University of Insubria

Varese, Italy

A NEW QUERY SUGGESTION ALGORITHM FOR TAXONOMY-BASED SEARCH ENGINES

Roberto Zanon, Simone Albertini,
Moreno Carullo, Ignazio Gallo

albertini.simone@gmail.com

http://artelab.dicom.uninsubria.it/

October 5th, 2012

Query Suggestion

- Queries are unstructured data
- Must be exploited to support the user in its search mission
- With query suggestion we mean the task of proposing a set of different possible alternative search texts to a user who submitted a query
- Accomplish the search mission

Query Suggestion

- Two main approaches

Document-based

Exploits the documents, that is the results produced by the query that the user selects

Session-based

Exploits the consecutiveness of the queries in the same user session

Objective

- Realize a query suggestion algorithm in order to support the search system of a commercial website: www.shoppydoo.it

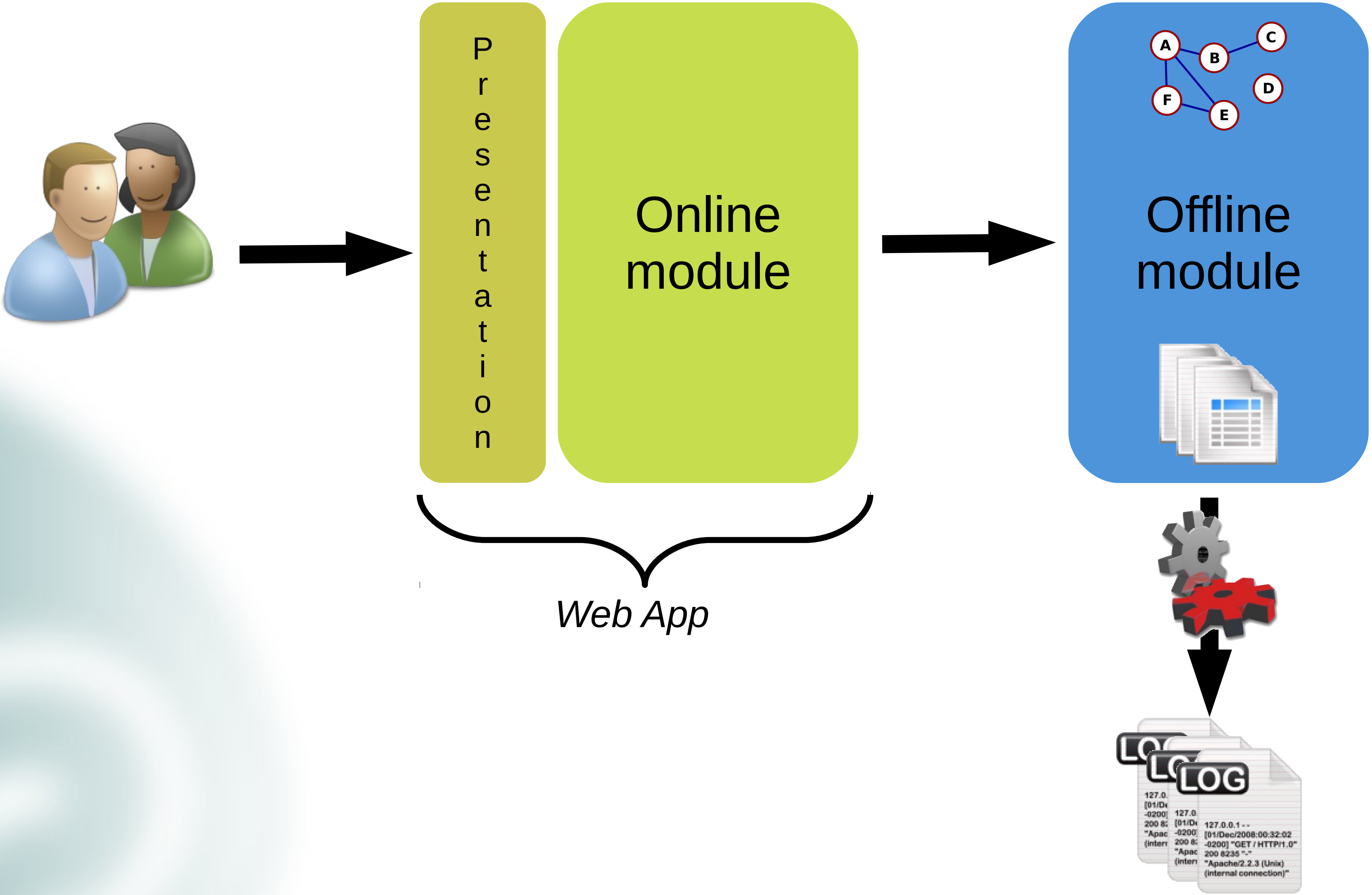
ShoppyDoo

  - Session-based, as it is the best approach [1]
  - Incremental
  - Non context-aware

