TDDD43

Theme 2.2: Keyword Search in Databases

Fang Wei-Kleiner
http://www.ida.liu.se/~TDDD43

Keyword search

- Keyword search is a well known mechanism for retrieving relevant information from a set of documents.
  - Google is a familiar example !
- What about structured data?
  - Such as **XML** documents or **Relational Databases**?
- Current enterprise search engines in structured data requires:
  - **Knowledge** of schema
  - **Knowledge** of a query language
  - **Knowledge** of the role of the keywords
- Do users have all of the above **Knowledge** ?

Keyword search in rel. DB

- Users need a simple system that receives some **keywords** as input and returns a **set of nodes** that together cover all or part of the input keywords as output.

- Relational databases can be modeled using **graphs**:
  - Tuples are **nodes** of the graph.
  - Foreign key relationships are **edges** that connect two nodes (tuples) to each other.

Keyword search in rel. DB

Cities

ID	Name	Country
22	Toronto	CA
16	New York	US

Organizations

ID	Name	Head Q.
135	UN	16
175	EU	81

Countries

Code	Name
CA	Canada
US	United States

Memberships

Country	Org.
CA	135
US	135

New York is Located in United States

Cities

ID	Name	Country
22	Toronto	CA
16	New York	US

Organizations

ID	Name	Head Q.
135	UN	16
175	EU	81

Countries

Code	Name
CA	Canada
US	United States

Memberships

Country	Org.
CA	135
US	135

Keywords : "New York" "United States"

New York hosts UN and Canada is a member

Cities

ID	Name	Country
22	Toronto	CA
16	New York	US

Organizations

ID	Name	Head Q.
135	UN	16
175	EU	81

Countries

Code	Name
CA	Canada
US	United States

Memberships

Country	Org.
CA	135
US	135

Keywords : "New York" "Canada"

Keyword search in DBs

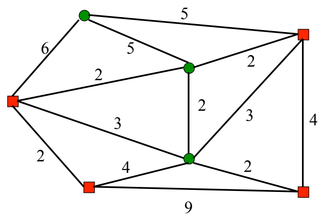
- Database is modeled as a graph where vertices can be objects, tuples, etc.) and edges reflect the relationship between vertices.
  - Undirected graph vs. directed graph
  - Weighted graph vs. un-weighted graph
  - Labeled edges vs. non-labeled edges

- Keywords → vertices in the graph
- Keyword search in DB → Steiner tree computation
- Ranked keyword search → ranked Steiner tree

Steiner tree problem

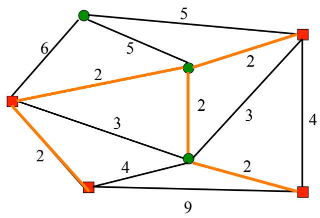
- Given a weighted graph $G = (V, E)$ of order $n = |V|$ and a set $S \subseteq V$ of $k = |S|$ to find a minimum cost sub-tree $T = T(S)$ of G connecting (spanning) all terminal nodes.
  - Minimal cost → the sum of weights of the edges in T is a minimum.

Steiner tree example



terminals

Steiner tree example



Special cases

- $k = 2$, STP is a shortest path problem between the two given terminals.
- $k = 3$, for three given points $z1$, $z2$ and $z3$, STP is to find a vertex v that minimizes $d(v; z1) + d(v; z2) + d(v; z3)$
- $k = n$, STP is a minimal spanning tree problem (MSTP) of G

Steiner tree: NP-hardness

- Steiner tree problem is hard.

The decision problem:

Given an undirected graph $G = (V,E)$, a subset Y of V and a value i, is there a Steiner tree T spanning Y with the length of T which is shorter than i?

is NP-hard.

Dreyfuss-Wagner algorithm

- The Dreyfus–Wagner algorithm solves the Steiner tree problem for $S \subseteq V$ by dynamic programming.
- It computes optimal trees $T(X \cup v)$ for all $X \subseteq S$ and $v \in V$ recursively.
  - Assume first that v is a leaf of the (unknown) optimal tree $T(X \cup v)$.
  - Then v is joined in $T(X \cup v)$ to some node w of $T(X \cup v)$ along a shortest path P_{vw}, such that either $w \in X$ or $w \notin X$. In both cases we have

$$T(X \cup v) = P_{vw} \cup T(X \cup w)$$

Dreyfuss-Wagner algorithm

- We can decompose

$$T(X \cup w) = T(X' \cup w) \cup T(X'' \cup w)$$

- for some nontrivial bipartition $X = X' \cup X''$.

- We may thus write

$$T(X \cup v) = \min (P_{vw} \cup T(X' \cup w) \cup T(X'' \cup w))$$

- where the minimum is taken over all $w \in V$ and all nontrivial bipartitions $X = X' \cup X''$.

