# Coniunge et Impera: Multiple-Graph Mining for Query-Log Analysis

Ilaria Bordino[1,⋆], Debora Donato[2], and Ricardo Baeza-Yates[3]

[1] Sapienza Università di Roma, Rome, Italy
bordino@dis.uniroma1.it
[2] Yahoo! Labs, California, US
debora@yahoo-inc.com
[3] Yahoo! Research, Spain
rbaeza@acm.org

**Abstract.** Query logs of search engines record a huge amount of data about the actions of the users who search for information on the Web. Hence, they contain a wealth of valuable knowledge about the users' interests and preferences, as well as the implicit feedback that Web searchers provide when they click on the results obtained for their queries.

In this paper we propose a general and completely unsupervised methodology for query-log analysis, which consists of aggregating multiple graph representations of a query log, tailored to capturing different semantic information. The combination is carried out by applying simple but efficient graph-mining techniques. We show that our approach achieves very good performance for two different applications, which are classifying query transitions and recognizing spam queries.

## 1 Introduction

The Web-search experience is nowadays part of the life of a continuously growing number of people. According to a recent study released by ComScore[1], the global Web-Search Market has increased by 41% in 2009, reaching more than 113 billion searches. Unique users and penetration[2] have experienced an impressive growth in the last decade. The most up-to-date estimations report the Internet is currently used by more than one fourth of the whole world population. Moreover, the highest growth rates have been observed within those segments of the world market where penetration is still very low.

The results of these studies provide clear evidence of the fact that, despite the massive user adoption and the maturity of the services offered by the Web industry, further efforts are still needed to conquer those fractions of the market where the Web experience is newer and penetration less exhaustive.

Commercial search engines attempt to improve the appeal and the usability of their services by offering tools like query recommendation, user profiling and

⋆ Part of this work was done while visiting Yahoo! Research Labs, Barcelona.
[1] http://www.comscore.com/
[2] http://www.internetworldstats.com/stats.htm

J.L. Balcázar et al. (Eds.): ECML PKDD 2010, Part I, LNAI 6321, pp. 168–183, 2010.

spam detection. They invest considerable amounts of resources in the development of instruments for gathering novel knowledge about the users' interests.

In this scenario, query-log analysis is broadly applied to obtain useful insights about the way users refine their queries, and the search strategies they apply to satisfy their information needs. Recent studies [7,17,20] have shown that the wealth of information stored in search logs can be successfully used to build a very accurate characterization of query reformulation types, which is a key step towards improving the service provided by search engines and developing innovative web-search paradigms. However, the most commonly adopted approaches share a major limitation, which is to be found in the usage of supervised learning, be it as a sole learning mechanism, or combined with classifiers and exact look-up on dictionaries. These editorial resources are very costly, and thus difficult to obtain for specific languages or cultures.

In this paper, we study the problem of developing effective approaches for query-log analysis. We aim to develop methods that are simple and at the same time able to deal with huge data volumes. The approach we propose is completely unsupervised and based on the map/reduce paradigm.

We use graph structures to build compact and navigable representations of the information extracted from query logs. The idea of inferring graphs from query logs has been extensively explored by recent research [5,6,7,12,14,24]. The above works focus on the analysis of a single graph, which tipically captures only some particular aspects of the interactions between users and search engines. None of these approaches is able to exploit exhaustively the huge amount of hidden information available in the log.

In this work, we propose to analyze query-log data through the joint mining of multiple graphs, as the combination of them is the ultimate wisdom-of-crowds approach. More specifically, our main contributions are listed below.

- We present *Coniunge et Impera*, a general framework for query-log analysis, designed to provide an effective support for the execution of many tasks that are relevant from a search-engine point of view. The fundamental building blocks of our framework consist of (*i*) a collection of graph projections extracted from a query log according to different semantic criteria, and (*ii*) a set of operations to be used for mining and maintenance purposes.
- Concerning the choice of the graph representations, we build three graphs that relate queries according to various types of information: common words, common clicked results, session information. The definitions we use are introduced by Baeza-Yates [2].
- The toolbox built for graph analysis includes set operations and more complex graph operations, like extraction of subgraphs, connected components and articulation points. The choice of the operations was driven by the applications we had in mind, which are extensively described in the paper. Although interesting results can be obtained by applying operations as simple as set operations, we will show that more complex graph algorithms are needed to exploit all the wealth of information enclosed in the graphs.

