Finding Top-k Min-Cost Connected Trees in Databases

Bolin Ding[1] Jeffrey Xu Yu[1] Shan Wang[2] Lu Qin[1]
Xiao Zhang[2] Xuemin Lin[3]

[1]Department of System Engineering and Engineering Management
The Chinese University of Hong Kong

[2]School of Informaion
Renmin University of China

[3]School of Computer Science and Engineering
The University of New South Wales

IEEE 23rd International Conference on Data Engineering

Outline

1. Keyword Search in Relational Databases
   - Database Graph, Query, and Answer
   - The Hardness of This Problem
2. Our New Parameterized Solutions
   - Finding Top-1 Answer
   - Finding Top-k Answers
3. Existing Solutions
   - Other Graph-Based Solutions
4. Experimental Studies
   - Some Representative Experimental Results

Outline

Outline

1. Keyword Search in Relational Databases
   - Database Graph, Query, and Answer
   - The Hardness of This Problem
2. Our New Parameterized Solutions
   - Finding Top-1 Answer
   - Finding Top-k Answers
3. Existing Solutions
   - Other Graph-Based Solutions
4. Experimental Studies
   - Some Representative Experimental Results

Outline

1. Keyword Search in Relational Databases
   - Database Graph, Query, and Answer
   - The Hardness of This Problem
2. Our New Parameterized Solutions
   - Finding Top-1 Answer
   - Finding Top-k Answers
3. Existing Solutions
   - Other Graph-Based Solutions
4. Experimental Studies
   - Some Representative Experimental Results

Outline

Weighted Database Graph $G(V, E, W)$

Node set V

- Nodes - tuples in database, $|V| = n$

Edge set E

- Edges - foreign key references between tuples, $|E| = m$

Edge Weight W

- Edge weight $w_e((v, u)) = \log_2(1 + \max\{d_v, d_u\})$ (d_x - degree of node x)
- The lower, the tighter
- Intuition: *the relationship between one node and the others is distributed*

Weighted Database Graph $G(V, E, W)$

Node set V

- Nodes - tuples in database, $|V| = n$

Edge set E

- Edges - foreign key references between tuples, $|E| = m$

Edge Weight W

- Edge weight $w_e((v, u)) = \log_2(1 + \max\{d_v, d_u\})$ (d_x - degree of node x)
- The lower, the tighter
- Intuition: *the relationship between one node and the others is distributed*

Weighted Database Graph $G(V, E, W)$

Node set V

- Nodes - tuples in database, $|V| = n$

Edge set E

- Edges - foreign key references between tuples, $|E| = m$

Edge Weight W

- Edge weight $w_e((v, u)) = \log_2(1 + \max\{d_v, d_u\})$ (d_x - degree of node x)
- The lower, the tighter
- Intuition: *the relationship between one node and the others is distributed*

Query and Answer

Query

- l keywords $p_1, p_2, \cdots, p_l$
- or l subsets $V_1, V_2, \cdots, V_l \subseteq V$ (V_i contains keyword p_i)

Answer

- Connected subtree T in G containing the l keywords
- or Group Steiner tree T, s.t. $V(T) \cap V_i \neq \varnothing$ $(i = 1, \cdots, l)$

Objective

- Cost of answer T: $s(T) = \sum_{(u,v)\in E(T)} w_e((u, v))$ (linear combination of node/edge weight)
- Output answers $T_1, \cdots, T_k$, with top-k minimum costs

(Top-k) Minimum Group Steiner Tree Problem

Query and Answer

Query

- l keywords $p_1, p_2, \cdots, p_l$
- or l subsets $V_1, V_2, \cdots, V_l \subseteq V$ (V_i contains keyword p_i)

Answer

- Connected subtree T in G containing the l keywords
- or Group Steiner tree T, s.t. $V(T) \cap V_i \neq \varnothing$ $(i = 1, \cdots, l)$

Objective

- Cost of answer T: $s(T) = \sum_{(u,v)\in E(T)} w_e((u,v))$ (linear combination of node/edge weight)
- Output answers $T_1, \cdots, T_k$, with top-k minimum costs

(Top-k) Minimum Group Steiner Tree Problem

Example

Database

Author

AID	Name
a1	Jim
a2	Robin

Citation

Cite	Cited
t1	t2
t3	t2
t5	t4
t6	t7

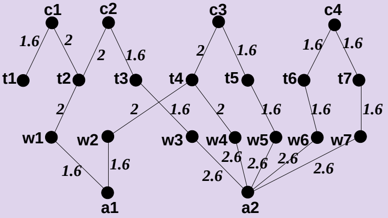
Paper

PID	Title
t1	Keyword Search on RDBMS
t2	Steiner Problem in DB
t3	Efficient IR−Query over DB
t4	Online Cluster Problems
t5	Keyword Query over Web
t6	Query Optimization on DB
t7	Parameterized Complexity

Paper−Author

PID	AID
t2	a1
t4	a1
t3	a2
t4	a2
t5	a2
t6	a2
t7	a2

Database Graph



Query

Keyword (p_1), Query (p_2), DB (p_3), and Jim (p_4)