[1] M.P. Kato, T. Sakai, K. T. (2011). *Query session data vs. clickthrough data as query suggestion resources.* In ECIR 2011 Workshop on Information Retrieval Over Query Sessions

[2] Broccolo, D., Frieder, O., Nardini, F. M., Perego, R., and Silvestri, F. (2010). *Incremental Algorithms for Effective and Efficient Query Recommendation.*

Architecture



Presentation

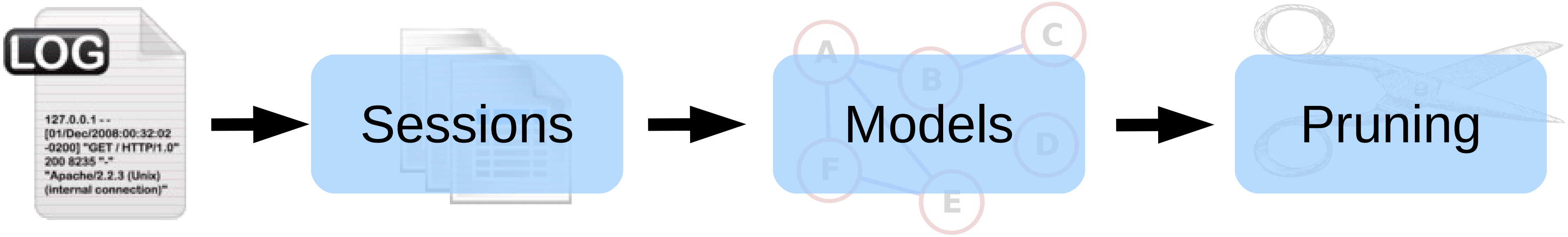
Online module

Offline module

Web App

```
127.0.0.1 - -
[01/Dec/2008:00:32:02
-0200] "GET / HTTP/1.0"
200 8235 "-"
"Apache/2.2.3 (Unix)
(internal connection)"
```

Offline process



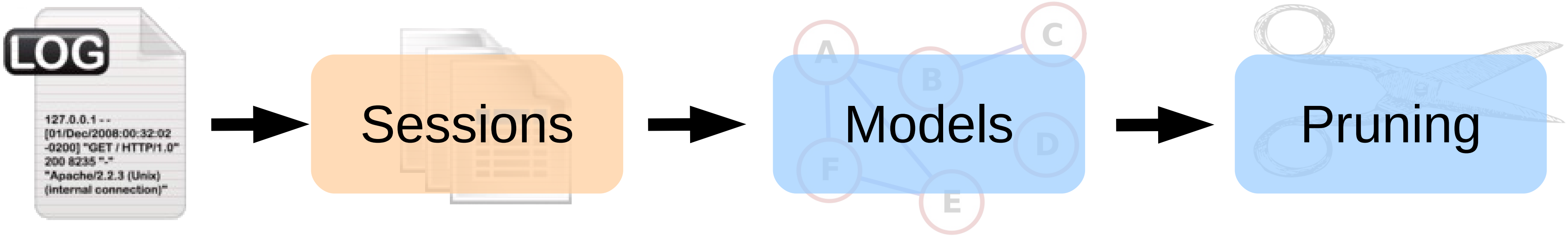
```
127.0.0.1 - -
[01/Dec/2008:00:32:02
-0200] "GET / HTTP/1.0"
200 8235 "-"
"Apache/2.2.3 (Unix)
(internal connection)"
```

Sessions

Models

Pruning

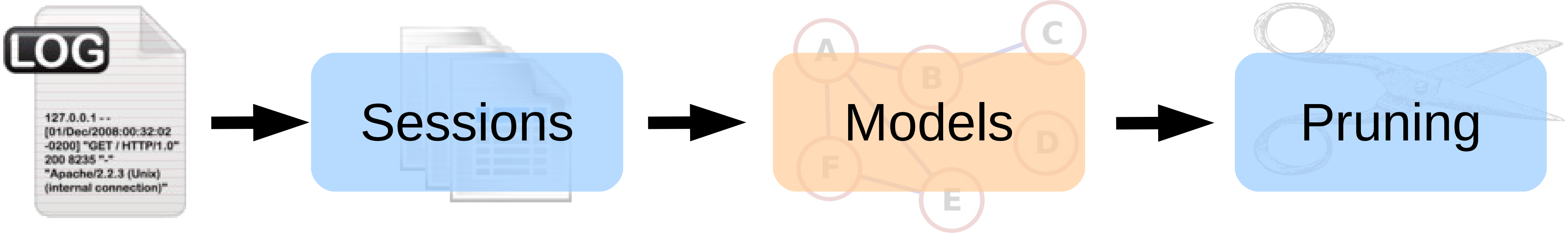
Offline process



Sessions → Models → Pruning

A logical session is a sequence of queries with the same user id and where for each pair of queries they were submitted not far more than 30 minutes.