- The operations are also used for the maintenance of the framework: for example, the union of two graphs can be computed to merge the data extracted from consecutive snapshots of a query log. The extraction of subgraphs can be applied to restrict the analysis to a subset of queries satisfying some particular conditions.
- To demonstrate the practical applicability and usefulness of our method, we analyze a large query log provided by a commercial search engine and we show how to customize our approach for two different applications, namely, classifying query transitions and recognizing spam queries.

We remark the fact that our work aims at providing a general instrument that can be useful for many different problems. For this reason, we impose no restrictions on the building blocks of our system, and we intentionally leave room for extensions. For example, more complex graph definitions could be introduced to analyze the semantics behind queries at a finer granularity.

It is also worth to observe that the approach we follow is completely unsupervised: it does not require the usage of any knowledge bases or other expensive editorial resources. This characteristic makes the method very general and applicable to different data, without any kind of linguistic and culture-specific issues.

The remainder of the paper is organized as follows. Section 2 discusses previous work. In section 3, we present the definitions used to extract different graphs from raw logs. Section 4 introduces the set of instruments that we included in our framework for mining and maintenance purposes. Various examples of the practical usefulness of the chosen operations are provided. Next, we show how to customize the *Coniunge et Impera* methodology for classifying query transitions (see Section 5) and for detecting spam queries (Section 6). Finally, Section 7 offers our concluding remarks.

## 2 Related Work

**Inferring graphs from query logs.** The idea of extracting a graph structure from query-logs has been introduced in the last decade. The attention on the possibilities offered by using graphs to mine different semantic information implicitly contained in query-logs was formalized by the work of Baeza-Yates [2]. Here five different types of such graphs, relying on different criteria to connect queries by edges (e.g. common words, common clicked URLs), are studied.

*Click graphs* [6,12,24] have been extensively used for various purposes. A click graph is a query-document bipartite graph, where a query is connected to the documents that were clicked in the associated result list.

Inspired by the works on click graphs, Baeza-Yates and Tiberi [5] propose a novel way to represent queries in a vector space based on a graph derived from the query-click bipartite graph. This graph is then used to find similar queries.

Starting from the click graph, Castillo et al. [10] define two alternative graphs, the *view graph* and the *anti-click graph*. In the view graph the edge set of the click graph is replaced with the one containing edges that relate a query to the documents whose URL has been viewed in the answer list returned to the

user, but not necessarily clicked. The view graph is a generalization of the click graph since each click is also a view. The anti-click graph intends to capture the negative feedback that users implicitly give to a top-ranked document when they ignore it by clicking on documents ranked below. This graph contains one edge between a query and any not-clicked documents ranked higher than one clicked by the user.

Boldi et al. [7] introduce the *query-flow graph*, which models user behavioral patterns and query dependencies. In this graph, a directed edge from one query to another means that the two queries are likely to be part of the same search mission. Any path over the query-flow graph may be seen as a searching behavior, whose likelihood is given by the strength of the edges along the path.

**Similarity or distance between queries.** Projecting query-logs on graphs is not the only way to infer semantic relations from implicit user feedback. The most explored approach consists of defining a similarity function between queries.

Raghavan and Sever [21] propose to measure the similarity between two queries through the mining of the differences in the ordering of the documents retrieved. Due to a matter of scalability, this technique is not of practical use when one has to deal with the whole Web.

Wen et al. [24] cluster similar queries to support recommendations for queries frequently submitted to search engines. They use several notions of query distance, based either on the keywords composing the query text or on the set of common clicked URLs.

Fonseca et al. [15] use association rules to discover related queries. Their approach views the log as a set of transactions, thus failing in discovering the most interesting related queries, which are the ones submitted by different users. It also encounters problems in determining successive query sessions that belong to the same search process.