Example

Database

Author

AID	Name
a1	Jim
a2	Robin

Citation

Cite	Cited
t1	t2
t3	t2
t5	t4
t6	t7

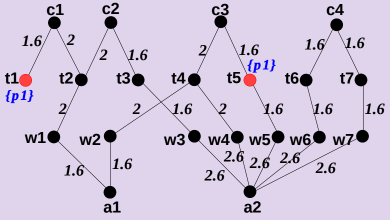
Paper

PID	Title
t1	Keyword Search on RDBMS
t2	Steiner Problem in DB
t3	Efficient IR−Query over DB
t4	Online Cluster Problems
t5	Keyword Query over Web
t6	Query Optimization on DB
t7	Parameterized Complexity

Paper−Author

PID	AID
t2	a1
t4	a1
t3	a2
t4	a2
t5	a2
t6	a2
t7	a2

Database Graph



Query

Keyword (p_1), Query (p_2), DB (p_3), and Jim (p_4)

Example

Database

Author

AID	Name
a1	Jim
a2	Robin

Citation

Cite	Cited
t1	t2
t3	t2
t5	t4
t6	t7

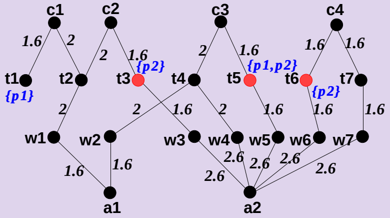
Paper

PID	Title
t1	Keyword Search on RDBMS
t2	Steiner Problem in DB
t3	Efficient IR−Query over DB
t4	Online Cluster Problems
t5	Keyword Query over Web
t6	Query Optimization on DB
t7	Parameterized Complexity

Paper−Author

PID	AID
t2	a1
t4	a1
t3	a2
t4	a2
t5	a2
t6	a2
t7	a2

Database Graph



Query

Keyword (p_1), Query (p_2), DB (p_3), and Jim (p_4)

Example

Database

Author

AID	Name
a1	Jim
a2	Robin

Citation

Cite	Cited
t1	t2
t3	t2
t5	t4
t6	t7

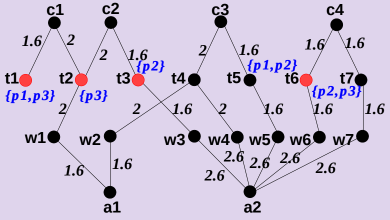
Paper

PID	Title
t1	Keyword Search on RDBMS
t2	Steiner Problem in DB
t3	Efficient IR–Query over DB
t4	Online Cluster Problems
t5	Keyword Query over Web
t6	Query Optimization on DB
t7	Parameterized Complexity

Paper–Author

PID	AID
t2	a1
t4	a1
t3	a2
t4	a2
t5	a2
t6	a2
t7	a2

Database Graph



Query

Keyword (p_1), Query (p_2), DB (p_3), and Jim (p_4)

Example

Database

Author

AID	Name
a1	Jim
a2	Robin

Paper

PID	Title
t1	Keyword Search on RDBMS
t2	Steiner Problem in DB
t3	Efficient IR−Query over DB
t4	Online Cluster Problems
t5	Keyword Query over Web
t6	Query Optimization on DB
t7	Parameterized Complexity

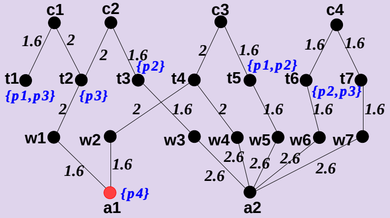
Paper−Author

PID	AID
t2	a1
t4	a1
t3	a2
t4	a2
t5	a2
t6	a2
t7	a2

Citation

Cite	Cited
t1	t2
t3	t2
t5	t4
t6	t7

Database Graph



Query

Keyword (p_1), Query (p_2), DB (p_3), and Jim (p_4)

Example

Database

Author

AID	Name
a1	Jim
a2	Robin

Citation

Cite	Cited
t1	t2
t3	t2
t5	t4
t6	t7

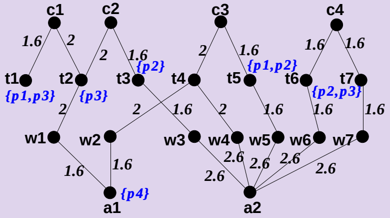
Paper

PID	Title
t1	Keyword Search on RDBMS
t2	Steiner Problem in DB
t3	Efficient IR−Query over DB
t4	Online Cluster Problems
t5	Keyword Query over Web
t6	Query Optimization on DB
t7	Parameterized Complexity

Paper−Author

PID	AID
t2	a1
t4	a1
t3	a2
t4	a2
t5	a2
t6	a2
t7	a2

Database Graph

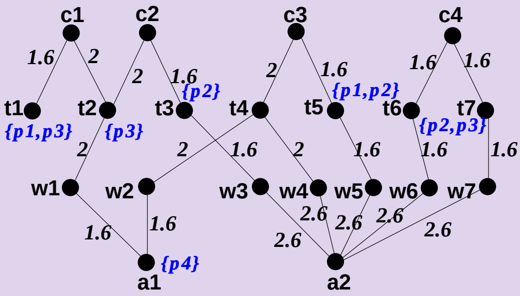


Query

Keyword (p_1), Query (p_2), DB (p_3), and Jim (p_4)

Example

Database Graph



Node sets containing keywords:
$V_1 = \{t1, t5\}$, $V_2 = \{t3, t5, t6\}$,
$V_3 = \{t1, t2, t6\}$, and $V_4 = \{a1\}$.

