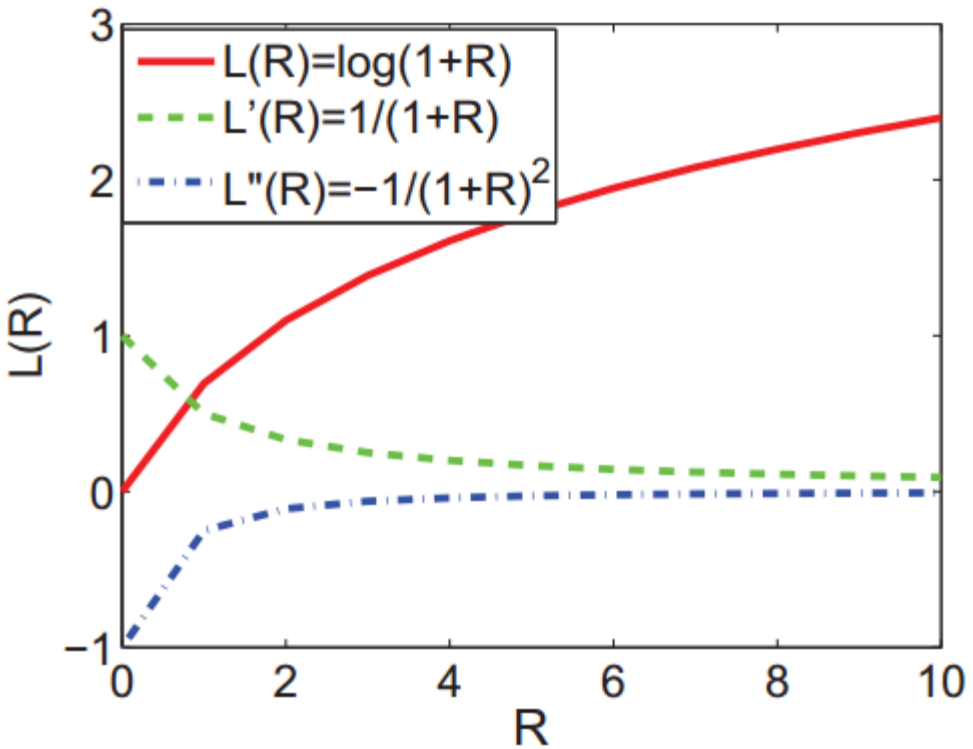
X = Training Adjacency Matrix

$\widehat{X}$ = Ranking Score

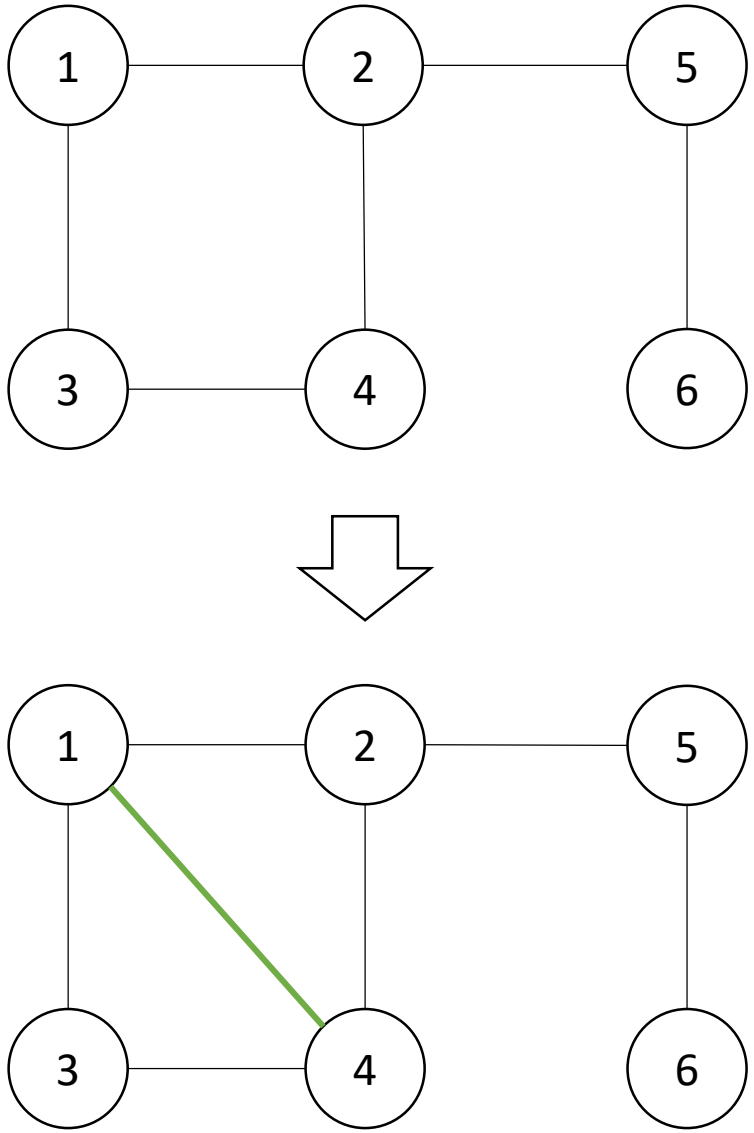
$$I(a=1) = \begin{cases} 1 & a = 1 \\ 0 & \text{else} \end{cases}$$

$$R(X_{ij}, \widehat{X}^i) = I(X_{ij} = 1) \cdot \sum_{s=1}^{n} I(\widehat{X}_{ij} \leq \widehat{X}_{is}) I(X_{is} = 0)$$

(2,4),(4,5),(3,4),(1,4),(4,6)



$$L\left(R(X_{ij}, \widehat{X}^i)\right) = \log\left(1 + R(X_{ij}, \widehat{X}^i)\right)$$

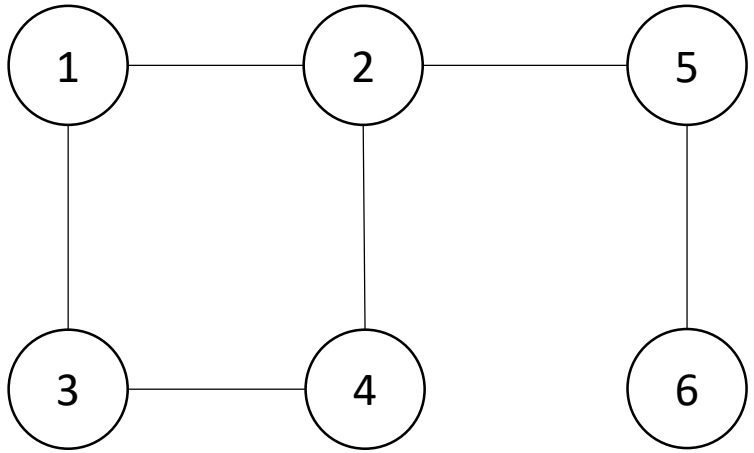


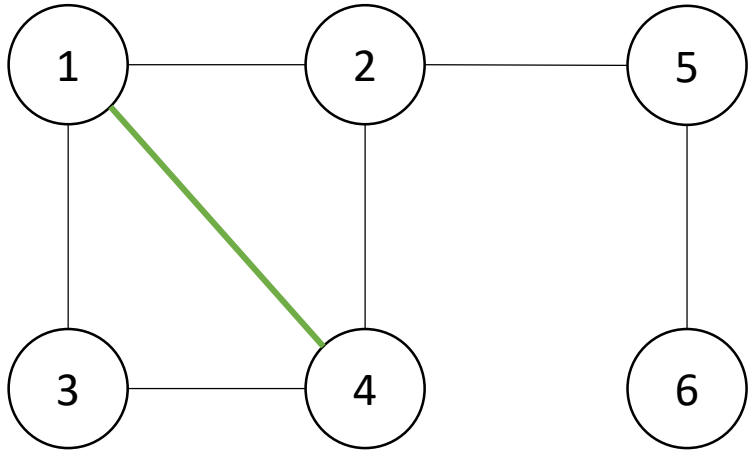
Predictions for user 4:

Ground truth: (1,4),(2,4),(3,4),(4,5),(4,6)

Example prediction 1: (2,4),(3,4),(4,5),(1,4),(4,6)

Example prediction 2: (2,4),(4,5),(3,4),(1,4),(4,6)





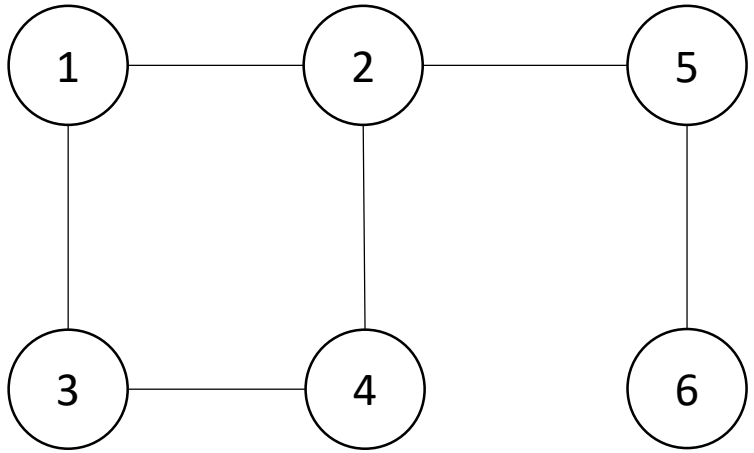
Predictions for user 4:

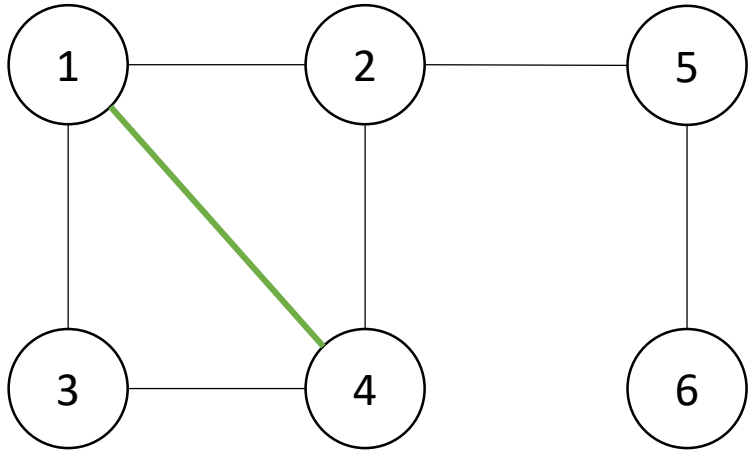
Ground truth: (1,4),(2,4),(3,4),(4,5),(4,6)

Example prediction 1: (2,4),(3,4),(4,5),(1,4),(4,6)

Example prediction 2: (2,4),(4,5),(3,4),(1,4),(4,6)

$$L(R(X_{41}, \widehat{X}^4)) = \log(1 + R(X_{41}, \widehat{X}^4))$$





Predictions for user 4:

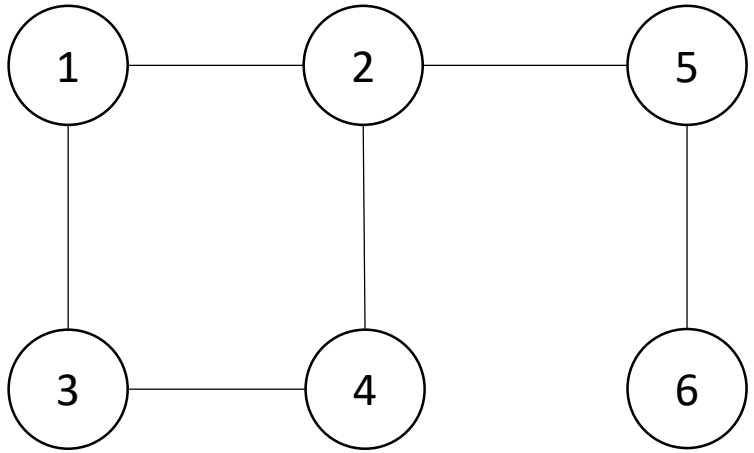
Ground truth: (1,4),(2,4),(3,4),(4,5),(4,6)

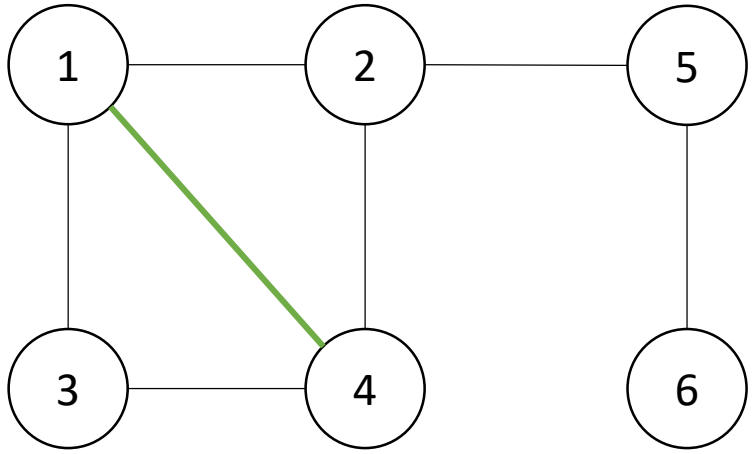
Example prediction 1: (2,4),(3,4),(4,5),(1,4),(4,6)

Example prediction 2: (2,4),(4,5),(3,4),(1,4),(4,6)

$$L(R(X_{41}, \widehat{X}^4)) = \log(1 + R(X_{41}, \widehat{X}^4))$$

Prediction 1: 0 + 0 + 1





Predictions for user 4:

Ground truth: (1,4),(2,4),(3,4),(4,5),(4,6)

Example prediction 1: (2,4),(3,4),(4,5),(1,4),(4,6)

Example prediction 2: (2,4),(4,5),(3,4),(1,4),(4,6)

$$L(R(X_{41}, \widehat{X}^4)) = \log(1 + R(X_{41}, \widehat{X}^4))$$

Prediction 1: 0 + 0 + 1

Prediction 2: 0 + 1 + 1

$$\widehat{R}(X_{ij}, \widehat{X}^i) = I(X_{ij} = 1) \cdot \sum_{s=1}^{n} g(\widehat{X}_{is} - \widehat{X}_{ij}) I(X_{is} = 0)$$

$$\widehat{R}(X_{ij}, \widehat{X}^i) = I(X_{ij} = 1) \cdot \sum_{s=1}^{n} g(\widehat{X}_{is} - \widehat{X}_{ij}) I(X_{is} = 0)$$

Sigmoid function: $g(z) = 1/(1 + \exp(-z))$

$$\widehat{R}(X_{ij}, \widehat{X}^i) = I(X_{ij} = 1) \cdot \sum_{s=1}^{n} g(\widehat{X}_{is} - \widehat{X}_{ij}) I(X_{is} = 0)$$

Sigmoid function: $g(z) = 1/(1 + \exp(-z))$

Now its derivable !

$$\widehat{R}(X_{ij}, \widehat{X}^i) = I(X_{ij} = 1) \cdot \sum_{s=1}^{n} g(\widehat{X}_{is} - \widehat{X}_{ij}) I(X_{is} = 0)$$

Sigmoid function: $g(z) = 1/(1 + \exp(-z))$

Now its derivable !

yay

$$\widehat{R}(X_{ij}, \widehat{X}^i) = I(X_{ij} = 1) \cdot \sum_{s=1}^{n} g(\widehat{X}_{is} - \widehat{X}_{ij}) I(X_{is} = 0)$$

Sigmoid function: $g(z) = 1/(1 + \exp(-z))$

Now its derivable !

$$\begin{aligned}
&\mathcal{O}(U, V) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} \phi(X_{ij}, \widehat{X}^i) + \frac{\lambda}{2} \sum_{i=1}^{n} U_i^T U_i + \frac{\mu}{2} \sum_{j=1}^{n} V_j^T V_j \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} \phi\left(\widehat{R}(X_{ij}, \widehat{X}^i)\right) + \frac{\lambda}{2} \sum_{i=1}^{n} U_i^T U_i + \frac{\mu}{2} \sum_{j=1}^{n} V_j^T V_j
\end{aligned}$$

$$\widehat{X}_{ij} = U_i^T V_j$$

$$\widehat{X}_{ij} = U_i^T V_j$$

$$\widehat{X}_{ij} = U_i^T V_j + \boldsymbol{g}_{ij}^T \boldsymbol{\theta}$$

$$\widehat{X}_{ij} = U_i^T V_j$$

$$\widehat{X}_{ij} = U_i^T V_j + \boldsymbol{g}_{ij}^T \boldsymbol{\theta}$$

$$g_{ij} = \begin{pmatrix} \text{degree}(i) \\ \text{degree}(j) \\ |\Gamma(i) \cap \Gamma(j)| \end{pmatrix} \qquad \theta \in \mathbb{R}^3$$

$$\widehat{X}_{ij} = U_i^T V_j$$

$$\widehat{X}_{ij} = U_i^T V_j + \boldsymbol{g}_{ij}^T \boldsymbol{\theta}$$

$$g_{ij} = \begin{pmatrix} \text{degree}(i) \\ \text{degree}(j) \\ |\Gamma(i) \cap \Gamma(j)| \end{pmatrix} \qquad \theta \in \mathbb{R}^3$$

Algorithm 2 Top-k link recommendation algorithm with both latent features and explicit features.

Input: X, λ, μ, α, T (maximum iteration).
Initialize: set $t = 0$, initialize U and V randomly.
repeat
  $t = t + 1$
  Randomly pick up an observed link X_{ij};
  Fix user i, uniformly draw b unknown status links $X_{is} = 0$;
  if $\exists$ s such that $U_i^T V_j + \boldsymbol{g}_{ij}^T \boldsymbol{\theta} < U_i^T V_s + \boldsymbol{g}_{is}^T \boldsymbol{\theta}$
    Calculate $\frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial U_i}$ based upon Eq. 12;
    Calculate $\frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial V_j}$ with Eq. 13;
    Calculate $\frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial V_s}$ with Eq. 14;
    Calculate $\frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\boldsymbol{\theta}}$ with Eq. 15;
    $U = U - \alpha \frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial U}$;
    $V = V - \alpha \frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial V}$;
    $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\boldsymbol{\theta}}$;
  end
until validation error does not improve or $t > T$.

$$\widehat{X}_{ij} = U_i^T V_j$$

$$\widehat{X}_{ij} = U_i^T V_j + \boldsymbol{g}_{ij}^T \boldsymbol{\theta}$$

$$g_{ij} = \begin{pmatrix} \text{degree}(i) \\ \text{degree}(j) \\ |\Gamma(i) \cap \Gamma(j)| \end{pmatrix} \qquad \theta \in \mathbb{R}^3$$

Algorithm 2 Top-k link recommendation algorithm with both latent features and explicit features.