Baeza-Yates et al. [3,4] introduce a term-weight vector model to represent queries. In this model, a weight is assigned to each term occurring in the content of the Web pages clicked after a query, depending on the number of occurrences of the query within the log and on the number of clicks of the documents which the term appears in.

## 3 Preliminaries

### 3.1 Query Graphs

We analyze query-log data by building different *query graphs* to capture the various semantic aspects of the relations between queries. We use the term *query graph* to refer to a graph $G = (V, E, w_V, w_E)$ extracted from a snapshot of a search-engine log, such that the set of nodes $V$ comprises all the distinct queries appearing in the log, whereas the edges in the set $E$ denote the existence of a particular semantic relation between queries, depending on a specific criterion considered in the graph definition. The weighting functions $W_V : V \to \mathcal{N}$ and $W_E : E \to \mathcal{R}$ are respectively used to associate nodes and edges with a weight whose meaning changes depending on the information used to generate the graph.

**Table 1.** Graph definitions: Summary

| Graph feature | Word graph | Session graph | Click graph |
|---|---|---|---|
| **Edge** | Common words | Consecutive appearance in a session | Common clicked results |
| **Node weight** | # Occurrences | # Sessions | # Occurrences |
| **Edge weight** | # Common words in the text of $q_i$ and $q_j$ | # Sessions in which $q_i$ and $q_j$ appear sequentially | Cosine similarity of the vectors of URLs clicked for $q_i$ and $q_j$ |

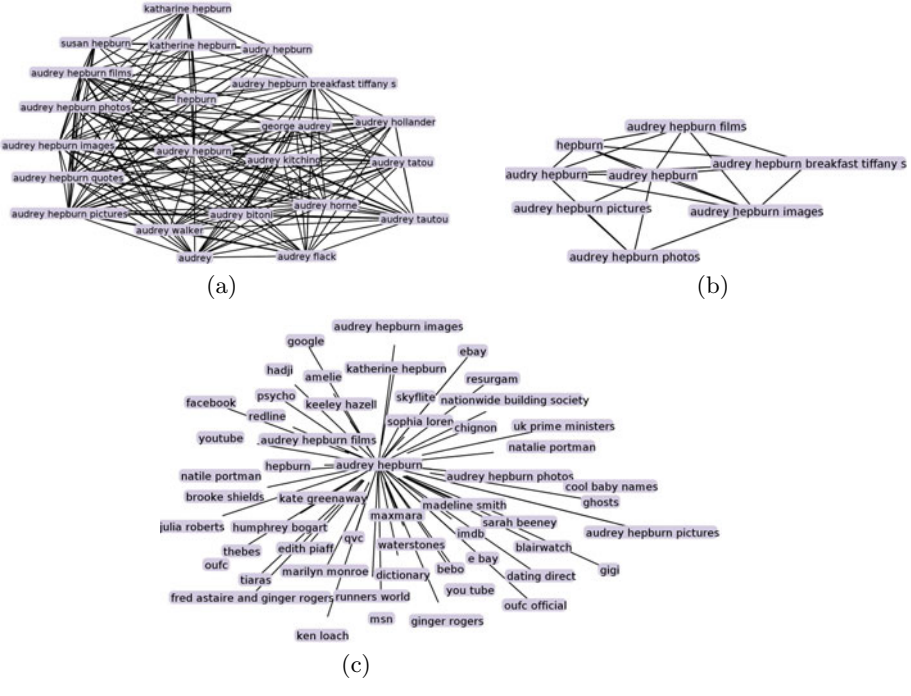


**Fig. 1.** Sample graphs for *audrey hepburn*: Word (a), Click (b) and Session (c) graph

In this paper, we generate three query graphs using the definitions of *Word Graph*, *Session Graph* and *Click Graph* (called *Url Cover Graph* in the original paper) formalized by Baeza-Yates [2]. We summarize the definitions in Table 1 and give an example of each graph in Figure 1.