Answer T_1

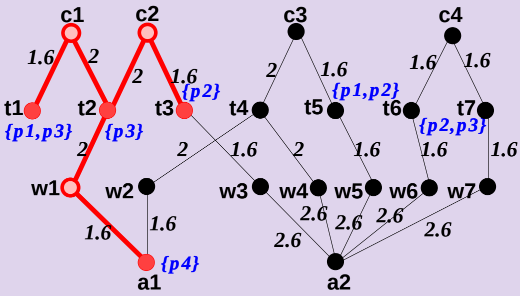
- Jim (a_1) writes a paper, t_2, which is cited by two papers, t_1 and t_3.
- Cost: $s(T_1) = 10.8$

Answer T_2

- Jim (a_1) writes a paper, t_2, which is cited by t_3; the author of t_3, Robin (a_2), writes another paper t_5.
- Cost: $s(T_2) = 15.6$

Example

Database Graph



Node sets containing keywords:
$V_1 = \{t1, t5\}$, $V_2 = \{t3, t5, t6\}$,
$V_3 = \{t1, t2, t6\}$, and $V_4 = \{a1\}$.

Answer T_1

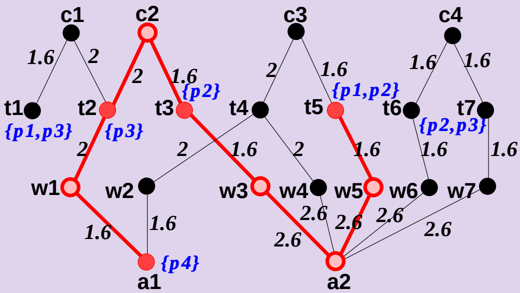
- Jim (a_1) writes a paper, t_2, which is cited by two papers, t_1 and t_3.
- Cost: $s(T_1) = 10.8$

Answer T_2

- Jim (a_1) writes a paper, t_2, which is cited by t_3; the author of t_3, Robin (a_2), writes another paper t_5.
- Cost: $s(T_2) = 15.6$

Example

Database Graph



Node sets containing keywords:
$V_1 = \{t1, t5\}$, $V_2 = \{t3, t5, t6\}$,
$V_3 = \{t1, t2, t6\}$, and $V_4 = \{a1\}$.

Answer T_1

- Jim (a_1) writes a paper, t_2, which is cited by two papers, t_1 and t_3.
- Cost: $s(T_1) = 10.8$

Answer T_2

- Jim (a_1) writes a paper, t_2, which is cited by t_3; the author of t_3, Robin (a_2), writes another paper t_5.
- Cost: $s(T_2) = 15.6$

Outline

1. Keyword Search in Relational Databases
   - Database Graph, Query, and Answer
   - The Hardness of This Problem
2. Our New Parameterized Solutions
   - Finding Top-1 Answer
   - Finding Top-k Answers
3. Existing Solutions
   - Other Graph-Based Solutions
4. Experimental Studies
   - Some Representative Experimental Results

Hardness Results

The Hardness of (Top-k) Minimum Group Steiner Tree Problem

- NP-Complete
- Harder than Undirected Minimum Steiner Tree Problem
- Equivalent to Directed Minimum Steiner Tree Problem
- NP-Hard to find $(1+\epsilon)$-approximation for any fixed $\epsilon > 0$
- NP-Hard to find $(1-\epsilon)\log l$-approximation for any fixed $\epsilon > 0$

Summary of Our Solution

Two Steps

- Optimal top-1 answer with the minimum cost
- Approximate top-k answers

Parameterized Complexity

- Time complexity: $O(3^l n + 2^l ((l + \log n)n + m))$
- Space complexity: $O(2^l n)$
- (For fixed l) Time complexity: $O(n \log n + m)$
- (For fixed l) Space complexity: $O(n)$
- Efficient because: n (the number of tuples) is large; m (the number of foreign keys) is large; l (the number of keywords) is small

Summary of Our Solution

Two Steps

- Optimal top-1 answer with the minimum cost
- Approximate top-k answers

Parameterized Complexity

- Time complexity: $O(3^l n + 2^l((l + \log n)n + m))$
- Space complexity: $O(2^l n)$
- (For fixed l) Time complexity: $O(n \log n + m)$
- (For fixed l) Space complexity: $O(n)$
- Efficient because: n (the number of tuples) is large; m (the number of foreign keys) is large; l (the number of keywords) is small

