Optimizing Index for Taxonomy Keyword Search

Speaker: Po-Hsien Shih
Advisor: Jia-Ling Koh
Source: SIGMOD’12
Date: 03 January, 2013
Outline

• Introduction
• Taxonomy keyword search
• Processing Taxonomy
• Index Optimization
• Experiment
• Conclusion
Introduction

- What’s the new query type? some research do...
  - Query rewriting
  - Finding query substitution
  - automatically discovering relationships among term by mining webpages and search engine click log.

if a query : q = “IT company, Seattle”
=> “Amazon .... Seattle” , “Microsoft....Seattle”
Introduction

• Goal...

• How to efficiently process a keyword query by answering all of its possible substitutes.

• How to optimize the index structure for this purpose.

P.S. Assume we have complete information of term substitutions.
Outline

• Introduction
• Taxonomy keyword search
• Processing Taxonomy
• Index Optimization
• Experiment
• Conclusion
Taxonomy Keyword Search

- \((\mathcal{T}, \sqsubseteq)\) - a taxonomy
  a universe of terms - \(\mathcal{T}\)
  a term-term concept-instance relationship \(\sqsubseteq\)

- Concept-instance relationship
  ex. “JPEG” \(\sqsubseteq\) “file format”

- Substitution relationship
  ex. “puppy” \(\sqsubseteq\) “dog” \(\sqsubseteq\) “pet”; “puppy” \(\preceq\) “pet”

- Can be a resulting directed graph
Inverted lists - $I(t) \subseteq D$,

$D$: a collection of documents consisting of terms from a vocabulary - $\mathcal{T}$.

Ex. $t = \text{"windows7"}$, then $I(t)$ is the inverted list of the set of document containing the term $t$.

$S(t) = \{t' \in \mathcal{T} | t' \preceq t\}$

"Windows7" $\sqsubseteq \text{"MS Windows"}$ $\sqsubseteq \text{"operating system"}$

"Windows7" $\preceq \text{"operating system"}$

$R(t) = \bigcup_{s \in S(t)} I(s)$

$R(t) =$ is the inverted list of the set of document containing all the substitutions of term $t$.

$R(\text{"windows7"}) = I(\text{"Windows7"}) + I(\text{MS Windows}) + I(\text{operating system})$
When $q = \{t_1, t_2, \ldots, t_k\}$:

$$\mathcal{R}(q) = \bigcap_{i=1}^{k} \mathcal{R}(t_i) = \bigcap_{i=1}^{k} \left( \bigcup_{s \in S(t_i)} \mathcal{I}(s) \right)$$

$$= \bigcup_{\{s_1, \ldots, s_m\} \in S(t_1) \times \ldots \times S(t_k)} \left( \bigcap_{i=1}^{k} \mathcal{I}(s_i) \right)$$

$$= R(t_1) + R(t_2) + \ldots + R(t_k)$$

$$= R(“windows”) \cap R(“Lumia”) \cap \ldots \cap R(“Nokia”)$$

$$= \ldots$$

What do we face?
Challenges

- Naive Method I
  Retrieve $I(s)$ for each instance $s$ of each term $t_i$ and storing it.
  But...
  
  If $t_i$ has a large number of instances, evaluating becomes more inefficient.

- Naive Method II
  We precompute and store $R(t) = \bigcup_{s \in S(t)} I(s)$ for all terms $t$'s in the offline stage. This approach doesn't need to compute unions of $I(s)$'s during query time. It still has the drawback that it requires excessive amount of storage of $R(t)$.

We want to solve it!
Outline

• Introduction
• Taxonomy keyword search
• **Processing Taxonomy**
• Index Optimization
• Experiment
• Conclusion
Assume $I(t)$ is precomputed and stored in the system. Because of it’s too space-consuming to also store result list $R(t)$ and $R(t)$’s size is always larger than $I(t)$’s size.

We select a subset $P \subseteq T$, and precompute and store

$$R(t) = \bigcup_{s \in S(t)} I(s) \text{ for each } t \in P$$

Once $P$ is fixed, we can compute $R(t)$’s for $t \subseteq P$ by merge lists together.
Online

When a query: \( q = \{t_1, \ldots, t_k\} \), we want to take advantage of the precomputed list \( R(t) \)'s as much as possible.

