The SphereSearch Engine for Unified Ranked Retrieval of Heterogeneous XML and Web Documents

Jens Graupmann, Ralf Schenkel, Gerhard Weikum.
Max-Planck Institut
2005

Presented by:
Eddie Galfrin
Motivation

Or, where do search engines fail?
Concept-Awareness fail
Which company develops operating systems and performs bioinformatics research?
Abstraction-Awareness fail

In a books database (contain digital copy of books), how can we find a book who:

• Main character pulled a sword from a rock
• Became a king
How can we do better?
SphereSearch from the bird's eye
The goal is to increase recall and precision for hard queries on linked and heterogeneous data.

- Unified search for unstructured, semi-structured, structured data from heterogeneous sources.
- Graph based model.
- Annotation engines – recognizes classes of named entities (names, locations, dates, money, etc.).
The goal II

- Compactness based score – “How compact the wanted data is”

**support:**
- Concept awareness
- Context awareness
- Abstraction awareness
SphereSearch engine overview

Pre-processing stage:
- Convert the document to XML file
- Annotate the files using information extraction tools
- Creates a graph out of the XML files

Query calculation stages:
- Calculates the local score of each node
- Calculate the sphere score of each node
- Return the top-k results
Data transformation and annotation
Data transformation

Transforms any type of (data) document (HTML page, doc, pdf, plain text document) to XML file with meaningful tags using several heuristic rules.
Smart XML transformation I

HTML document

<table>
<thead>
<tr>
<th>&lt;H1&gt;Experiment&lt;/H1&gt;</th>
<th>&lt;Experiment&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>In this experiment we will....</td>
<td>in this experiment....</td>
</tr>
<tr>
<td>&lt;H2&gt;Settings&lt;/H2&gt;</td>
<td>&lt;Settings&gt;</td>
</tr>
<tr>
<td>We’ve prepared.....</td>
<td>We’ve prepared ...</td>
</tr>
<tr>
<td>&lt;H1&gt;...</td>
<td>&lt;/Settings&gt;</td>
</tr>
</tbody>
</table>

Transformed XML document

```xml
<Experiment>
  in this experiment....
  <Settings>
    We’ve prepared ...
  </Settings>
</Experiment>
```
Smart XML transformation II

**HTML document**

```html
<b>Title:</b>
War and peace </br>
<b>Author:</b></br>
Lev Tolstoy
```

**XML document**

```xml
<Title>
War and peace
</Title>
<Author>
Lev Tolstoy
</Author>
```
Smart XML transformation III

And many other heuristics such as:

• Smart table conversion
• Formatting tags removal (like </br>)
• Etc.
Data annotation I

• Use information extraction tools to automatically give meaningful tags to certain objects.

• Use out-of-the-box GATE’s ANNIE component in this implementation. Easy to change \ add tools.
Original data
The Pelican Hotel in Salvador, operated by Roberto Cardoso, offers comfortable rooms starting at $100 a night, including breakfast. Please check in before 7pm.

Annotated data
The <company> Pelican Hotel </company> in <location> Salvador </location>, operated by <person> Roberto Cardoso </person>, offers comfortable rooms starting at <price> $100 </price> a night, including breakfast. Please check in before <time> 7pm </time>. 
Data annotation III

**HTML data**

```html
<H1>Professor: Yaron Kanza, Technion</H1>
<b>research:</b>DBs
<b>course:</b>seminar in DBs
<H1>...
```

**Annotated data**

```html
<Professor>
  Yaron Kanza
  <location>
    Technion
  </location>
  <research>DBs</research>
  <course>seminar in DBs</course>
</Professor>
```
Graphs

Annotated data

<Professor>
  Yaron Kanza
</Professor>

<location>
  Technion
</location>

<research>
  DBs
</research>

<course>
  seminar in DBs
</course>

Graph

- docid=1
  tag="Professor"
  content="Yaron Kanza"

- docid=1
  tag="location"
  content="Technion"

- docid=1
  tag="course"
  content="seminar in DBs"

- docid=1
  tag="research"
  content="DBs"
Query language
Query language I

• Similarity:
The “like” operator
~Professor, ~house....