Outline

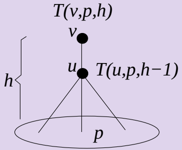
1. Keyword Search in Relational Databases
   - Database Graph, Query, and Answer
   - The Hardness of This Problem
2. Our New Parameterized Solutions
   - Finding Top-1 Answer
   - Finding Top-k Answers
3. Existing Solutions
   - Other Graph-Based Solutions
4. Experimental Studies
   - Some Representative Experimental Results

Naive Dynamic Programming

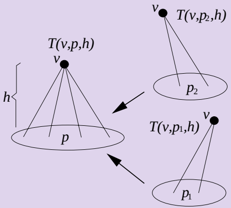
Optimal Substructure $T(v, \mathbf{p}, h)$

Subtree $T(v, \mathbf{p}, h)$: (i) rooted at node $v \in V$; (ii) with height $\leq h$; (iii) containing a set of keywords $\mathbf{p}$; (iv) with the minimum cost.

Tree Grow Case



Tree Merge Case



Dynamic Programming Equation

Tree Grow: $\mathrm{T_g}(v, \mathbf{p}, h) = \min\limits_{u \in N(v)} \{(v, u) \oplus T(u, \mathbf{p}, h-1)\}$

Tree Merge: $\mathrm{T_m}(v, \mathbf{p}_1 \cup \mathbf{p}_2, h) = \min\limits_{\mathbf{p}_1 \cap \mathbf{p}_2 = \emptyset} \{T(v, \mathbf{p}_1, h) \oplus T(v, \mathbf{p}_2, h)\}$

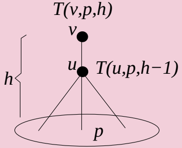
($\oplus$: merge two trees into a new tree)

Naive Dynamic Programming

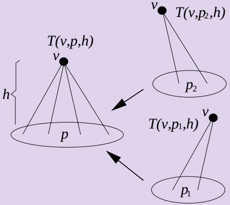
Optimal Substructure $T(v, \mathbf{p}, h)$

Subtree $T(v, \mathbf{p}, h)$: (i) rooted at node $v \in V$; (ii) with height $\leq h$; (iii) containing a set of keywords $\mathbf{p}$; (iv) with the minimum cost.

Tree Grow Case



Tree Merge Case



Dynamic Programming Equation

Tree Grow: $\mathrm{T_g}(v, \mathbf{p}, h) = \min_{u \in N(v)} \{(v, u) \oplus T(u, \mathbf{p}, h-1)\}$

Tree Merge: $\mathrm{T_m}(v, \mathbf{p}_1 \cup \mathbf{p}_2, h) = \min_{\mathbf{p}_1 \cap \mathbf{p}_2 = \emptyset} \{T(v, \mathbf{p}_1, h) \oplus T(v, \mathbf{p}_2, h)\}$

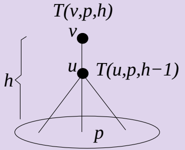
($\oplus$: merge two trees into a new tree)

Naive Dynamic Programming

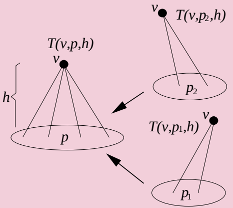
Optimal Substructure $T(v, \mathbf{p}, h)$

Subtree $T(v, \mathbf{p}, h)$: (i) rooted at node $v \in V$; (ii) with height $\leq h$; (iii) containing a set of keywords $\mathbf{p}$; (iv) with the minimum cost.

Tree Grow Case



Tree Merge Case



Dynamic Programming Equation

Tree Grow: $\mathrm{T_g}(v, \mathbf{p}, h) = \min\limits_{u \in N(v)} \{(v, u) \oplus T(u, \mathbf{p}, h-1)\}$

Tree Merge: $\mathrm{T_m}(v, \mathbf{p}_1 \cup \mathbf{p}_2, h) = \min\limits_{\mathbf{p}_1 \cap \mathbf{p}_2 = \emptyset} \{T(v, \mathbf{p}_1, h) \oplus T(v, \mathbf{p}_2, h)\}$

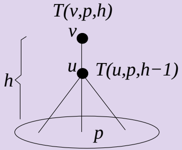
($\oplus$: merge two trees into a new tree)

Naive Dynamic Programming

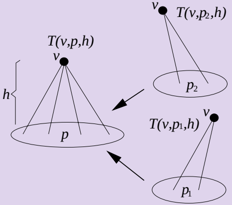
Optimal Substructure $T(v, \mathbf{p}, h)$

Subtree $T(v, \mathbf{p}, h)$: (i) rooted at node $v \in V$; (ii) with height $\leq h$; (iii) containing a set of keywords $\mathbf{p}$; (iv) with the minimum cost.