Input: X, λ, μ, α, T (maximum iteration).
Initialize: set $t = 0$, initialize U and V randomly.
repeat
  $t = t + 1$
  Randomly pick up an observed link X_{ij};
  Fix user i, uniformly draw b unknown status links $X_{is} = 0$;
  if $\exists$ s such that $U_i^T V_j + \boldsymbol{g}_{ij}^T \boldsymbol{\theta} < U_i^T V_s + \boldsymbol{g}_{is}^T \boldsymbol{\theta}$
    Calculate $\frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial U_i}$ based upon Eq. 12;
    Calculate $\frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial V_j}$ with Eq. 13;
    Calculate $\frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial V_s}$ with Eq. 14;
    Calculate $\frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\boldsymbol{\theta}}$ with Eq. 15;
    $U = U - \alpha \frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial U}$;
    $V = V - \alpha \frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial V}$;
    $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\boldsymbol{\theta}}$;
  end
until validation error does not improve or $t > T$.

(2,4),(3,4),(4,5),(1,4),(4,6)

$$\widehat{X}_{ij} = U_i^T V_j$$

$$\widehat{X}_{ij} = U_i^T V_j + \boldsymbol{g}_{ij}^T \boldsymbol{\theta}$$

$$g_{ij} = \begin{pmatrix} \text{degree}(i) \\ \text{degree}(j) \\ |\Gamma(i) \cap \Gamma(j)| \end{pmatrix} \qquad \theta \in \mathbb{R}^3$$

Algorithm 2 Top-k link recommendation algorithm with both latent features and explicit features.

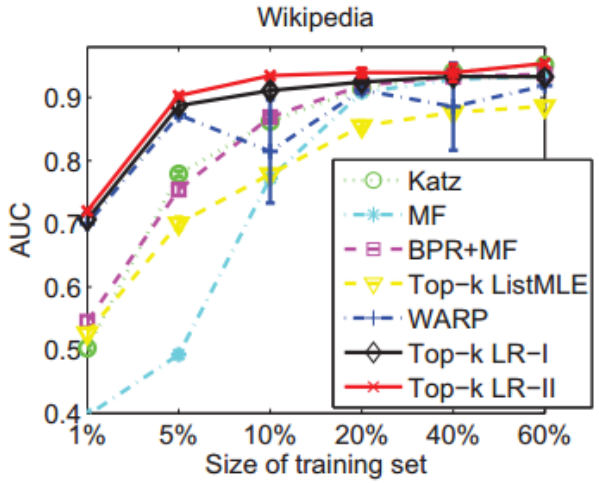
Input: X, λ, μ, α, T (maximum iteration).
Initialize: set $t = 0$, initialize U and V randomly.
repeat
  $t = t + 1$
  Randomly pick up an observed link X_{ij};
  Fix user i, uniformly draw b unknown status links $X_{is} = 0$;
  if $\exists\ s$ such that $U_i^T V_j + \boldsymbol{g}_{ij}^T \boldsymbol{\theta} < U_i^T V_s + \boldsymbol{g}_{is}^T \boldsymbol{\theta}$
    Calculate $\frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial U_i}$ based upon Eq. 12;
    Calculate $\frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial V_j}$ with Eq. 13;
    Calculate $\frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial V_s}$ with Eq. 14;
    Calculate $\frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\boldsymbol{\theta}}$ with Eq. 15;
    $U = U - \alpha \frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial U}$;
    $V = V - \alpha \frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\partial V}$;
    $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \frac{\partial \overline{\mathcal{O}}(U,V,\boldsymbol{\theta})}{\boldsymbol{\theta}}$;
  end
until validation error does not improve or $t > T$.

(2,4),(3,4),(4,5),(1,4),(4,6)

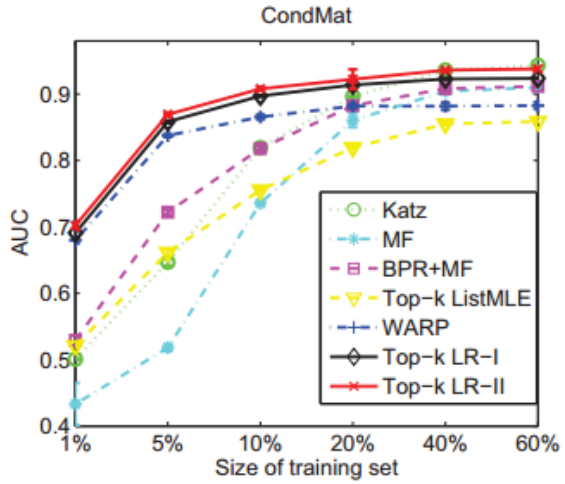
⇩

(1,4),(2,4),(3,4),(4,5),(4,6)

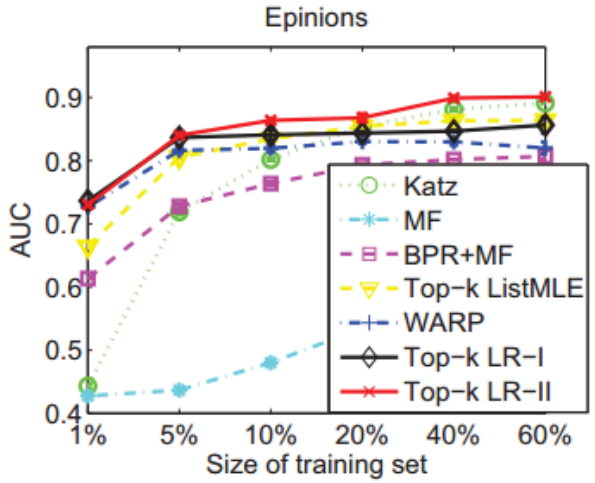
Dataset	Wikipedia	CondMat	Epinions	MovieLens 1M
Nodes	7,115	23,133	75,879	6040/3592
Links	103,689	186,994	508,837	1,000,209
Links: Non-links	1:488	1:2,929	1:11,317	1:20.70
Average degree	14.57	7.89	6.70	165.59
Type	Directed	Undirected	Directed	Directed



(a) AUC on Wikipedia

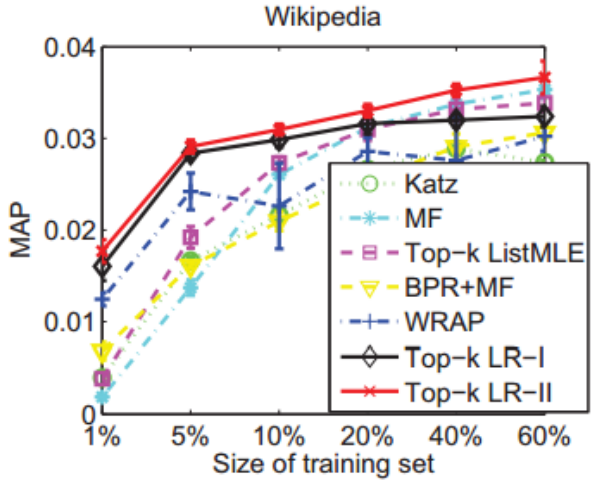


(b) AUC on CondMat

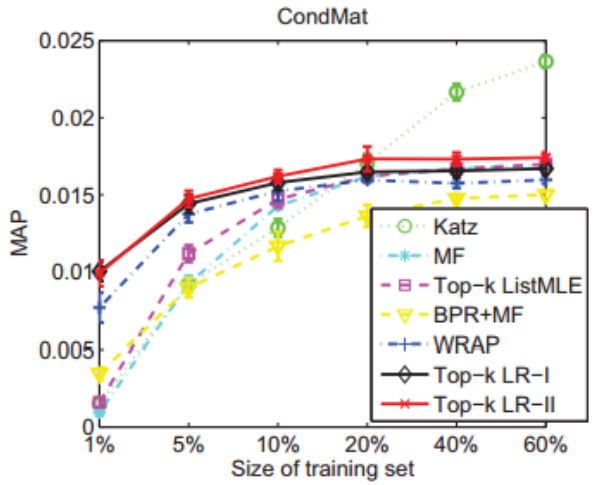


(c) AUC on Epinions

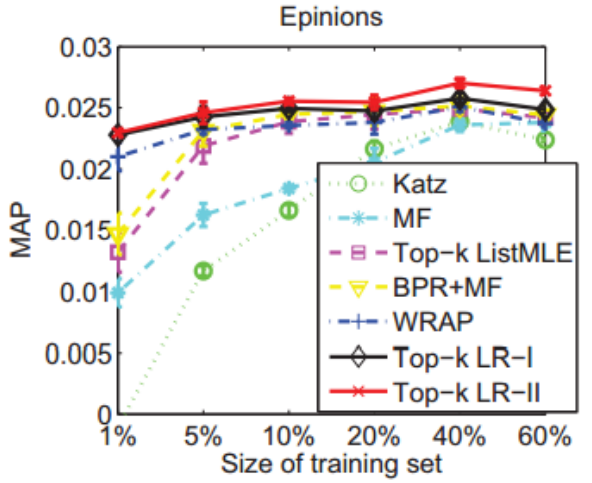
Figure 2: (a) to (c) are AUC obtained on Wikipedia, CondMat, and Epinions when the size of training set varies from 1%, 5%, 10%, 20%, 40%, to 60%. Error bars denote the standard deviations.



(a) MAP on Wikipedia

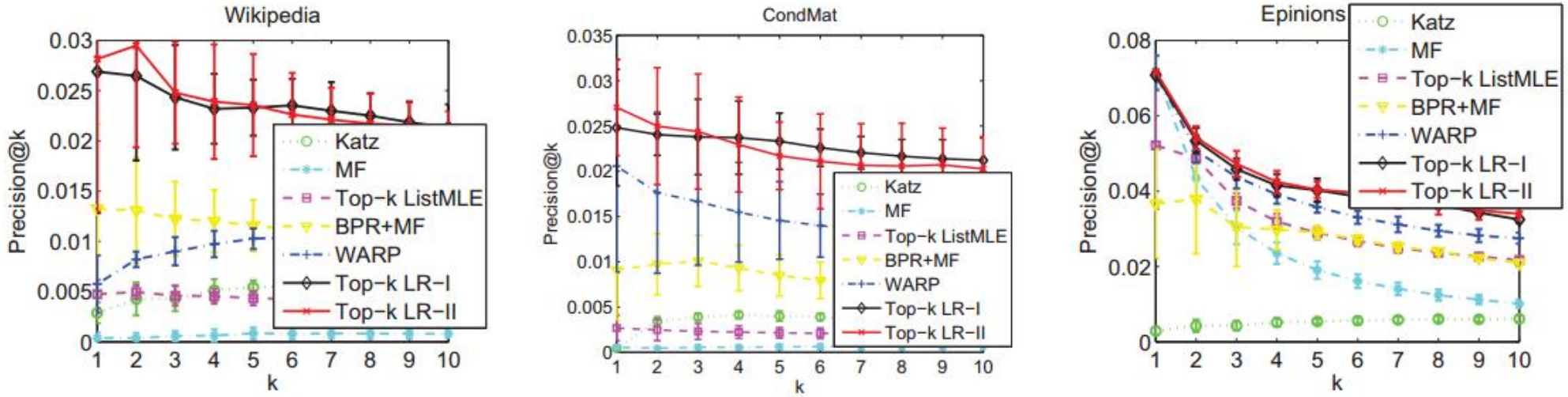


(b) MAP on CondMat



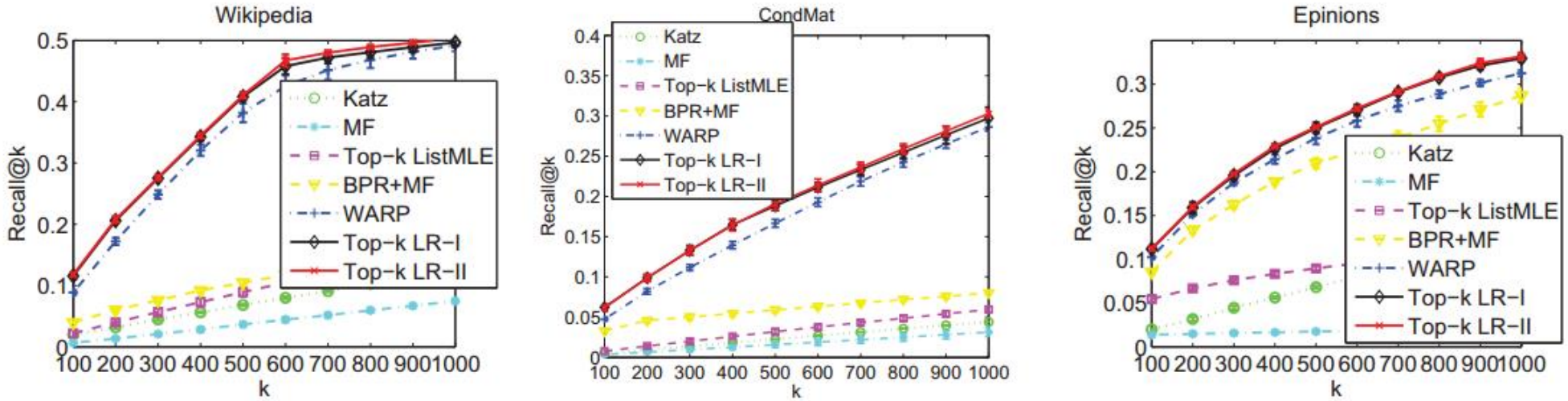
(c) MAP on Epinions

Figure 3: (a) to (c) are MAP obtained on Wikipedia, CondMat, and Epinions when the size of training set varies from 1%, 5%, 10%, 20%, 40%, to 60%. Error bars denote the standard deviations.



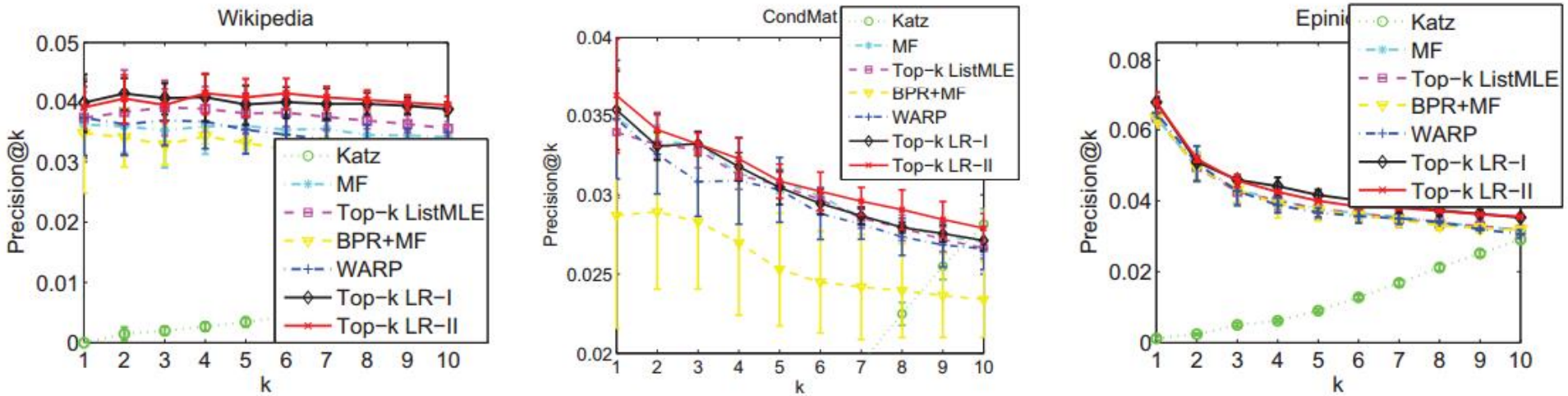
(a) Precision@k on Wiki (1%) (b) Precision@k CondMat (1%) (c) Precision@k Epinions (1%)

Figure 4: Precision@k for Wikipedia, CondMat, and Epinions when the size of training set is 1%. Error bars denote the standard deviations.



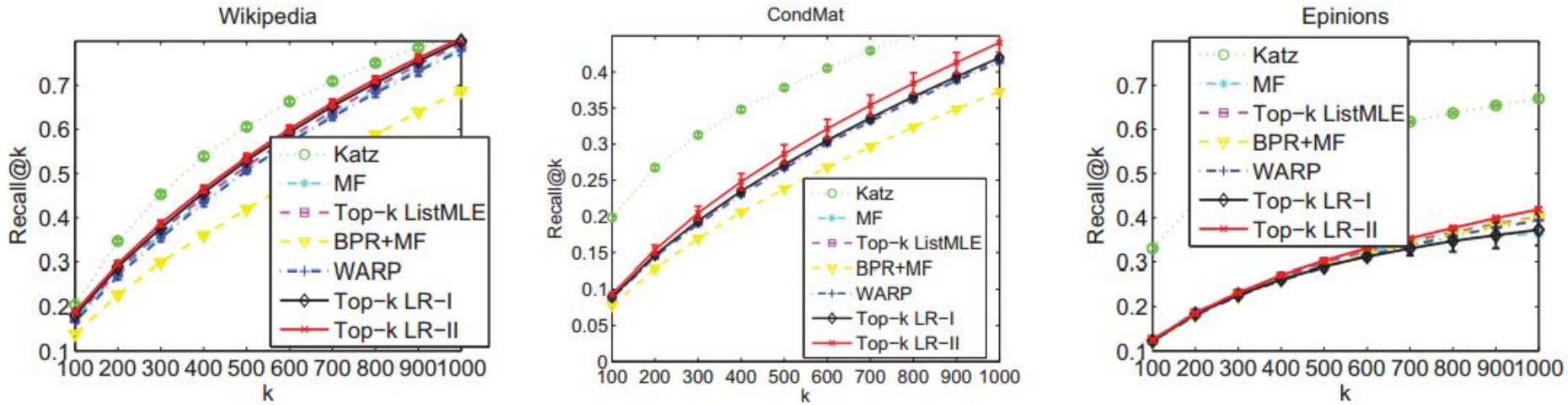
(a) Recall@k on Wiki (1%) (b) Recall@k CondMat (1%) (c) Recall@k Epinions (1%)

Figure 5: Recall@k for Wikipedia, CondMat, and Epinions when the size of training set is 1%. Error bars denote the standard deviations.