Offline process

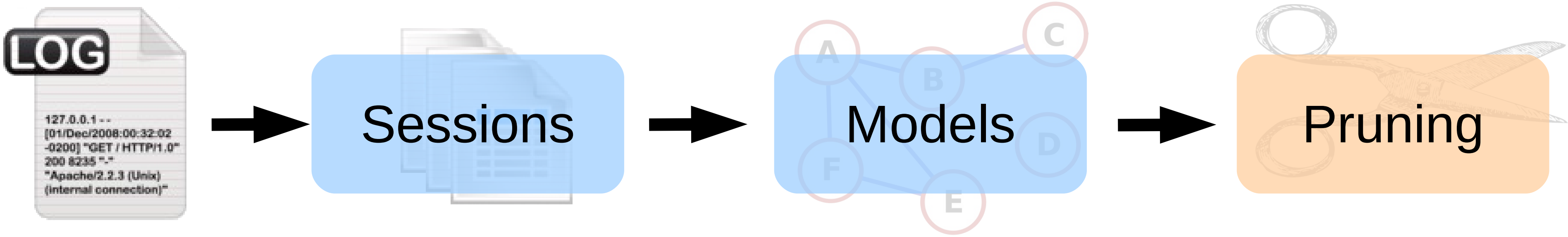


```
127.0.0.1 - -
[01/Dec/2008:00:32:02
-0200] "GET / HTTP/1.0"
200 8235 "-"
"Apache/2.2.3 (Unix)
(internal connection)"
```

Sessions → Models → Pruning

- *Representation of the sessions in the Query graph*
- *Representation of the similarity in the Similarity graph*
- *Inverted indexes for fast access*

Offline process



```
127.0.0.1 - -
[01/Dec/2008:00:32:02
-0200] "GET / HTTP/1.0"
200 8235 "-"
"Apache/2.2.3 (Unix)
(internal connection)"
```

Sessions → Models → Pruning

Reduction of the data structures dimensions:
Cut off weak (low weight) edges in both the graphs

Query graph

- Similar to the query flow graph [3]

- Node is <*category, query text*>
  - Links depends from the consecutiveness in the query sessions
  - Edge weights depends from the frequency of the consecutiveness

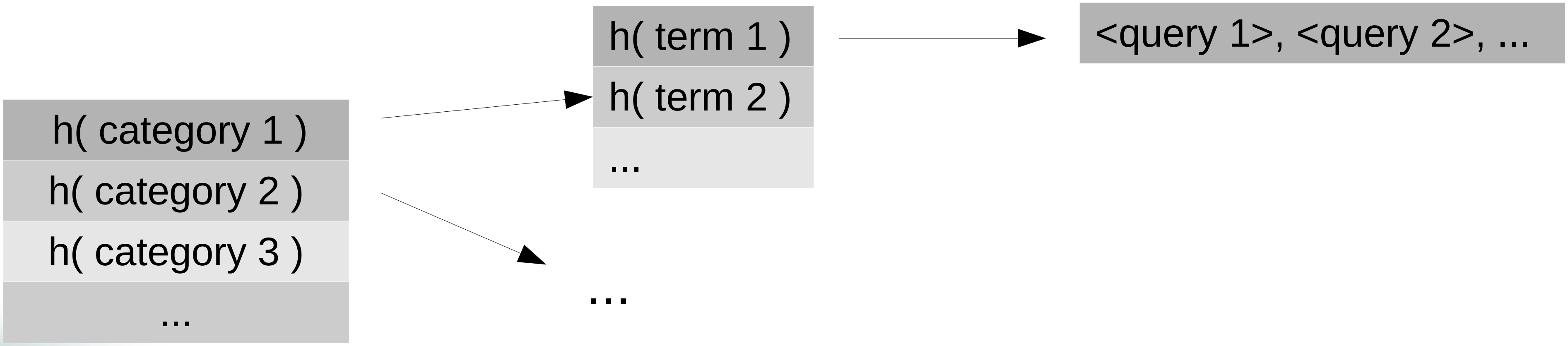
[3] Boldi, P., Bonchi, F., Castillo, C., Donato, D., Gionis, A., and Vigna, S. (2008). *The query-flow graph: model and applications.* In International Conference on Information and Knowledge Management.