Tree Grow Case



Tree Merge Case



Dynamic Programming Equation

Tree Grow: $\mathrm{T_g}(v, \mathbf{p}, h) = \min_{u \in N(v)} \{(v, u) \oplus T(u, \mathbf{p}, h-1)\}$

Tree Merge: $\mathrm{T_m}(v, \mathbf{p}_1 \cup \mathbf{p}_2, h) = \min_{\mathbf{p}_1 \cap \mathbf{p}_2 = \emptyset} \{T(v, \mathbf{p}_1, h) \oplus T(v, \mathbf{p}_2, h)\}$

($\oplus$: merge two trees into a new tree)

Naive Dynamic Programming

Dynamic Programming Equation

- $T(v, \mathbf{p}, h) = \min\{\mathrm{T_g}(v, \mathbf{p}, h), \mathrm{T_m}(v, \mathbf{p}, h), T(v, \mathbf{p}, h-1)\}$
- $T(v, \mathbf{p}, 0) = 0$, for v containing keywords $\mathbf{p}$

Naive Dynamic Programming Algorithm

- Compute $T(v, \mathbf{p}, h)$ in the ascending order of h
- Optimal top-1 answer: $\min_{v \in V}\{T(v, \{\mathbf{p}_1, \cdots, \mathbf{p}_l\}, n)\}$
- Time complexity: $O(3^l n^2 + 2^l nm)$
- Space complexity: $O(2^l n^2)$

Speedup of Dynamic Programming

Main Idea

1. Reduce the search space of dynamic programming
2. Speed up the computation of dynamic programming equation

Complexity

- Time complexity: $O(3^l n + 2^l((l + \log n)n + m))$
- Space complexity: $O(2^l n)$

Reducing the Search Space: $T(v, \mathbf{p}, h) \rightarrow T(v, \mathbf{p})$

Origin Dynamic Programming Equation

$$
\begin{aligned}
T(v, \mathbf{p}, 0) &= 0 \text{ for } v \text{ containing keywords } \mathbf{p}, \\
T(v, \mathbf{p}, h) &= \min\{\mathrm{T_g}(v, \mathbf{p}, h), \mathrm{T_m}(v, \mathbf{p}, h), T(v, \mathbf{p}, h-1)\}, \\
\text{where } \mathrm{T_g}(v, \mathbf{p}, h) &= \min_{u \in N(v)} \{(v, u) \oplus T(u, \mathbf{p}, h-1)\}, \\
\mathrm{T_m}(v, \mathbf{p}_1 \cup \mathbf{p}_2, h) &= \min_{\mathbf{p}_1 \cap \mathbf{p}_2 = \emptyset} \{T(v, \mathbf{p}_1, h) \oplus T(v, \mathbf{p}_2, h)\}.
\end{aligned}
$$

Presence of Height h

- Promising the correctness
- Size of the search space: $O(n^2 2^l)$
- Serving as an order to compute $T(v, \mathbf{p}, h)$

Reducing the Search Space: $T(v, \mathbf{p}, h) \rightarrow T(v, \mathbf{p})$

Origin Dynamic Programming Equation

$$
\begin{aligned}
T(v, \mathbf{p}, 0) &= 0 \text{ for } v \text{ containing keywords } \mathbf{p}, \\
T(v, \mathbf{p}, h) &= \min\{\mathrm{T_g}(v, \mathbf{p}, h), \mathrm{T_m}(v, \mathbf{p}, h), T(v, \mathbf{p}, h-1)\}, \\
\text{where } \mathrm{T_g}(v, \mathbf{p}, h) &= \min_{u \in N(v)} \{(v, u) \oplus T(u, \mathbf{p}, h-1)\}, \\
\mathrm{T_m}(v, \mathbf{p}_1 \cup \mathbf{p}_2, h) &= \min_{\mathbf{p}_1 \cap \mathbf{p}_2 = \emptyset} \{T(v, \mathbf{p}_1, h) \oplus T(v, \mathbf{p}_2, h)\}.
\end{aligned}
$$

Presence of Height h

- Promising the correctness
- Size of the search space: $O(n^2 2^l)$
- Serving as an order to compute $T(v, \mathbf{p}, h)$

Reducing the Search Space: $T(v, \mathbf{p}, h) \to T(v, \mathbf{p})$

Simplified Dynamic Programming Equation

$$
\begin{aligned}
T(v, \mathbf{p}, 0) &= 0 \text{ for } v \text{ containing keywords } \mathbf{p}, \\
T(v, \mathbf{p}, h) &= \min\{\mathrm{T_g}(v, \mathbf{p}, h), \mathrm{T_m}(v, \mathbf{p}, h), T(v, \mathbf{p}, h-1)\}, \\
\text{where } \mathrm{T_g}(v, \mathbf{p}, h) &= \min_{u \in N(v)} \{(v, u) \oplus T(u, \mathbf{p}, h-1)\}, \\
\mathrm{T_m}(v, \mathbf{p}_1 \cup \mathbf{p}_2, h) &= \min_{\mathbf{p}_1 \cap \mathbf{p}_2 = \emptyset} \{T(v, \mathbf{p}_1, h) \oplus T(v, \mathbf{p}_2, h)\}.
\end{aligned}
$$