• Concept-based conditions:
  person=Max Plank
  year=1900
  ~location=Haifa
Query language II

• Grouping - key words together:
  C(concert, Haifa) R(~restaurant, Haifa)
  professor R(research, DB) T(teach, WWW)

• Join conditions:
  A(Madonna, husband) B(director)
  A.person=B.director
SphereSearch in details

Let’s dive in...
Definitions

Let’s go formal...
The graph

Let $X=(D,L)$ a collection of XML documents – $D$, with a set $L$ of links (of type href, Xpointer and Xlink).
The element-level graph $G(X) = (V(X), E(X))$
While $V(X)$ is the group of all the elements in $D$, and $E(X)$ is the groups of undirected edges that correspond to parent-child relationship and links (the group $L$).
Queries

A SphereSherch query defined as: \( S = (Q, J) \)
While \( Q = \{G_1, \ldots, G_g\} \) of \( g > 0 \) groups and
\( J = \{J_1, \ldots, J_m\}, m \geq 0 \) join conditions.
A group \( Q_i \) consists of a set of keyword
conditions \( t_1^i, \ldots, t_{k_i}^i \) and \( c_1^i = v_1^i, \ldots, c_{l_i}^i = v_{l_i}^i \)
concept-value conditions.
When either keyword, concept or values are
exact match or include the similarity operator
(\(\sim\)).
Results

The result is a g-tuples \((e_1, ... e_g)\) of elements in \(E(X)\) where each \(e_i\) is a result for query group \(G_i\) sorted by a score that measures the expected quality of the result.
Scores
Node score

• For exact-match condition \( t \), the node score \( ns(n, t) \) of node \( n \) is computed using the Okapi BM25 scoring model (adapted to XML).

• For a similarity keyword \( \sim K \), first compute the set \( \exp(K) \) (ontologically similar words to \( K \)).

The node score:

\[
ns(n, \sim K) = \max_{x \in \exp(K)} \left[ sim(K, x) \times ns(n, x) \right]
\]

• The \( \sim \) operator gives the system abstraction-awareness.
Spheres

We want to promote the score of elements where the requested keyword appears frequently in the context of the element, i.e., in the content of other elements in its neighborhood.

This will give the system context-awareness.
Formally, the sphere $S_d(n)$ of node $n$ at distance $d$ is the set of all nodes whose distance from $n$ is $d$.

The sphere-score $S_d(n, t)$ at distance $d$ of a node $n$ and keyword $t$ is:

$$S_d(n, t) = \sum_{v \in S_d(n)} ns(v, t)$$

And the sphere-score of node $n$ with keyword $t$ is:

$$S(n, t) = \sum_{i=1}^{D} s_i(n, t) \times \alpha^i$$

With sphere size limit $D$, and damping coefficient $\alpha \in [0, 1]$. 
In the following example we have 3 linked documents and their elements (as held in the element graph). The ‘t’s near an element means that the word t appears in the element #t times.
For a keyword score using a local scoring algorithm, node 2 would get the highest score. In sphereSearch node 2 will only get the best node score.

(Let’s suppose that the local score=$\#(t)$ in a node)
Let’s compute the sphere-score of nodes 1 and 2 at the configuration of $D = 3, \alpha = 0.5$

The sphere score for node 2 is:

$$s(2, t) = 3 + 0 \times 0.5 + 0 \times 0.5^2 + 1 \times 0.5^3$$

$$= 3.125$$
And the sphere score of node 1 is:
\[ s(1, t) = 1 + 4 \times 0.5 + 2 \times 0.5^2 + 5 \times 0.5^3 \]
\[ = 4.175 \]

So node 1 is a better result from node 2.
In case of concept-value condition, we apply a similar scoring method. For a condition value \( c=v \), the only change is in the node score computation:

\[
ns(n, c = v) = \begin{cases} 
0 & \text{name}(n) \neq c \\
ns(n, v) & \text{otherwise}
\end{cases}
\]

And for \( \sim c=v \) the node score will be

\[
ns(n, \sim c = v) = \text{sim}(\text{name}(n), c) \cdot ns(n, v)
\]

This will give the system concept-awareness.
The sphere score of a node $n$ with respect to a query group $G$ is then the sum of the node’s sphere scores for each condition of the query group:

$$s(n, G) = \sum_{j=1}^{k} s(n, t_j) + \sum_{j=1}^{l} s(n, c_j = v_j)$$
Compactness

Because usually we are searching for related entities in a single query, only the sphere score by itself isn’t enough to assess the relevance of an answer.