**Dataset.** We studied a sample log of early 2008 from the Yahoo! UK search engine. The raw data consisted of a sequence of annotated sessions, containing more than 20 million distinct queries. Following [2] and [7], we removed the queries containing non-English characters. Motivated by preliminary assessments, we

**Table 2.** Basic statistics

| Graph | # Nodes | # Edges | # Isolated nodes | # CC | Size of largest CC |
|---|---|---|---|---|---|
| Word graph | 801 876 | 565 095 123 | 121 423 | 135 942 | 649 298 |
| Click graph | 801 876 | 21 971 751 | 161 344 | 195 379 | 549 217 |
| Session graph | 801 876 | 6, 654 614 | 25 657 | 26 199 | 775 096 |

filtered out the queries with less than 5 occurrences in the log, and we retained a common subset of queries to avoid the dishomogeneties caused by the different definitions. Table 2 reports statistics about the three graphs we built.

## 4 A Software Framework for Query-Log Graphs

We now describe the algorithms included in our framework. We carefully studied the operations that were necessary for the purposes of (*i*) data mining, and (*ii*) manteinance of the framework. We selected a collection of initially minimal and crucial mining tools that are listed below. The presentation is intentionally informal and descriptive. The list should by no means be considered exhaustive. Different operations might be needed for different applications.

A recent work [16] proposes a formal algebra that manipulates graphs as basic units of information, maintaining the same basic characteristics and the same expressive power as the relational algebra. We don't follow a similar approach because we aim at handling complex graph problems, which cannot be efficiently represented and studied by means of relational algebra.

### 4.1 Operations

We use both *binary* operations and *unary* operations.

**Binary operations.** The binary operations are the fundamental set operations that can be applied on graphs: union, intersection, difference.

- **Union.** Given two query graphs $G$ and $H$, their union is represented by a graph $F = G \cup H$ such that $V(F) = V(G) \cup V(H)$ and $E(F) = E(G) \cup E(H)$.
  **Example.** The union of two graphs is a critical operation for the maintenance of our framework: it is necessary to merge data extracted from different snapshots of the log. Updates are needed from time to time to regenerate the graphs, given that the information extracted from query logs suffers a slow aging effect, as demonstrated in [18].
- **Difference.** The set difference of two graphs $G$ and $H$ is a graph $F = G \setminus H$ such that $V(F) = V(G) \setminus V(H)$ and $E(F) = E(G) \setminus E(H)$.
  **Example: Similar queries.** Identifying queries that express closely related information needs is crucial task for generating query recommendations [4]. In the *Coniunge et Impera* framework, a very simple solution for the problem of detecting queries that are semantically, but not sintactically similar consists of computing the set difference between the Click graph and the Word

graph: the edges included in the result connect queries that have common clicked results, and thus are likely to be related to the same topic despite the fact that their texts have no common words. The edge weights of the click graph can be used to filter out the least significant relations.

- **Intersection.** The intersection of two graphs $G$ and $H$ is a graph $F = G \cap H$ such that $V(F) = V(G) \cap V(H)$ and $E(F) = E(G) \cap E(H)$.
  **Example: Finding logical sessions.** Identifying changes in the *search mission*, that is, in the information need [7,17] expressed by a user within a given session, is a critical issue for modeling user behavior and predicting user satisfaction. To detect different search missions within a user session, we compute the set difference between the Session graph and the intersection of the Click graph and the Word graph.

**Note.** The set operations can be applied on graphs that have weights both on the nodes and on the edges. We impose no restrictions on how to combine the weights, as the best choice depends on the particular task that one has in mind.