Absence of Height h

- Still promising the correctness
- Size of the search space: $O(n2^l)$
- Need to find another order to compute $T(v, \mathbf{p})$

Reducing the Search Space: $T(v, \mathbf{p}, h) \rightarrow T(v, \mathbf{p})$

Simplified Dynamic Programming Equation

$$
\begin{aligned}
T(v, \mathbf{p}) &= 0 \text{ for } v \text{ containing keywords } \mathbf{p}, \\
T(v, \mathbf{p}) &= \min(\mathrm{T_g}(v, \mathbf{p}), \mathrm{T_m}(v, \mathbf{p})), \\
\text{where } \mathrm{T_g}(v, \mathbf{p}) &= \min_{u \in N(v)} \{(v, u) \oplus T(u, \mathbf{p})\}, \\
\mathrm{T_m}(v, \mathbf{p}_1 \cup \mathbf{p}_2) &= \min_{\mathbf{p}_1 \cap \mathbf{p}_2 = \emptyset} \{T(v, \mathbf{p}_1) \oplus T(v, \mathbf{p}_2)\}.
\end{aligned}
$$

Absence of Height h

- Still promising the correctness
- Size of the search space: $O(n2^l)$
- Need to find another order to compute $T(v, \mathbf{p})$

The Order to Compute $T(v, \mathbf{p})$

Requirement of the Order to Compute $T(v, \mathbf{p})$

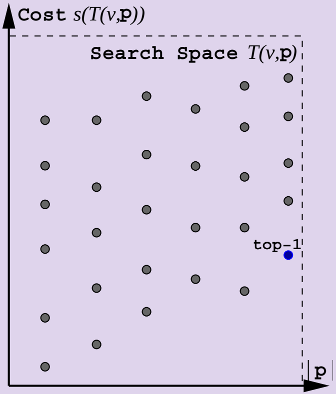
IF $T(v', \mathbf{p}')$ is a subtree of $T(v, \mathbf{p})$,
THEN $T(v', \mathbf{p}')$ must be computed earlier than $T(v, \mathbf{p})$.

Two Possible Orders

1. The ascending order of $|\mathbf{p}|$ (size of keywords set $\mathbf{p}$) - an unpublished work by Benny Kimelfeld and Yehoshua Sagiv (www.cs.huji.ac.il/~bennyk/papers/steiner06.pdf)
2. The ascending order of $s(T(v, \mathbf{p}))$ (cost of tree $T(v, \mathbf{p})$) - our solution