(a) P@k on Wiki (20%) (b) P@k CondMat (20%) (c) P@k Epinions (20%)

Figure 6: Precision@k over Wikipedia, CondMat, and Epinions when the size of training set is 20%. Error bars denote the standard deviations.



(a) Recall@k on Wiki (20%) (b) Recall@k CondMat (20%) (c) Recall@k Epinions (20%)

Figure 7: Recall@k over Wikipedia, CondMat, and Epinions when the size of training set is 20%. Error bars denote the standard deviations.

Sources

Dongjin Song, David A. Meyer, Dacheng Tao, "Top-k Link Recommendation in Social Networks", *2015 IEEE International Conference on Data Mining (ICDM)*, vol. 00, no. , pp. 389-398, 2015, doi:10.1109/ICDM.2015.136

Graph Mining Course Slides

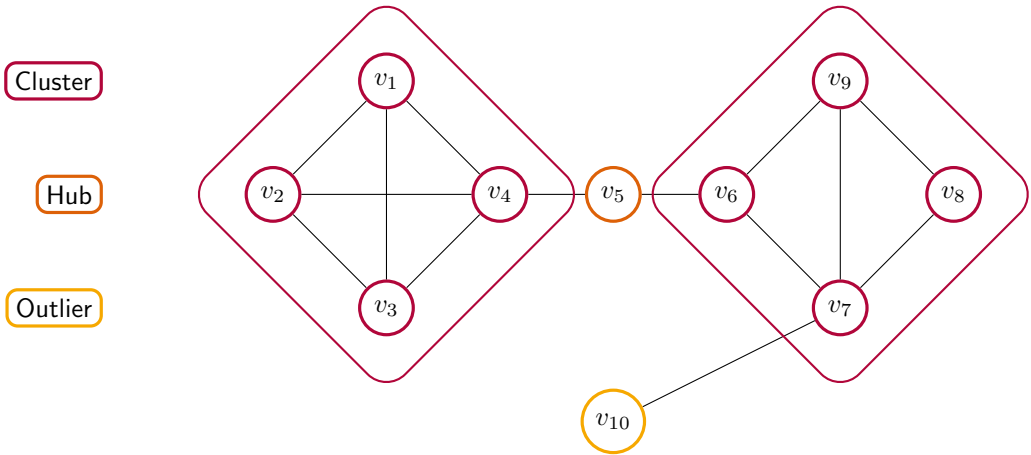
quicklatex.com

pSCAN: Fast and Exact Structural Graph Clustering

Lukas Wenzel

Graph Mining WT 2016/17

Structural Graph Clustering



Agenda

Principles of SCAN

Improvements of pSCAN

Experimental Results

Principles of SCAN

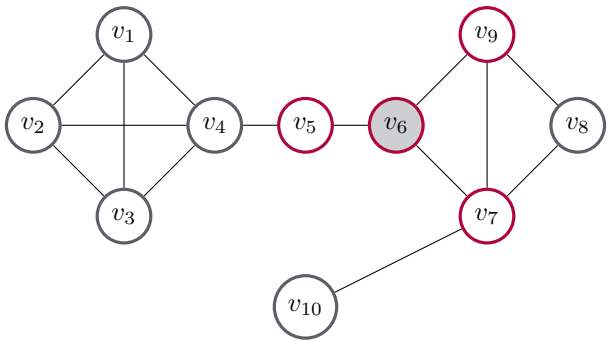
- Graph $G = (V, E)$

Structural Neighborhood:

$$N_u = \{v \in V | (u, v) \in E\} \cup \{u\}$$
$$d_u = |N_u|$$

Vertex u

Neighborhood N_u



Principles of SCAN

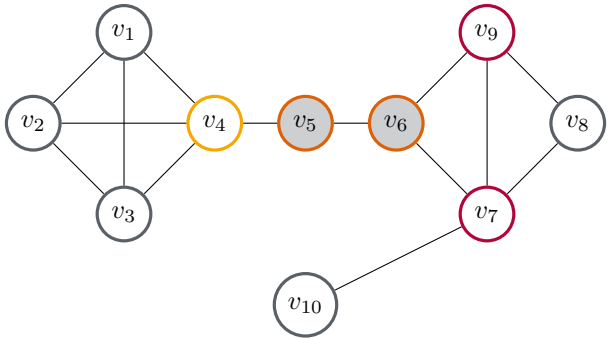
Structural Similarity:

$$\sigma(u, v) = \frac{|N_u \cap N_v|}{\sqrt{d_u \cdot d_v}}$$

Neighborhood N_u

Neighborhood N_v

$N_u \cap N_v$



$\sigma(v_6, v_5) = \frac{2}{\sqrt{3 \cdot 4}} \approx 0.57$

Similarity Threshold and Core Vertices:

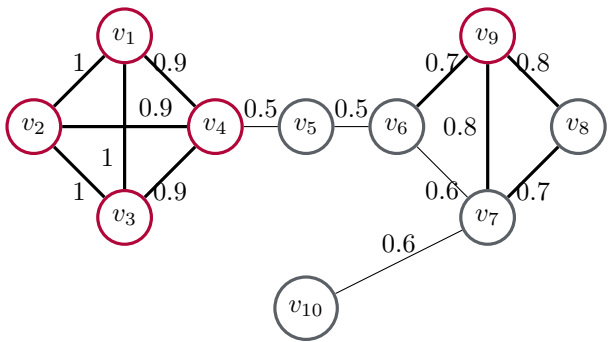
$$N_u^\epsilon = \{v \in N_u | \sigma(u, v) \geq \epsilon\}$$
$$V^C = \{v \in V | N_v^\epsilon \geq \mu\}$$

Core Vertex

ϵ-edge

$\epsilon = 0.7$

$\mu = 4$



Principles of SCAN

Structure Reachability:

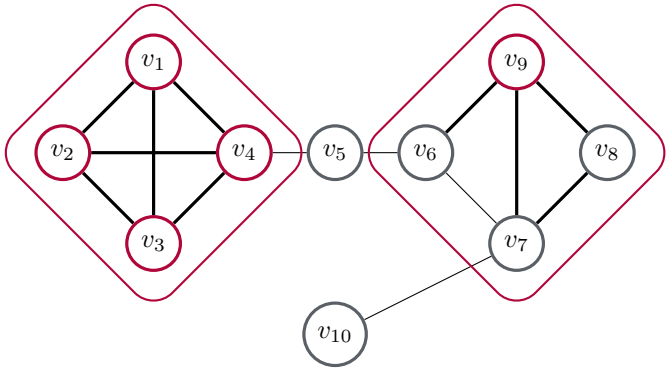
- u is structure reachable by v if there is a Path of ϵ-edges and intermediate core vertices
- Cluster $C \subseteq V$ such that all Vertices $v \in C$ are mutually structure reachable and no Vertex $x \in V \setminus C$ is structure reachable by any Vertex in C

Core Vertex

ϵ-edge

$\epsilon = 0.7$

$\mu = 4$



Principles of SCAN

Algorithm:

- SCAN requires structural similarity calculation along all edges in G

$\Rightarrow$ equivalent to enumeration of all triangles in G

- complexity of $\mathcal{O}(|E|^{1.5})$ which is worst case optimum

Agenda

Principles of SCAN

Improvements of pSCAN

Experimental Results

Improvements of pSCAN

Paradigm:

1. cluster core vertices
2. expand clusters by core vertices' structural neighborhood

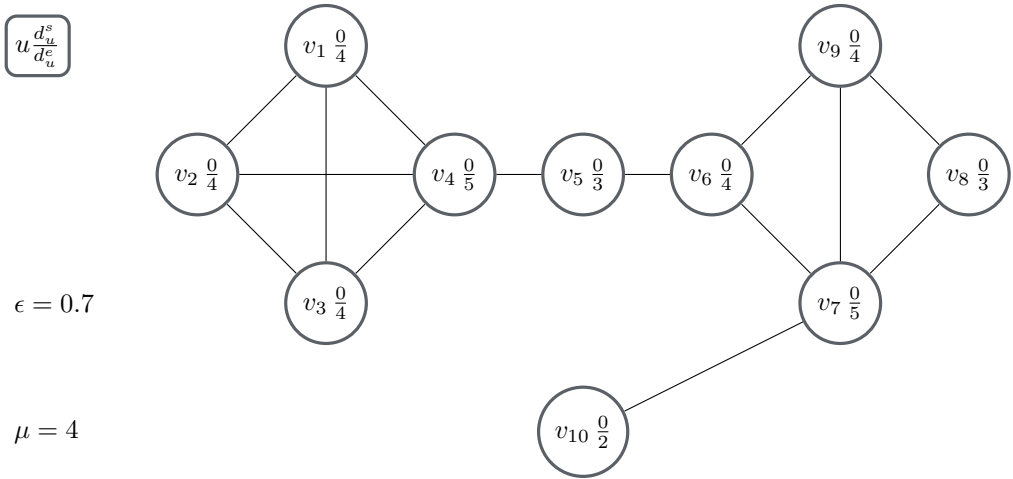
Effective Degree and Similar Degree:

- vertex $u \in V$ with incomplete similarity measures in its neighborhood
- effective degree: $d_u^e = d_u - x$ where x is the number of neighbors $v \in V$ with $\sigma(u, v) < \epsilon$ known
- similar degree: d_u^s number of neighbors $v \in V$ with $\sigma(u, v) \geq \epsilon$ known

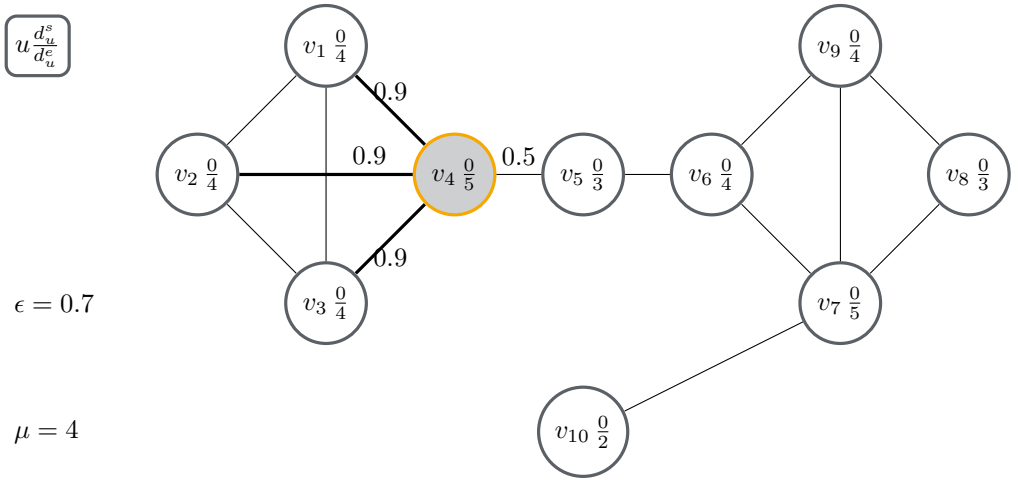
$\Rightarrow$ d_u^e and d_u^s provide an upper and lower bound for $|N_u^\epsilon|$

- calculate structural neighborhoods for vertices in non-increasing d^e-order
- update d^e and d^s for every calculated σ; omit neighborhood calculation if d^e or d^s are sufficient to identify core vertex