Assume \( P = \{t_3, t_6, t_{10}, t_{12}, t_{17}\} \), query \( q = \{t_2, t_{10}\} \):

\[
R(t_2) = I(t_2) + I(t_3) + \ldots + I(t_9),
\]

because we can get \( R(t_3), R(t_6), R(t_{10}) \). \( t_3, t_6, t_{10} \) are in \( P \). We can get: \( (R(t_3) \cup R(t_6) \cup I(t_2) \cup I(t_9)) \cap (R(t_{10})) \)

---

**PROCESSQUERY**\((q, P)\)

1: For each \( t_i \in q \) do:
2: \[ L(t_i) \leftarrow \text{PROCESSTERM}(t_i, P) \].

In linear scan model:
3: Compute \( R(t_i) \) from the set of lists \( L(t_i) \) \((i = 1, \ldots, k)\).
4: Return \( \mathcal{R}(q) = \cap_{i=1}^{k} \mathcal{R}(t_i) \).

Or, in hash lookup model: (initially, \( \mathcal{R}(q) = \emptyset \))
5: Compute \( \mathcal{R}(t_1) \) from the set of lists \( L(t_1) \).
6: For each \( d \in \mathcal{R}(t_1) \) do
7: \[ \text{For each } L(t_i) \ (i = 2, \ldots, k) \text{ check:} \]
8: \[ \text{whether } d \in L \text{ for some list } L \in L(t_i); \]
8: \[ \text{If “yes” for all } i = 2, \ldots, k, \mathcal{R}(q) \leftarrow \mathcal{R}(q) \cup \{d\}. \]
9: Return \( \mathcal{R}(q) \).

**PROCESSTERM**\((t, P)\)

(It returns pointers to the lists \( \mathcal{I}(t') \)'s and \( \mathcal{R}(t') \)'s as \( L \))
1: If \( t \in P \) then return \( L(t) \leftarrow \{\mathcal{R}(t)\} \);
2: Else: \[ L = \{\mathcal{I}(t)\}; \]
3: For each child \( t' \) of \( t \): \[ L \leftarrow L \cup \text{PROCESSTERM}(t'); \]
4: Return \( L(t) \leftarrow L \).

---

**Figure 1:** A taxonomy and \( P = \{t_3, t_6, t_{10}, t_{12}, t_{17}\} \)

We can get \( L(t_{10}) = \{R(t_{10})\} \),
\( L(t_2) = \{R(t_3), R(t_6), I(t_2), I(t_9)\} \)
Proposition
With the selection of $C(t, P)$, $C(t, P)$, $R(t)$ can be computed with the minimum number of union operations.

$$C(t, P) = \{t' \in P \mid t' \leq t \land \#x \in P : t' < x \leq t\}$$

Any term $t'$ in $C(t, P)$ is the closest descendant of $t$ on the path from $t$ to $t'$ in $P$. ex: $C(t_{10}, P) = \{t_{10}\}$, $C(t_{2}, P) = \{t_{3}, t_{6}\}$

$$\overline{C}(t, P) = S(t) - \bigcup_{x \in C(t, P)} S(x)$$

Those terms that are substitutes of $t$ (in $S(t)$) but not covered by $C(t, P)$. When ProcessTerm accesses those terms in $C(t, P)$, it adds $I(t)$ to $L$ and continue the traversal. ex: $C(t_{2}, P) = \{t_{2}, t_{9}\}$ and $C(t_{10}, P) = \emptyset$

$$L(t) = \{R(t') \mid t' \in C(t, P)\} \cup \{I(t'') \mid t'' \in \overline{C}(t, P)\},$$

$$R(t) = \left(\bigcup_{t' \in C(t, P)} R(t')\right) \bigcup \left(\bigcup_{t'' \in \overline{C}(t, P)} I(t'')\right).$$
Two way to implement the list

**Linear model**

All Elements in I(t) and R(t) are stored as sorted linear list.