**Unary operations.** We consider the following unary operations:

- **Node filter.** Given a query graph $G$ and a mathematical condition $c$, this operation returns a query graph $H$ corresponding to the induced subgraph of $G$ whose vertex set is formed by all the vertices in $V(G)$ associated with weights satisfying $c$.
- **Edge filter.** Given a query graph $G$ and a mathematical condition $c$, this operation returns the (non-induced) subgraph of $G$ whose vertex set includes all the vertices in $V(G)$, while the set of edges is formed by the edges in $E(G)$ associated with a weight that satisfies the specified condition.
  **Examples.** We used the filter operations in the preliminary phases of our study, when we extensively analyzed the structural properties of the graphs extracted from the log. We tested various thresholds on the weights of nodes and edges, to study the differences between the resulting denser or sparser versions of the graphs.
  The filter operations allow to select portions of the graphs that are relevant for a particular application. For example, if we aim to detect *similar queries*, we can apply an edge filter on the Click graph, selecting the edges with a large weight. In the Click graph, a large-weight edge indicates a significant degree of similarity in the results clicked for the two queries. We applied this idea in our methods for detecting error corrections (see Section 5).
  An opposite filter is needed to identify *spam queries*, i.e., queries that attract a high number of spam pages in their result set. The heuristic we developed for this application, which is presented in Section 6, exploits the fact that a low-weight edge in the Click graph is indication of a relation of very poor quality between the queries connected by such edge. The clicked results shared by these queries are usually spam or multi-topical URLs. The queries containing these URLs in their set of answers are included into a base set of spam candidates.

- **Connected components.** This operation takes a query graph $G$ and returns a set of graphs $S = \{G_1, G_2, \ldots, G_k\}$, such that, for each $i = 1, 2, \ldots, k$, $G_i$ is isomorphic to a maximal connected subgraph of $G$.
  **Example: Clustering queries.** The computation of the connected components in the Session graph and/or in the Click graph is required for the task of clustering queries, which is useful for many purposes, such as re-ranking, query recommendation, and finding logical sessions.
- **Biconnected components.** This operation takes a query graph $G$ and returns a set of graphs $S = \{G_1, G_2, \ldots, G_k\}$ such that, for each $i = 1, 2, \ldots, k$, $G$ is isomorphic to a maximal biconnected subgraph of $G$.
- **Articulation points.** Given a query graph $G$, this operation returns a set of query instances $S = \{q_1, q_2, \ldots, q_k\}$ such that $S \subseteq V(G)$ and, for $i = 1, 2, \ldots, k$ $q_i$ is an articulation point in $G$. An articulation point may belong to one or more biconnected component.
  **Example: Polysemic queries.** Polysemic queries [19,23] are related by a syntactic point of view (for instance, they share one or more common words in their text representations), but they cover unrelated semantic aspects. Articulation points in the Click graph and/or in the Word graph are natural candidates for word polysemy.

### 4.2 Graph Extraction and Representation

The raw data we analyzed consisted of a set of annotated sessions, containing 20 million distinct queries.

For efficiency and scalability, we used Hadoop's MapReduce[3] to extract the query graphs. MapReduce [13] is a popular programming model for processing large-scale data. This model allows users to write map/reduce components that are scheduled to distributed resources for processing, with a transparent handling of problems like parallelization, network communication, and fault tolerance.

Due to lack of space, we do not describe the three algorithms we implemented to generate the graphs. Our procedures cascade a sequence of 3-5 MapReduce jobs, which are used to gather the specific data that is needed to interconnect queries and to create the adjacency list of each query.

We ran the MapReduce routines on a server with 3G RAM. We needed 2 hours to generate the Session graph and 4 hours to create the Word graph and the Click graph. The time required for the Session graph was lower because the particular format of the input data made the extraction of the information required for this graph immediate. Altogether, the time used for generating the three graphs is less than half a day for our sample log. We also observe that our toolbox allows an incremental construction of the query graphs.

The operations included in our framework are complex and procedural by nature; they require global computations that involve all the nodes in a graph. This consideration oriented us towards the choice of compressed graph representations to store and manipulate our data.

[3] `http://hadoop.apache.org/common/docs/current/mapred_tutorial.html`

Compressed graph representations have become very attractive, as they allow to overcome the limitations encountered by many modern applications, whose storage requirements exceed the capacity of the faster memories.

Web search uses graphs as natural models for the Web structure. Several algorithms that are used by search engines to rank pages, discover communities and so on are run on these Web graphs.