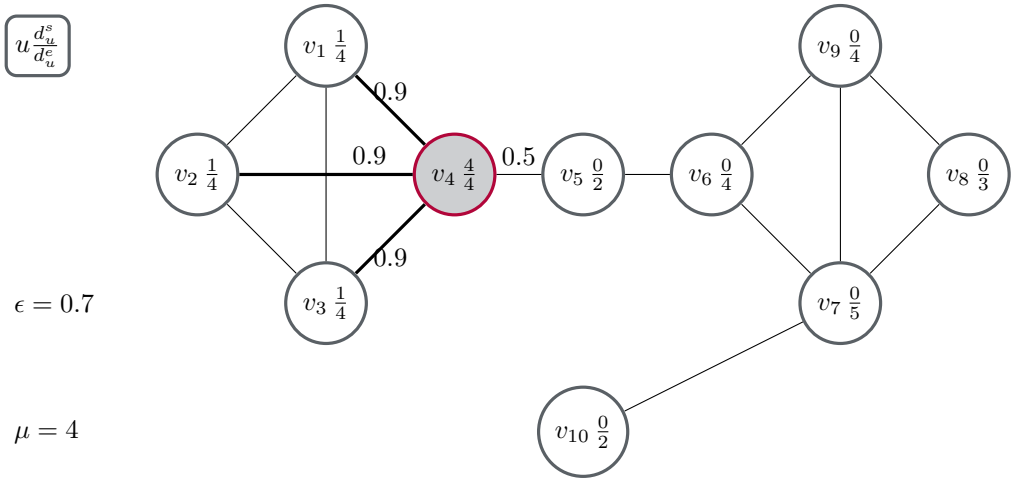
Improvements of pSCAN



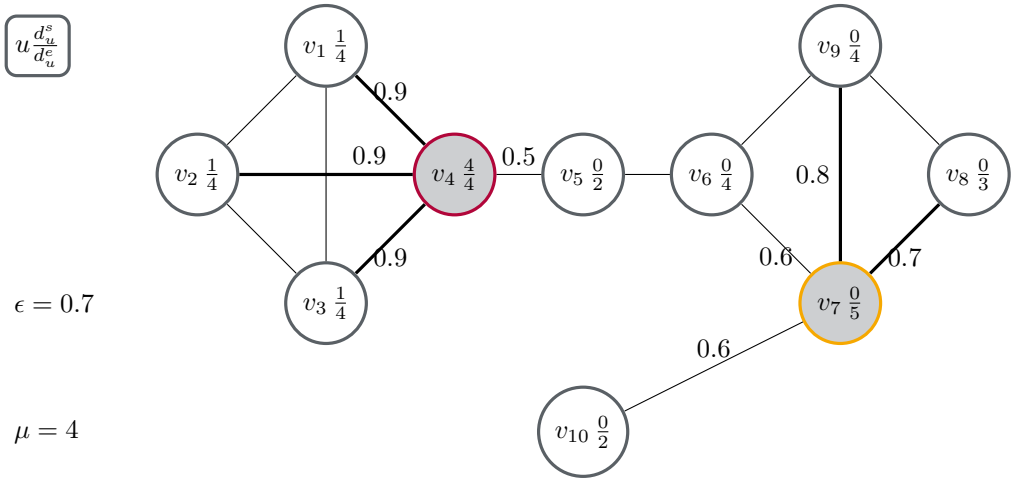
Improvements of pSCAN



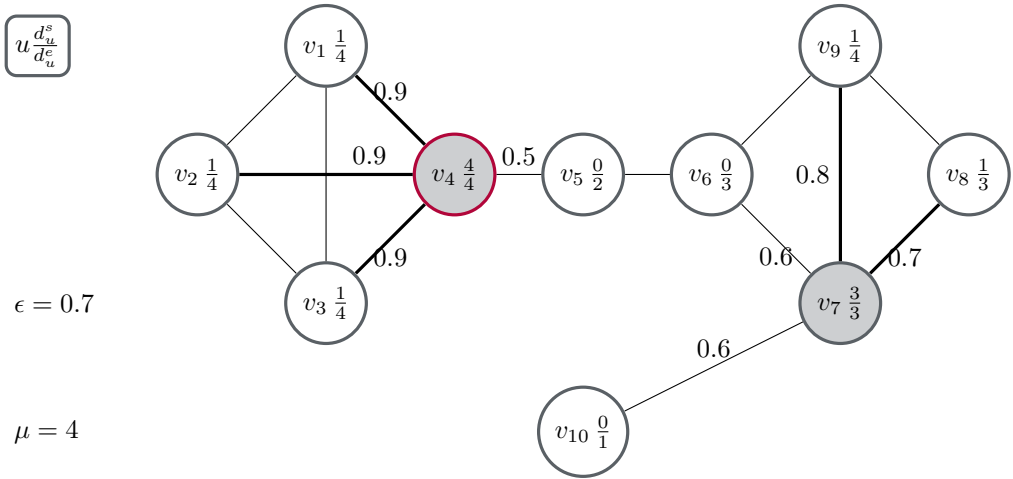
Improvements of pSCAN



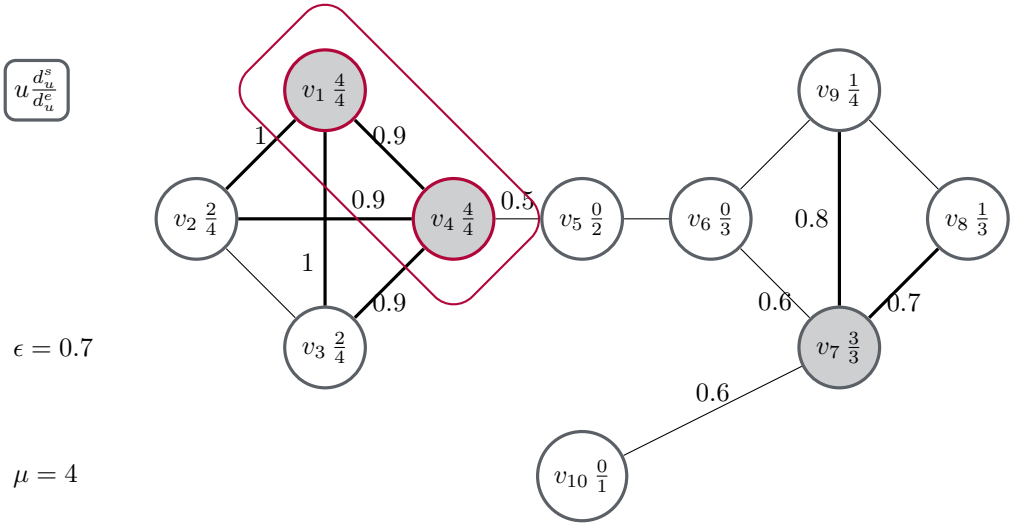
Improvements of pSCAN



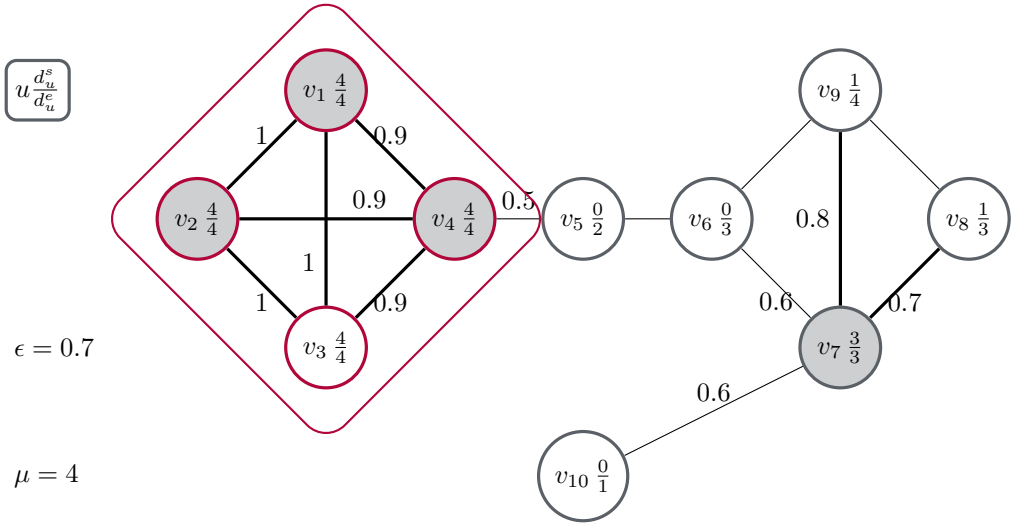
Improvements of pSCAN



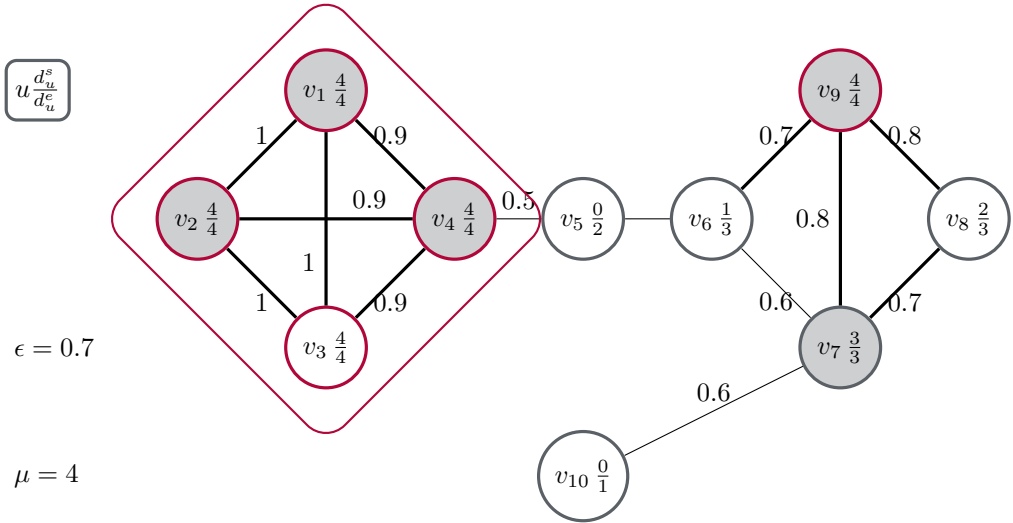
Improvements of pSCAN



Improvements of pSCAN



Improvements of pSCAN



Agenda

Principles of SCAN

Improvements of pSCAN

Experimental Results

Experimental Results

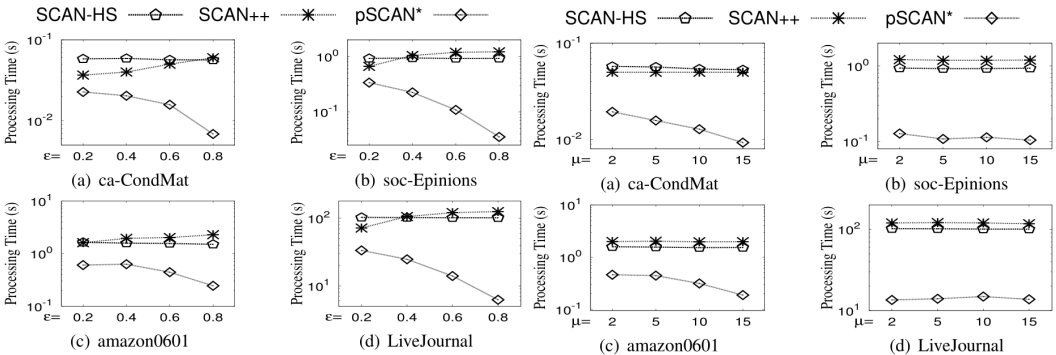
Algorithms:

- SCAN
- SCAN++ (approximative version of SCAN omitting some similarity calculations)
- pSCAN (5 variants with different degrees of optimization)

Data:

- 13 real world networks (13k to 1.2G edges)
- 11 generated networks with varying characteristics

Experimental Results



- ~ 1 order of magnitude faster than either SCAN or SCAN++

Clustering Large Probabilistic Graphs

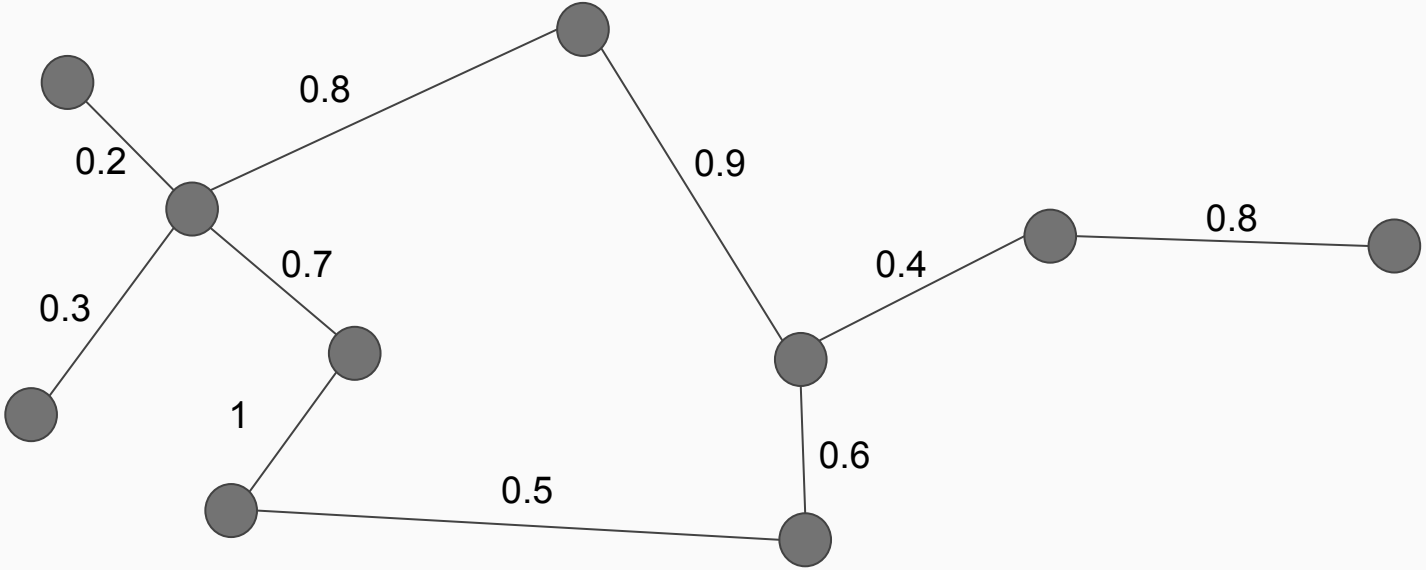
Marianne Thieffry

Agenda

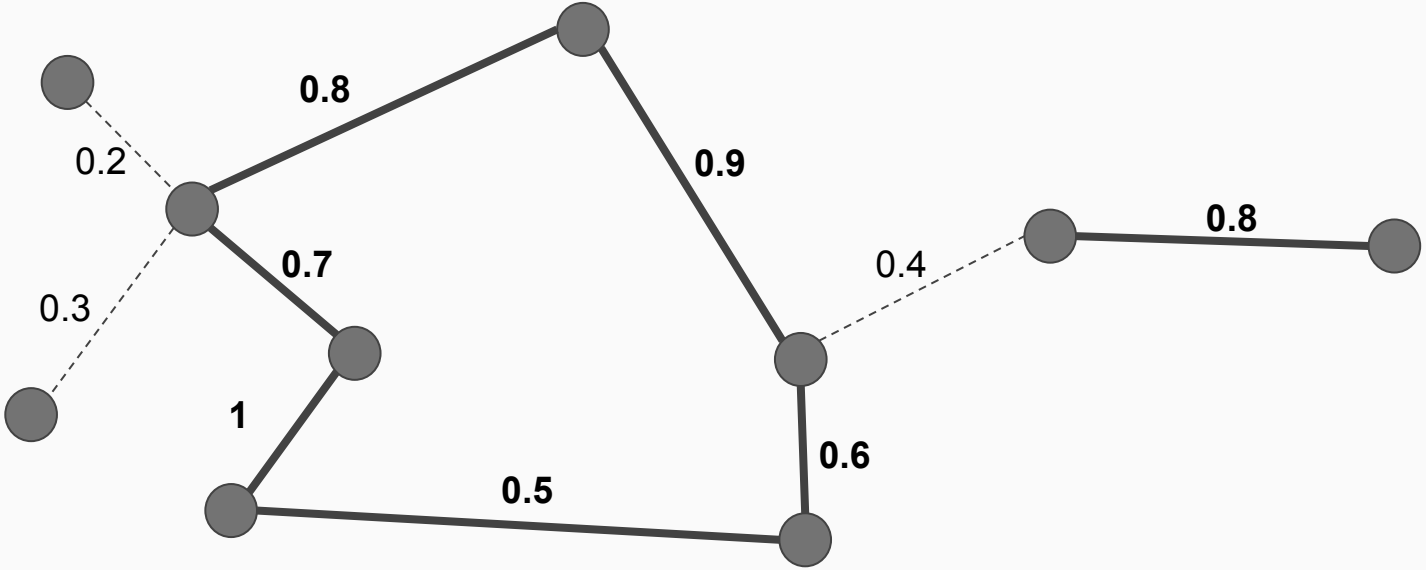
1. Definition of `pClusterEdit`
2. Algorithms for `pClusterEdit`
3. Generalizations of `pClusterEdit`

Definition of pClusterEdit

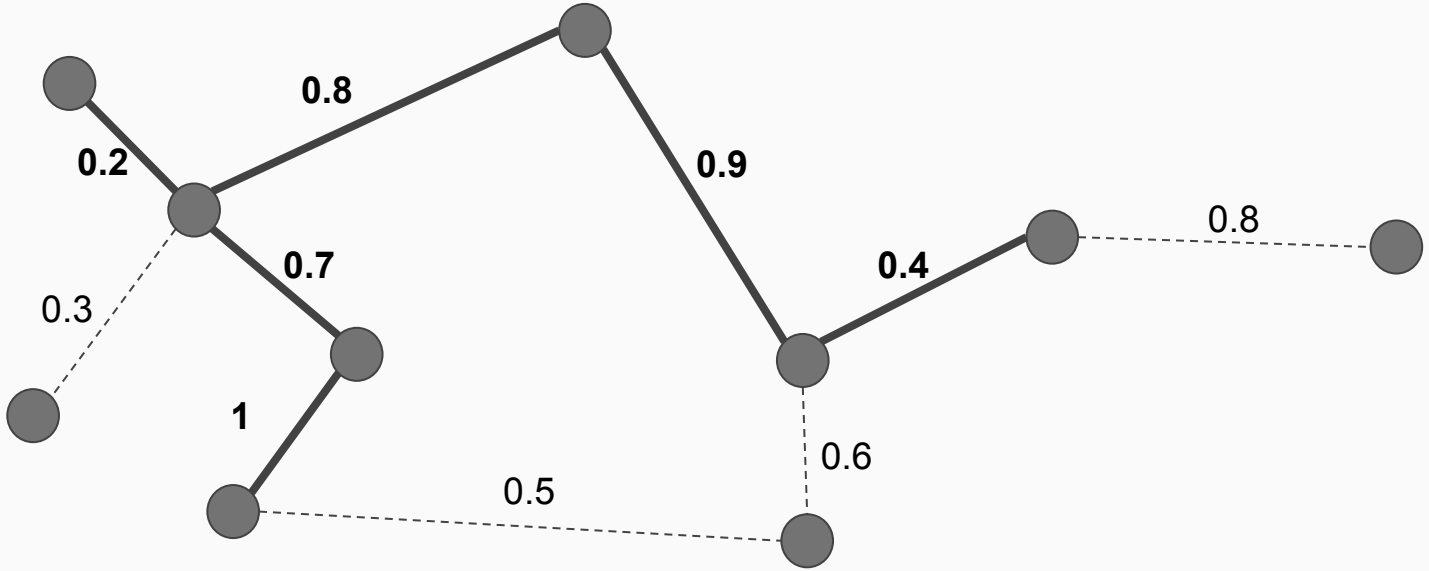
Probabilistic Graphs



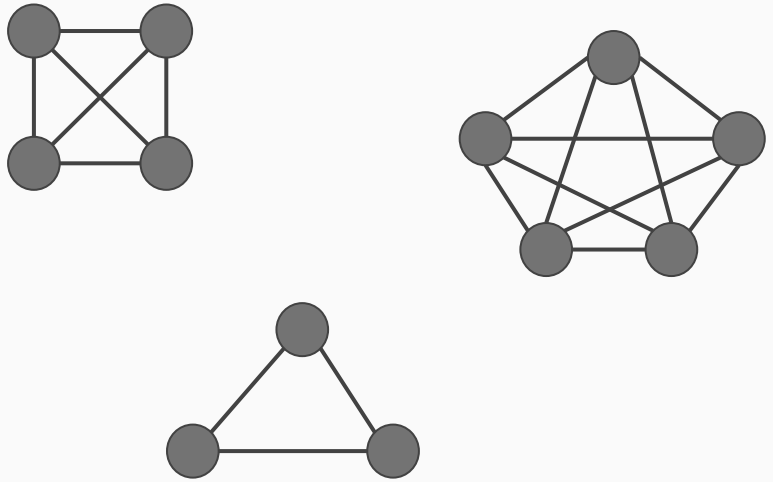
Probabilistic Graphs



Probabilistic Graphs



Cluster graphs



Edit distance

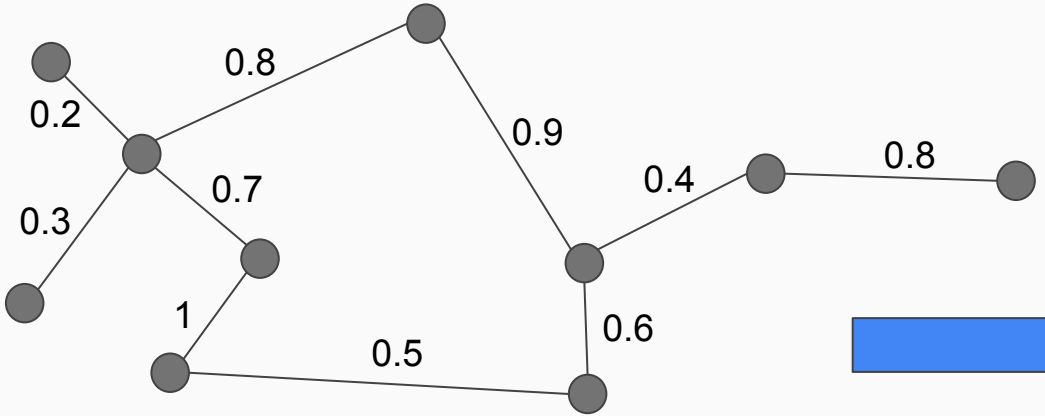
 # edges added + # edges removed

Edit distance

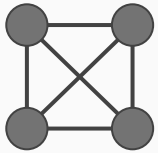
 # edges added + # edges removed

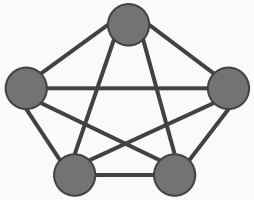
[between all deterministic instances of a given probabilistic graph and a given deterministic graph]

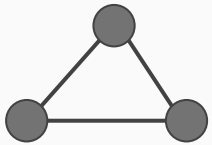
pClusterEdit



[while minimizing
the edit distance]