Two Step to evaluate R(q):

(i) Compute \( R(t_i) \) for each \( t_i \in q \)
(ii) Take the intersection of the sorted linear list \( R(t_i) \).

\[
\text{cost}_s(t, P) = \sum_{t' \in C(t, P)} |R(t')| + \sum_{t'' \in \overline{C}(t, P)} |\overline{I}(t'')|.
\]

\[
\text{cost}_s(q, P) = \sum_{t_i \in q} \text{cost}_s(t_i, P)
\]

**PROCESSQUERY**\( (q, P) \)
1: For each \( t_i \in q \) do:
2: \( \mathcal{L}(t_i) \leftarrow \text{PROCESSTERM}(t_i, P). \)

In linear scan model:
3: Compute \( \mathcal{R}(t_i) \) from the set of lists \( \mathcal{L}(t_i) \) \( (i = 1, \ldots, k) \)
4: Return \( \mathcal{R}(q) = \cap_{i=1}^k \mathcal{R}(t_i). \)

Or, in hash lookup model: (initially, \( \mathcal{R}(q) = \emptyset \))
5: Compute \( \mathcal{R}(t_1) \) from the set of lists \( \mathcal{L}(t_1). \)
6: For each \( d \in \mathcal{R}(t_1) \) do
7: For each \( \mathcal{L}(t_i) \) \( (i = 2, \ldots, k) \) check:
8: If “yes” for all \( i = 2, \ldots, k \), \( \mathcal{R}(q) \leftarrow \mathcal{R}(q) \cup \{d\}. \)
9: Return \( \mathcal{R}(q). \)

**PROCESSTERM**\( (t, P) \)
( It returns pointers to the lists \( I(t') \)’s and \( R(t') \)’s as \( \mathcal{L} \))
1: If \( t \in P \) then return \( \mathcal{L}(t) \leftarrow \{R(t)\}; \)
2: Else: \( \mathcal{L} = \{I(t)\}; \)
3: For each child \( t' \) of \( t \): \( \mathcal{L} \leftarrow \mathcal{L} \cup \text{PROCESSTERM}(t'); \)
4: Return \( \mathcal{L}(t) \leftarrow \mathcal{L}. \)
Hash look model

Give a query \( q = \{t_1, t_2, \ldots, t_k\} \)
Find \( t_0 = \arg \min_{t_i \in q} |R(t_i)| \). (in \( q \))

Suppose \( t_0 = t_1 \), compute \( R(t_1) \), then adding each document in each \( R(t') \) or \( I(t'') \) in \( L(t) \)
Then, for each document \( d \) in \( R(t_1) \), we use hash lookups to check whether it also appears in every other \( R(t_i) \)'s—if yes, we include it into the query answer \( R(q) \).

\[
\mathcal{N}(t, P) = |C(t, P)| + |\overline{C}(t, P)|.
\]

\[
\text{cost}_h(q, P) = |R(t_1)| \cdot \sum_{t_i \in q} \mathcal{N}(t_i, P)
\]

well, the most important thing is how to select \( P \)?
Outline

- Introduction
- Taxonomy keyword search
- Processing Taxonomy
- Index Optimization
- Experiment
- Conclusion
In the IndexSelection problem, suppose we have a collection of documents $D$ for which a taxonomy $(T, \sqsubseteq)$ is predefined. Given a query $Q$ and a space budget $B_0$, our goal is to find a subset of terms $P \subseteq T$, and the objective is to

\[
\begin{align*}
\text{minimize} & \quad \text{cost}_{\text{exp}}(Q, P) \\
\text{s.t.} & \quad \text{space}(P) \leq B_0, \quad P \subseteq T.
\end{align*}
\]

\[
\text{cost}_{\text{exp}}(Q, P) = \sum_{q \in Q} \text{cost}(q, P) \cdot w(q), \quad \text{space}(P) = \sum_{t \in P} |\mathcal{R}(t)|.
\]

$w(q)$ is the frequency of query $q$. 