For graph representation, we use Webgraph [9], a framework that obtains state-of-the-art results in terms of compression through the combination of several mechanisms, such as node reordering, differential encoding, compact interval representations, and references to similar adjacency lists.

Starting from a raw input of $\sim$ 30GB, we obtained three representations of the following sizes: $500M$ for the Word graph, $500M$ for the Click graph, and $20M$ for the Session graph.

Exploting the fact that our graphs are compact and can be efficiently kept in main memory for navigation purposes, we developed main-memory implementations of our algorithms. Due to lack of space, we omit details about the specific realizations, which are all based on standard algorithms. All the operations require time linear in the size of the input graphs. In our experiments, we reported running times in the order of minutes for all the algorithms implemented.

We also explored the possibility of developing MapReduce realizations for our algorithms. We decomposed each operation into a series of MapReduce processes, adopting an approach similar to [11]. The obtained implementations are still inefficient, mostly because many operations are based on graph traversal, which requires a large number of iterations because a mapper can only read a random record for each map operation. J.Ekanayake [1] has recently developed *i-MapReduce*, a streaming-based framework that supports iterative MapReduce computations. We plan to explore the usage of this instrument in future work.

## 5 Classifying Query Transitions

We now show how to apply our methodology to the task of classifying query transitions. By *transition* we mean a pair of queries that a user submitted consecutively to a search engine. The analysis of these query sequences is extremely important for the purpose of understanding how the users reformulate their queries to gather information of better quality.

Retrieving information from the Web is a process that requires a continual interaction between the user and the search engine. Only in half of the cases an information need is satisfied with just a single query [22]. In the other cases, the user submits a refined query, because she is not satisfied with the documents obtained in the first attempt. This might happen for various reasons, for example because the former query contained errors, or it was either too general or too specific for the purpose the user had in mind. A transition between queries that are part of the same search mission is usually referred to as a *query reformulation*.

Characterizing query reformulation patterns is a task of critical importance to improve the relevance of web-search results, or to refine tools for query

recommendation. Following the taxonomy introduced and used in [7,8], we focus on the task of recognizing the following types of query transitions:

- **Error correction:** the user is trying a different spelling or capitalization of a query. Example: *audry hepburn, audrey hepburn*.
- **Generalization:** the second query is more general than the first one. Example: *audrey hepburn quotes*, *audrey hepburn*.
- **Specialization:** the first query is more specific than the second one. Example: *audrey hepburn films*, *audrey hepburn breakfast tiffany s*.
- **Parallel move:** the user is modifying her query to search for something related but not equivalent. Example: *audrey hepburn*, *sophia loren*.
- **Different mission:** the user is trying to satisfy a completely different information need. Example: *audrey hepburn*, *runners world*.

The last two types of query transitions, *Parallel move* and *Different mission*, can be considered as belonging to a broader category, which comprises the transitions that represent a change in the information need expressed by the user. In the work of Jones et al. [17], these two transition types are both labeled as *Different chain*. We adopt the same approach, aggregating Parallel move and Different mission into a more general category, which we call *Different goal*. In the remainder of this section we describe how to extract and aggregate information from the Word graph, the Click graph and the Session graph to classify query transitions.

### 5.1 Unsupervised Approach to Query Transition Classification

Our method for classifying query transitions combines three independent heuristics, which are tailored to labelling non-overlapping sets of transitions. This is an extremely advantageous characteristic, which implies that the three building blocks of our approach can be either applied in parallel, for example on a grid, or they can be executed consecutively in a sequential setting, from the most to the least aggressive approach, so that the amount of data to be analyzed is drastically reduced at each step.

**Algorithm 1: Identifying different goals.** Our first algorithm aims at recognizing transitions between two queries that express a different search goal. The algorithm computes the set difference between the Session graph and the union of the Click graph and the Word graph. This corresponds to filtering the edges of the Session graph to retain only the transitions connecting two queries that have no common clicked results and no common words in their text representations.