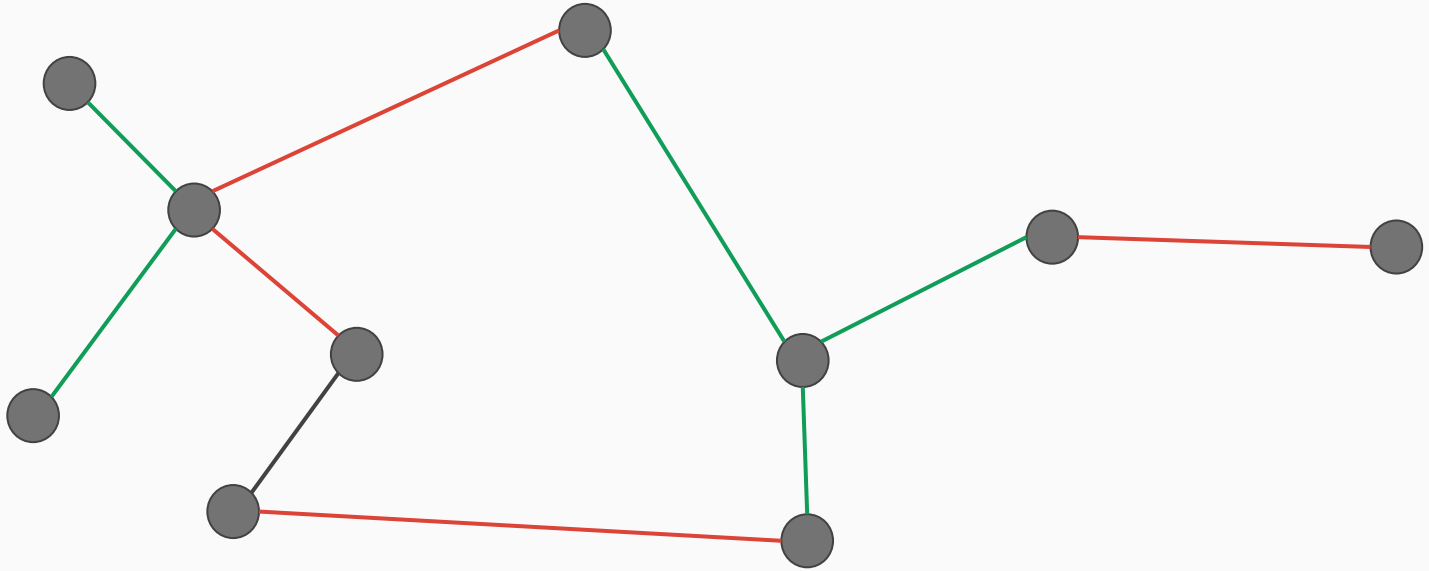




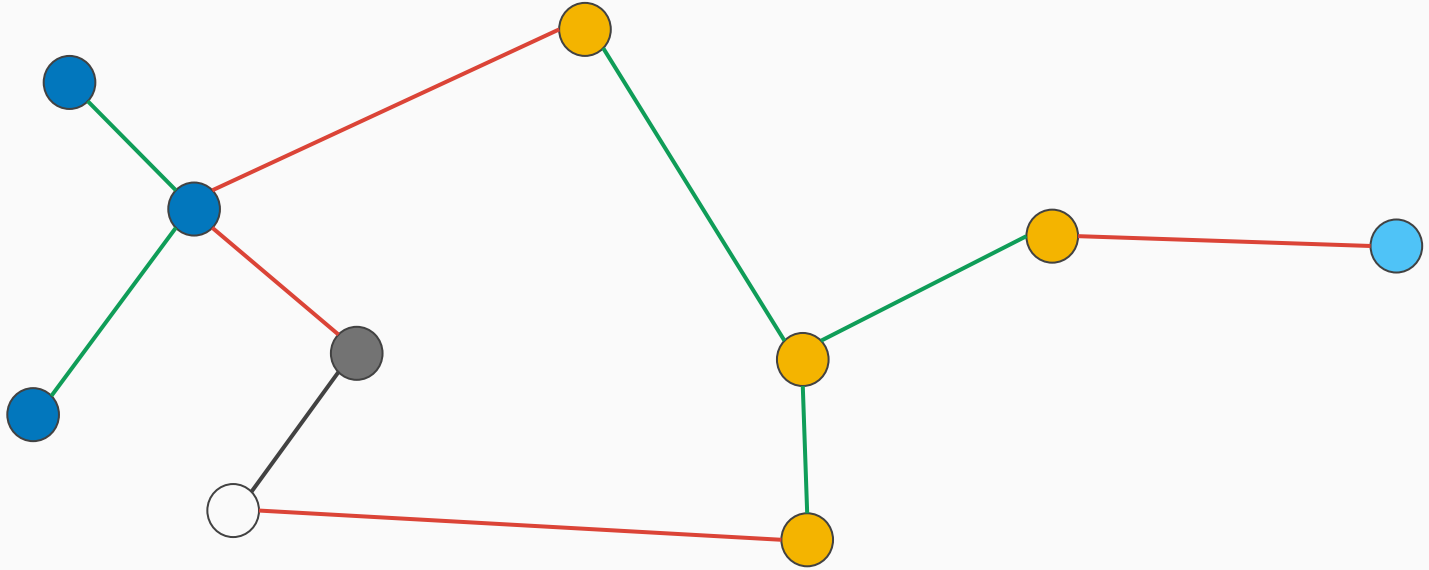
pClusterEdit

$$\mathop{\mathbb{E}}_{G \sqsubseteq \mathcal{G}} \left[\sum_{u<v} X_{uv} \right] = \sum_{u<v} \left(\mathop{\mathbb{E}}_{G \sqsubseteq \mathcal{G}} X_{uv} \right) = \sum_{\{u,v\} \in E_Q} (1 - P_{uv}) + \sum_{\{u,v\} \notin E_Q} P_{uv}$$

CorrelationClustering



CorrelationClustering

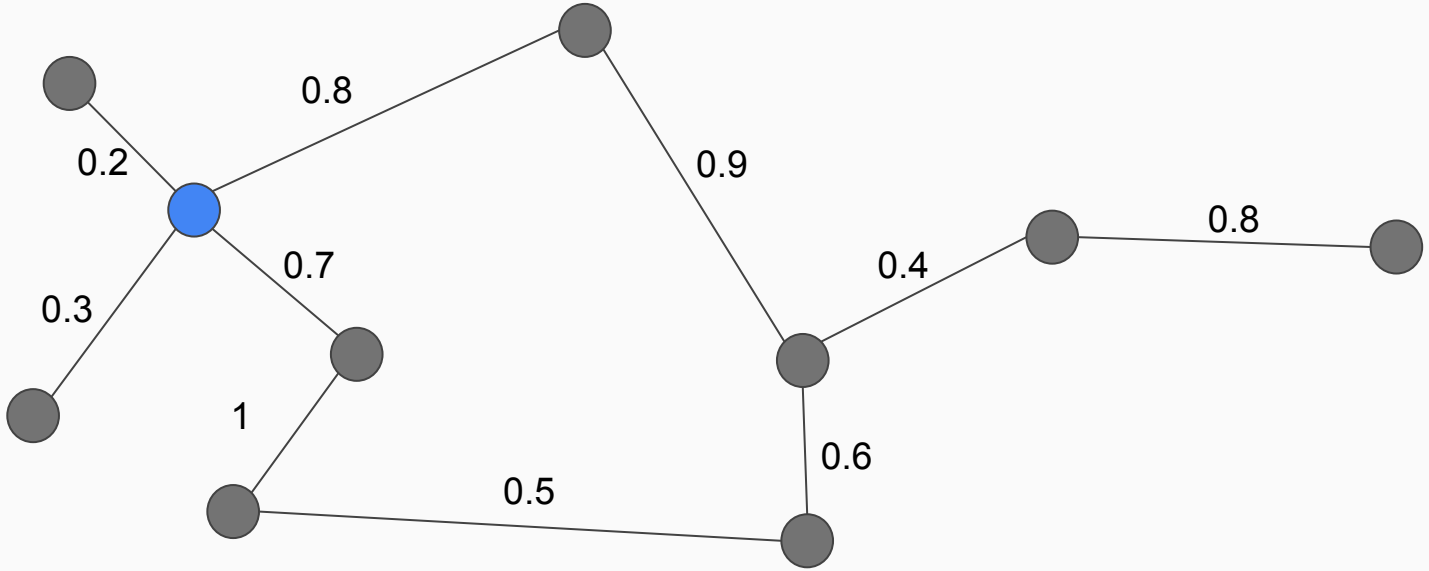


CorrelationClustering

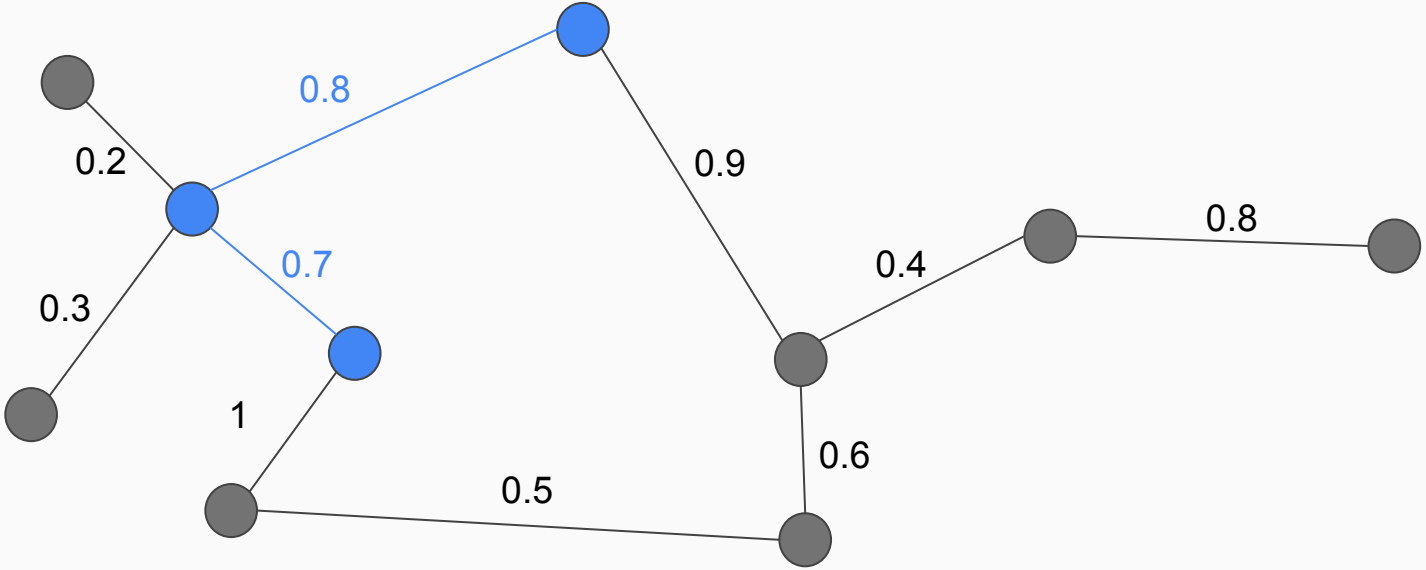
$$\mathrm{Cc}(\mathcal{P}) = \sum_{\substack{(u,v) \\ \mathcal{P}(u)=\mathcal{P}(v)}} W^-_{uv} + \sum_{\substack{(u,v) \\ \mathcal{P}(u)\neq\mathcal{P}(v)}} (1 - W^-_{uv})$$

