Graph Databases and Linked Data

So far: Objects are considered as iid
(independent and identical distributed)

$\Rightarrow$ the meaning of objects depends exclusively on the description

$\Rightarrow$ objects do not influence each other

In the following: Link-Mining

Objects are connected and dependent.

Examples: Publications are measures based on citations.

$\Rightarrow$ objects might depend on any connected object

$\Rightarrow$ databases become large networks (knowledge graphs)

Node Ranking

Idea: Select and rank nodes w.r.t. their relevance or interestingness in large networks.

Interestingness might depend on :

- influence to the complete networks
- key nodes for network flows

Applications:

- Ranking web sites and web pages
- Rank researchers in citation networks
- Rank importance of nodes representing crossing or routers in transportation networks

Centrality Measures

Idea: Centrality depends on the position of a node to the other nodes w.r.t. networks distance (=cost optimal path between two nodes)

Let d(v,t) be the length of the shortest path from v to t (v,t $\in$V) in G(V,E):

- **Closeness Centrality:** $C_C(v) = \frac{1}{\sum_{t \in V} d(v,t)}$
- **Graph Centrality:** $C_G(v) = \frac{1}{\max_{t \in V}(d(v,t))}$

Let σ_{st} be the number of shortest paths from s to t and
let $\sigma_{st}(v)$ be the number of shortest path from s to t containing v.

- **Stress Centrality:** $C_S(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v)$
- **Betweenness Centrality:** $C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$

Centrality Measures

Example: Let nodes represent routers in a computer network.
If the router having the highest betweenness centrality goes offline the most direct connections are affected.

Computation: Set of all-pair-shortest paths can be computed in *O(n³) time and using O(n²) memory by the* Floyd-Warshal algorithm.

theorem*: v is on the shortest path between s and t if and only if*

d(s,t) = d(s,v)+ d(v,t)

$$\Rightarrow \sigma_{st}(v) = \begin{cases} 0 & if \quad d(s,t) < d(s,v) + d(v,t) \\ \sigma_{sv} \cdot \sigma_{vt} & else \end{cases}$$

$\Rightarrow$ to compute the betweenness centrality it is not necessary to compute all paths

$\Rightarrow$ *there are faster solution:*

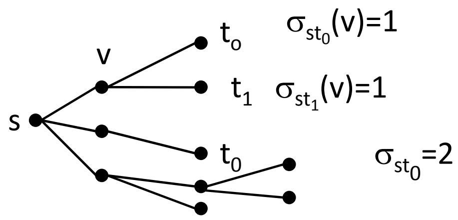
- *O(nm)* without edge weights
- *O(nm+n²log n)* in graphs having edge weights

where *n = |V|* and *m = |E|* in the graph *G(V,E)*

Computing Betweeness Centrality

Basic idea:

- Start a single source all target search from each node s. The result is a tree (called Dijkstra tree) containing all shortest paths starting with s.
- The Dijkstra tree also induces a distance ranking of all nodes to s.
- Visit each node v with descending distance to s and count all nodes t lying behind v in the tree ($\sigma_{st}(v)$) and the set of shortest paths from s to t (σ_{st})



Algorithm for unweighted graphs(1)

Variables and expressions:

- S: Stack storing nodes w.r.t to their distance to s
- Q: Priority Queue for the Dijkstra search (ordered by the distance to s)
- P[v]: List storing all predecessors of v
- d[v]: distance of the shortest path from s to v
- σ[v]: number of shortest paths from s to v
- δ[v]: Given $\delta_{st}[v]=\frac{\sigma_{st}[v]}{\sigma_{st}}$ then $\delta[v]=\delta_{s\bullet}(v)=\sum_{t\in V}\delta_{st}[v]=\sum_{w:v\in P[w]}\frac{\sigma_{sv}}{\sigma_{sw}}\cdot\left(1+\delta_{s\bullet}(w)\right)$

Workflow for each starting node s:

1. Phase: Algorithm computes the Dijkstra tree of s
2. Phase: traverse stack S and count the number of nodes behind each visited node v

