湖南大學
HUNAN UNIVERSITY

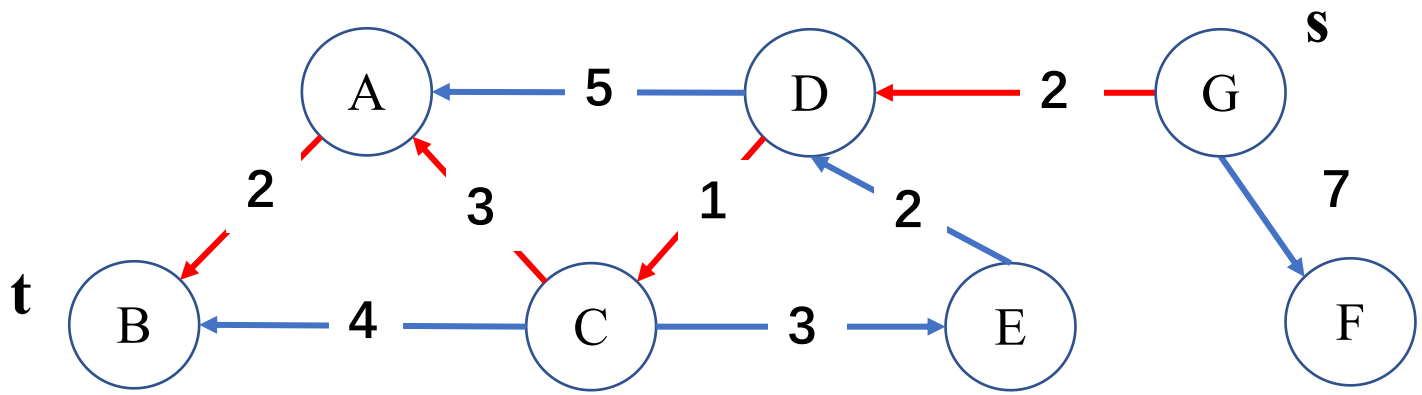
# 第十四章 图算法前沿

湖南大学信息科学与工程学院

# 目录

# 两点间最短路径

给定一个带权图G=(V, E, W)和两个点s和t， 计算G上从s到t的距离$\delta(s,t)$



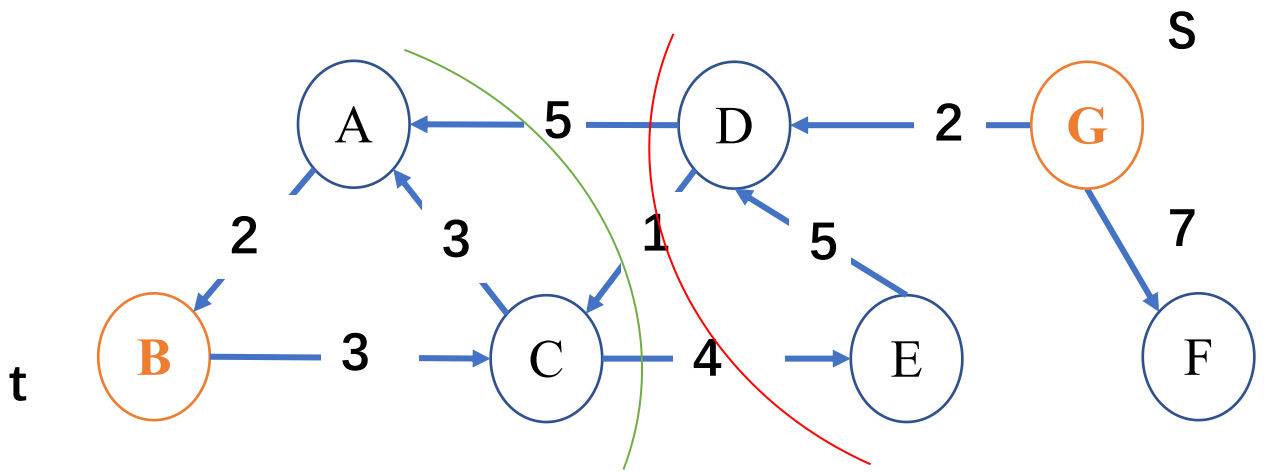
# 基本算法

```
1  Dijkstra(G, Adj, s)
2    Initialize();
3    Q = V
4    while Q ≠ ∅ ;
5      u = EXTRACT-MIN(Q); S = S ∪ {u};
6        if u == t
7            return δ(s, t)
8      for each v ∈ Adj[u]:
9          Relax(u,v,G)
```

遇到t的时候
直接终止

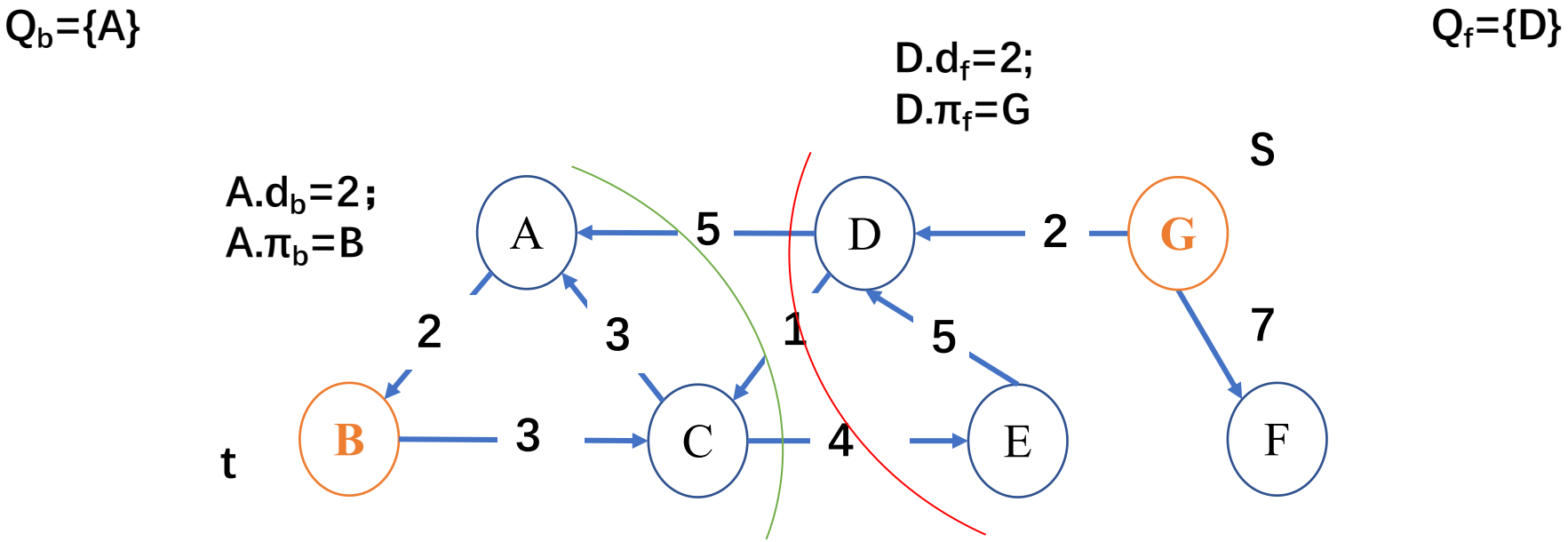
# 双向Dijkstra算法

给定一个带权图G=(V, E, W)和两个点s和t，从s向前做搜索（Forward Search）的同时交替地进行从t开始的反向搜索（Backward Search）



# 相关定义

给定一个点v，$v.d_f$和$v.d_b$定义为前向搜索和后向搜索过程中s到v和v到t的距离； $v.\pi_f$和$v.\pi_b$定义为前向搜索和后向搜索过程中s到v和v到t的路径上的前驱结点；$Q_f$和$Q_b$分别为前向搜索和后向搜索过程中优先队列



# 算法中止条件

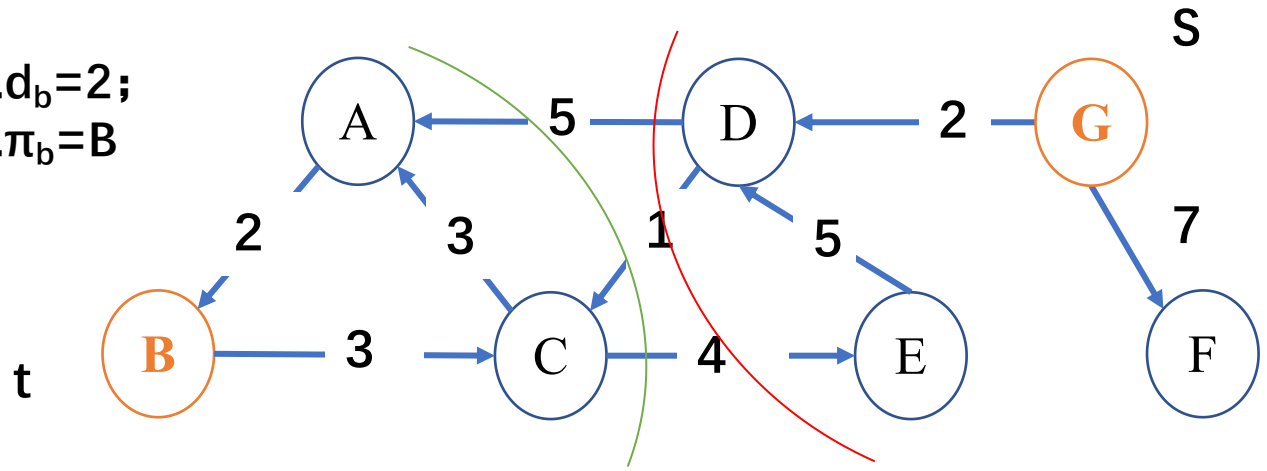
存在一个点w从$Q_f$和$Q_b$中都被删除的时候，算法停止

$D.d_f=2$;
$D.\pi_f=G$

$Q_b=\{C, D\}$

$Q_f=\{C, A, F\}$

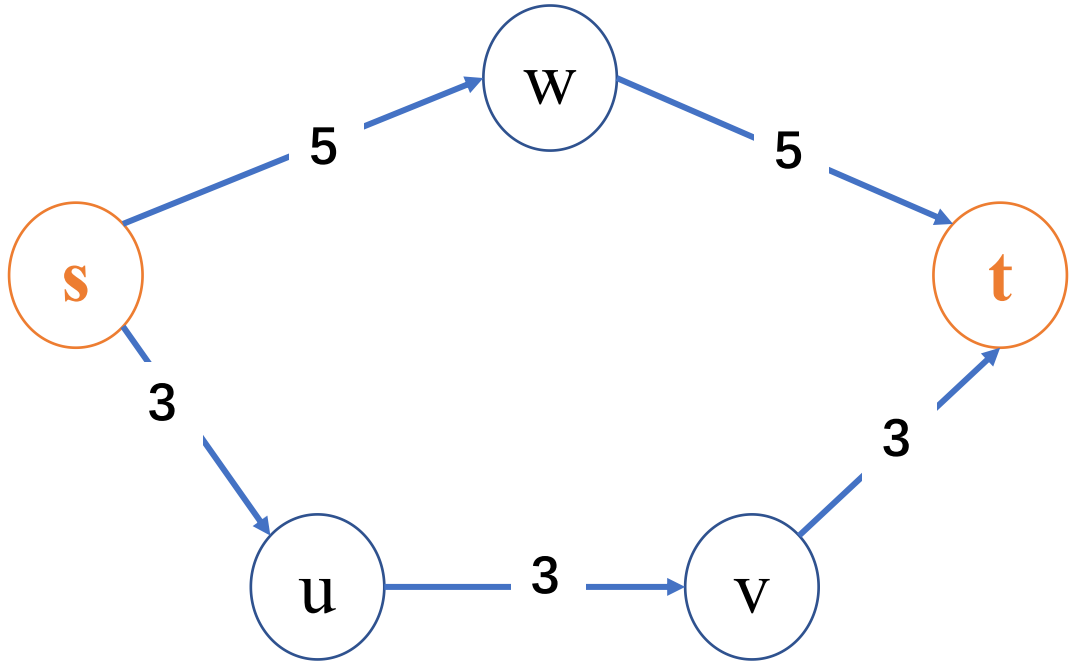
$A.d_b=2$;
$A.\pi_b=B$




$C.d_f=3$；$C.d_b=5$
$C.\pi_f=D$；$C.\pi_b=A$

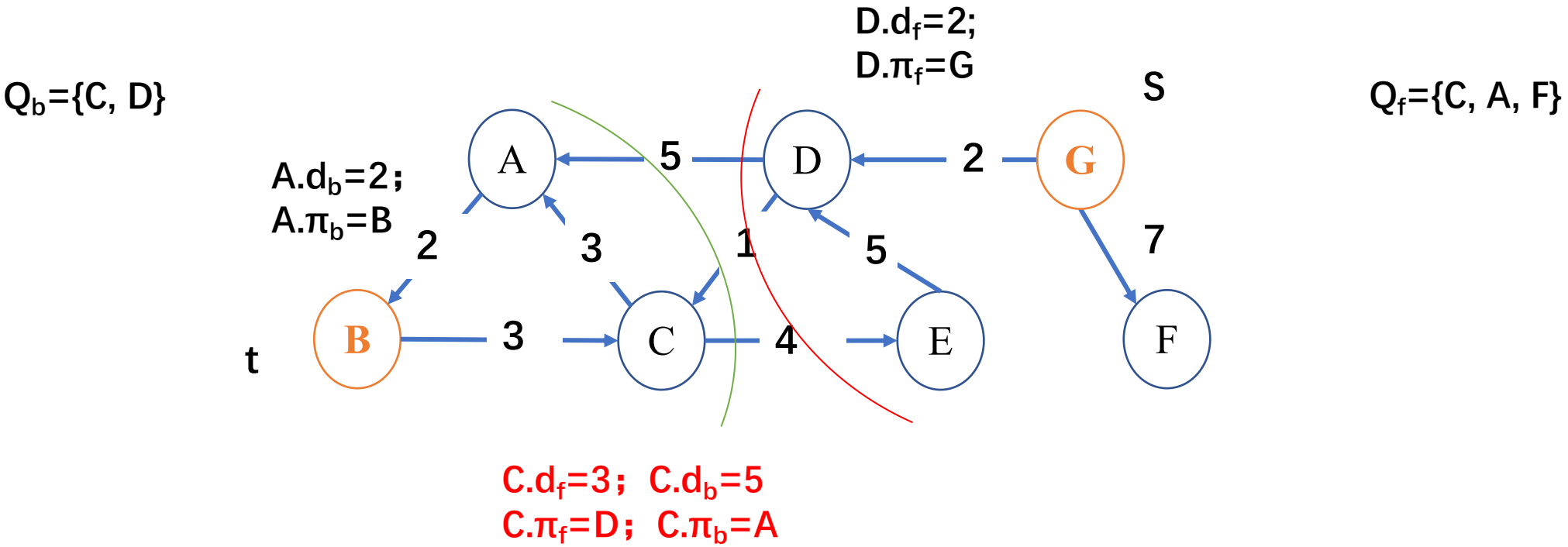
# 最终解计算

在点w从$Q_f$和$Q_b$中都被删除的时候，直接从w出发得到最终解是不行的



# 正确的最终解

正确的最终解计算方法应该是从$Q_f$和$Q_b$中找出连接s和t的最小路径来返回，也就是返回：$\min_{x \in Q_f \cup Q_b} \{d_f[x] + d_b[x]\}$