Algorithms for pClusterEdit

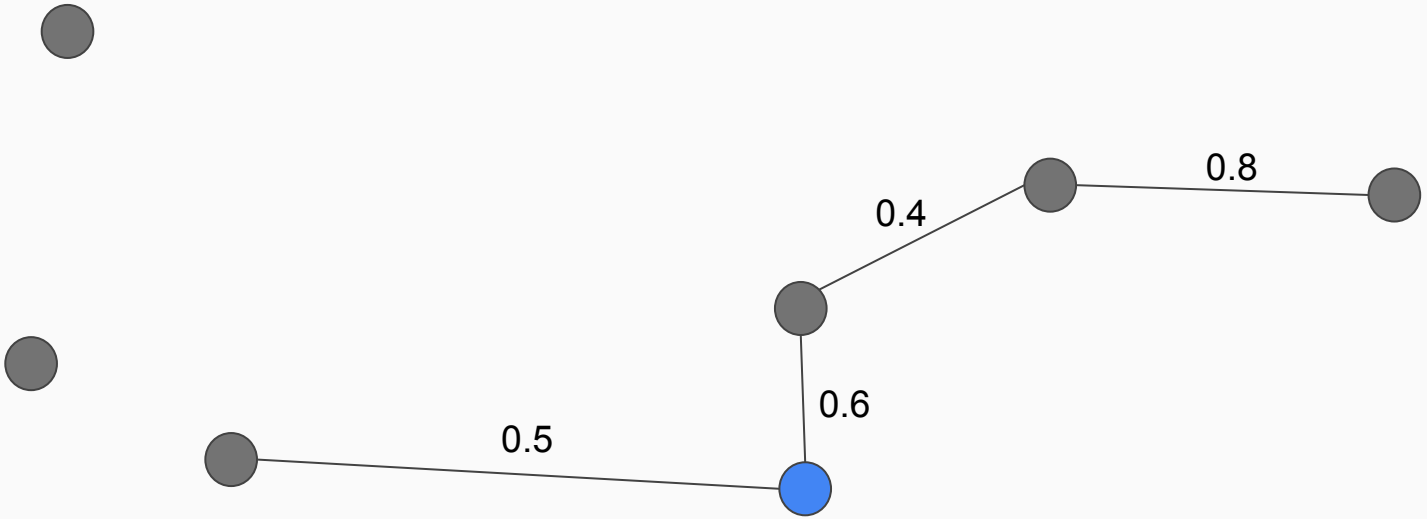
pKwikCluster



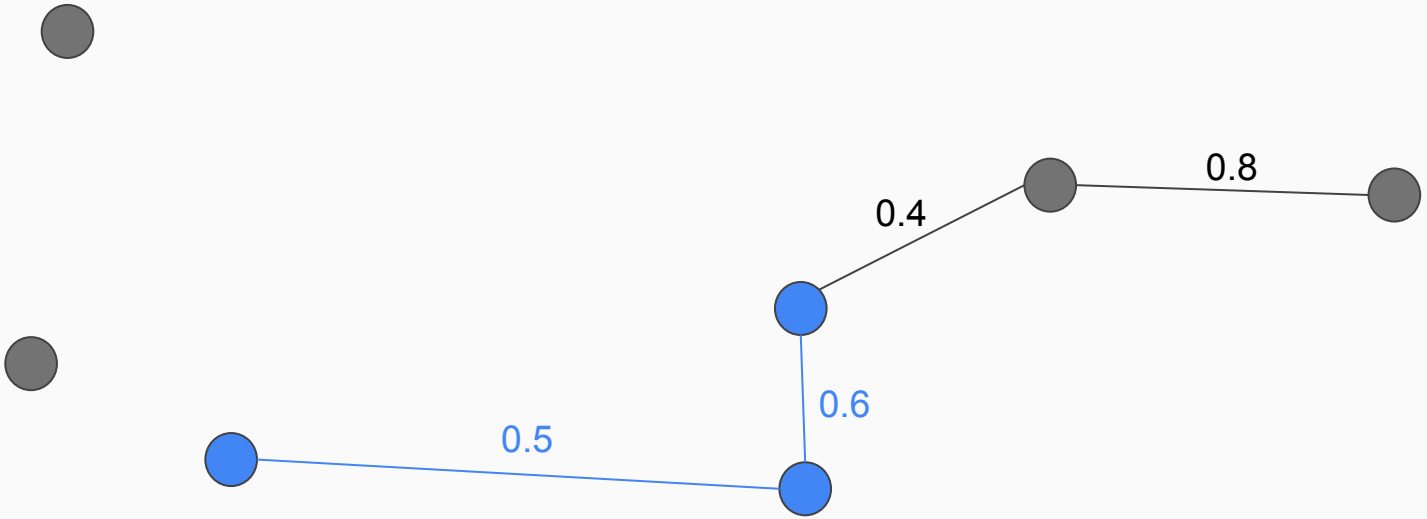
pKwikCluster



pKwikCluster

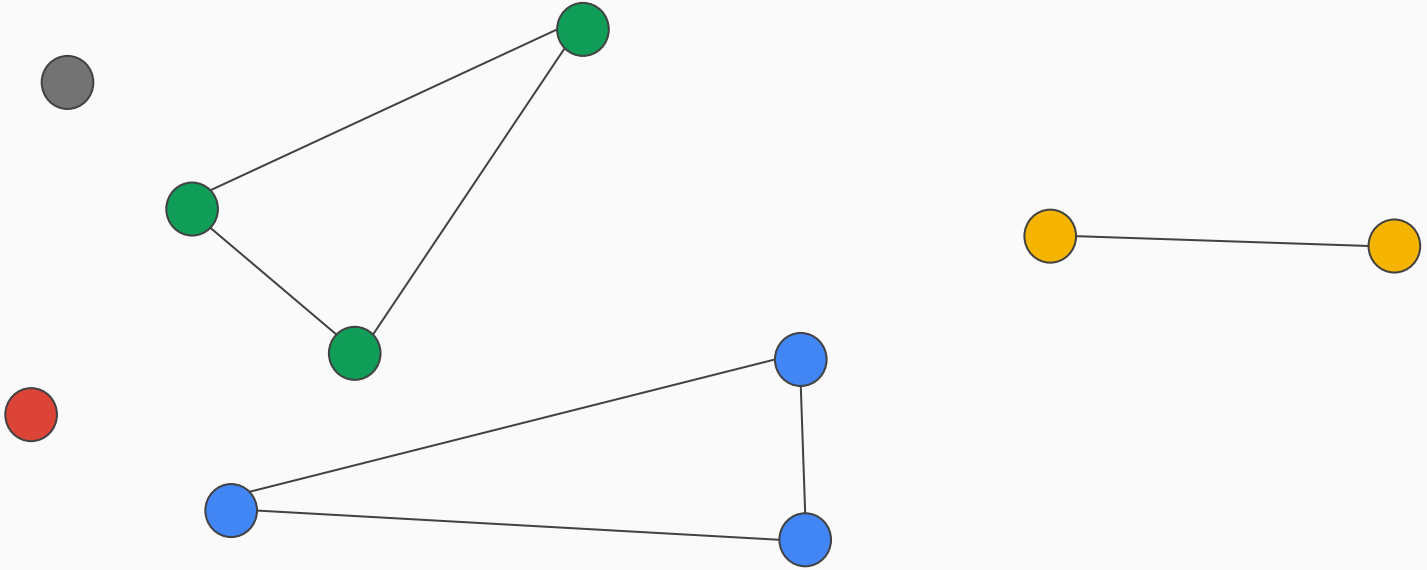


pKwikCluster



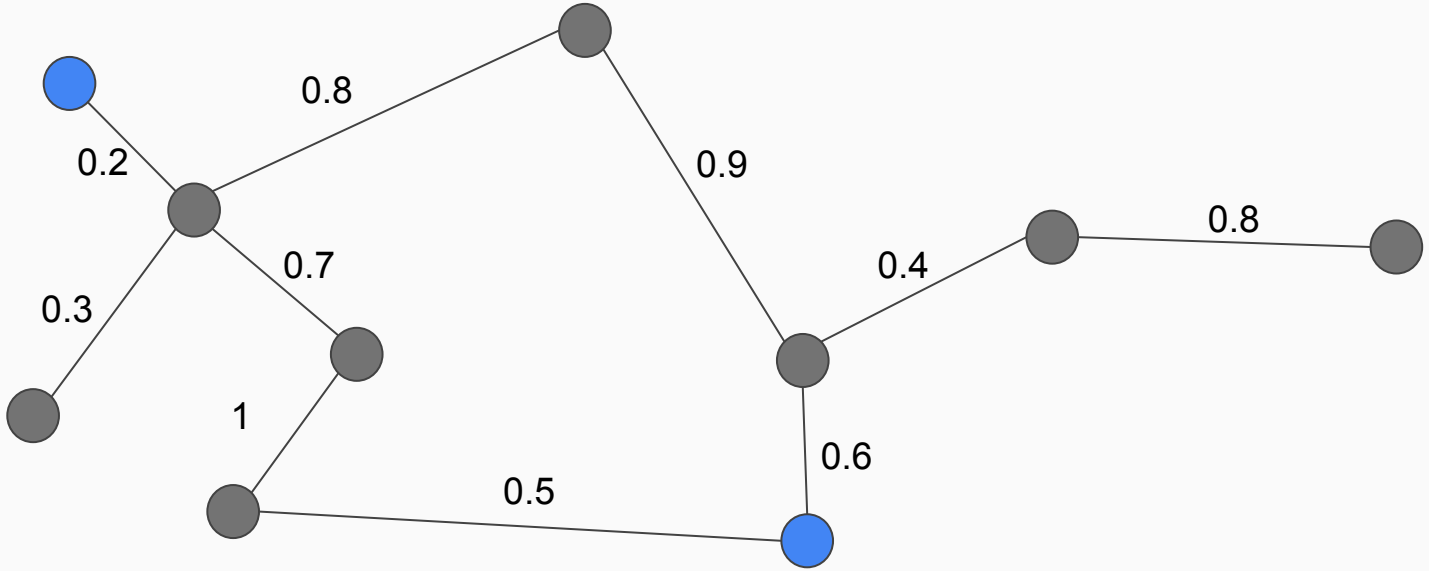
pKwikCluster

pKwikCluster

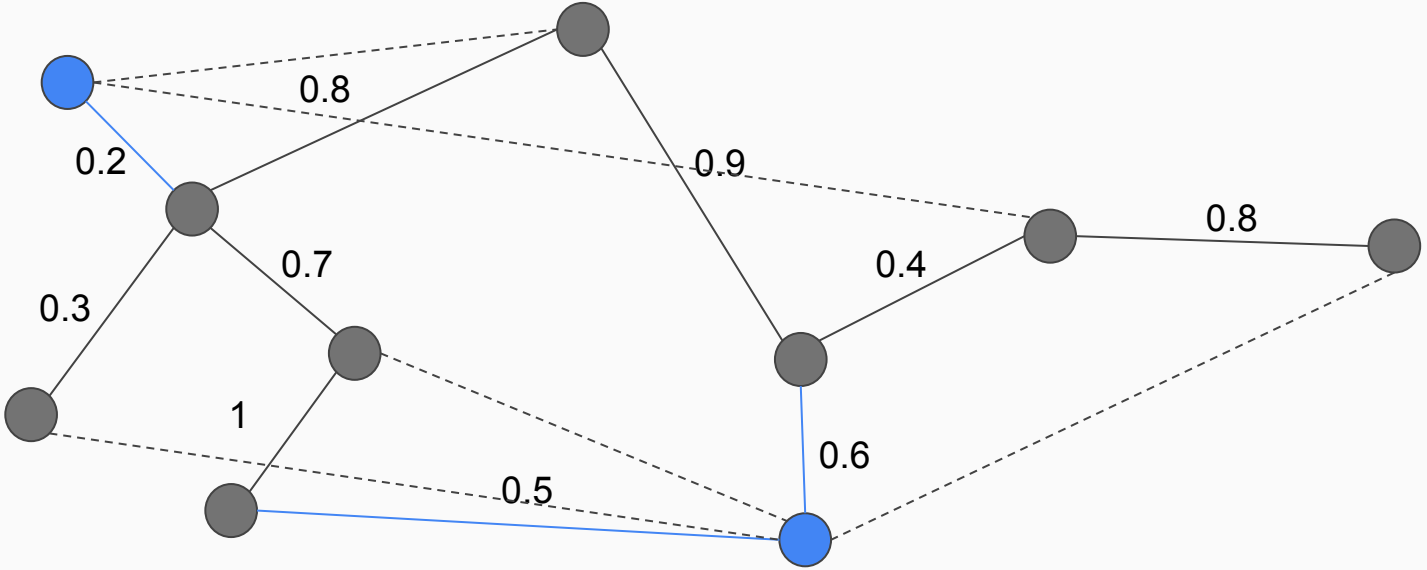


O(m)

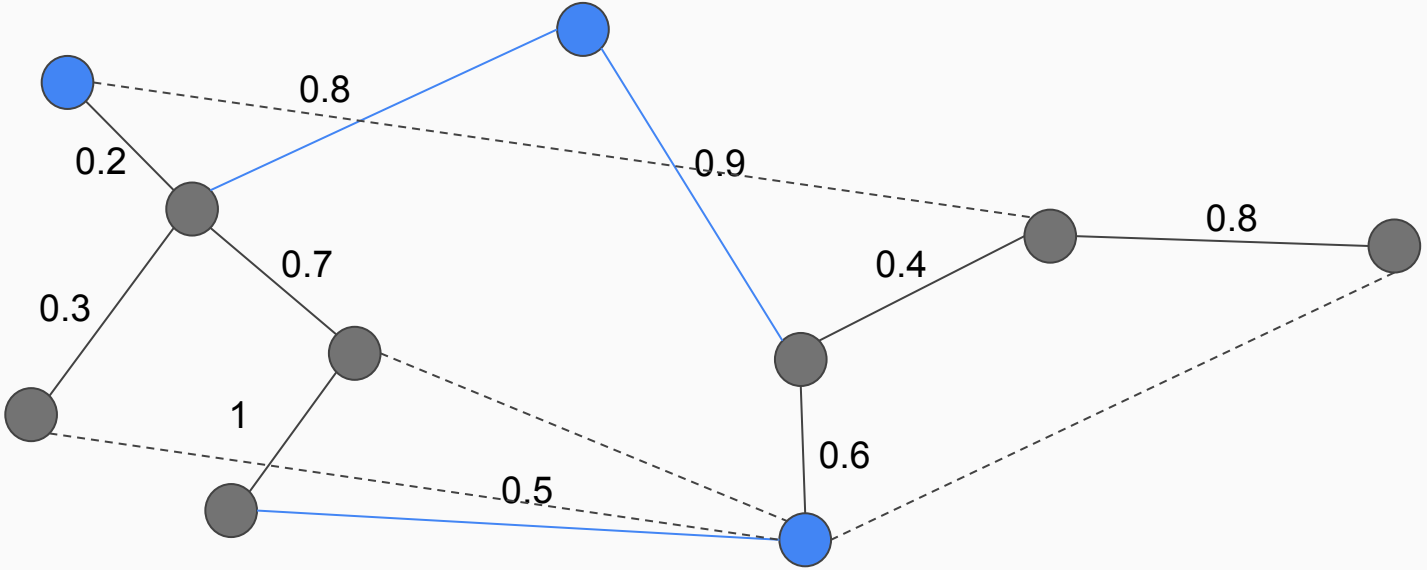
Furthest



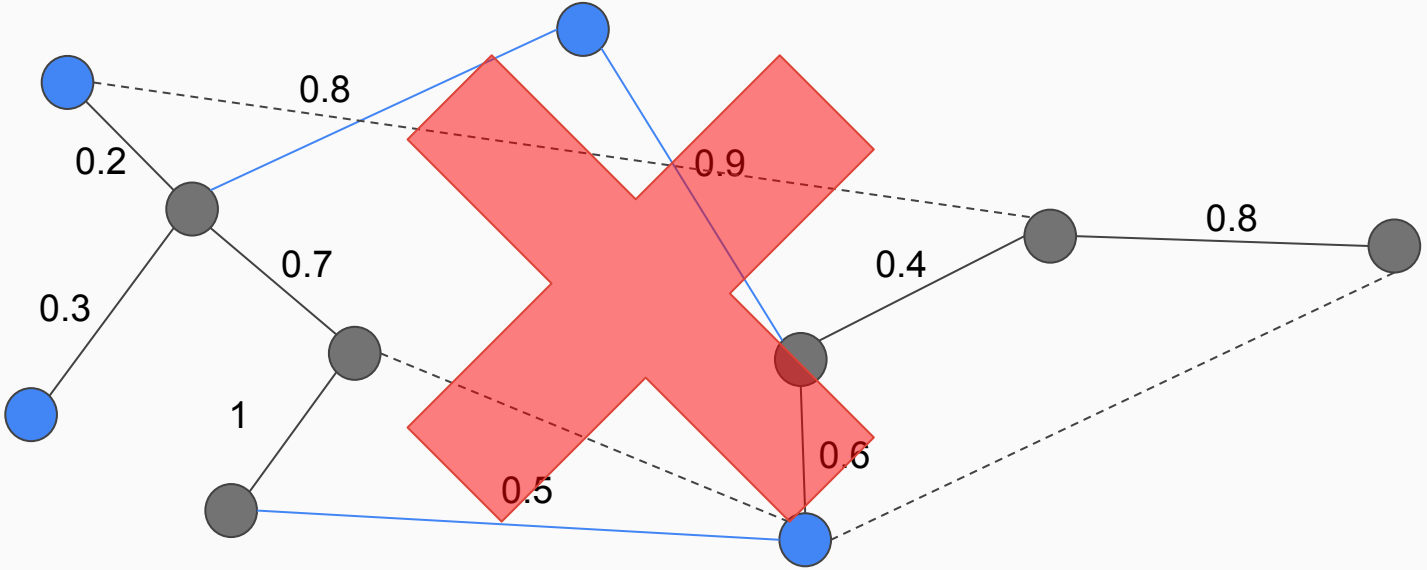
Furthest



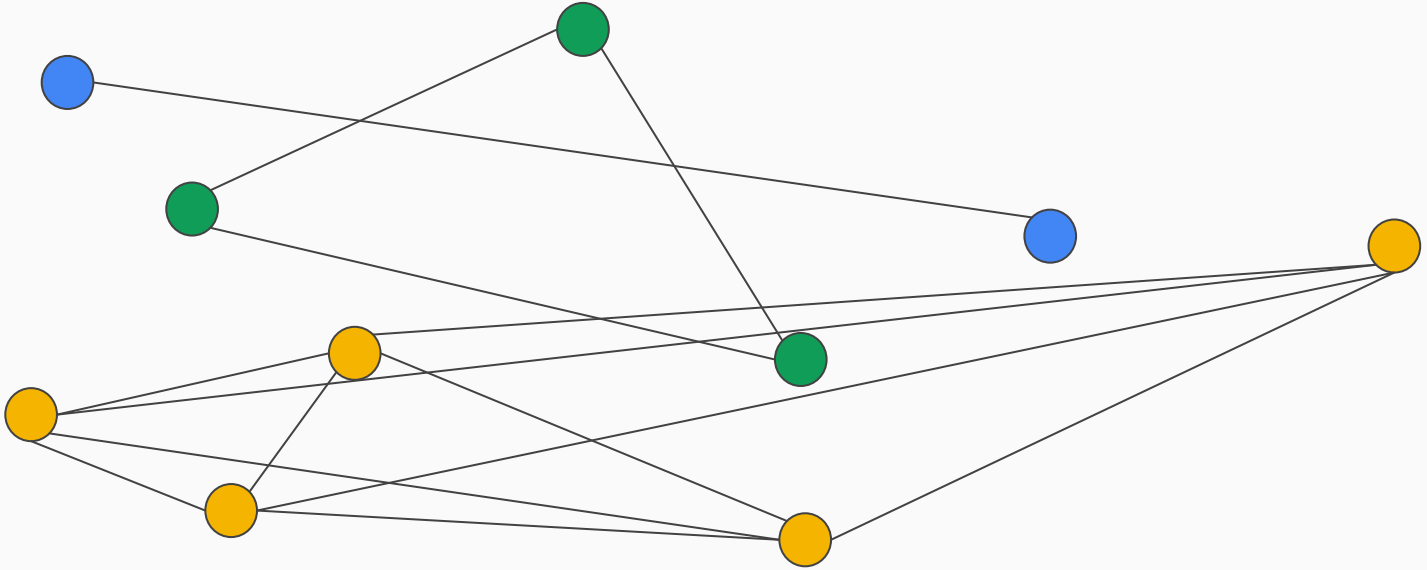
Furthest



Furthest



Furthest



$O(nk^2)$

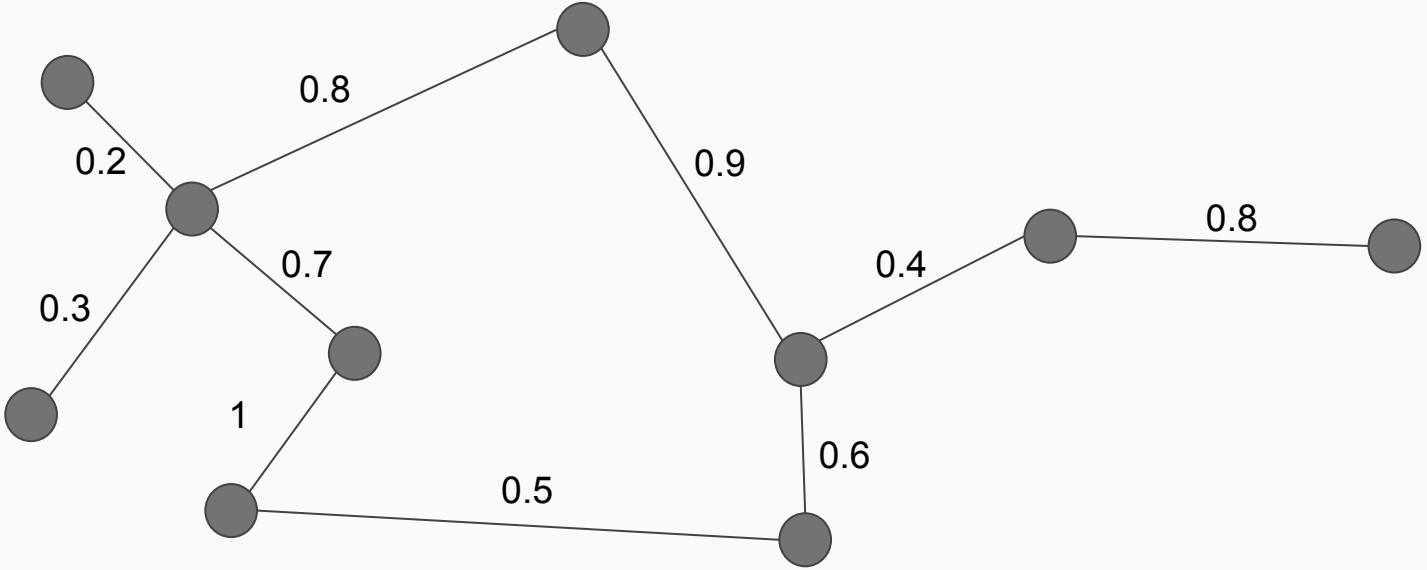
Agglomerative

Singleton clusters

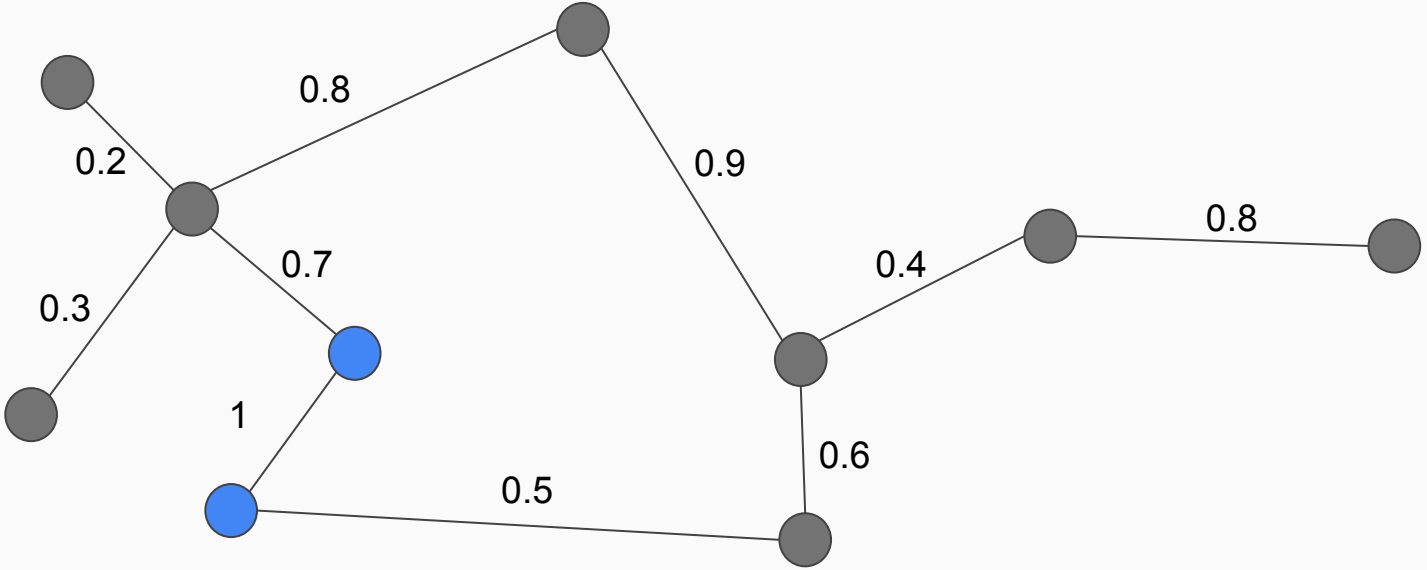
Merge clusters with highest average edge probability, until highest average edge probability is at most 0.5

Complexity O(kn log n)