Algorithmus für ungewichtet Graphen(2)

```
CB[v] := 0 ∀ v∈V
for s ∈ V
  S:= empty Stack;
  P[w] := empty List ∀ w∈V;
  σ[t] :=0  ∀t∈V; σ[s]:=1;
  d[t] :=-1 ∀t∈V; d[s]:0;
  Q := empty Queue;
  Q.push(0,s);
  while Q not empty do
    v := Q.pop();
    S.push(v);
    foreach neighbor w of v do
      if d[w] < 0 then
        d[w]:=d[v]+1;
        Q.push(d[w],w);
      end if
      if d[w]=d[v]+1 then
        σ(w):=σ(w)+σ(v)
        P[w].add(v)
      end if
    end for
  end while
  δ[v]:=0; v∈V;
  while S not empty do
    w:=S.pop();
    for v∈P[w] do
      δ[v]:= δ[v] + σ[v]/σ[w] · (1+δ[w]);
    end for
    if w≠s then
      CB[w]:=CB[w]+δ[w];
    end if
  end while
end for
```


Ranking nodes in hyperlinked Text

PageRank: (S.Brin/B. Page 1996)

- important component in ranking algorithms of search engines (in combination with other features)
- Data is considered a strongly connected, directed network G(V,E). (e.g. all HTML documents in a search engine)
- probabilistic surfer performs an infinite random walk. idea: visiting probability = importance of the page v.

DATABASE SYSTEMS GROUP

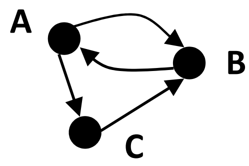
Anwendungen im Web Mining

LMU

Computing the PageRank

start distribution: *p0(u) = 1 / |V |*

$$E = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$



adjacency matrix: E

transition prob.: $L[u,v] = \dfrac{E[u,v]}{\sum_{\beta} E[u,\beta]}$

$$L = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

probability of page v at time i : $p_i[v] = \sum_{u \in V} L[u,v] p_{i-1}(u)$

distribution vector over all pages: $\vec{p}_i = L^T \vec{p}_{i-1}$

Computation by „Power Iterations“: $\vec{p}_i \leftarrow L^T \vec{p}_{i-1}$

after ca. 20-30 iterations result should be stable

Solution for none strongly connected graphs: 1. Remove nodes without outlink

2. Allow jumps during traversal

DATABASE SYSTEMS GROUP

Ranking Linked Objects

LMU

HITS (Kleinberg 1998): Hyperlink Induced Topic Search

- Consider only objects being relevant for *q* or being linked to relevant pages (in- and outlinks).

  $\Rightarrow$ $G_q(V_q, E_q)$ *for query q*

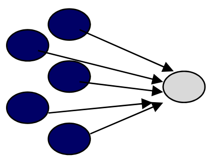
- *there are two types of objects* :

  Hubs: link to relevant objects (authorities)

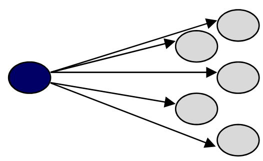
  Authorities: relevant objects being linked by hubs.

  => each object has an authority score and a hub score

  for each object u, h[u] denotes its hub score and a[u] its authority score.



a good authority is linked by many good hubs



a good hubs links to many good authorities.

Anwendungen im Web Mining

Computing HITS:

- $\vec{a}$ vector of authority scores over all objects v $\in V_q$
- $\vec{h}$ vector of hub scores over all objects v $\in V_q$
- Computation by mutual iterations: $\vec{a} = E^T \vec{h}$ (authority score)

  $\vec{h} = E\vec{a}$ (hub score)

Complete algorithm:

1. determine relevant objects (root set).
2. determine all pages linking relevant objects .(extended set)
3. iterate over all hub- and authority scores
4. Order the relevant pages by the authority scores

Link Prediction

Input: A graph G(V,E) and 2 nodes v,u $\in$ V where (v,u) $\notin$ E.

Output: Predict the existence of link (v,u) if:

- the existence is unknown.
- the link might develop at a future point in time

Examples:

- Links in social networks
- unknown protein interaction
- Customer product recommendations in bipartite graphs (Collaborative Filtering)

Feature-based Link Prediction

Idea: Use the features of pairs of objects to describe their relationship.

Example:

- Common interests in social networks
- Co-authors do research in the same area
- proteins have complementary active regions

⇒ Links do develop by accident, there are reasons which might be found in the feature values

⇒ Link Prediction: Learn a classifier that maps pairs of feature descriptions to link probabilities

⇒ Formal: Let u,v ∈ V and let F(v),F(u) be their feature descriptions. Then, Link Prediction is the task to learn a function P: (F(v),F(u))-> L.
(L is either discrete {link, no link} or real-valued [0,..max_Strength])

Topology-Based Linke Prediction

Problem: Feature-based approach do not consider network proximity.

Example:

- Persons having similar interests might not have any contact
- Proteins might dock but do not appear in the same natural surrounding

Solution: Integrate the neighborhood of v and u in G.