# 基于Landmark的优化

可以按照之前的边重赋权方法对所有边进行重赋权，即找出一个函数h，对于任意边(u,v)，我们有

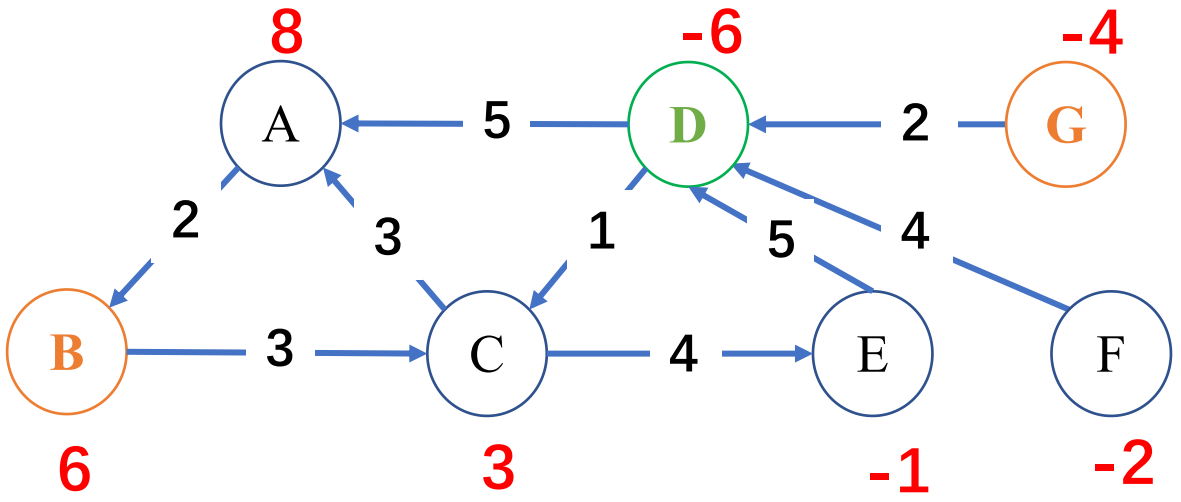
$$w_h(u,v) = w(u,v) + h(v) - h(u)$$

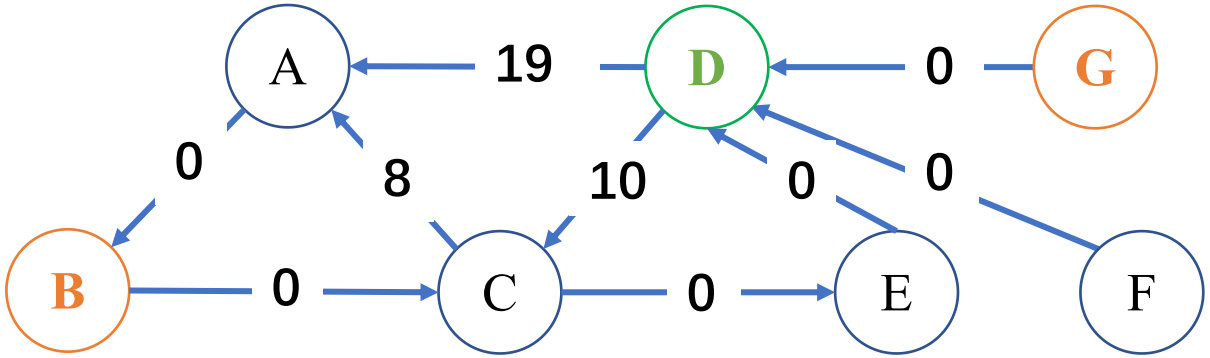
之后，找出一个landmark l来定义h以加快Dijkstra算法计算过程，即

$$\mathrm{h(u)} = \delta(u,l) - \delta(l,t)$$

所有$\delta(u,l)$和$\delta(l,t)$可以在预处理阶段事先算好

# 基于Landmark的优化



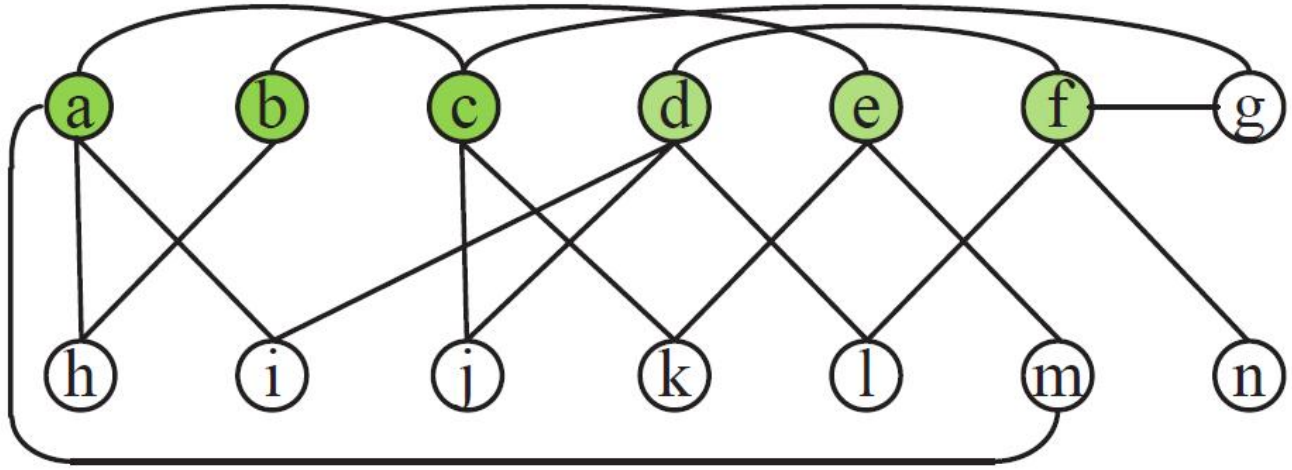



# 基于点覆盖的最短路径树计算

James Cheng, Yiping Ke, Shumo Chu, Carter Cheng. Efficient processing of distance queries in large graphs: a vertex cover approach. SIGMOD 2012. 457-468

# 点覆盖

对于无向图G=(V,E)中的一个点覆盖是一个集合C⊆V使得每一条边至少有一个端点在C中



# 点覆盖性质

性质1：给定G=(V, E)的一个点覆盖C，V中任意一个点v要么属于C，要么所有邻居都属于C；

性质2：如果G中两点u和v之间存在一条路径p，p中除了u和v以外的所有点都不在C中，那么p长度最多为2

# 基本算法

给定图G=(V,E)的一个点覆盖C，计算出C中任意两点距离存在硬盘上

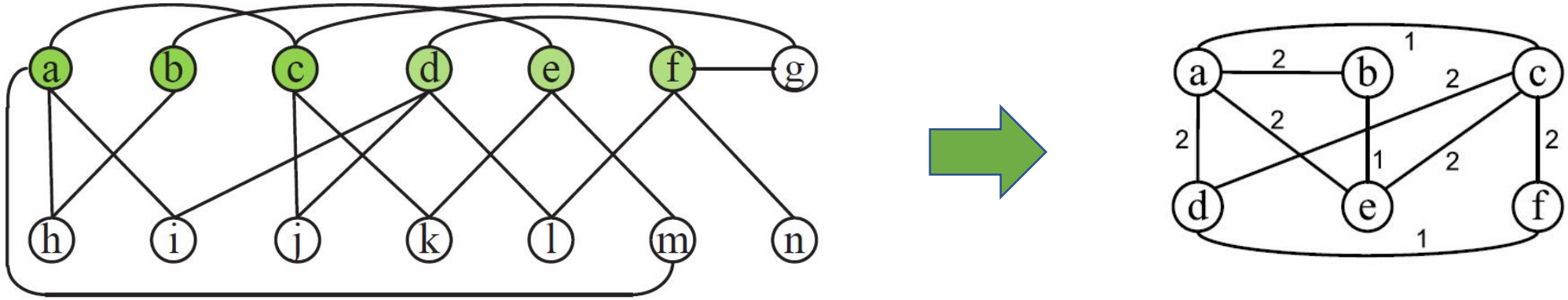
当查询点s进来后，根据性质1计算出s所有点的距离

缺点：空间代价过大

# 改进算法

给定图G=(V, E)的点覆盖C，根据性质2，将图G压缩成图G’

G’中点都是C中点，如果C中两个点u和v在G中两步可达，u和v之间有条边，且边长为u和v之间距离



# 改进算法

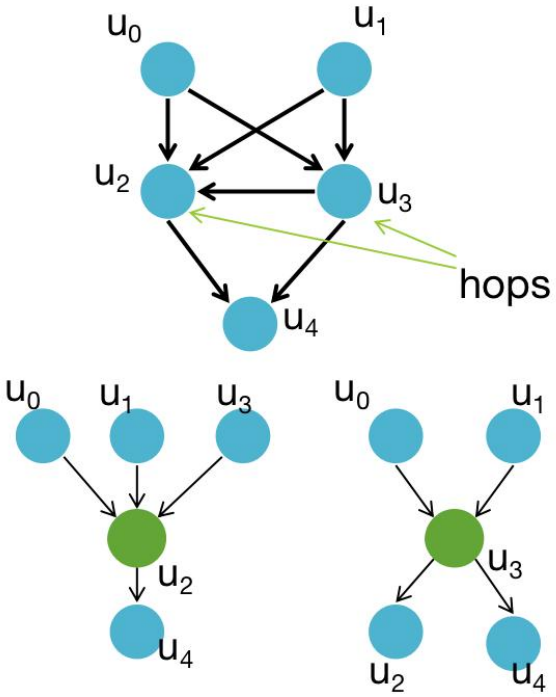
于是，不需要存储C中任意两点距离，而是在计算源点s到所有点距离的时候用Dijkstra计算G’中两点间距离

G’中点的数量小于G中点数量，所以G’上Dijkstra更快

# 2-Hop Labeling

Edith Cohen, Eran Halperin, Haim Kaplan, Uri Zwick. Reachability and Distance Queries via 2-Hop Labels. SIAM J. Comput. 32(5): 1338-1355 (2003)

# 示例

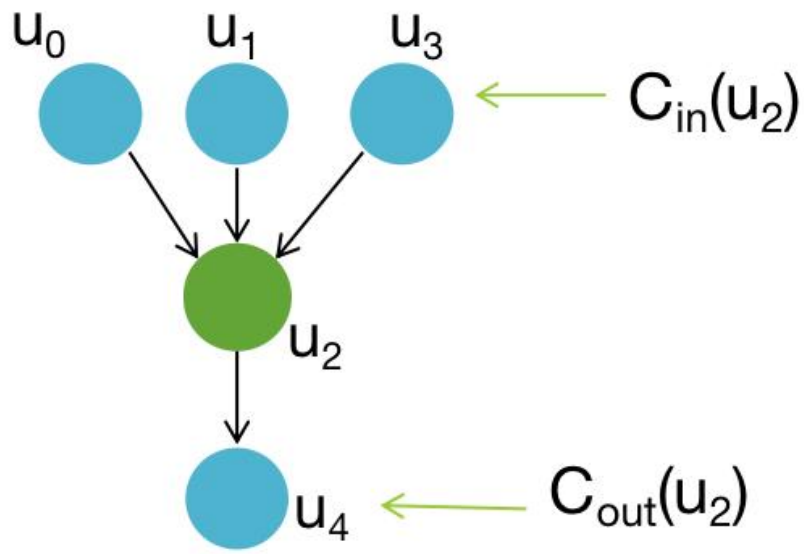



| vertex | 2-hop Labeling |
|---|---|
| u0 | $L_{in}(u_0)=\{\}$ , $L_{out}(u_0)=\{u_2, u_3\}$ |
| u1 | $L_{in}(u_1)=\{\}$, $L_{out}(u_1)=\{u_2,u_3\}$ |
| u2 | $L_{in}(u_2)=\{u_2, u_3\}$, $L_{out}(u_2)=\{u_2\}$ |
| u3 | $L_{in}(u_3)=\{u_3\}$, $L_{out}(u_3)=\{u_2,u_3\}$ |
| u4 | $L_{in}(u_4)=\{u_2, u_3\}$, $L_{out}(u_4)=\{\}$ |

$$\forall u_i, u_j \in V(G), u_i \to u_j \Leftrightarrow | L_{out}(u_i) \cap L_{in}(u_j) | > 0$$

$$e.g.\ \ u_1 \to u_4 \Leftrightarrow | \{L_{out}(u_1) \cap L_{in}(u_2)\} | = | \{u_2, u_3\} | > 0$$

# 示例




2-hop Cover for hop $u_2$.