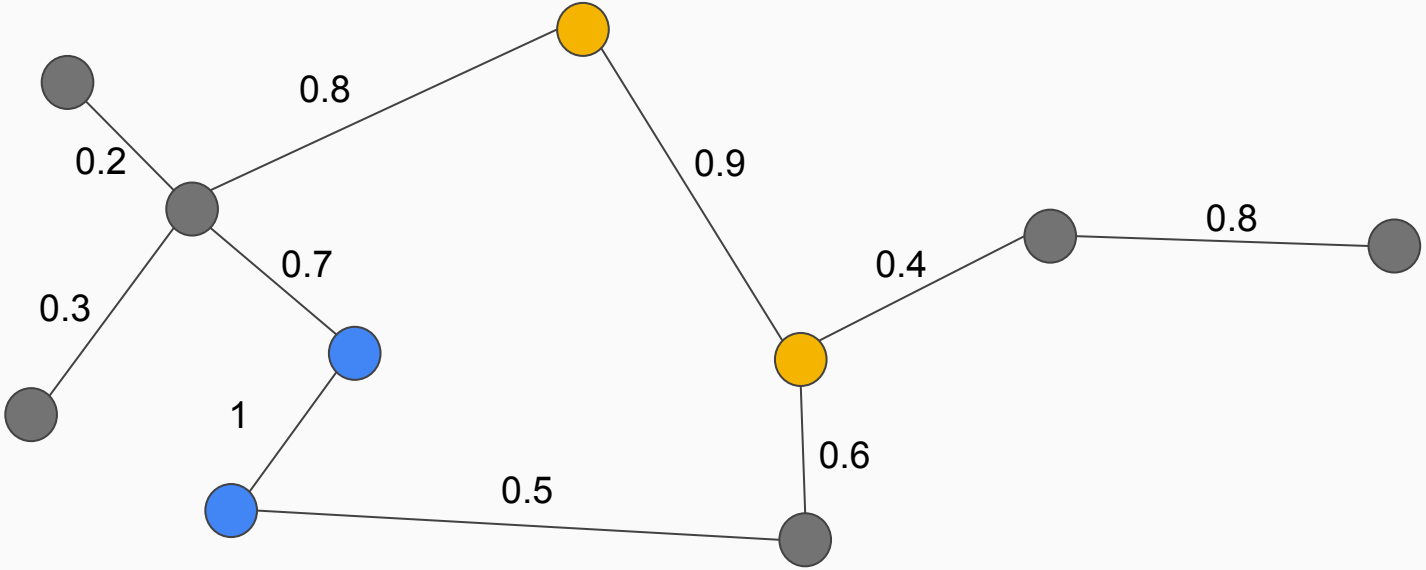
pKwikCluster



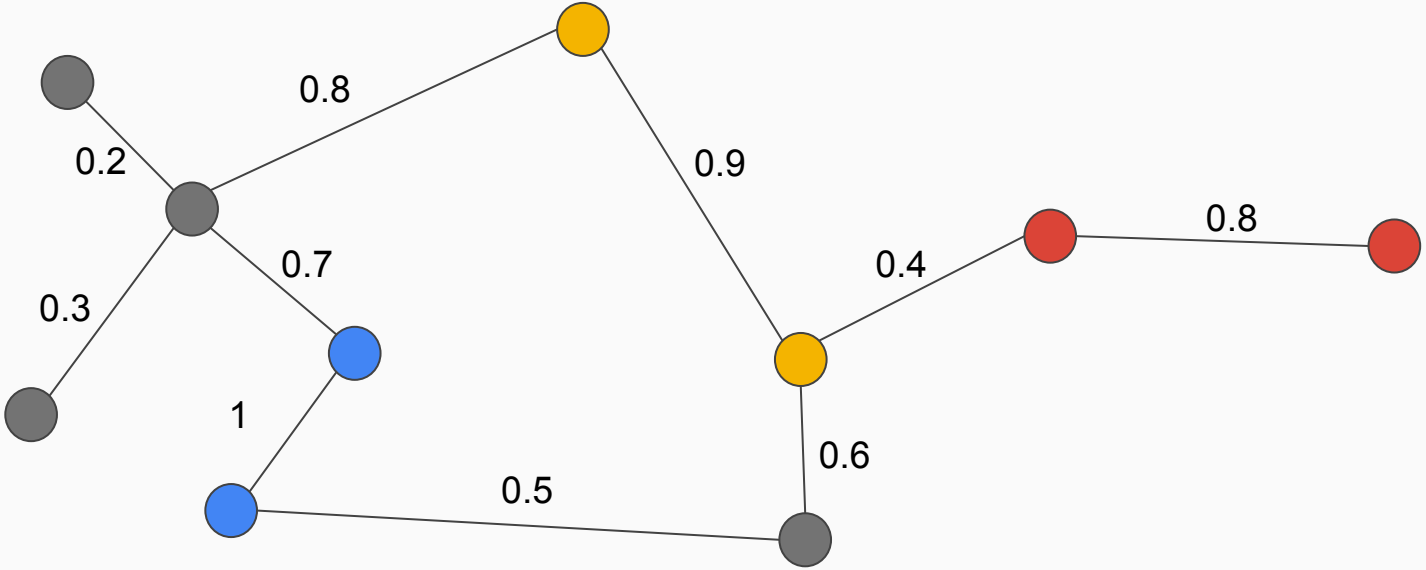
pKwikCluster



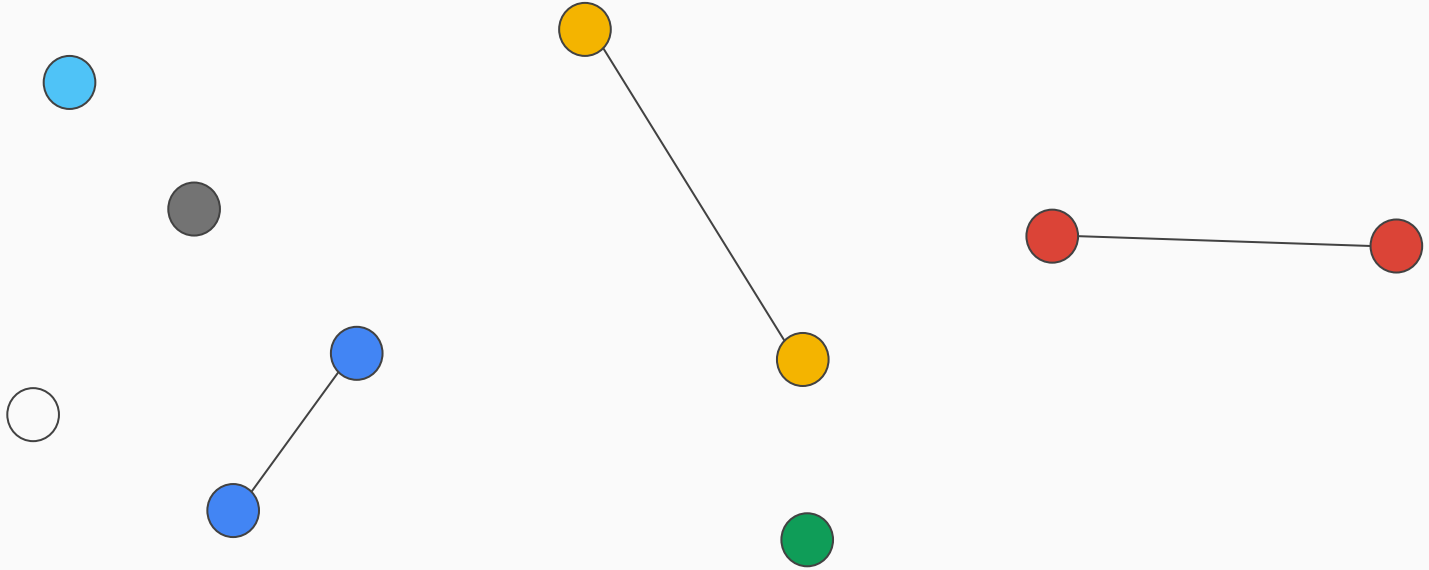
pKwikCluster



pKwikCluster



pKwikCluster



O(kn log n)

Significance

$$\text{E-VAL}(\mathcal{G}, C, R) = \Pr\left[D(\mathcal{G}, C) \leq D(\mathcal{G}, R(C))\right]$$

Evaluation - CORE dataset

- protein-protein interaction network
- 2708 nodes
- 7123 edges

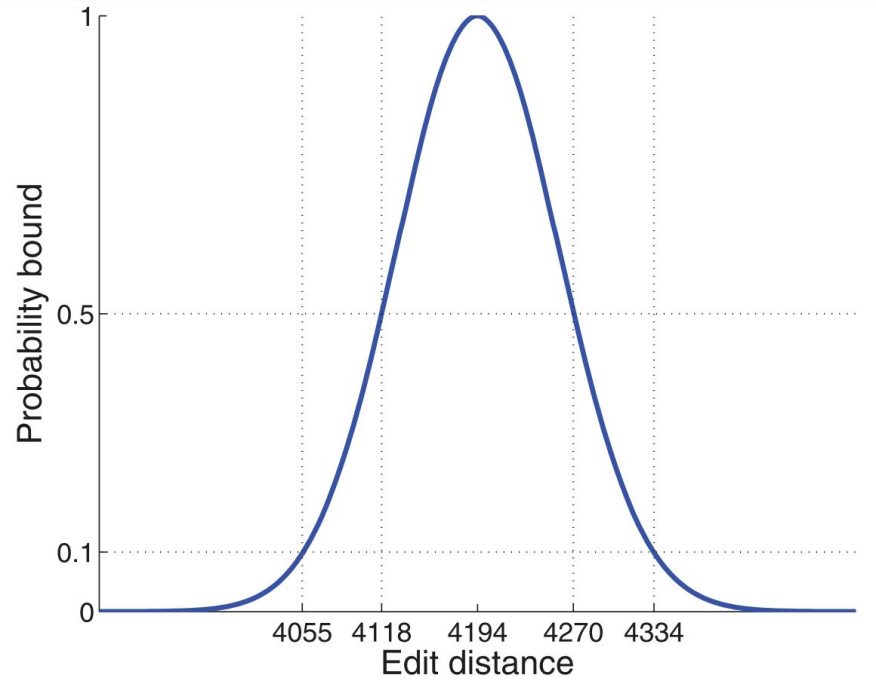
Evaluation - CORE dataset

Algorithm	Edit Distance	Wallclock Time (sec)
Reference	12230	n/a
pKwikCluster	4194	0.005
pCast	4502	100
Agglomerative	3420	10
Furthest	4612	150
Balls	4960	8

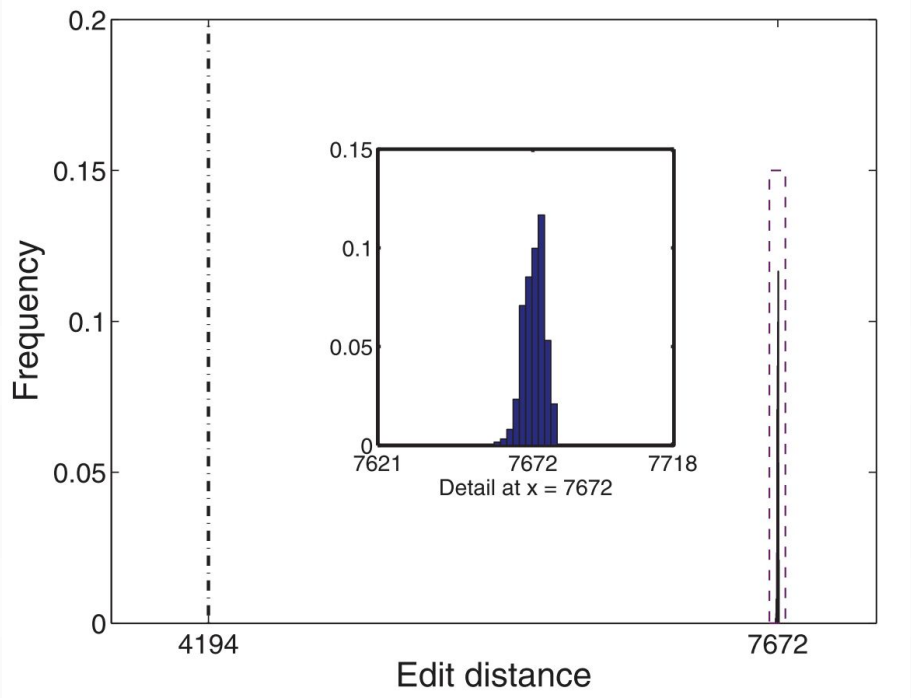
Evaluation - CORE dataset

Algorithm	#clusters	TP	FP	FN
Reference	547	1791	11635	3589
pKwikCluster	491	838	2003	4542
pCast	1189	633	1338	4747
Agglomerative	543	946	1357	4434
Furthest	619	894	2322	4486
Balls	757	1120	1743	4260

Evaluation - CORE dataset

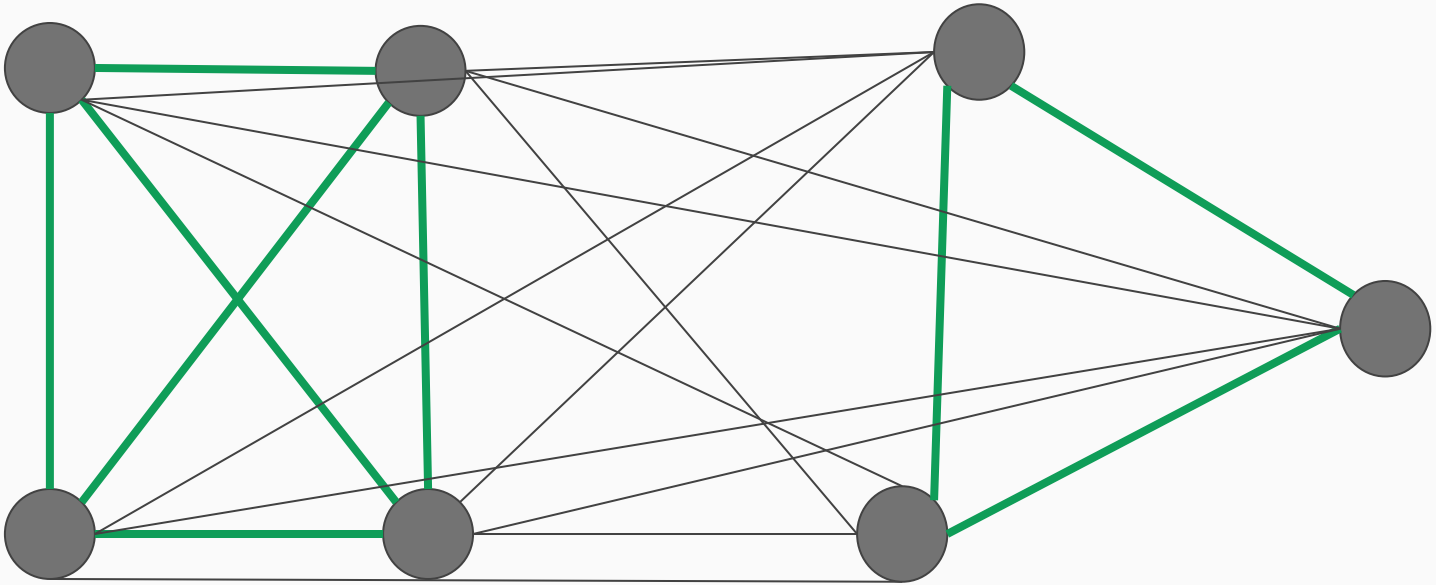


Evaluation - CORE dataset



Generalizations of pClusterEdit

Noisy cluster graphs



p2ClusterEdit & minα

- Find α-cluster with minimal edit distance
- Find α that produces minimum α-cluster with minimal edit distance

The shortest Path is not always a straight line

From Vasiliki Kalavri, Tiago Simas & Dionysios Logothetis

Philippe ZOU

Graph Mining - WS 2016/2017

Davide Mottin & Konstantina Lazaridou

Agenda

1. Overview
2. Implementation
3. Evaluation
4. Conclusion

Overview

Motivations

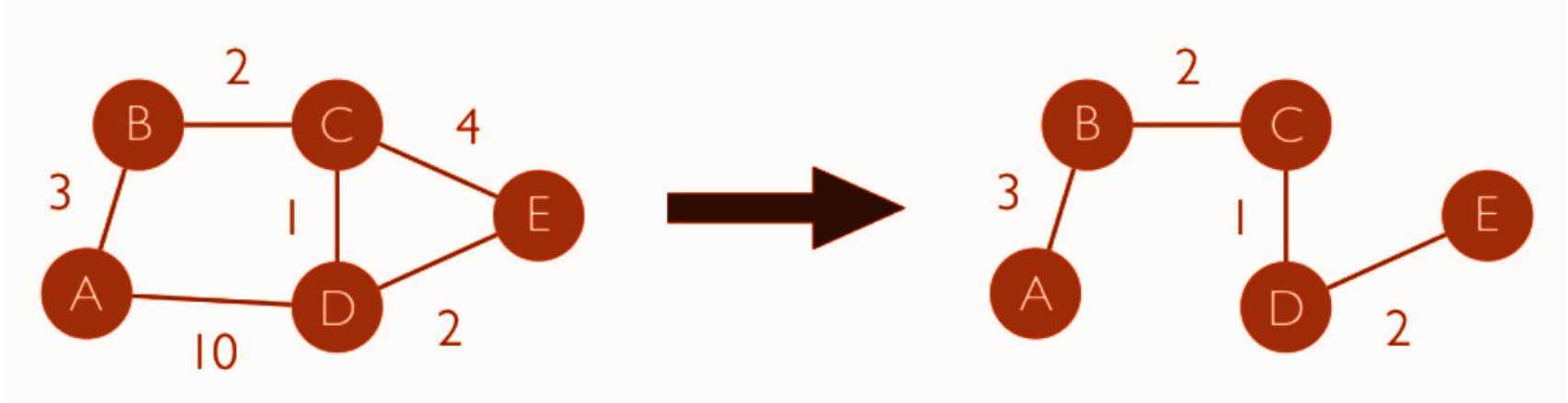
Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

- Leverage the Backbone metric
- “The Blackbone metric is the minimum subgraph that preserves the shortest paths of a weighted graph”
- Improve graph analysis tools and computing performance
  - Large graph
  - reduce running time and storage

Backbone Metric

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

For instance, in the example they give :

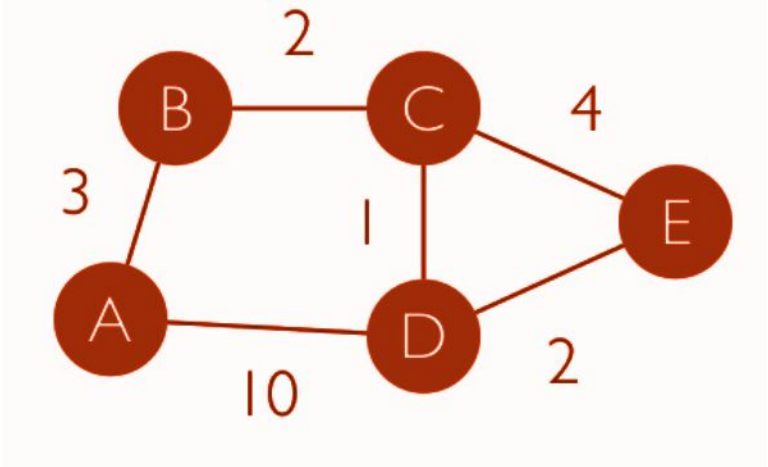


Concept of semi-metricity

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

An edge is semi-metric if you can find an indirect shorter path between the edges.