The Order to Compute $T(v, \mathbf{p})$

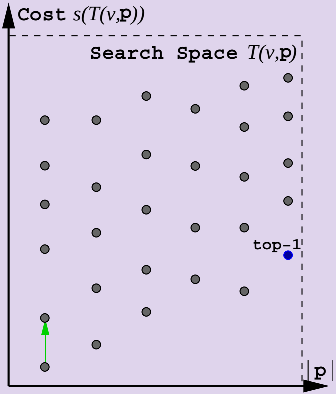
Ascending Order of $|\mathbf{p}|$



Ascending Order of $s(T(v, \mathbf{p}))$

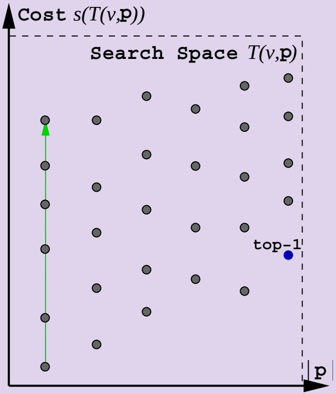
The Order to Compute $T(v, \mathbf{p})$

Ascending Order of $|\mathbf{p}|$



The Order to Compute $T(v, \mathbf{p})$

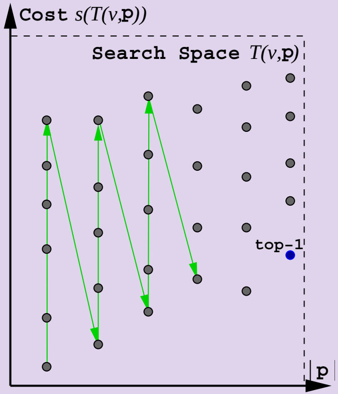
Ascending Order of $|\mathbf{p}|$



Ascending Order of $s(T(v, \mathbf{p}))$

The Order to Compute $T(v, \mathbf{p})$

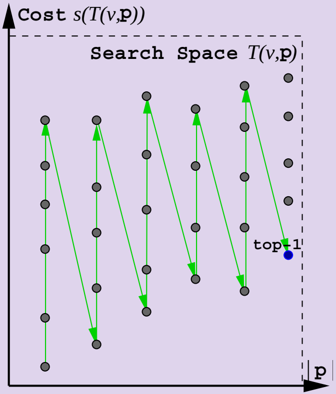
Ascending Order of $|\mathbf{p}|$



Ascending Order of $s(T(v, \mathbf{p}))$

The Order to Compute $T(v, \mathbf{p})$

Ascending Order of $|\mathbf{p}|$

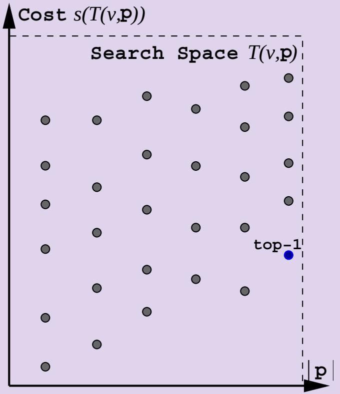


Ascending Order of $s(T(v, \mathbf{p}))$

The Order to Compute $T(v, \mathbf{p})$

Ascending Order of $|\mathbf{p}|$

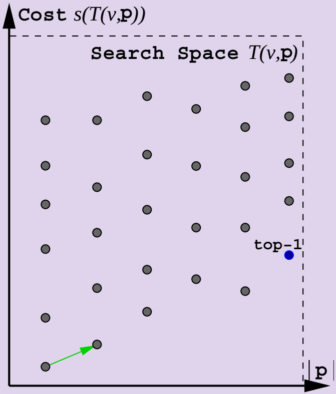
Ascending Order of $s(T(v, \mathbf{p}))$



The Order to Compute $T(v, \mathbf{p})$

Ascending Order of $|\mathbf{p}|$

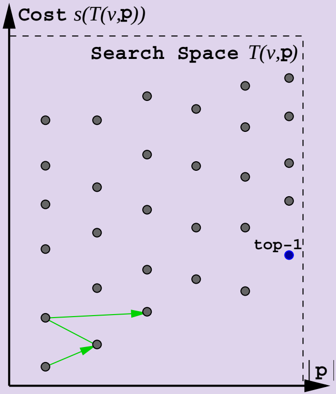
Ascending Order of $s(T(v, \mathbf{p}))$



The Order to Compute $T(v, \mathbf{p})$

Ascending Order of $|\mathbf{p}|$

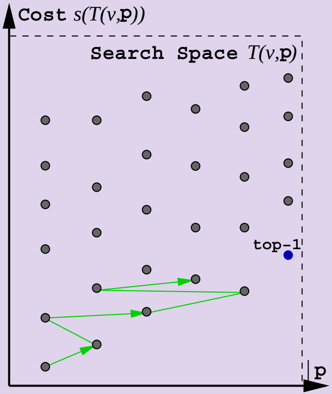
Ascending Order of $s(T(v, \mathbf{p}))$



The Order to Compute $T(v, \mathbf{p})$

Ascending Order of $|\mathbf{p}|$

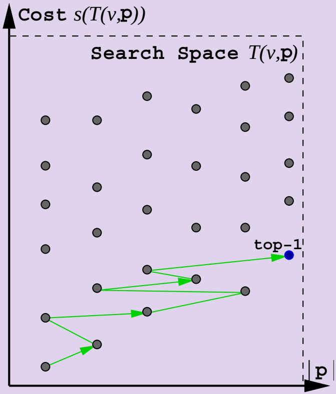
Ascending Order of $s(T(v, \mathbf{p}))$



The Order to Compute $T(v, \mathbf{p})$

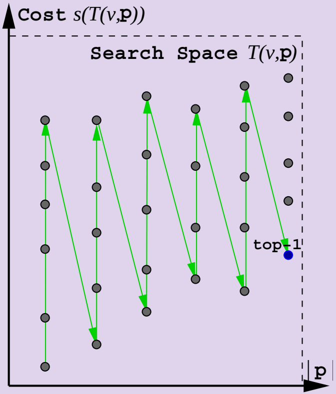
Ascending Order of $|\mathbf{p}|$

Ascending Order of $s(T(v, \mathbf{p}))$

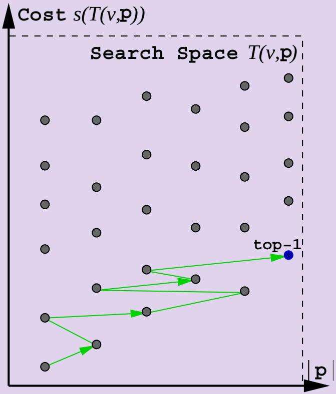


The Order to Compute $T(v, \mathbf{p})$

Ascending Order of $|\mathbf{p}|$



Ascending Order of $s(T(v, \mathbf{p}))$



Speedup of Dynamic Programming

Comparing the Two Orders

- In the same search space $\{T(v, \mathbf{p})\}$
- Ascending order of $|\mathbf{p}|$: visiting nearly the whole search space
- Ascending order of $s(T(v, \mathbf{p}))$: following a shortcut to the top-1

Our Second Dynamic Programming Algorithm

- Reduce the search space from $\{T(v, \mathbf{p}, h)\}$ to $\{T(v, \mathbf{p})\}$
- Follow the ascending order of $s(T(v, \mathbf{p}))$ to compute the dynamic programming equation of $T(v, \mathbf{p})$
- Visit only the necessary portion of the search space
- Achieve low time / space complexity