如何计算2-Hop覆盖?

$$\text{for all } v \in C_{in}(u): \; L_{out}(v) := L_{out}(v) \cup \{u\}$$

$$\text{for all } v \in C_{out}(u): \; L_{in}(v) := L_{in}(v) \cup \{u\}$$

# 问题定义

2-Hop Labeling的目标在于找出一个最优的编码策略，使得如下定义的编码代价最小

$$LabelSize = \sum_{u_i \in V(G)} (|L_{in}(u_i)| + |L_{out}(u_i)|)$$
$$= \sum_{w_j \in hops} (|C_{in}(w_j)| + |C_{out}(w_j)|)$$

可以证明，这个问题能转化为集合覆盖问题，进而难以找出多项式时间的算法

# 近似2-Hop编码求解算法

第一步，计算出所有点对之间的最短路径，形成集合T

第二步，挑选一个点w来最大化如下公式：

$$r(w)=\frac{|S(C_{in}(w),w,C_{out}(w))\cap T'|}{|C_{in}(w)|+|C_{out}(w)|}$$

其中T’表示T中还没有被覆盖的路径

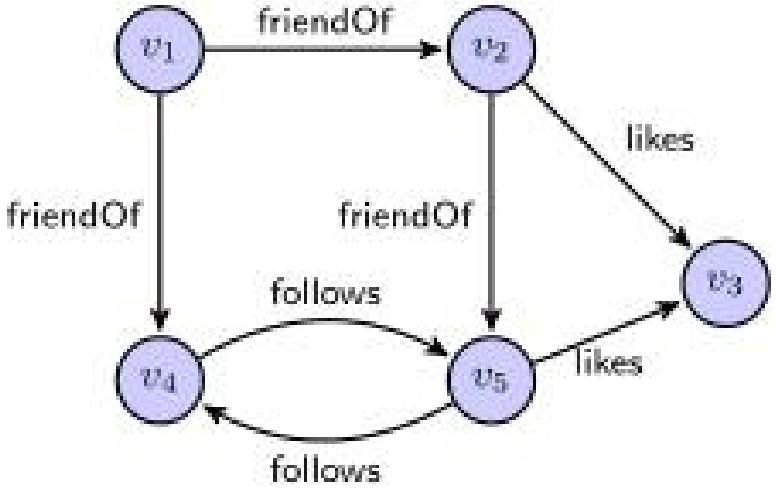
第三步，T=T-T’，然后迭代执行第1、2步直到T为空

# 标签限制的可达性查询

Lucien D. J. Valstar, George H. L. Fletcher, Yuichi Yoshida. Landmark Indexing for Evaluation of Label-Constrained Reachability Queries. SIGMOD Conference 2017: 345-358

# 问题定义

给定图G的顶点s和t以及G的所有边标签L集合的子集L'，仅使用L'中有标签的边来确定G中是否存在从s到t的路径。




**查询($v_1$, $v_5$, {friendOf})，返回True；**

**查询($v_1$, $v_3$, {friendOf})，返回Flase；**

# 基本思路

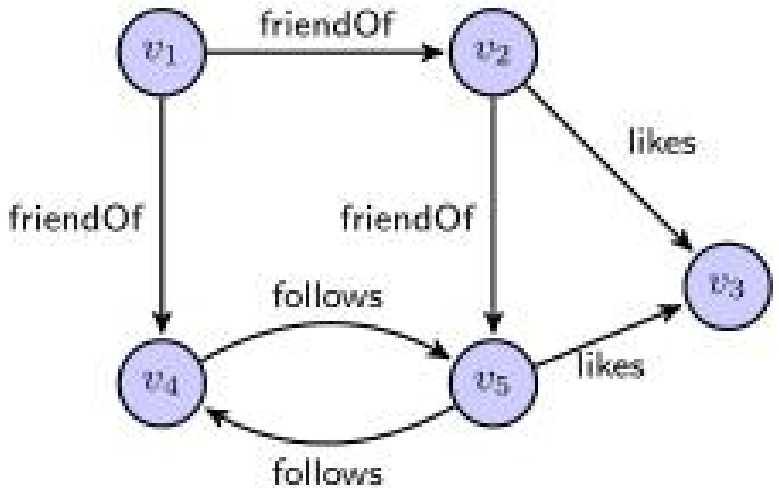
**离线阶段：**给定输入图G=（V，E，$\mathcal{L}$），对于每个顶点v，如果存在从v到w的L'路径，则每个（w，L'）对存储到v的索引中

**在线阶段：**通过检查对（w，L'）是否在v的索引中来回答任何LCR查询（v，w，L'）

# 优化策略

对于点v，只存储（w，L"），使得L"是连接v到w的最小标签集。

查询（v，w，L'）且L"是L'子集，则返回True




**查询($v_1$, $v_4$, {friendOf, follows}) 返回True，标签集 {friendOf, follows} 不是最小标签集合，因为查询 ($v_1$, $v_4$, {friendOf})也返回True**

# 方法概述

**离线阶段：**只为少量顶点（称为Landmark）构造索引

**在线阶段：**给定一个查询（s，t，L），我们从s中执行BFS（考虑标签）。当我们找到一个为其构造索引的地标x时，我们使用它立即获得答案。

# 基于Landmark的距离估计

Michalis Potamias, Francesco Bonchi, Carlos Castillo, Aristides Gionis: Fast shortest path distance estimation in large networks. CIKM 2009: 867-876

Andrey Gubichev, Srikanta J. Bedathur, Stephan Seufert, Gerhard Weikum: Fast and accurate estimation of shortest paths in large graphs. CIKM 2010: 499-508

Miao Qiao, Hong Cheng, Lijun Chang, Jeffrey Xu Yu: Approximate Shortest Distance Computing: A Query-Dependent Local Landmark Scheme. ICDE 2012: 462-473

Mingdao Li, Peng Peng, Yang Xu, Hao Xia, Zheng Qin:Distributed Landmark Selection for Lower Bound Estimation of Distances in Large Graphs. APWeb/WAIM (1) 2019: 223-239

# 问题定义

给定一个无向图G=(V, E)，所谓landmark，就是一些特定的点。我们记录并保存所有点到landmark的距离，进而对于每个点v形成一个向量

$$\phi(v) =< \delta(v, l_1),\ \delta(v, l_2), ..., \delta(v, l_n) >$$

对于两个查询点s和t而言，显然有

$$\delta_{\text{approx}}(s,\ t\ ) = \min_{1 \le k \le n} \{\delta(s, l_k) + \delta(t, l_k)\} \ge \delta(s,\ t\ )$$

$$\delta_{\text{approx}}(s,\ t\ ) = \max_{1 \le k \le n} \{|\delta(s, l_k) - \delta(t, l_k)|\} \le \delta(s,\ t\ )$$

# Potamias et.al@CIKM2009

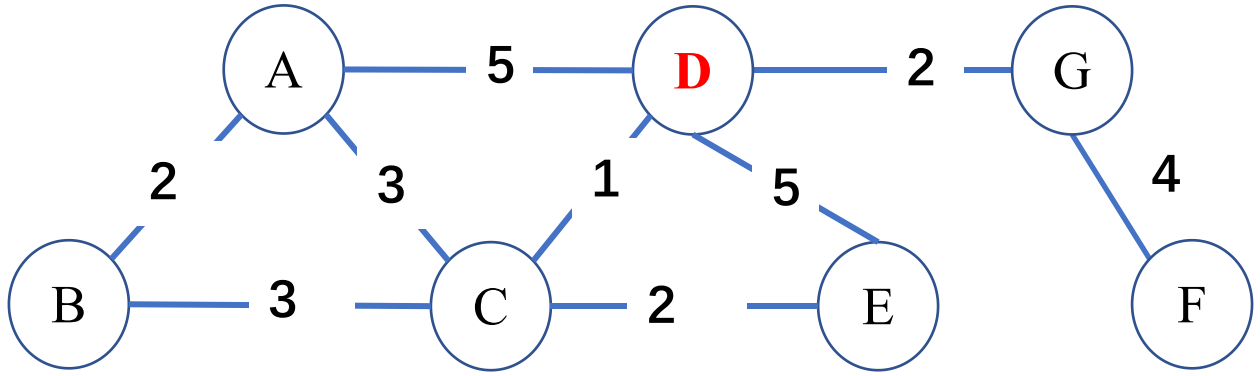
本文主要讨论了lankmark的选取策略问题

本文着重介绍了基于度中心度和介数中心度的两种策略

# 度中心度

度中心度（Degree Centrality）通过点的邻居数目来计算节点的重要程度

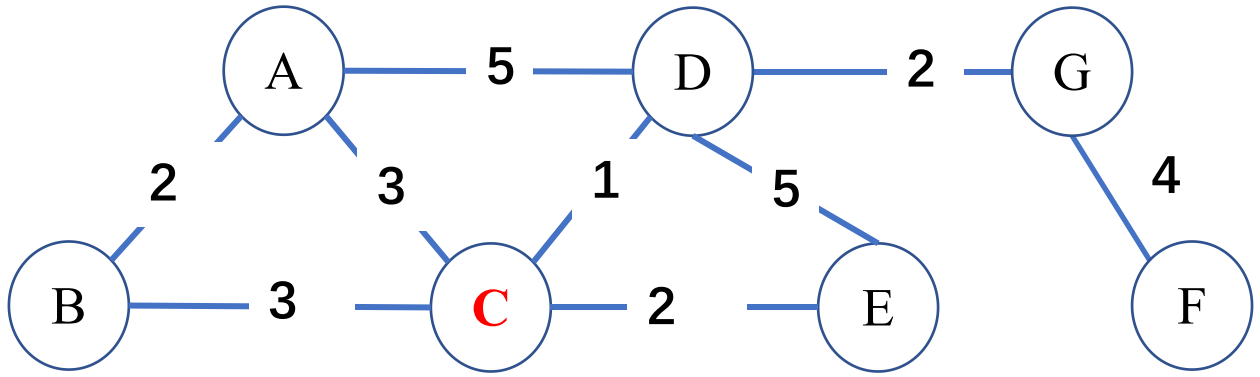
给定图G = (V,E)，点v的度中心度计算公式为：$C_D = \deg(v)$



# 介数中心度

介数中心度（Betweenness Centrality）定义为G 中除了v 以外的点对间最短路径有多少条最短路径经过了点v

介数中心度的计算公式为：$C_B = \sum_{s \neq v \neq t \in V} \frac{\pi_{st}(v)}{\pi_{st}}$，其中$\pi_{st}$表示s到t的最短路径，$\pi_{st}(v)$表示s到t经过v的最短路径