Let consider the previous graph

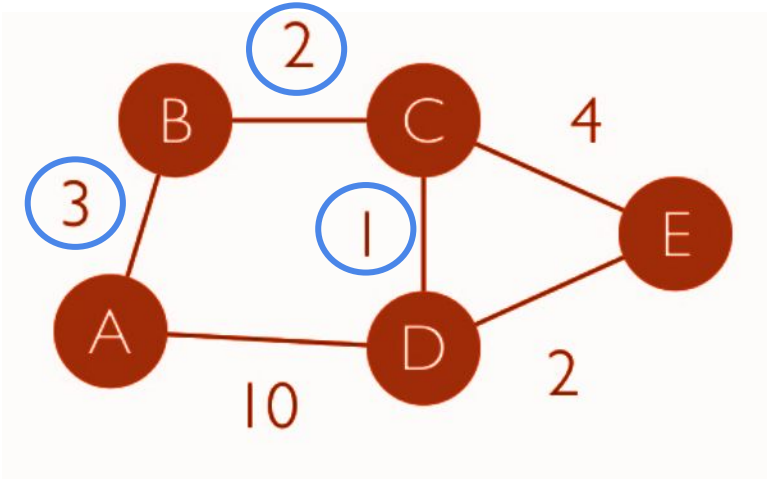


Concept of semi-metricity

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

An edge is semi-metric if you can find an indirect shorter path between the edges.

Let consider the previous graph

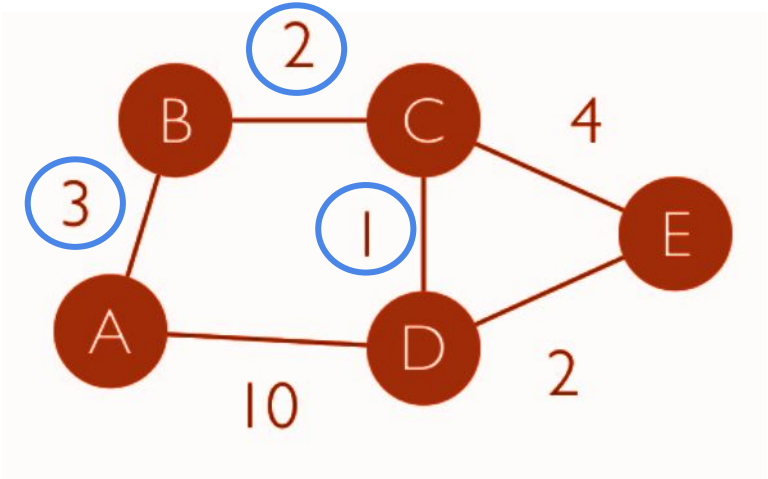


3 + 2 + 1 = 6 < 10
AD is semi-metric

(same for the node CE)

Concept of semi-metricity

Semi-metricity Degree : number of edges to add to reach the shortest path compared to the original (straight) one

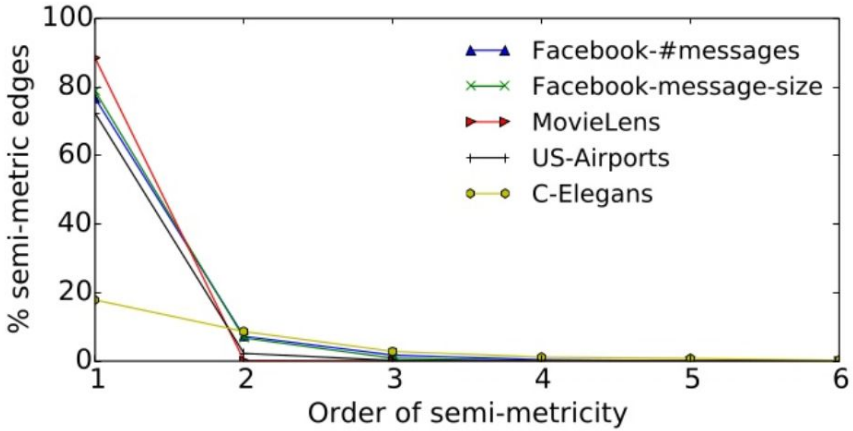


AD will be 2nd order

(and CE will be 1st order)

Concept of semi-metricity

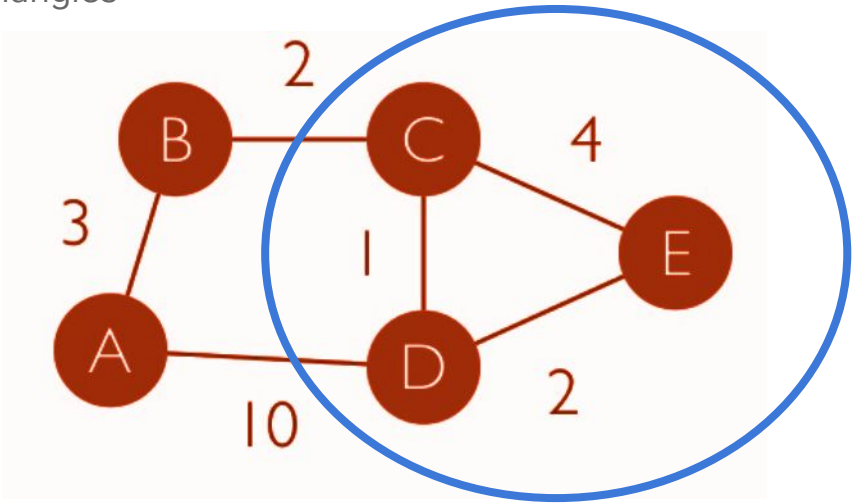
Performed an analysis on data sets



Implementation

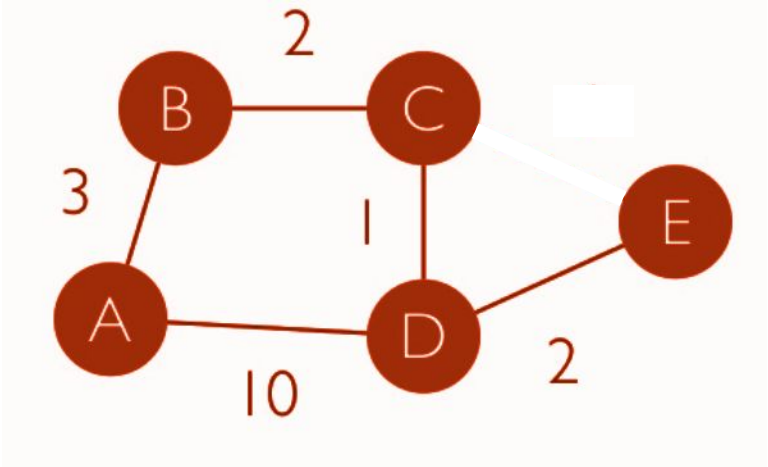
Backbone algorithm

- 3 phases
  - Find 1st order semi-metric edges
    - Only look at triangles



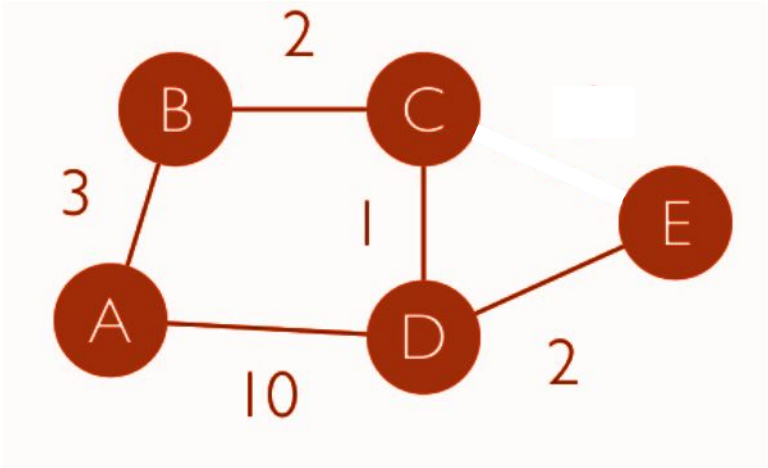
Backbone algorithm

- 3 phases
  - Find 1st order semi-metric edges
    - Only look at triangles



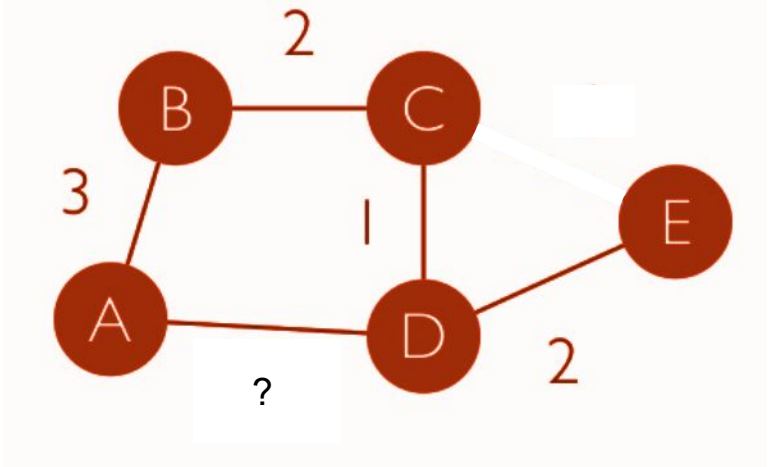
Backbone algorithm

- 3 phases
  - Focus on 2nd order semi-metric edges



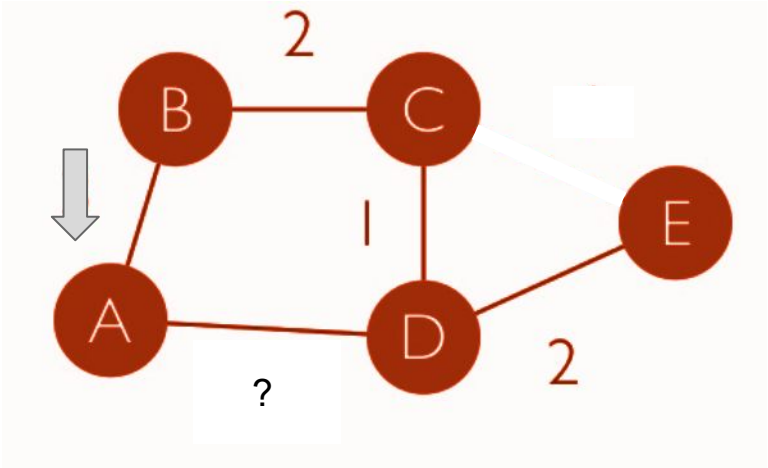
Backbone algorithm

- 3 phases
  - Focus on 2nd order semi-metric edges
    - Lowest weight edge of every node is metric



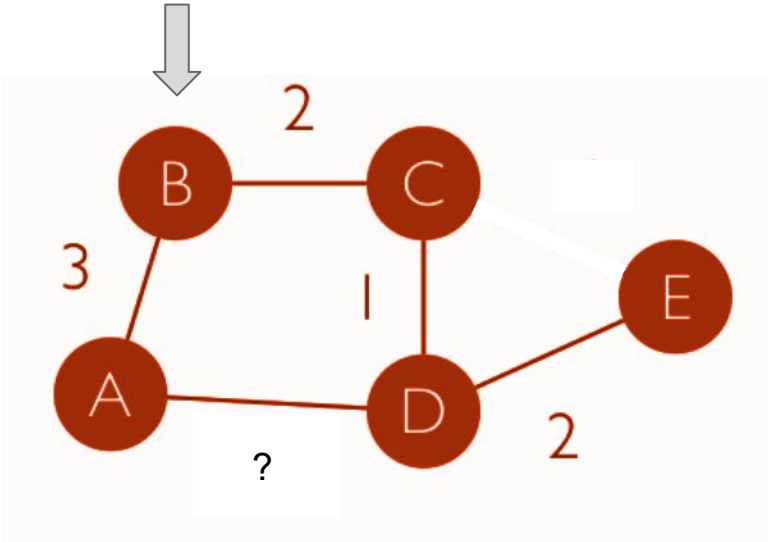
Backbone algorithm

- 3 phases
  - Breadth first search



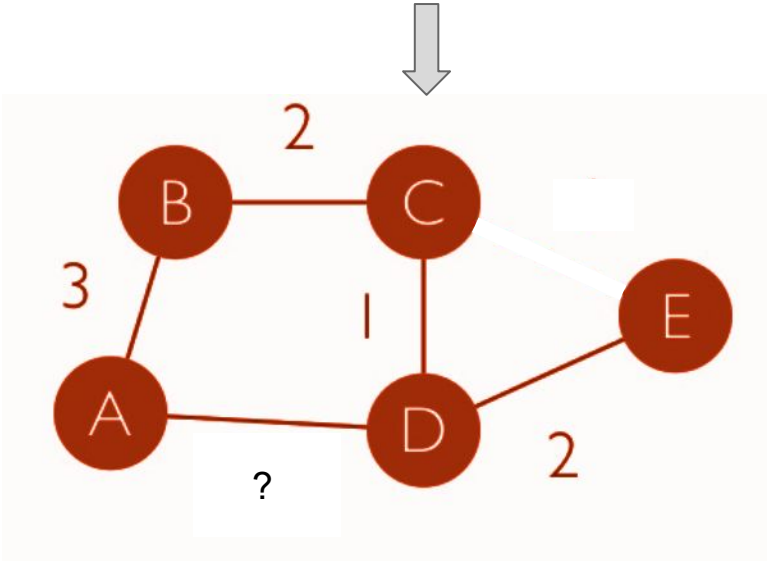
Backbone algorithm

- 3 phases
  - Breadth first search



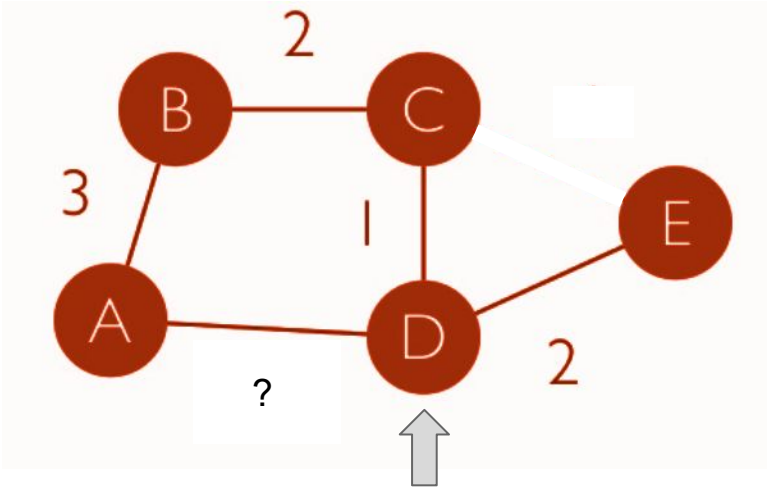
Backbone algorithm

- 3 phases
  - Breadth first search



Backbone algorithm

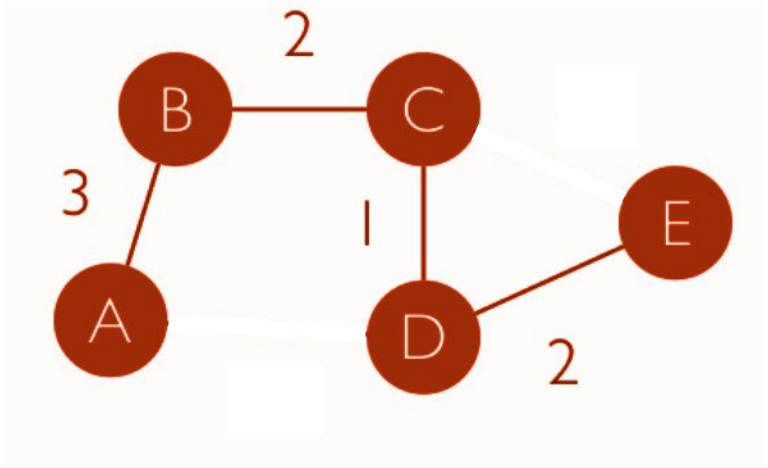
- 3 phases
  - Breadth first search



Target reached but with a shorter path -> Semi-metric edge

Backbone algorithm

- 3 phases
  - End of algorithm



Evaluation

Execution time for phase 1

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

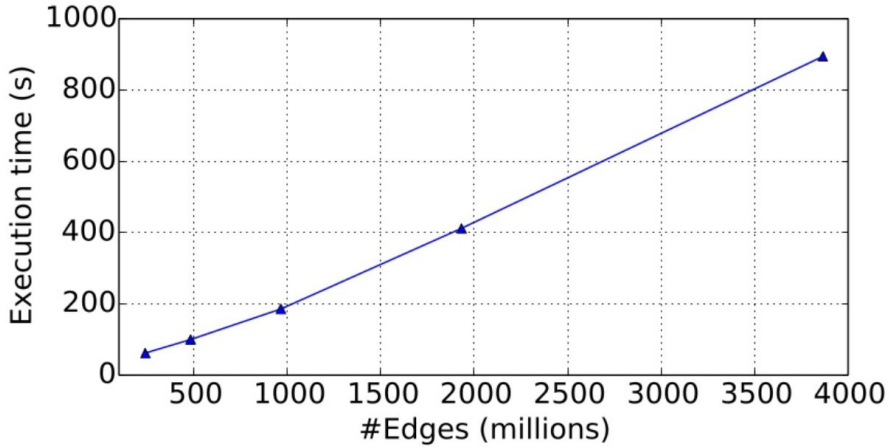
Execution time for finding triangles + removal 1st order semi-metric

Graph	|E|	Size (GB)	Time (s)
Twitter [32]	1.5B	72	3792
Tuenti [4]	685M	33	1305
LiveJournal [8]	34M	1.6	62
NotreDame-web [7]	1.5M	0.07	25
Twitter egonet [38]	1.7M	0.08	32
DBLP [52]	1M	0.05	20

Execution time for phase 1

Increase linearly.

15 Minutes for 4 Billions edges.



Storage

Size reduction of the network after removal

Graph	% of size reduction
Tuenti [4]	59.14
LiveJournal [8]	39.66
NotreDame-web [7]	16.67
DBLP [52]	22.62
Twitter egonet [38]	57.14

Conclusion

Conclusion

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

- Provide a new concept in graphs : semi-metricity

- Algorithm to lower the number of edges in order to improve computation and reduce time
  - show that direct nodes are not necessarily the shortest ones

- Challenge the large graphes issue