Similarity graph

- Similar to the Word graph [4]
- Two queries are similar if they are related to the same category and have a similar search text
- Built on the same set of nodes as the query graph
- Weights of the edges are the similarity score
- Similarity score is calculated with the Jaccard index on the set of terms of the two queries

[4] Baeza-Yates, R. (2007). *Graphs from search engine queries.* In SOFSEM 2007: Theory and Practice of Computer Science, volume 4362 of Lecture Notes in Computer Science, pages 1–8

Indexes



- Provides fast access to the nodes of the graph
- Built during the construction of the query graph
- Used during the similarity graph construction

Online Process

1. Look for the similar queries by category in the similarity graph
   - If the query is not present, then calculate them
2. For each similar query, add to the set of similar queries all the queries that follow them in the query graph
3. Sort by the ranking function
   - Sums the normalized weights of the edges of the graphs

Online Process

1. Look for the similar queries by category in the similarity graph
   - If the query is not present, then calculate them
2. For each similar query, add to the set of similar queries all the queries that follow them in the query graph
3. Sort by the ranking function
   - Sums the normalized weights of the edges of the graphs
4. Return the first *m* queries to be suggested

Experimental evaluation

- Two datasets with real data from two websites

Trova Pr€zzi ShoppyDoo

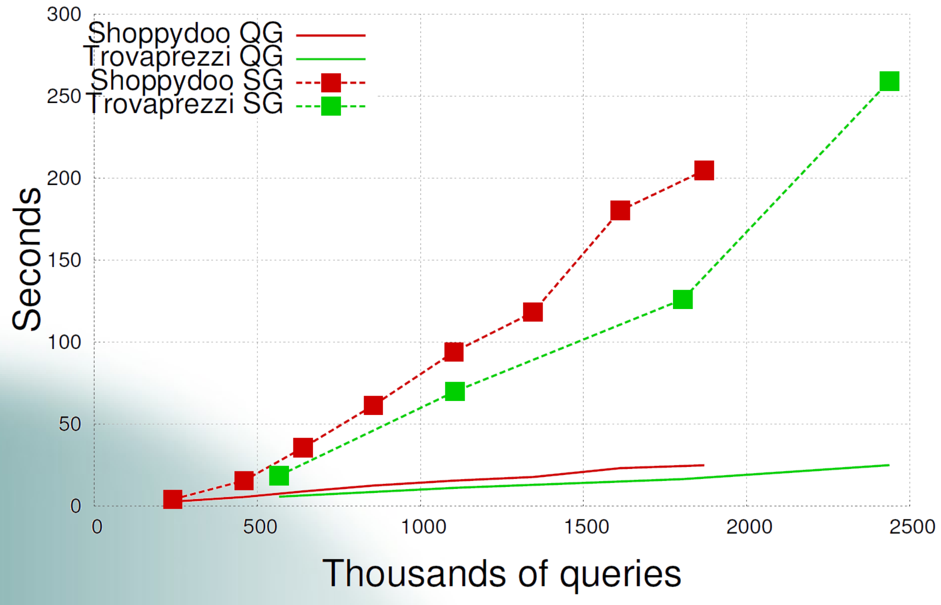
- Quantitative experiment:

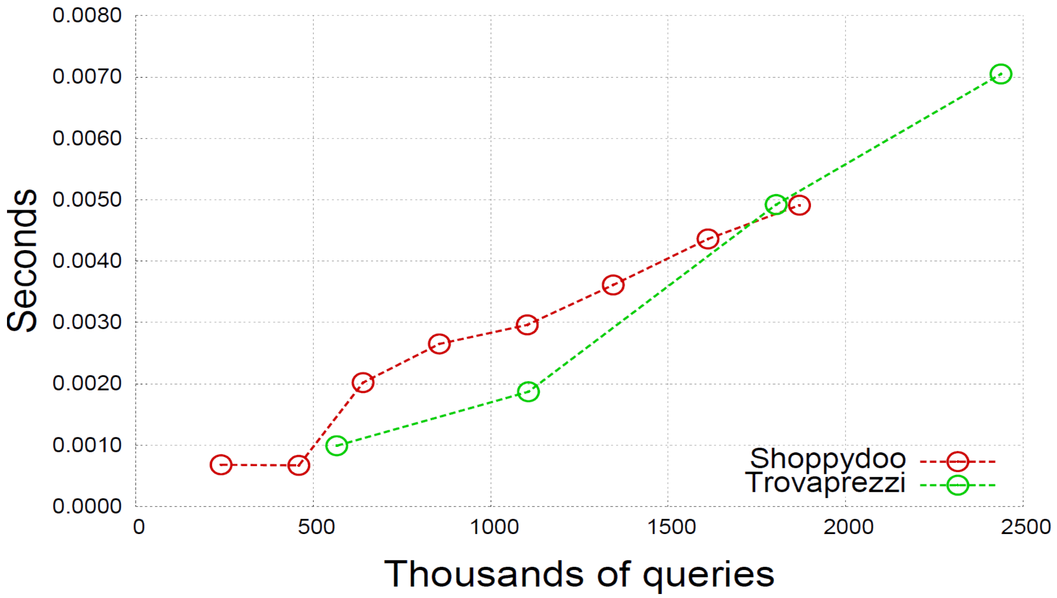
  Temporal complexity experimentally measured