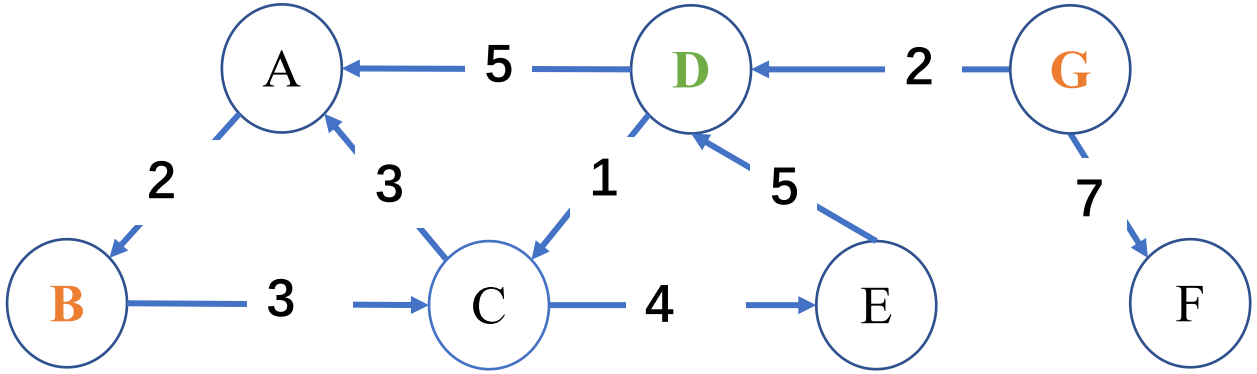


# Gubichev et.al@CIKM2010

本文主要讨论了基于landmark进一步提高距离预测准确度的方法

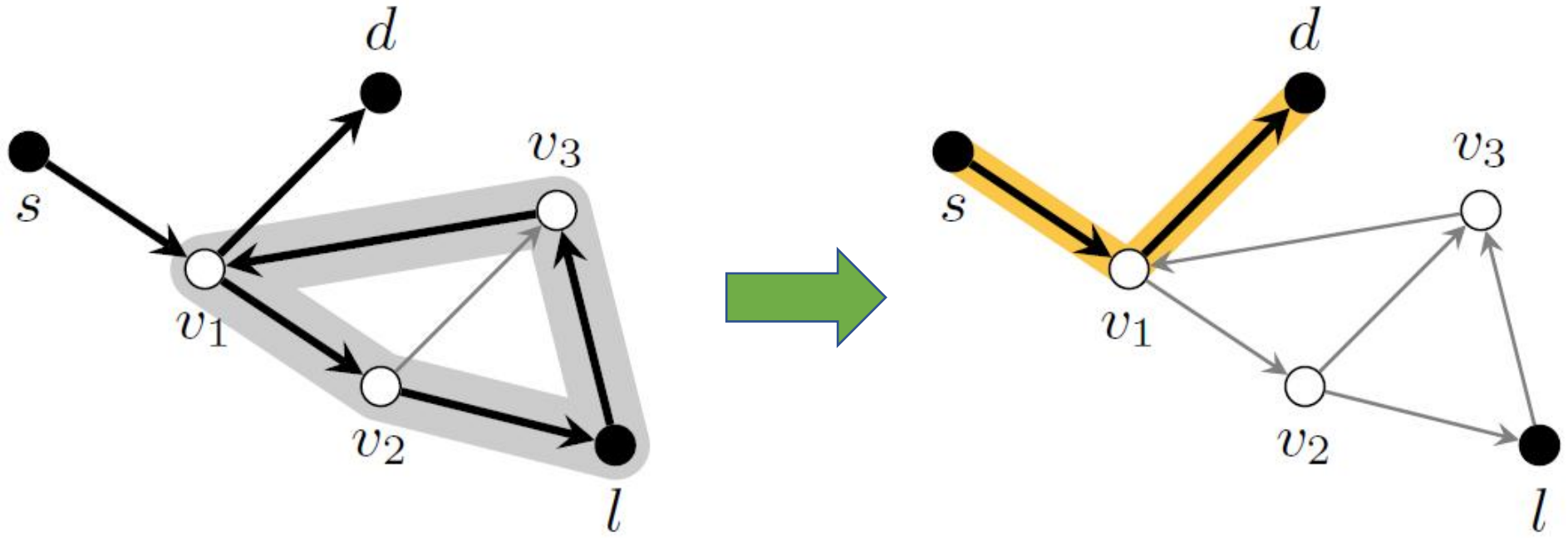
在预处理阶段，预计算出所有Landmark的最短路径树

在查询处理阶段，$\pi_{st}$就等于$\pi_{sl}$和$\pi_{lt}$的拼接



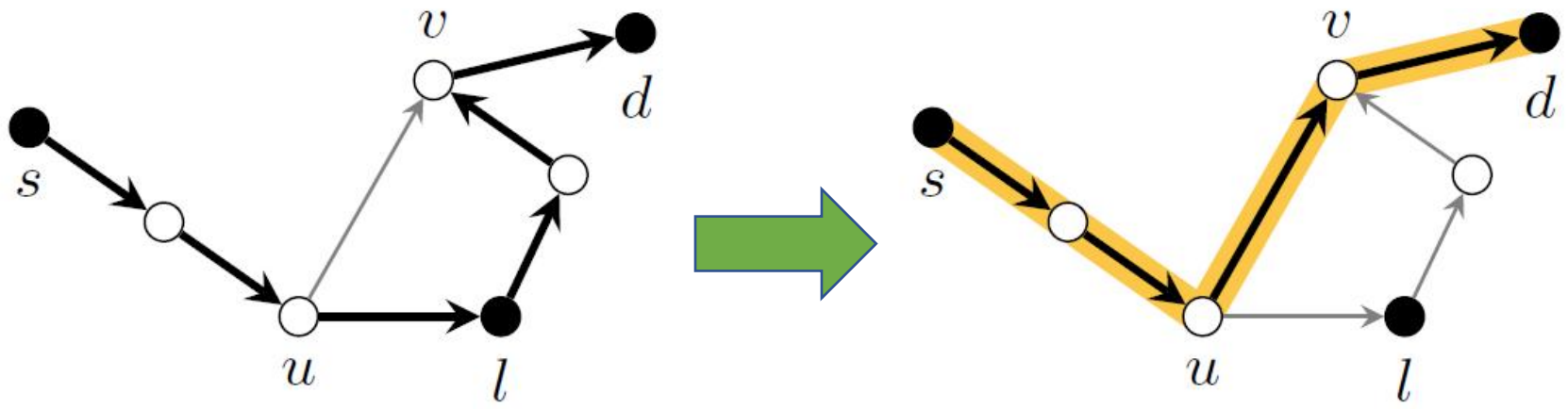
# Cycle Elimination

$\pi_{sl}$和$\pi_{lt}$的拼接过程中消除期间的环



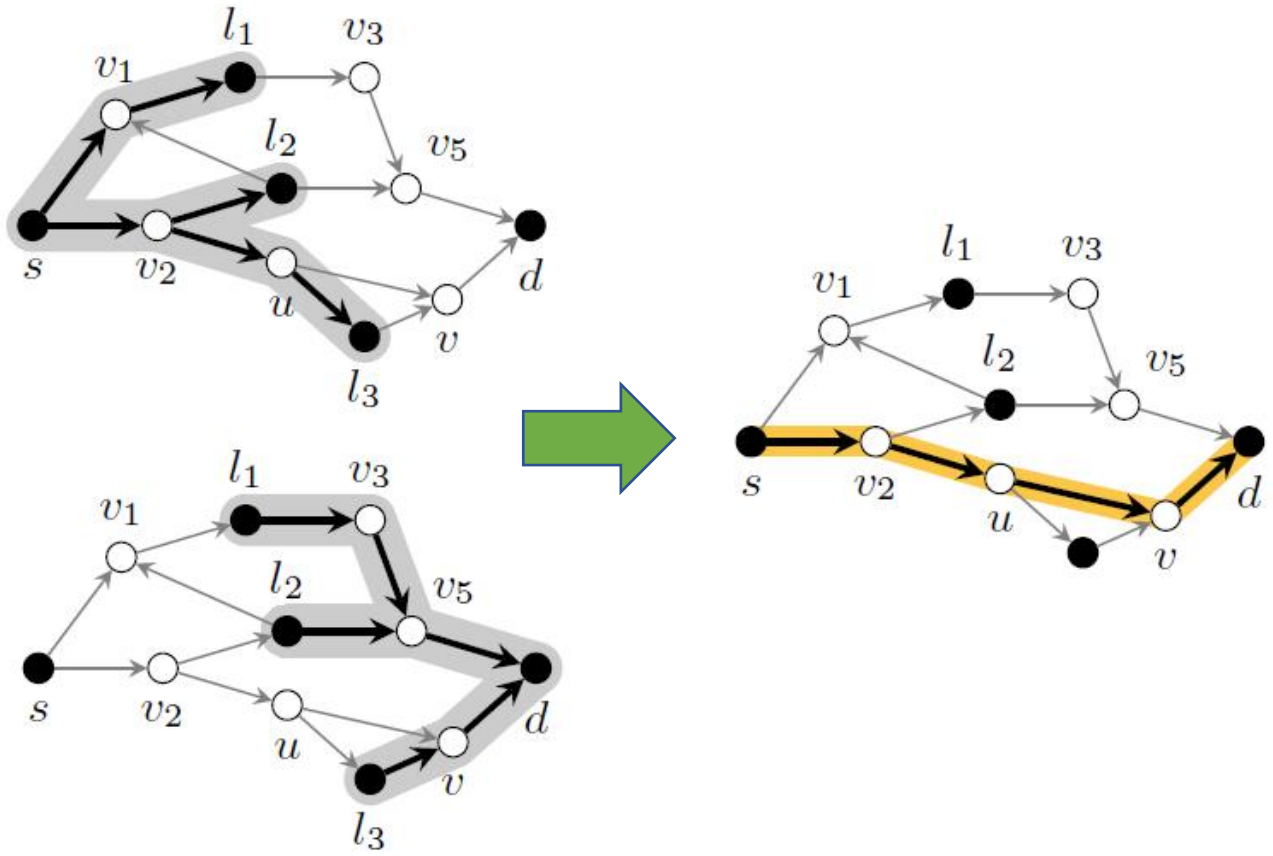
# Shortcutting

$\pi_{sl}$和$\pi_{lt}$的拼接过程中找出期间的捷径



# Tree Algorithm

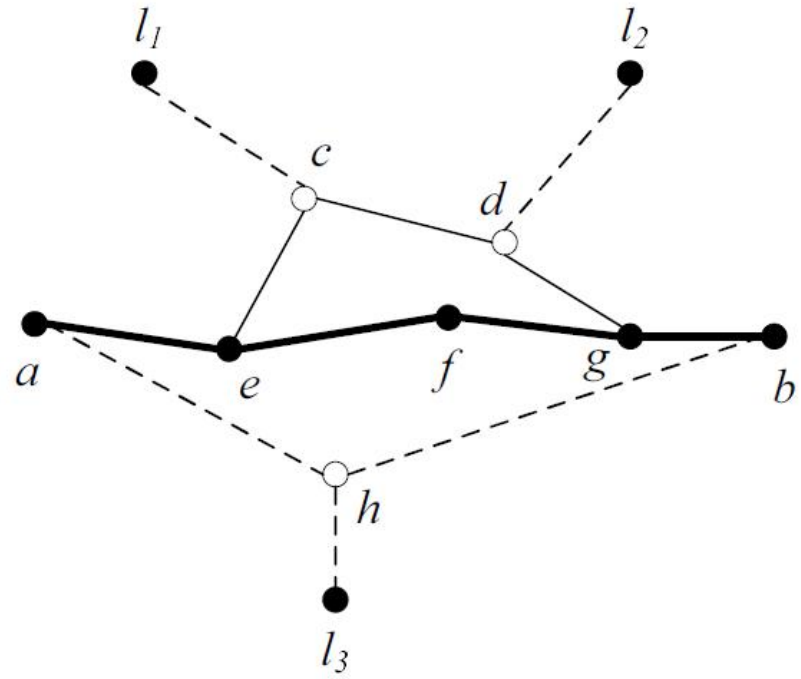
$\pi_{sl}$和$\pi_{lt}$的拼接过程中从最短路径树上找出跨树的路径



# Qiao et.al@ICDE2012

本文再进一步讨论了如何进一步构建索引提高效率

具体而言，本文重点讨论了基于最短路径树上最近公共祖先的优化



湖南大學 HUNAN UNIVERSITY

# 目录

# 链接分析内容

什么是链接分析？

- 链接分析也称链接分析法或超链分析，可广义理解为以Web中页面间的超链接为研究对象的分析活动。

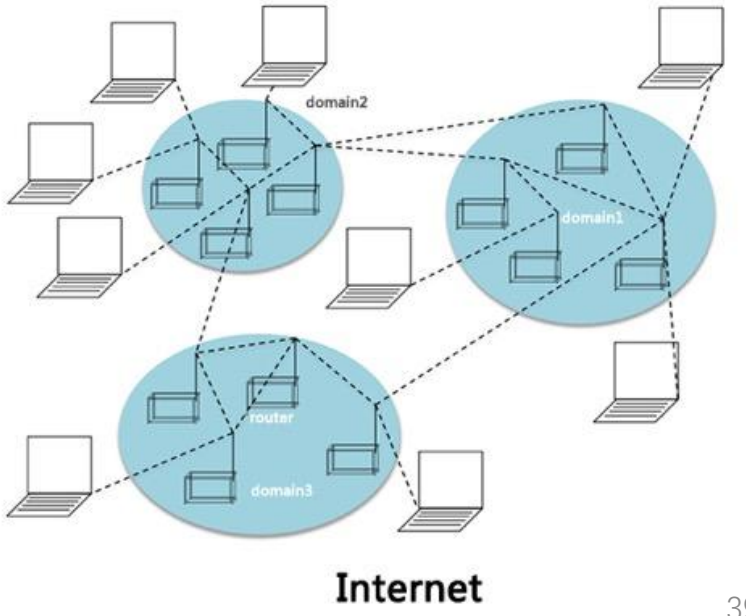
涉及的文档

- 四个层面：页面、目录、域名、站点

主要内容

- Web结构研究
- 链接增长规律研究
- 链接分类研究
- 链接分析算法
- 链接分析工具研究

重点：Web链接分析!!!



# 链接分析应用领域

## 搜索引擎与网站设计

- 链接分析算法用于对检索结果排序，有效提高检索效率
- 网站根据搜索引擎的原理和排序算法改进网络结构、提高网站质量以增加其可见度

# 链接分析历史

1978年，Nelson在《Dream Machine》中提到“链接”，指出链接带来了文件连通的可能性，并在1981年描述了全球化大文档想法

1995年， Brazilian Marcia J. Bossy首次提出可将信息技术应用于因特网


1996年，Larson在《Bibliometrics of the World Wide Web: An Exploratory Analysis of the Intellectual Structure of Cyberspace》一文中明确将信息技术从文献计量学移植到网络

1997年，Almind和Ingwersen提出了“网络计量学（webometrics）”一词，旨在定量分析网络现象

1998年，Google的创始人Brin和Page（拉里·佩奇）公开了PageRank算法的核心部分

# PageRank

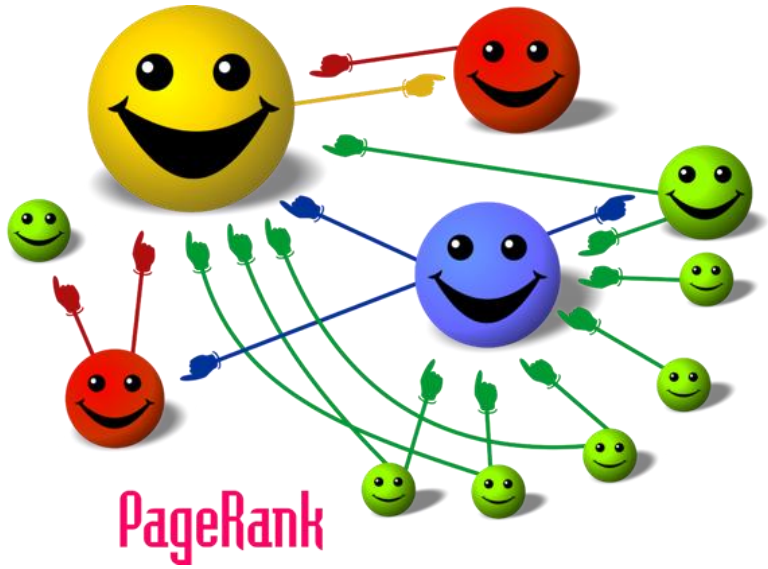
基本思路：链接即投票

- 网页链接数越多越重要
- 入链多少？出链多少？

将指向某网页的导入链作为该网页获得的投票

所有的导入链都是相同的吗？

- 从重要的网页获得的投票应该给予更高权重
- 递归问题！（多轮投票）



# 链接即投票

http://www.alexa.com

What sites link to hnu.edu.cn?

Total Sites Linking In 331

| | Site | Page |
|---|---|---|
| 1. | baidu.com | tieba.baidu.com/f?kw=湖南大学 |
| 2. | sina.com.cn | kaoshi.edu.sina.com.cn/college/c/10532... |
| 3. | so.com | baike.so.com/doc/5348407-5583860.html |
| 4. | china.com.cn | u.edu.china.com.cn |
| 5. | rednet.cn | sk.rednet.cn |

What sites link to csu.edu.cn?