Algorithm

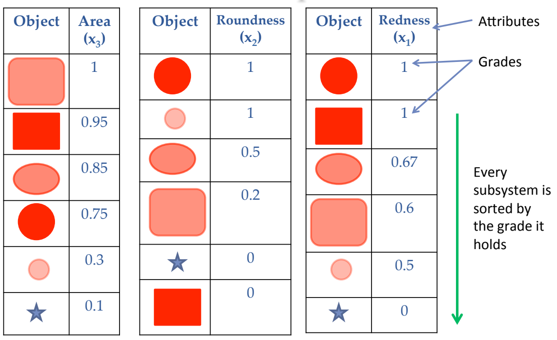
Algorithm (Dreyfus-Wagner algorithm)
Step 1: Compute P_{wv} for all $w, v \in V$.
Step 2: For $i = 2$ to $k - 1$,
For any $|X| = i$ and any $w \in Y$, $v \in V \setminus X$
$T(X \cup v) = \min P_{vw} \cup T(X' \cup w) \cup T(X'' \cup w)$.

Complexity: $O(n \cdot 3^k + n^2 \cdot 2^k + n^3)$

Ranked queries

- Given a collection of objects, our goal is to find Top-k objects, whose scores are greater than the remaining objects.

Example



Top-k object problem

- Naïve algorithm
- Basic Idea:
  - For for each object, use the aggregation function to get the score
  - According to the scores, get the top k.
- Problem: inefficiency
- Question:
  - Do we need to count the score for every object in the database?
  - Can we SAFELY ignore some objects whose scores are lower than what we already have?

Fagin's algorithm

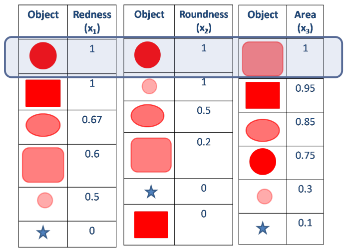
- Do Sorted access in parallel at all the lists
- Stop when we have *k* objects which appear in all the lists
- Calculate score value of all the objects
- Compute Top-k objects

Example: Fagin's algorithm

1 Objects appear in every list:

{ }

Objects seen so far:



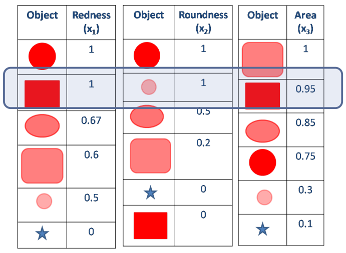
k = 3

Example: Fagin's algorithm

2 Objects appear in every list:

{ }

Objects seen so far:

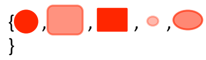


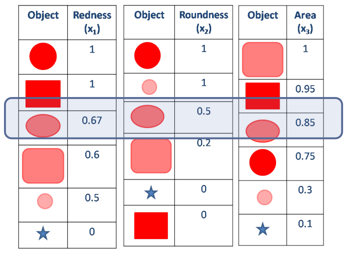
k = 3

Example: Fagin's algorithm

3 Objects appear in every list:

Objects seen so far:

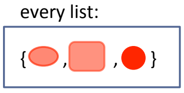




k = 3

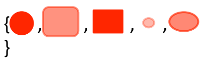
Example: Fagin's algorithm

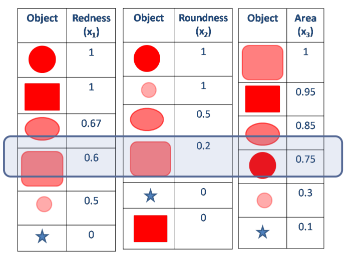
4 Objects appear in every list:



We got enough objects

Objects seen so far:

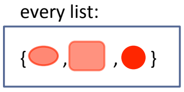




k = 3

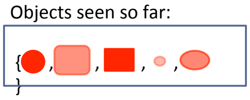
Example: Fagin's algorithm

4 Objects appear in every list:

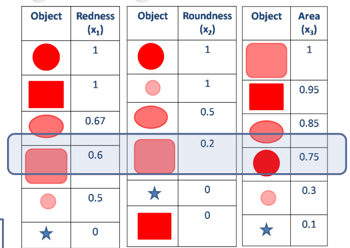


We got enough objects

Objects seen so far:



For all these, calculate the score and get the Top-k



k = 3

Ranked Steiner tree with 3 terminals

- For three given points *z1*, *z2* and *z3*, STP is to find a vertex *v* that minimizes *d(v; z1) + d(v; z2) + d(v; z3)*

Ranked Steiner tree with 3 terminals

v	*d(v,z1)*
12	1
256	3
9	4
55	6
137	7
474	8
987	10
33	12
787	15

v	*d(v,z2)*
256	3
345	13
678	14
347	16
55	17
890	18
235	25
57	32
564	35

v	*d(v,z3)*
999	20
64	21
954	22
332	23
256	24
55	24
687	26
1	37
33	40

Find the top-2 Steiner trees of terminals (z1, z2, z3)