- Qualitative experiment

  End-to-end results evaluated by users

Temporal complexity





Time needed to build the QG and the SG

Time needed by the online suggestion system

Both complexities are confirmed to be <u>between linear and N log N</u> (worst case) in function of the number of queries used to generate the models

User experience evaluation

Metrics:

Coverage

Quality

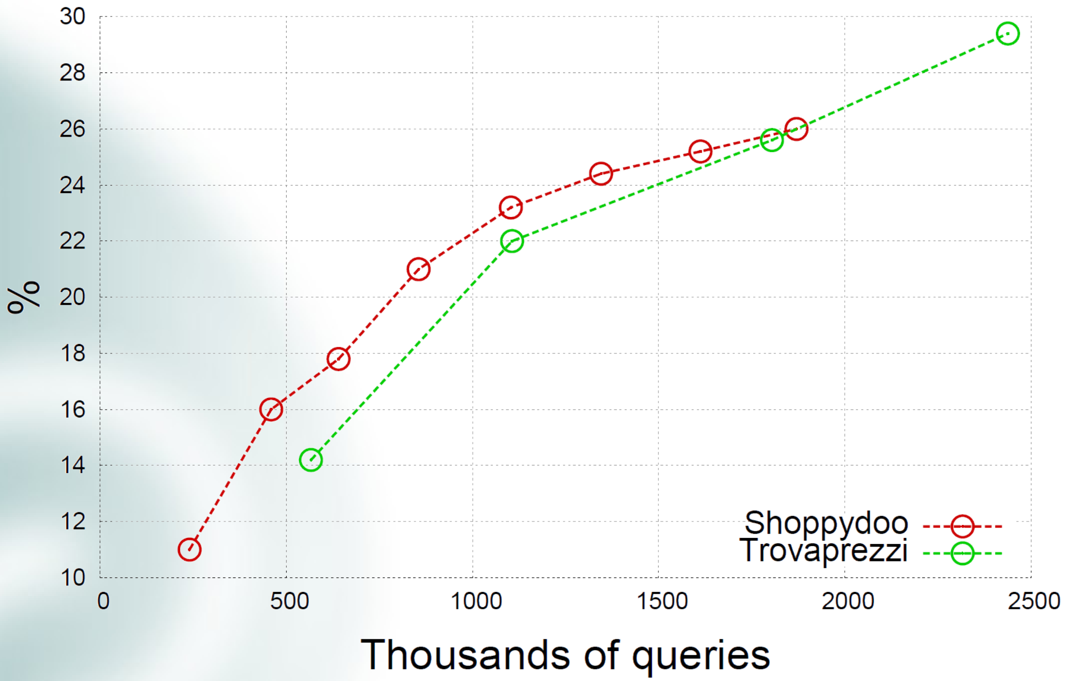
User experience evaluation

Metrics:

Coverage

Quality

Indicates for how many input queries the algorithm returns at least a minimum amount of suggestions



Another experiment has been run using the logs of five days from Trovaprezzi (6,2 millions of queries) reaching a coverage value of 37,5%

User experience evaluation

Metrics: Coverage Quality

How many suggestions which are useful to the user are obtained

Evaluation is conducted by humans which evaluates the end-to-end results produced by the algorithm using a simple web application.

Quality value obtained: 70,1%

User experience evaluation

Metrics: Coverage Quality

Overall results:

- Quality: 70,1%
- Coverage: 37,2%
- Average time for a suggestion: 0,006 seconds

Linux workstation with 64bit 2,6 Ghz CPU, 8Gb of RAM memory
Implementation with the Ruby language.