Total Sites Linking In 856

| | Site | Page |
|---|---|---|
| 1. | sina.com.cn | blog.sina.com.cn/kuangcuilin |
| 2. | 360.cn | edu.360.cn/daxue |
| 3. | stackoverflow.com | stackoverflow.com/users/2374716/sarly |
| 4. | pixnet.net | shingi015.pixnet.net/blog/4 |
| 5. | so.com | baike.so.com/doc/2840941-2998095.html |

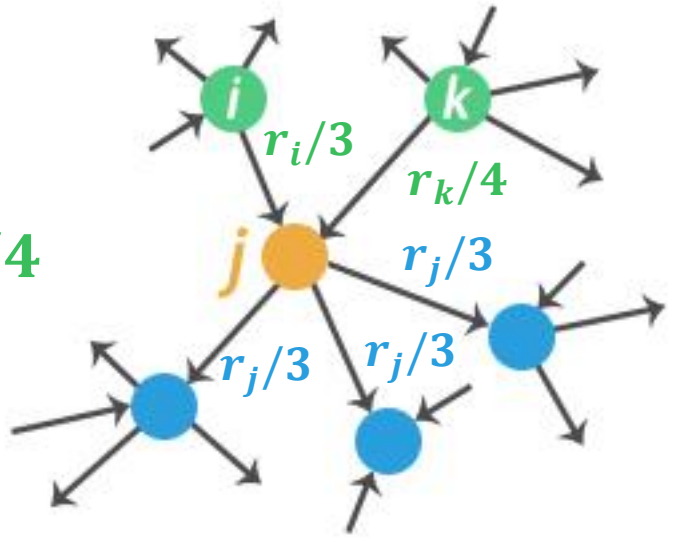
# 简单的递归公式

每条链接的投票数正比于其源网页的重要程度

如果重要程度是$r_j$的网页j有n条出链，那么每条出链拥有$r_j/n$个投票数

网页j的重要程度即为其入链投票数之和

$$r_j = r_i/3 + r_k/4$$



# 递归公式

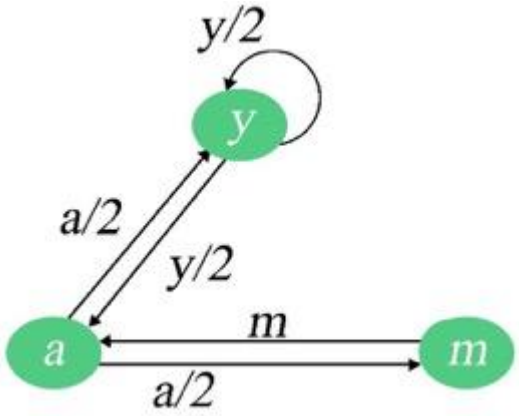
从重要的网页获得的投票更有价值

被其他重要网页链接的网页也重要

对网页j定义排名$r_j$

$$r_j = \sum_{i \to j} \frac{r_i}{deg_i}$$

其中$d_i$是节点$i$的出度




“流”等式：

$$r_y = r_y/2 + r_a/2$$

$$r_a = r_y/2 + r_m$$

$$r_m = r_a/2$$

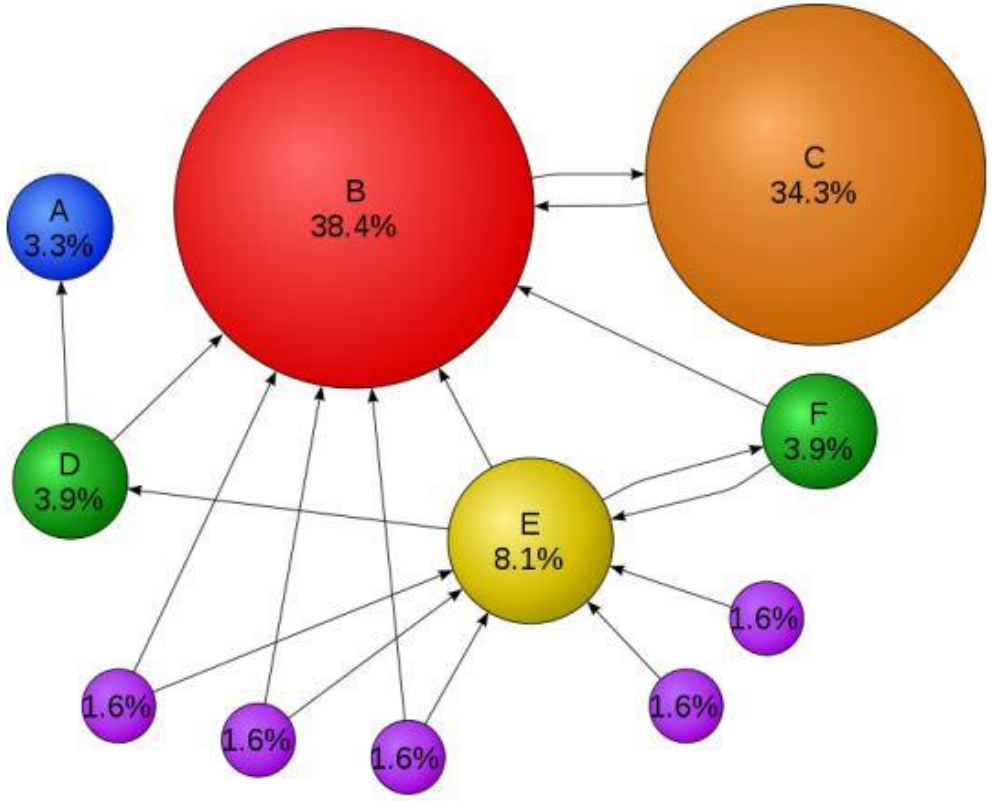
# 阻尼系数

有一些网页是dead ends（没有出链），这些网页导致投票被“泄露”；此外，用户很可能不沿着超链接跳转了，而是直接另开一个网页

阻尼系数d 用来表示在PageRank 迭代过程中一个点沿着出边跳转到下一个点的概率。于是，1 − d 表示浏览过程随机跳转到下一个点的概率。于是

$$r_j = \frac{1-d}{|V|} + d \times \sum_{i \to j} \frac{r_i}{deg_i}$$

# 例子：PageRank 分数

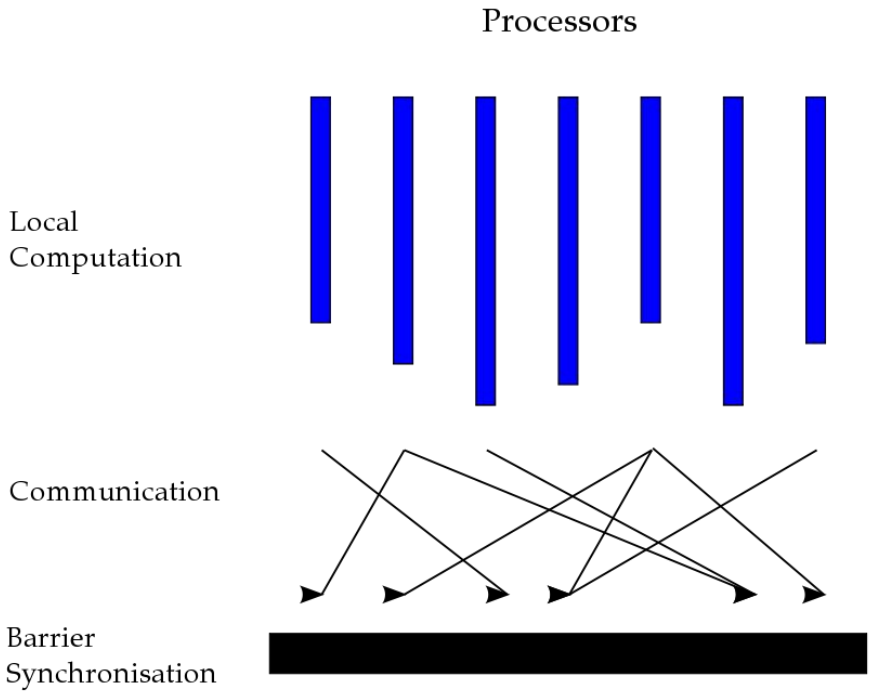


# PageRank

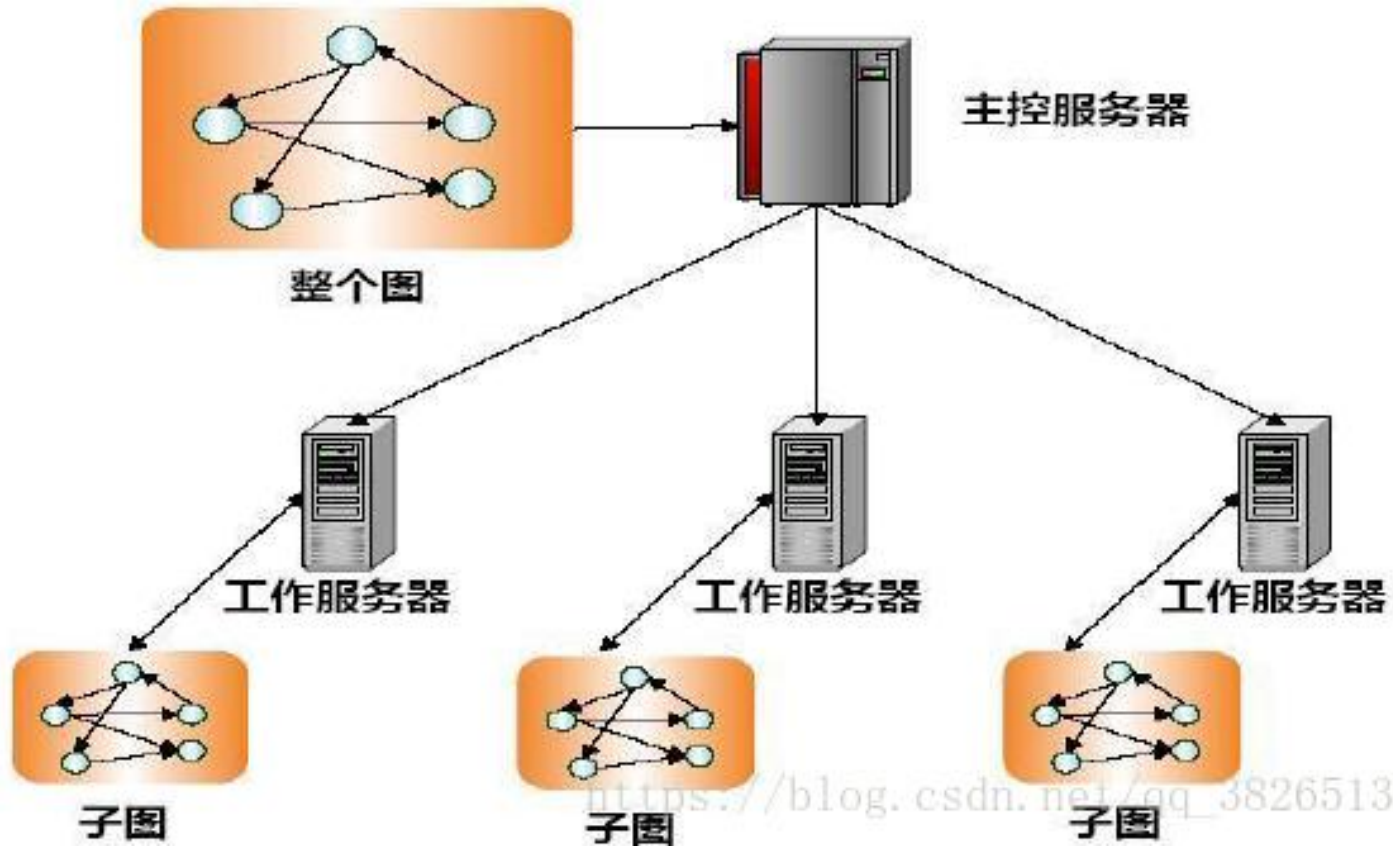
针对大规模网络上的网页链接分析等实际问题，Pregel是谷歌提出的大规模分布式图计算平台

Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski: Pregel: a system for large-scale graph processing. SIGMOD Conference 2010: 135-146