**Algorithm 2: Error corrections with no common words.** An *Error correction* is a query reformulation that the user submits to fix a typo, usually trying a different spelling or capitalization of a query. Our second algorithm detects error corrections involving queries with no words in common.
**Step 1: Edge filter on the Click Graph.** We start from the Click graph and we filter out the edges whose weight is less than 0.3. In this way, we only retain

the top 10% strongest semantic relations, which is useful because in the case of an error correction the search goal remains the same.
**Step 2: Intersection with the Session graph.** We intersect the subset of the Click graph that was computed in the previous step with the Session graph: This is needed to identify query transitions.
**Step 3: Set difference with the Word graph.** We compute the set difference between the subgraph obtained at the previous step and the Word graph: this is required to remove transitions with common words. (These transitions are considered by our third algorithm).
**Step 4: Filter by Click labels.** We refine the partial result obtained in the previous step by selecting the edges whose type label in the Click graph is equal to 2, meaning that the set of results clicked for the first query is strictly contained in the set of answers associated with the second query. Upon detecting errors in the text of the query submitted by a user, search engines suggest the proper spelling, presenting the top-ranked answers of the correct formulation in the first positions of the result list. This practice makes the above condition on click labels very likely to be satisfied by error-correction transitions.
**Step 4: Filter by edit distance.** Finally, we use Levehnstein distance to isolate error corrections from other cases. We retain only the transitions for which the edit distance between the two queries is no greater than 0.2. The threshold was chosen after experimenting with different values (0.2, 0.3, 0.4): As expected, the lower the threshold, the more we are effective in isolating error corrections from other reformulations.

**Algorithm 3: Transitions involving queries with common words.** Our last algorithm analyzes the transitions involving queries with common words in their text representations.
**Step 1: Intersection between the Session graph and the Word graph.** This is needed to isolate the set of transitions we want to focus on. Many types of reformulations are characterized by overlapping query texts. A number of tests are applied to distinguish the various cases.
**Step 2.** The text of the first query is a subset of the text of the second query: then we label the transition as Specialization.
**Step 3.** The text of the first query is a superset of the text of the second query: then we label the transition as Generalization.
**Step 4.** We use edit distance to isolate error corrections.

### 5.2 Evaluation

We used an annotated query-flow graph [8] extracted from the same log snapshot as the ground truth for evaluating the quality of results. This graph originally contained $\sim$ 21M nodes and $\sim$ 43M edges. Restricting to the nodes included in our query graphs, we extracted a subgraph that contains $\sim$ 3.5M edges.

The model introduced by Boldi et al. [8] and used to annotate the edges of the query-flow graph is able to produce an automatic classification of query

**Table 3.** Classification of query transitions: quality of results

| Transition type | TP | TN | FP | FN | Precision | Recall | Accuracy |
|---|---|---|---|---|---|---|---|
| Different Goal | 2 594 560 | 318 125 | 516 501 | 4 970 | 0.834 | 0.998 | 0.848 |
| Generalization | 58 627 | 3 366 295 | 2 660 | 6 574 | 0.956 | 0.899 | 0.997 |
| Specialization | 179 774 | 2 827 724 | 19 | 426 639 | 0.999 | 0.296 | 0.878 |
| Error Correction | 60 330 | 3 249 459 | 21 685 | 102 682 | 0.736 | 0.370 | 0.964 |

reformulation types with an accuracy as high as 92%. This method represents the state of the art for classifying query transitions: However, we remark the fact that this is a supervised approach, based on machine learning from a human-labeled log sample.

To evaluate the quality of our method, we compared the classification generated by our algorithms with the labels associated to the same transitions in the query-flow graph. We computed True Positives, True Negatives, False Positives, False Negatives, Precision, Recall, and Accuracy. Results are shown in Table 3.

We report the global evaluation of our methodology in Table 3. We remark two main results. The first one is that every part of our methodology relies on merging the information coming from at least two different query graphs. The second one is that the approach is totally unsupervised, and uses operations which are linear in the size of the graphs involved. Moreover, since the subsets of transitions labeled by each heuristic do not overlap, the number of edges to consider decreases at each step.