Speedup of Dynamic Programming

Comparing the Two Orders

- In the same search space $\{T(v, \mathbf{p})\}$
- Ascending order of $|\mathbf{p}|$: visiting nearly the whole search space
- Ascending order of $s(T(v, \mathbf{p}))$: following a shortcut to the top-1

Our Second Dynamic Programming Algorithm

- Reduce the search space from $\{T(v, \mathbf{p}, h)\}$ to $\{T(v, \mathbf{p})\}$
- Follow the ascending order of $s(T(v, \mathbf{p}))$ to compute the dynamic programming equation of $T(v, \mathbf{p})$
- Visit only the necessary portion of the search space
- Achieve low time / space complexity

Outline

Finding Top-k Answers

A Progressive Method

- Finding $T(v, \{p_1, \cdots, p_l\})$'s with top-$k$ minimum costs as the top-k answers
- Advantage I: time / space complexity unchanges
- Advantage II: no sorting is needed

Outline

1. Keyword Search in Relational Databases
   - Database Graph, Query, and Answer
   - The Hardness of This Problem
2. Our New Parameterized Solutions
   - Finding Top-1 Answer
   - Finding Top-k Answers
3. Existing Solutions
   - Other Graph-Based Solutions
4. Experimental Studies
   - Some Representative Experimental Results

Other Graph-Based Solutions

1-Star Tree

- Combining l shortest-paths from leaves (containing keywords) to the roots
- $O(l)$-approximation for linear cost functions
- BANKS I: Gaurav Bhalotia, et. al., ICDE 2002
- BANKS II: Varun Kacholia, et. al., VLDB 2005

Spanning and Cleanup

- Spanning a set of trees until some of them cover all the l keywords
- $O(l)$-approximation for linear cost functions
- RIU-E: Wen-Syan Li, et. al., WWW 2001, TKDE 2002

Outline

Experiment Setup

Implementation

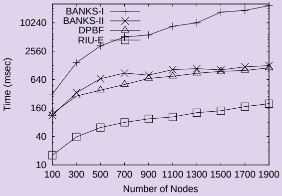
- Compare our solution (DPBF) with: RIU-E, BANKS-I, and BANKS-II
- Implement these algorithms in memory
- Use linear cost function (sum of edge weights)
- Environment: 3.4GHz CPU and 2G memory PC running XP

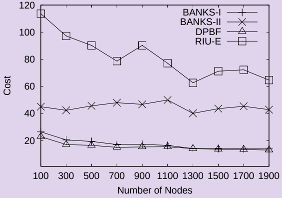
Datasets - 10 Subsets of DBLP

100K (up to 1982), 300K (up to 1987), 500K (up to 1993), 700K (up to 1996), 900K (up to 1997), 1100K (up to 1999), 1300K (up to 2000), 1500K (up to 2001), 1700K (up to 2002), and 1900K (up to 2004)

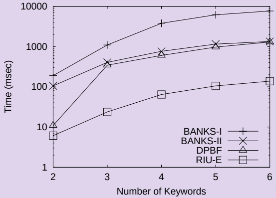
Some Results

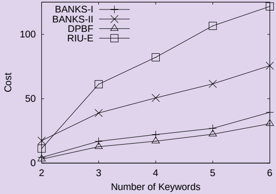
Varying the Size of Database n — 4-Keyword Queries, $k = 1$





Varying the Number of Keywords l — DBLP 500K, $k = 1$





Summary

- Model the keyword search problem as the (top-k) minimum group Steiner tree problem.
- Propose a parameterized algorithm for this problem.
  1. Bounded time / space complexity
  2. Efficient in practice
- Support undirected / directed model, node / edge weights, and cost function in the form of linear combination of weights.

Related Work: Other Graph-Based Solutions

- BANKS I: Gaurav Bhalotia, et. al.
  Keyword Searching and Browsing in Databases using BANKS.
  ICDE'02, pages 431-440, 2002.
- BANKS II: Varun Kacholia, et. al.
  Bidirectional Expansion For Keyword Search on Graph Databases.
  VLDB'05, pages 505-516, 2005.
- RIU-E: Wen-Syan Li, et. al.
  Query Relaxation by Structure and Semantics for Retrieval of Logical Web Documents.
  IEEE Trans. Knowl. Data Eng., 14(4):768-791, 2002.
- Benny Kimelfeld, et. al.
  Finding and Approximating Top-k Answers in Keyword Proximity Search.
  PODS'06, pages 173-182, 2006.

Related Work: Database-Based Solutions

Sanjay Agrawal, et. al.
DBXplorer: A System for Keyword-Based Search over Relational Databases.
ICDE'02, pages 5-16, 2002.

Vagelis Hristidis, et. al.
Efficient IR-Style Keyword Search over Relational Databases.
VLDB'03, pages 850-861, 2003.

Fang Liu, et. al.
Effective Keyword Search in Relational Databases.
SIGMOD'06, pages 563-574, 2006.

Yi Luo, et. al.
SPARK: Top-k Keyword Query in Relational Databases.
To Appear in SIGMOD'07, 2007.