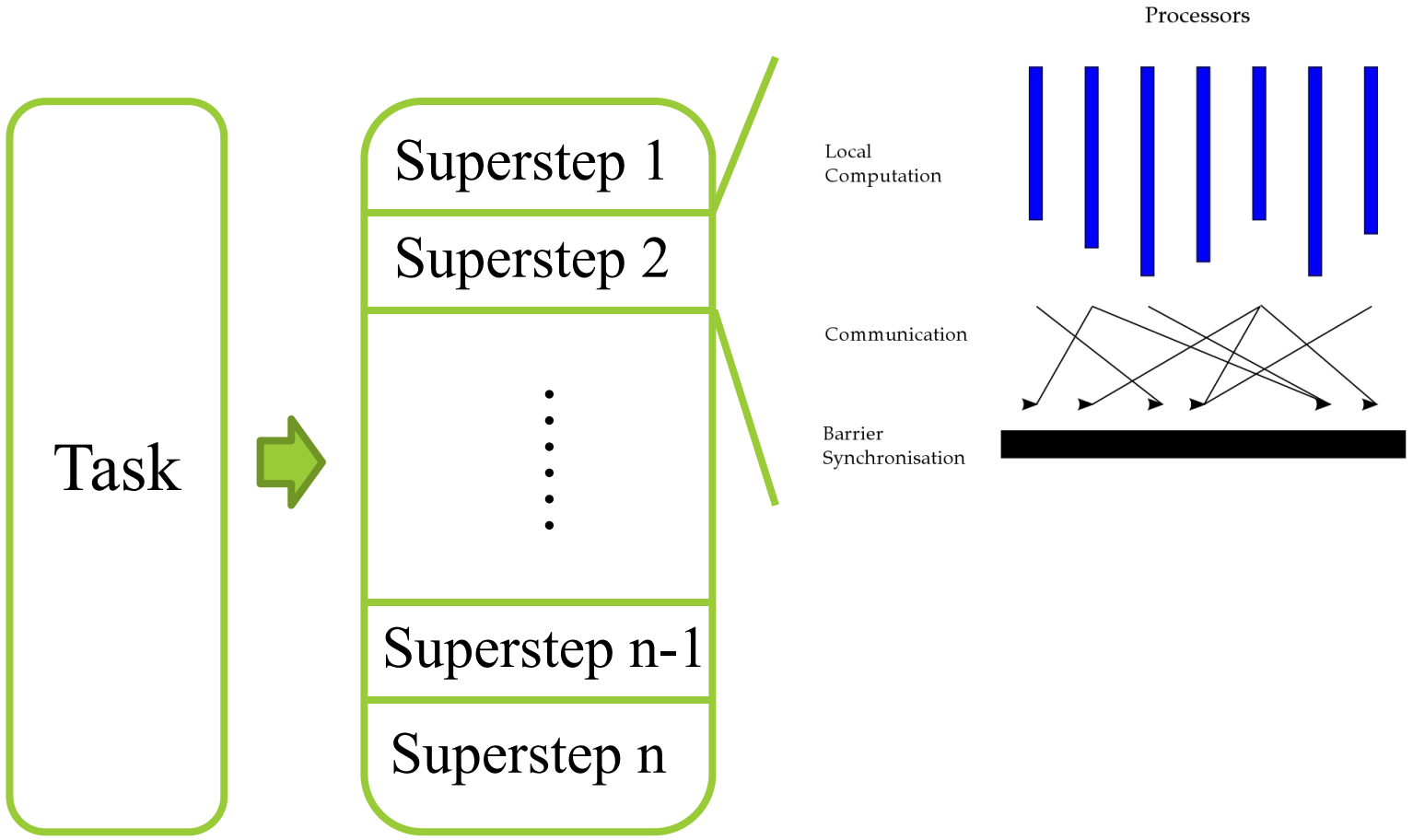
# 计算模型（BSP）


Processors
Local
Computation
Communication
Barrier
Synchronisation

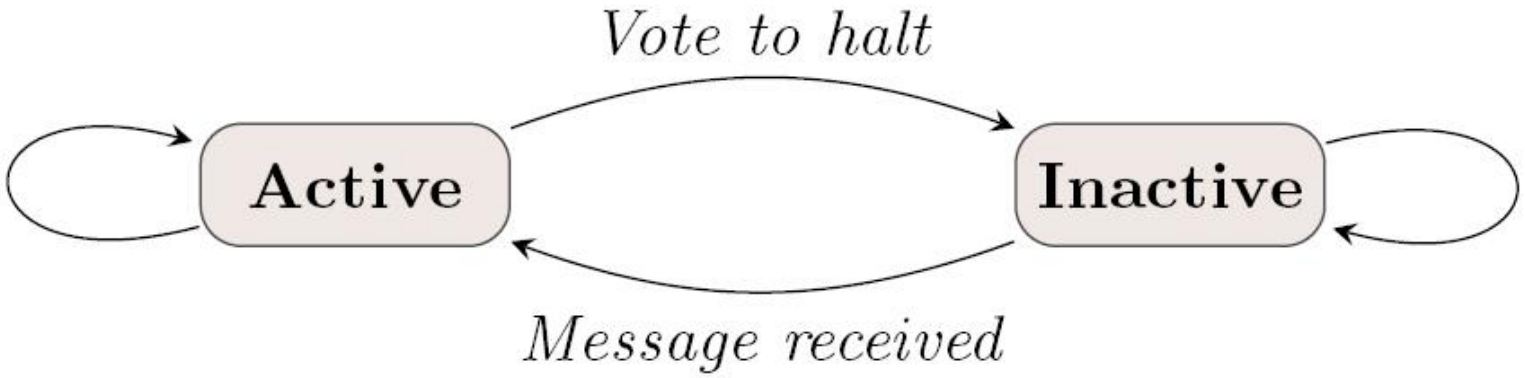
# 系统架构



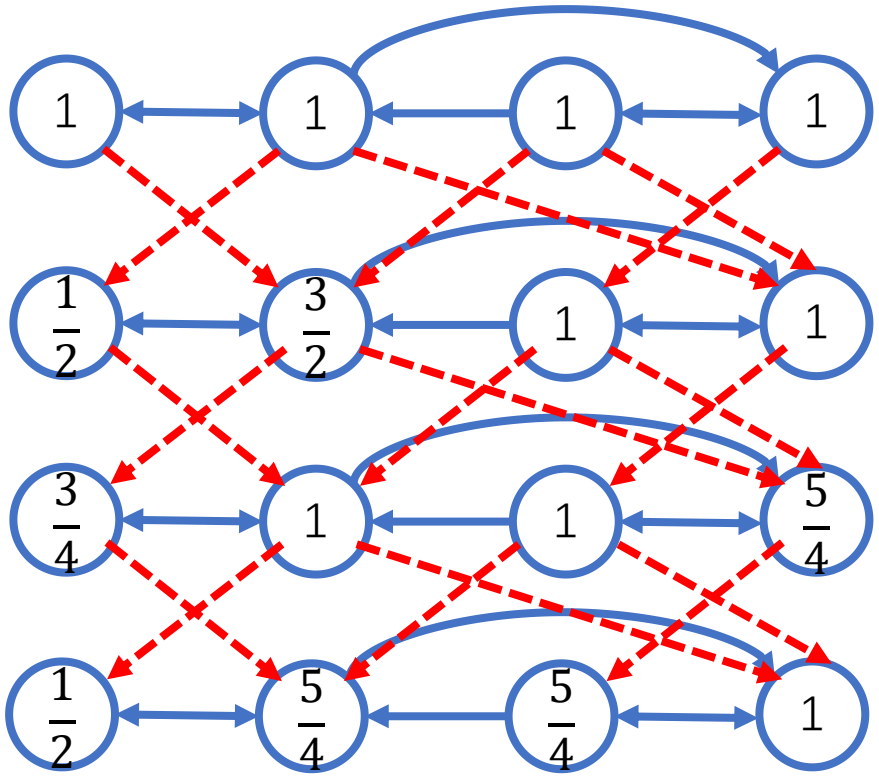
# Pregel@SIGMOD2010



# 以点为核心的计算模型



# 示例




Superstep 1

Superstep 2

Superstep 3

Superstep 4

# Pregel相关

Giraph(Apache)，GraphLab(CMU)，GPS(Stanford)，Pregel+(CUHK)，GraphX (UC Berkeley)，PAGE(PKU)

湖南大学 HUNAN UNIVERSITY

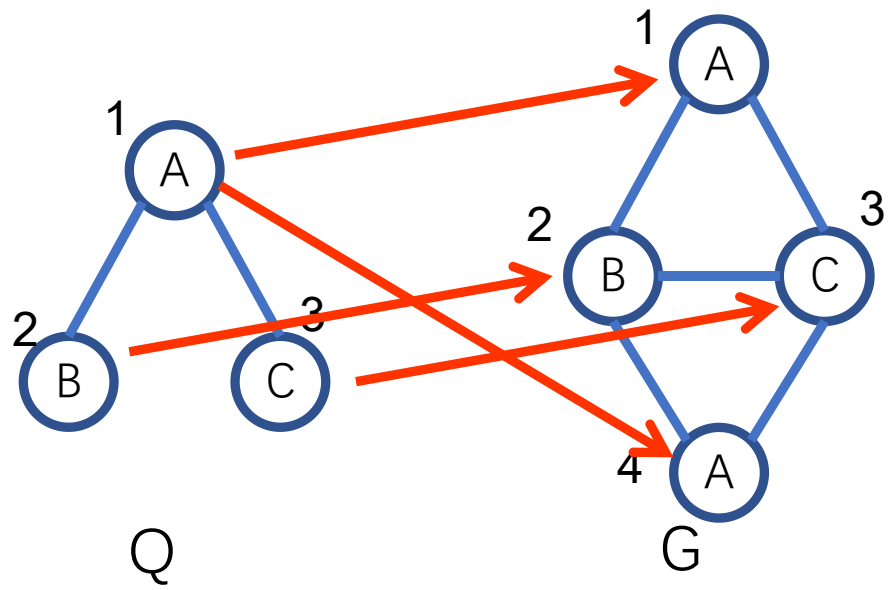
# 目录

# 子图匹配

给定一个数据图G和一个查询图Q，找出一个Q在G上的匹配g，使得

$$\forall v \in V(Q), F(v) = F'(g(v)); and$$

$$\forall v_1, v_2 \in V(Q), \overrightarrow{v_1 v_2} \in E(Q) \Rightarrow \overrightarrow{g(v_1)g(v_2)} \in E(G)$$

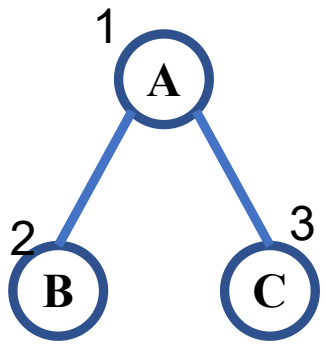


# VF2

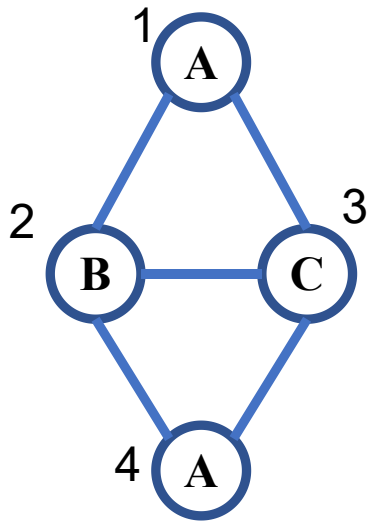
L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub) graph isomorphism algorithm for matching large graphs. TPAMI, 26(10):1367–1372, 2004.

# VF2

找出点对(u,v)的集合，其中Q上点u在G上的匹配是v




Q

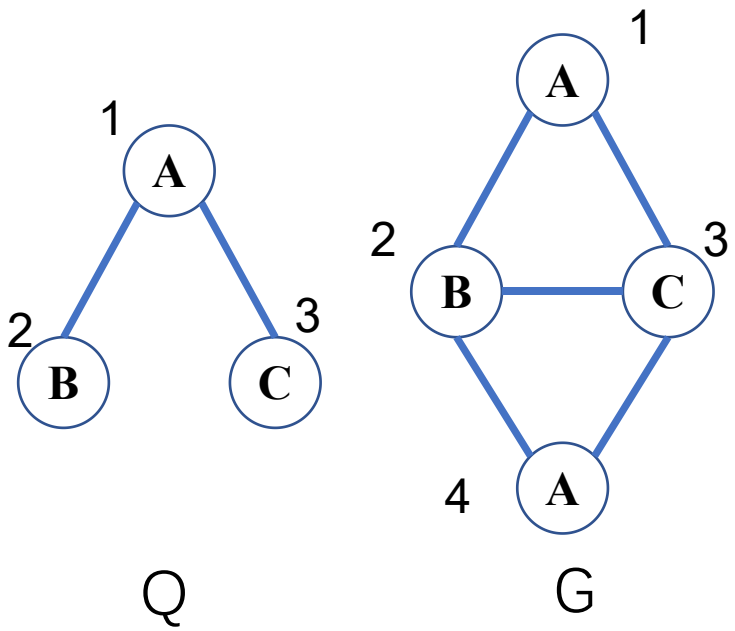



G

| $S_1$ | $S_2$ |
| --- | --- |
| (1, 1) | (1, 4) |
| (2, 2) | (2, 2) |
| (3, 3) | (3, 3) |

# VF2 Algorithm

根据每个查询点的可能匹配点对，在图上进行搜索得到最终解




如何减少每个查询点可能匹配点对的数目？

Candidate Pair Sets

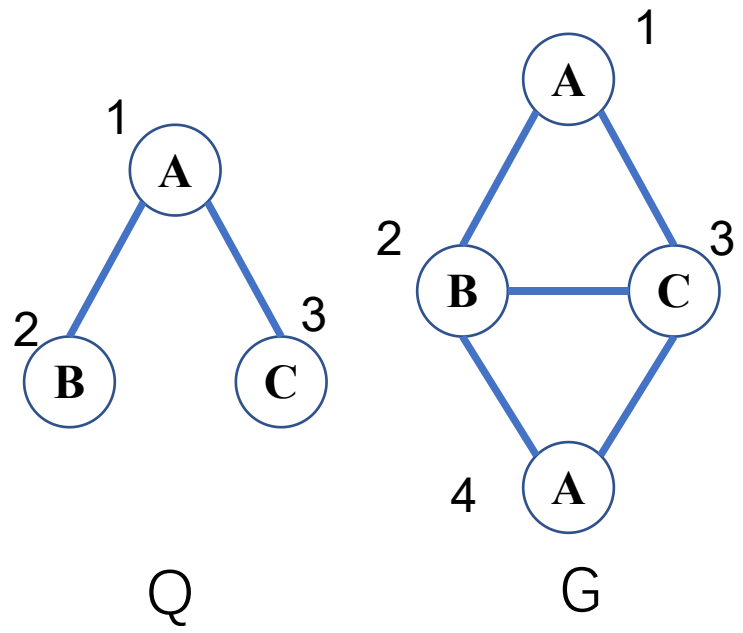
(2, 2)　(1, 1)　(1, 4)