Goal: We want to select $P$ to minimize the query processing cost for a workload of queries $Q$, subject to space budget.
A naive algorithm

To include the top-$k$ **most frequent** terms into $P$, where $k$ is selected to be the max one so that the space budget $B_0$ is not violated.

**Two drawback**

i) Frequency is not the only criteria to quantify the benefit of including a term into $P$.

ii) The benefit of materializing $R(t)$ is not additive but dependent on the terms already in $P$.

---

**Benefit function - Rewrite cost function**

**Linear**

$$\text{cost}_{\text{exp}}(Q, P) = \sum_{q \in Q} \sum_{t \in q} \text{cost}_s(t, P) \cdot w(q)$$

$$= \sum_{t \in T} \left( \text{cost}_s(t, P) \cdot \sum_{q \in Q : t \in q} w(q) \right)$$

$$= \sum_{t \in T} \text{cost}_s(t, P) \cdot w_s(t, Q),$$

$$w_s(t, Q) = \sum_{q \in Q : t \in q} w(q)$$

**Hash lookup ($C(q) = |R(t_1)|$)**

$$\text{cost}_{\text{exp}}(Q, P) = \sum_{q \in Q} \sum_{t \in q} C(q) \cdot N(t, P) \cdot w(q)$$

$$= \sum_{t \in T} \left( N(t, P) \cdot \sum_{q \in Q : t \in q} C(q) \cdot w(q) \right)$$

$$= \sum_{t \in T} N(t, P) \cdot w_h(t, Q),$$

$$w_h(t, Q) = \sum_{q \in Q : t \in q} C(q) \cdot w(q)$$
To get the optimal solution for IndexSelection, a approach is to enumerate all subsets of \( T \) as \( P \), where \( N = |T| \) is the total number of terms in the taxonomy. We use Dynamic programming method and greedy method to do it.
Outline

• Introduction
• Taxonomy keyword search
• Processing Taxonomy
• Index Optimization
• Experiment
• Conclusion
Experiment

- **DataSet**
  - i) Using a taxonomy automatically extracted from a corpus of 1.68 billion web pages. This taxonomy contains 279,109 terms, with term-term concept-instance relationship predefined.
  
  ii) The query log is extracted from queries to search engine Bing.com from Sept 2007 to Feb 2010, and we only keep the ones with frequency larger than 300. We have 1,260,526 different queries which appear totally more than 6 billion times.

  iii) With a sample from Bing.com’s web page corpus as the collection of documents, we build the inverted index, i.e., $I(\cdot)$, on these pages for all terms in our taxonomy. The inverted lists $I(t)$’s contain a total of 10,282,150 entries.

- **Measures**
  - We record the time and the number of (linear scan or hash lookup) operations needed to process workloads of queries.
Exp-I: Varying space budget B0.

Figure 2: Linear scan model: varying space budget
Exp-II: Handling future queries

To show that the performance of our index optimization techniques are stable when handling future queries.

Figure 4: Linear scan model: future queries

Figure 5: Hash lookup model: future queries
Exp-III: Varying size of inverted index.

Figure 6: Linear scan model: varying size of dataset
Exp-IV: Comparing linear scan processing model and hash lookup model.

![Graphs showing comparison between linear scan model and hash lookup model.](image)

(a) Linear scan model  
(b) Hash lookup model

**Figure 7:** Average processing time (ms) per query
Exp-V: Varying amount of queries for optimization

**Figure 8:** Linear scan model: varying the amount of queries used for index optimization
Exp-VI: Cost of index optimization and construction

(a) Optimization time (ms)  (b) Construction time (ms)

Figure 9: Index optimization and construction
Outline

- Introduction
- Taxonomy keyword search
- Processing Taxonomy
- Index Optimization
- Experiment
- Conclusion
Conclusion

• We introduce a new query type: taxonomy query for the purpose of flexible query substitution.

• It proposes processing models for taxonomy queries, and introduce how to build an additional index to support efficient query processing.

• We study the problem of how to optimize this additional index based on a workload of queries, with the goal of minimizing query processing cost, and propose algorithms with performance guarantees.

• Our index optimization techniques are tested using real datasets and are shown to be effective and robust.