⇒ common neighbors increase the likelihood of a link

⇒ describe a node by its adjacency list or the subnetwork being influenced by the node

Link Prediction and Matrix Factorisation

Input: Graph G(V,E) with adjacency matrix A and let $E_u \subseteq E$ be the set of links with unknown existence or strength.

Method:

- Factorizing A allows to find a latent k-dimensional space (k is the rank of A) (Factorization can be done regardless of missing entries)
- nodes can be expressed in this latent space
- remapping of the nodes to the |V| dimensional space fills up the unknown entries E_u .

Vorgehen:

- Factorize A in the n×k Matrix B while minimizing L(B) the : $A' = BB^T$

$$L(B) = \sum_{a_{i,j} \in A \setminus U} \left| a_{i,j} - a'_{i,j} \right|^2 = \sum_{a_{i,j} \in A \setminus U} \left| a_{i,j} - \left\langle b_{i,*}, b_{*,j} \right\rangle \right|^2$$

Computation: Gradient descent on the derivate of L(B).

Remark: Also applicable to bipartite graphs (customer/ product)

Dense Subgraph Discovery

- Find „dense“ subgraphs in a network G(V,E).
- Definitions of „dense“:
  - cliques (complete subgraphs)
  - quasi-cliques (at least x % of the edges must exist)
  - relative density of the surrounding: in node in subgraph G‘ has more links to other node from G‘ than to nodes G \ G‘.
- …
- Problem: almost all definitions lead to NP-hard search problems
  => heuristic solutions
  => practical use is limited

Graph Clustering

- class of clustering methods that treat the data set as graph
- Object= node; links distance, similarity, reachability distance...
- usually: only consider the k-nearest neighbors or an ε-range
  => directed and undirected network are considered

Clustering by weighted k-mincut: Partition a graph G into k disjunctive subgraphs having similar size while minimizing the number of removed edges.

=> Weighted k-mincut is also an NP-hard problem.

Graph Clustering: Spectral Clustering

- built a symmetric adjacency matrix S: $S_{i,j} = sim(x_i, x_j)$
- Transform S into a graph Laplacian matrix L:

$$L = I - D^{-\frac{1}{2}}\, S\, D^{-\frac{1}{2}} \qquad D_{i,j} = \begin{cases} \sum_k sim(x_i, x_k) & if \quad i = j \\ 0 & else \end{cases}$$

- after eigenvalue decomposition of L:
  - Eigenvectors with eigenvalues = 0, represent connected components
  - Eigenvectors describe the linear weights to represent a cluster representative

$$r_k = \sum_{i=1}^{|DB|} EV_i \cdot o_i$$

Conclusions Graph Mining

- Graph-Mining includes new data mining tasks
  - Ranking nodes
  - Link prediction
  - Dense subgraph discovery and community detection
  - Frequent Subgraph Mining

- Clustering can be formulated as a graph problem
  - Density-based clustering: find all connected components where links denote a similarity predicate
  - Spectral clustering
  - weighted k-mincut: Partition a graph into k subgraphs while minimizing the weights of the cut edges under size constraints w.r.t. the result subgraphs.

Literature

- Borgwardt K., Kriegel H.-P.: „Shortest-path kernels on graphs“. In Proc. Intl. Conf. Data Mining (ICDM 2005), 2005

- Borgwardt K.: „Graph Kernels“, Dissertation im Fach Informatik, Ludwig-Maximilians-Universität München, 2007

- Bunke, H. : „Recent developments in graph matching“. In ICPR, pages 2117–2124. 2000

- Gärtner, T., Flach, P., and Wrobel: „On graph kernels: Hardness results and efficient alternatives.“ Proc. Annual Conf. Computational Learning Theory, pages 129–143, 2003

- Wiener, H.: „Structural determination of paraffin boiling points“. J. Am. Chem. Soc., 69(1):17–20, 1947

Literature

- Yan X., Han J.:“gSpan: Graph-based substructure pattern mining“, In ICDM, 2002.

- Kuramochi M., Karypis G.:“Frequent Subgraph Discovery“, In ICDM, 2001

- Brin S., Page L.:“The anatomy of a large-scale hypertextual Web search engine“, Computer Networks and ISDN Systems, Vol 30, Nr.1-7, S.107-117,1998

- Kleinberg J. M.:“Authoritative sources in a hyperlinked environment“, Journal of the ACM,Vol.46, Nr. 5, S. 604-632,1999

- Brandes U.:“A faster Algorithm for Betweenness Centrality“, Journal of Mathematical Sociology, 25(2):163-177, 2001