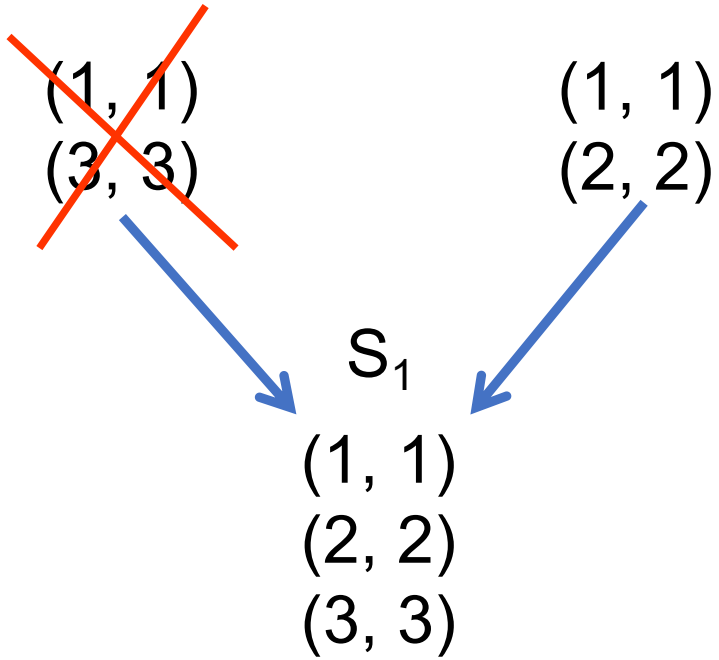
(3, 3)　(3, 3)

# VF2 Algorithm

一个匹配可以从都中搜索路径中搜索得到，于是可以给查询点定个顺序

Eg. (1 < 2 < 3)



# 基于最坏情况下最优连接的子图匹配

Amine Mhedhbi, Semih Salihoglu: Optimizing Subgraph Queries by Combining Binary and Worst-Case Optimal Joins. PVLDB 12(11): 1692-1704 (2019)

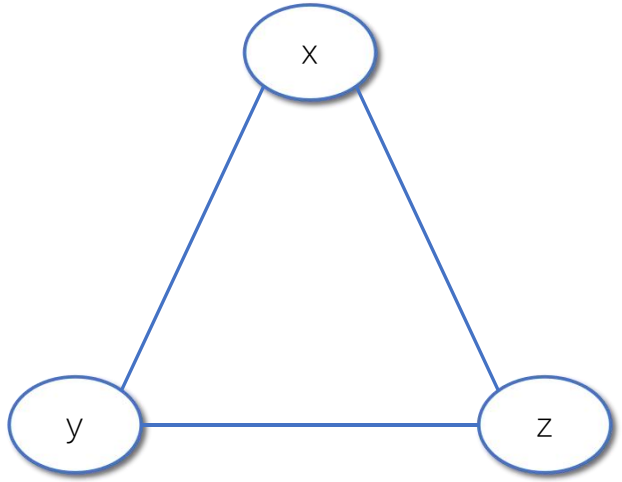
# Outline

## Worst-Case Optimal Joins

Optimizing WCO Plans

# Triangle Query

Consider the triangle query, $R(x,y) \bowtie S(y,z) \bowtie T(x,z)$ and let $|R|=|S|=|T|=N$. Then, the size of the query results is at most $N^{1.5}$



# Hypergraph

A hypergraph is a pair H = (V(H), E(H)), consisting of a nonempty set V(H) of vertices, and a set E(H) of subsets of V(H), the hyperedges of H

# Fractional Edge Cover

A fractional edge cover of a hypergraph H = (V, E) is a mapping $f: E \rightarrow [0, \infty)$ such that $\sum_{e \in E, v \in e} f(e) \geq 1$

for all v∈ V . The number $\sum_{e \in E} f(e)$ is the weight f. The fractional edge cover number $\rho^*(H)$ of H is the minimum of the weights of all fractional edge covers of H

# Upper Bound of Joins

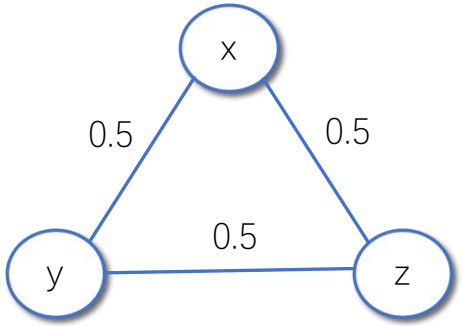
Given a set of relations for joining, there is a vertex for each attribute of the query and a hyperedge for each relation.

Then, given a fractional edge cover f, we have an upper bound of the query result size |out| as follows

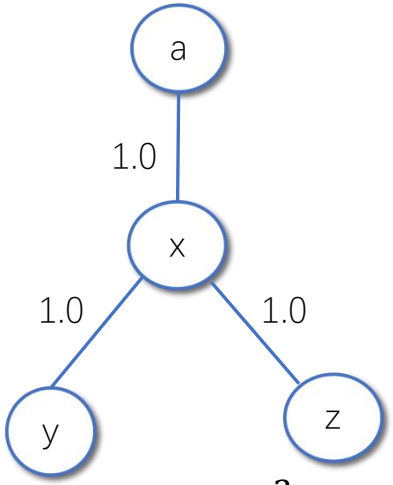
$$|OUT| \leq \prod_{e \in E} |R_e|^{f(e)}$$

# Examples for Worst-case Optimal Joins

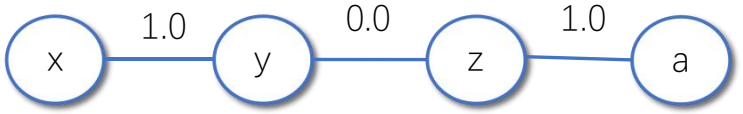
The following three queries have very different query result sizes




$|OUT| \leq N^{1.5}$




$|OUT| \leq N^3$




$|OUT| \leq N^2$

# Generic Join

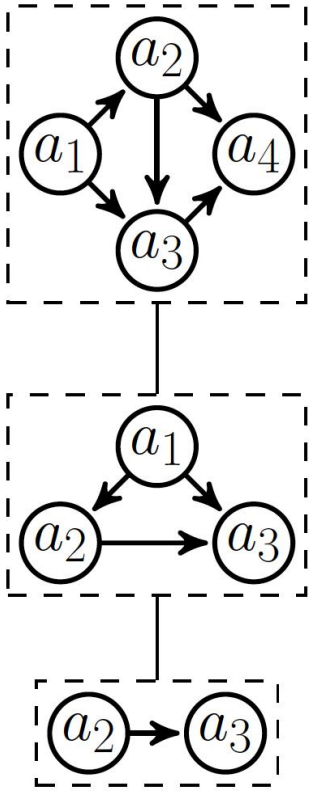
Generic Join [30] is a wort-case optimal (WCO) join algorithm that evaluates queries one attribute at a time

First, Generic Join picks an order of query vertices to match. It often assume that the subquery induced by the first k query vertices in the order is connected

Second, Generic Join iteratively evaluates one query vertex accoding to the order

# Evaluation Example

Query vertices order <$a_2$, $a_3$, $a_1$, $a_4$>



# How to Evaluate Query Vertices in Order

There are two basic operators to evaluate query vertices in order: Scan and Extend/Intersect

**Scan**: The first two vertices of the order, which match a single query edge, are evaluated by scaning the data graph to find matches of the query edge.

# Extend/Intersect

**Extend/Intersect**: After the subquery induced by the first $k$ query vertices in the order are evaluated, Generic Join extends each match $m$ of $k$ query vertices to one or more matches of $k+1$ query vertices.

The operator first computes the extension set $S$ of $m$ by intersecting the adjacency lists of $k$ query vertices' matches and then extends $m$ to $m \times S$

# Outline

Worst-Case Optimal Joins

**Optimizing WCO Plans**

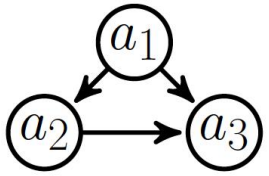
# Effects of Query Vertex Order(QVO)

Directions of Intersected Adjacency Lists

Number of Intermediate Partial Matches

Intersection Cache Hits

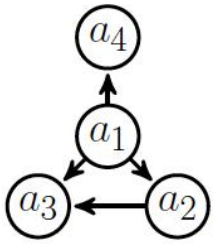
# Directions of Intersected Adjacency Lists




| | BerkStan | | | Live Journal | | |
|---|---|---|---|---|---|---|
| QVO | time | part. m. | i-cost | time | part. m. | i-cost |
| $a_1a_2a_3$ | 2.6 | 8M | 490M | 64.4 | 69M | 13.1B |
| $a_2a_3a_1$ | 15.2 | 8M | 55,8B | 75.2 | 69M | 15.9B |
| $a_1a_3a_2$ | 31.6 | 8M | 55,9B | 79.1 | 69M | 17.3B |

Which combination of adjacency list directions is more efficient depends on the structural properties of the input graph, e.g., forward and backward adjacency list distributions.

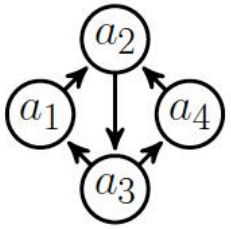
# Number of Intermediate Partial Matches




| | Amazon | | | Epinions | | |
|---|---|---|---|---|---|---|
| QVO | time | part. m. | i-cost | time | part. m. | i-cost |
| $a_1a_2a_3a_4$ | 0.9 | 15M | 176M | 0.9 | 4M | 0.9B |
| $a_1a_3a_2a_4$ | 1.4 | 15M | 267M | 1.0 | 4M | 0.9B |
| $a_2a_3a_1a_4$ | 2.4 | 15M | 267M | 1.7 | 4M | 1.0B |
| $a_1a_4a_2a_3$ | 4.3 | 35M | 640M | 56.5 | 55M | 32.5B |
| $a_1a_4a_3a_2$ | 4.6 | 35M | 1.4B | 72.0 | 55M | 36.5B |

Which QVOs will generate fewer intermediate matches depend on several factors: (i) the structure of the query; (ii) for labeled queries, on the selectivity of the labels on the query; and (3) the structural properties of the input graph, e.g., graphs with low clustering coefficient generate fewer intermediate triangles than those with a high clustering coefficient.

# Intersection Cache Hits




| | Amazon | | | Epinions | | |
|---|---|---|---|---|---|---|
| QVO | time | part. m. | i-cost | time | part. m. | i-cost |
| $a_2a_3a_1a_4$ | 1.0 | 11M | 0.1B | 0.9 | 2M | 0.1B |
| $a_1a_2a_3a_4$ | 3.0 | 11M | 0.3B | 4.0 | 2M | 1.0B |

The intersection cache of our E/I operator is utilized more if the QVO extends (k–1)-matches to $a_k$ using adjacency lists with indices from $a_1$, ..., $a_{k-2}$.

# Cost Metric for WCO Plans

We introduce a new cost metric called intersection cost (i-cost), which we define as the size of adjacency lists that will be accessed and intersected by different WCO plans.

$$\sum_{Q_{k-1} \in Q_2 \dots Q_{m-1}} \sum_{t \in Q_{k-1}} \sum_{\substack{(i,dir) \in A_{k-1} \\ \text{s.t. (i, dir) is accessed}}} |t[i].dir|$$

# HashJoin Operator

We use the classic hash join operator, which first creates a hash table of all of the tuples of $Q_{c1}$ on the common query vertices between $Q_{c1}$ and $Q_{c2}$. The table is then probed for each tuple of $Q_{c2}$.

# Dynamic Programming Optimizer

We initialize the cost of computing 2-vertex sub-queries of Q. Then starting from k = 3 up to $|V_Q|$, for each k-vertex sub-query $Q_k$ of Q, we find the lowest cost plan $P^*_{Q_k}$ to compute $Q_k$ in three different ways:

- $P^*_{Q_k}$ is the lowest cost WCO plan that we enumerated.
- $P^*_{Q_k}$ extends the best plan $P^*_{Q_{k-1}}$ for a $Q_{k-1}$ by an E/I operator.
- $P^*_{Q_k}$ merges two best plans $P^*_{Q_{c1}}$ and $P^*_{Q_{v2}}$ for $Q_{c1}$ and $Q_{c2}$, respectively, with a HASH-JOIN.

湖南大学 HUNAN UNIVERSITY

# 目录

# 社区结构

对整个复杂网络的研究一直是许多领域的研究热点，其中社区结构是复杂网络中的一个普遍特征，整个网络是由许多个社区组成的。

设图$G = G(V, E)$，所谓社区发现是指在图$G$中确定$n_C(\geq 1)$个社区

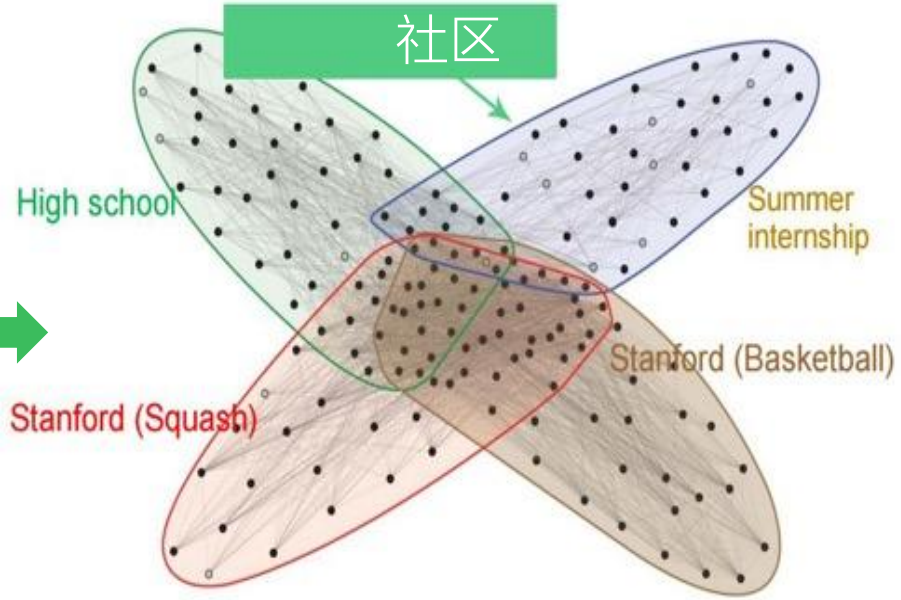
$$C = \{C_1, C_2, \cdots, C_{n_C}\}$$

使得各社区的顶点集合构成$V$的一个覆盖

# Facebook网络

我们能够找出社区吗?