Different parts of the element-level graph may be completely unrelated!
Intuitively, we should prefer answers that have nodes close to each other instead of spread all over the graph.

i.e. the answer nodes are in the same document or in a few clicks distance, and therefore often related to the same topic.

This will give the system context-awareness.
Computing compactness:
In order to compute the compactness of a result N, we create a connection graph G(N)=(V(N),E(N)) such that:
V(N) – elements of N
E(N) – undirected, weighted edge \{x,y\} exists iff \(\delta(x, y)\) (the distance between x and y in the element level graph) is finite.
The weight of the edge will be:
\[
\frac{1}{\delta(x, y) + 1}
\]
The calculation of the compactness \((C(N))\) is made by:

1. Calculate the maximum spanning tree of \(G(N)\).

2. Sum the weights from all the edges in the maximum spanning tree

If there is no spanning tree (the graph is not connected), then \(C(N) = -\infty\)
The score of a potential answer $N$ to a query $S$ is then defined as the weighted sum of the aggregated sphere scores and the compactness of the node set:

$$s(N, S) = \beta \cdot C(N) + (1 - \beta) \cdot \sum_{i=1}^{g} s(n_i, G_i)$$

When $C(N)$ is the compactness of $N$,
And $\beta \in [0,1]$
N sorted by $s(N, S)$ is the result of the query.
Given 3 query groups, and their results:

\[ G_1 = \{A, B\}; \quad G_2 = \{X\}; \quad G_3 = \{1,2\} \]
For this set of results, we can create 4 answers:

\[
N_1 = (A, X, 1) \\
N_2 = (A, X, 2) \\
N_3 = (B, X, 1) \\
N_4 = (B, X, 2)
\]

\[
G_1 = \{A, B\}; \\
G_2 = \{X\}; \\
G_3 = \{1, 2\}
\]
We can see that $N_1$ is the most compact result

$C(N_1) = \frac{5}{6} = 0.83$

$C(N_2) = \frac{3}{4} = 0.75$

$C(N_3) = \frac{7}{12} = 0.58$

$C(N_4) = \frac{2}{3} = 0.67$
Queries with join

Query groups can be connected by joins. For example, when visiting in Europe we would like to visit gothic churches and roman. Where we shall go?

A(gothic, church)  B(roman, church)
A.location=B.location
Queries with join

In case of an exact-match join condition $A.x=B.y$, for every two nodes, $n$ and $m$, that satisfy this condition, we simply add a temporary edge $(n,m)$ to the element-level graph. For example if B’s parent is joined with x, the result of $N_4$ will be now 0.83 – more compact than $N_1$.
Expriments
setup

• 3 databases:
  – Wikipedia
  – Wikipedia++ - Wikipedia with links to IMDB
  – DBLP++ - DBLP + links to publisher's home pages

• 50 queries from variety of people. Like:
  – A(actor birthday 1970<date<1980) western
  – G(California, governor) M(movie)

• Evaluation – precision@10
All queries were written in several language levels of SphereSearch:

- **SSE-KW** – simple keyword search
- **SSE-basic** – Keyword conditions using sphere-based scoring - lacks concept-awareness and does not use multiple query groups.
- **SSE-CV** – basic + support for concept value conditions.
- **SSE-QG** – CV + query groups.
- **SSE-join** – the full version with all the features.
Experimental Results on Wikipedia

precision@10 for Wikipedia

- GoogleWiki
- Google~wiki
- GoogleWeb
- google~Web
- SSE-KW
- SSE-basic
- SSE-CV
- SSE-QG
- SSE-join
More results

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision@10 for Wikipedia++</th>
<th>Precision@10 for DBLP++</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSE-KW</td>
<td>0.6</td>
<td>0.7</td>
</tr>
<tr>
<td>SSE-basic</td>
<td>0.5</td>
<td>0.6</td>
</tr>
<tr>
<td>SSE-CV</td>
<td>0.4</td>
<td>0.5</td>
</tr>
<tr>
<td>SSE-QG</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td>SSE-join</td>
<td>0.2</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Conclusions

• Simple and powerful search engine
• Including enhanced search features:
  – Concept awareness
  – Content awareness
  – Abstraction awareness
Thank You!