社区发现




节点：Facebook用户
边：朋友关系

节点：Facebook用户
边：朋友关系

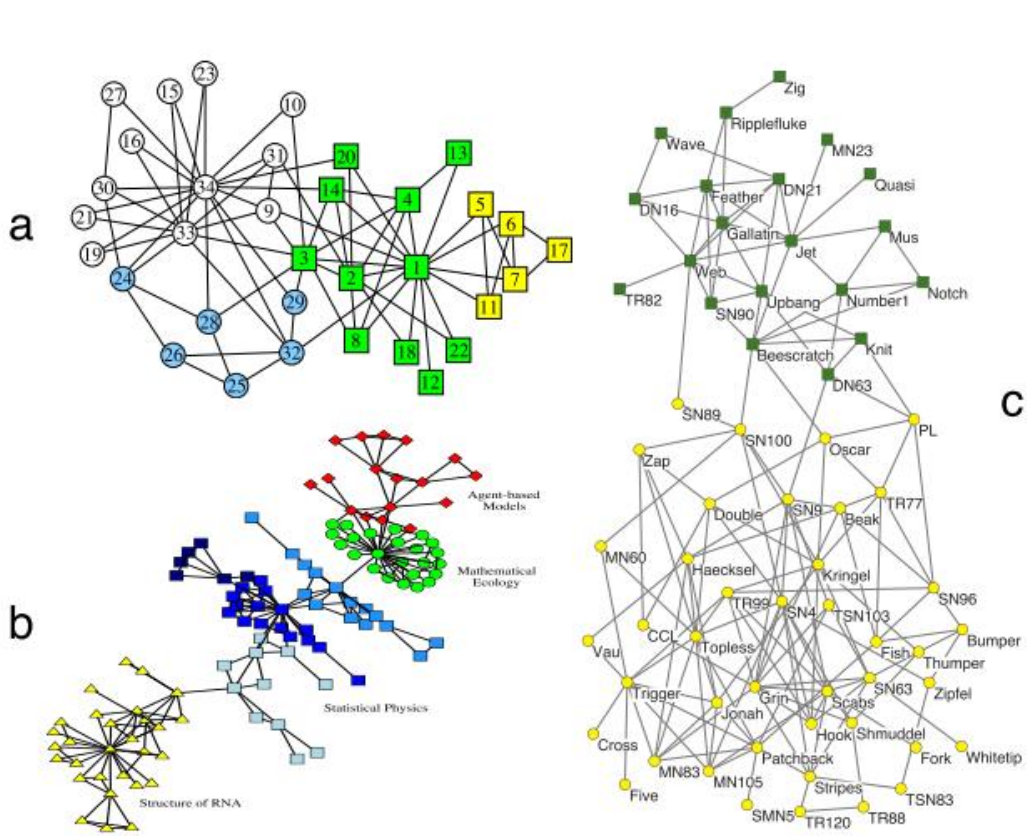
# 经典的社区研究案例

a.空手道俱乐部（karate club）

b.科学家合作网络（Collaboration network）

c.斑马群体的社交行为研究

其中著名的空手道俱乐部社区已经成为通常检验社区发现算法效果的标准（benchmark）之一



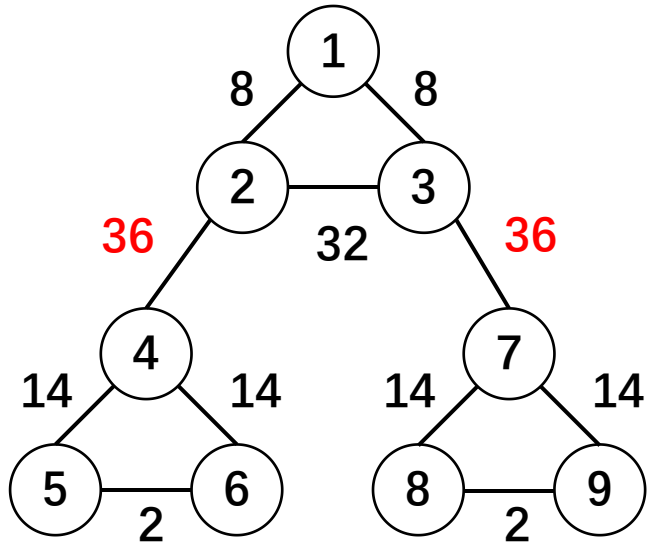
# GN算法

M. Girvan, M. E. J. Newman. Community structure in social and biological networks. Proceedings of the National Academy of Sciences of the United States of America. 2002. 99 (12): 7821–7826

# GN算法

GN算法是一种分裂型的社区结构发现算法，该算法的基本思想是，根据网络中社区内部高内聚、社区之间低内聚的特点，逐步去除社区之间的边，取得相对内聚的社区结构。即通过不断地移除介数最大的边，将网络逐步分解为不同的社区结构。

如果一条边连接两个社区，那么这两个社团节点之间的最短路径通过该边的次数就会最多，相应的边介数最大。如果删除该边，那么两个社团就会分割开



# 边介数中心度

具体地，介数中心度的计算公式为：

$$C_B(e) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(e)}{\sigma_{st}}$$

其中，$\sigma_{st}$表示从节点$s$到$t$的最短路径数量，$\sigma_{st}(e)$表示其中经过节点$e$的最短路径数量。

# 边介数中心度

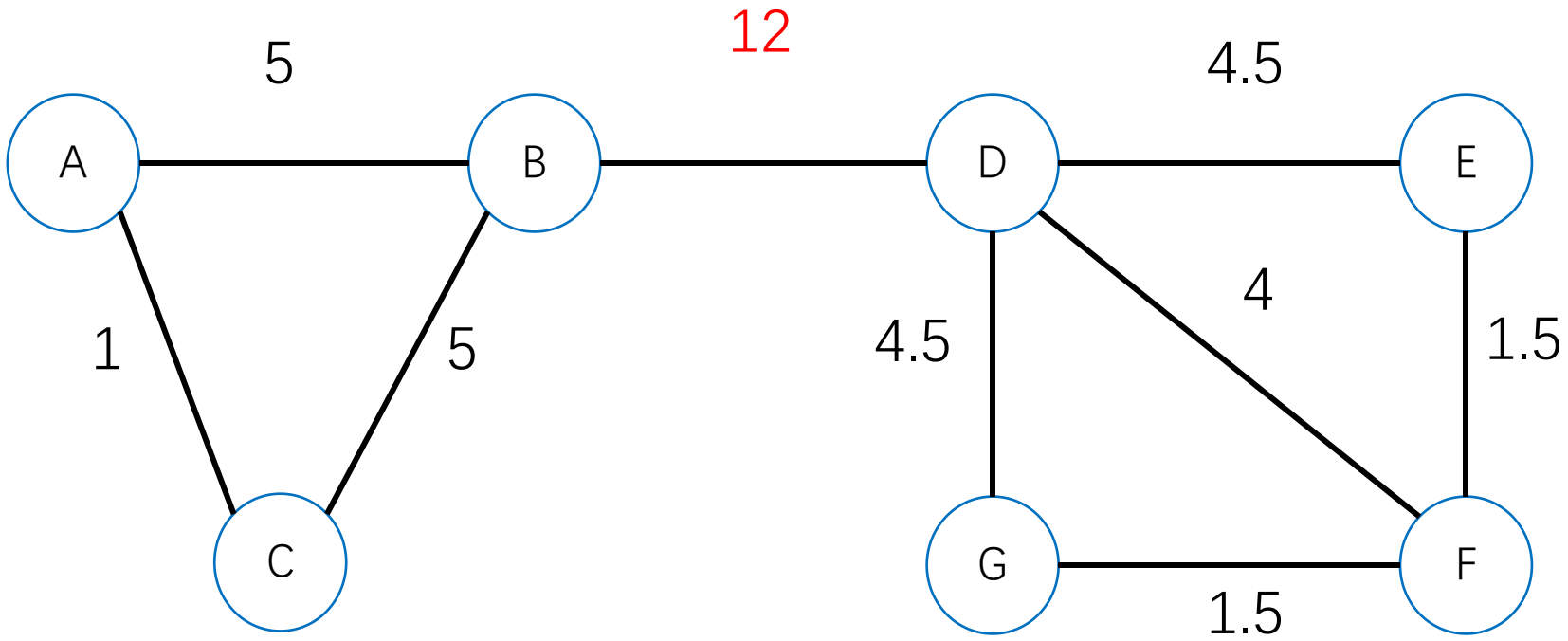
第一步：计算网络中的每条边的边介数；

第二步：删除网络中边介数的值最大的边；

第三步：重新计算网络中所有边的边介数；

第四步：重复执行第二、三步，直到网络中所有边被删除

# GN算法结果



# GN算法的优劣

优点：全局搜索算法，有很强实用性

缺点：GN算法对于网络中的社区结构没有一个量的定义，因此，它不能直接地从网络中的拓扑结构来判断所划分的社区结构是否是网络中真是存在的社区结构，所以需要一些附加的信息来判断划分的社区结构是否具有实际意义。另外，GN算法还不能判断算法终止位置

为了解决这个问题，Newman引入了模块度Q的概念。

# Newman快速算法

Newman M E. Fast algorithm for detecting community structure in networks[J]. Physical Review E, 2003.

# 模块度Q

模块度的定义如下：

模块度指网络中连接社区结构内部节点的边所占的比例与另外一个随机网络中连接社区结构内部节点的边所占比例的期望值相减得到的差值。

一系列推导后得到的公式为：

$$Q = \sum_{c \in C} \left( \frac{I_c}{m} - \left( \frac{D_c}{2m} \right)^2 \right)$$

其中$m$为网络中总边数，$I_c$为社区$c$中所有内部边的条数，$D_c$为社区$c$中所有节点的度之和

# 边介数中心度

Newman快速算法不仅解决了GN算法的速度问题，还提出了社区模块度$Q$ 的概念。

具体算法可描述如下：

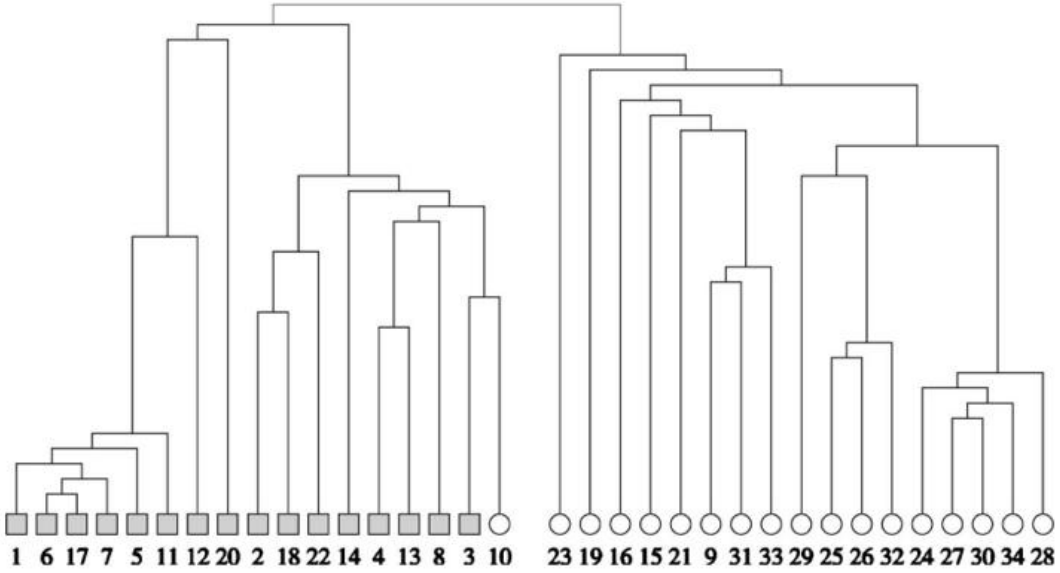
1）初始化：将每个节点视作一个社区，$Q$ 初值为0;

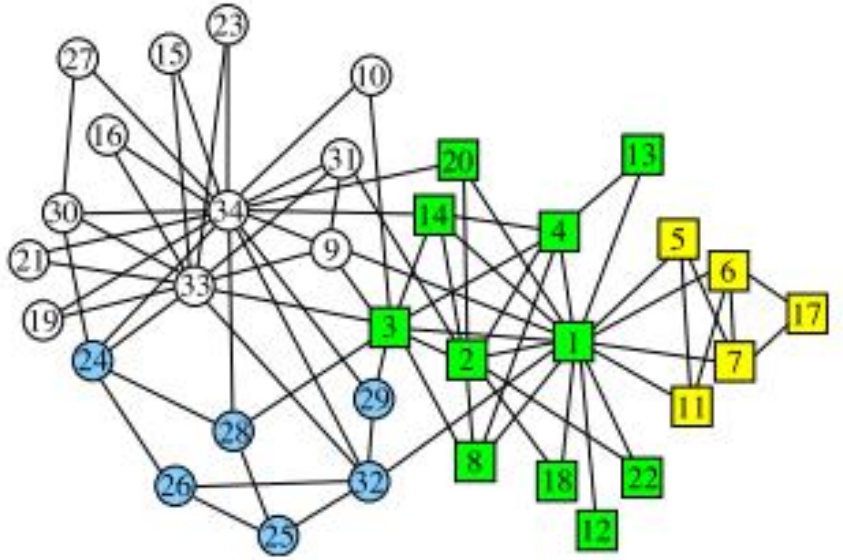
2）合并：计算将有边相连的两个社区合并后的$Q$ 值的增量$\Delta Q$ ，选择$\Delta Q$ 最大（或最小）的社区对进行合并；

3）终止条件：重复2），直到整个网络合并为一个社区

划分结果可以用树状图直观的表示出来，最优划分选择$Q$ 值最大的层次

# NF算法用于空手道俱乐部社区划分



湖南大學
HUNAN UNIVERSITY

# 谢谢观赏

实事求是　敢为人先