Altogether, the labels obtained with our method agreed with those assigned to the same transitions in the query-flow graph in 84% of the cases. More specifically, we obtained very good results in the cases of generalization and different goal, for which we measured an accuracy respectively of 99 and 85%. On the other end, we noticed a very low recall in the case of specialization and error correction. Since the query-flow graph was labeled using a model, we wondered whether this poor accuracy could be induced by errors in the query-flow graph rather than in our methodology.

To investigate this hypothesis, we manually evaluated a random sample of the transitions for which our method and the one based on the query-flow graph did not agree. We selected 1K transitions labeled as specialization in the query-flow graph, and $1K$ transitions labeled as error correction. We considered the transitions divided into buckets according to their frequency of occurrence in the log. Three human assessors were asked to evaluate the transitions, assigning them one of the reformulation types considered in this study.

We assumed the manual assessment to be the golden truth for the sample, and we measured how well the two automatic classification methods (ours, and the one using the query-flow graph) agreed with the golden truth. The evaluation gave the following results: the labels produced by our method agreed with the ones assigned by human assessors in 80.4% of cases, while the query-flow-graph labels resulted equal to the ground truth in 12.8% of cases.

To evaluate the statistical significance of this result, we associated each automatic method with a vector that has a cell for each sampled transition, containing a 1 if the label assigned by the method agreed with the ground truth, and a 0 otherwise. We them performed a Wilcoxon signed-rank test, comparing the positive differences between the judgements obtained for the two methods. The test determined a statistical significance at $p \ll 1\%$ for the difference between our methodology and the query-flow graph.

Hence, we believe that the actual precision of our approach is higher than the one obtained with this experiment, and that the disagreement with the query-flow graph is due to errors in the model used by the latter method to train the classifier.

## 6 A Heuristic for Detecting Spam Queries

*Spam queries* [10] are queries that generate a high number of spam pages in the top positions of their lists of answers. Identifying these queries can uncover meaningful information for the task of designing more robust spam-detection algorithms. We now focus on detecting queries that have collected many spam results. Spam pages typically include many unrelated keywords or links, advertising and machine-generated content. These characteristics make spam pages likely to be selected as answers for very different queries, which may express unrelated or poorly related information needs.

We use the Click graph to identify candidate spam pages. Given that spam pages often cover a large number of unrelated topics, we follow the approach suggested in [5] to identify multitopical URLs. In the Click graph, low-weight edges indicate a poor-quality relation between the involved queries. In our experiment, we consider all the edges whose weight is $< 0.005$; in this way, we isolate the least significant 25% edges in the graph.

We count each edge as a spamicity vote for all the URLs in the intersection of the result sets associated with the two queries. We consider the top 200 URLs that obtain the highest number of votes. All the queries that have one of these URLs in their result set form a clique in the Click graph. We retain the groups of queries that are also mutually connected in the other graphs.

As a result, we obtain a list of 9 140 queries. We filter out the queries that have more than $1K$ occurrences to get rid of navigational queries. We then sort the queries by decreasing degree in the Click graph and we select the top 10% of the queries with highest degree: the intuition is that queries that form small dense subgraphs in the three graphs, while maintaining many connections to other nodes in the graph, are good spam candidates. In the end, we extract a set of 928 queries.

### 6.1 Experimental Evaluation

The above method led us to isolate 928 queries. We tested the effectiveness of our strategy by conducting an experimental evaluation aimed to assess the

**Table 4.** (a): Top categories obtained from Yahoo! Directory (b): Examples of spam queries extracted from the graphs

(a)

| Level 1 | Level 2 |
|---|---|
| Regional | Shopping |
| Business and Economy | US States |
| Entertainment | Countries |
| Arts | Business |
| Society and Culture | Music |
| Recreation | Movies |
| Computers and Internet | TV Shows |
| News and Media | Humanities |
| Government | Actors |

(b)

| Query | Spam | 1st Category | Cat. |
|---|---|---|---|
| filma shqiptar | 5 | Video | 1 |
| pro wresting | 1 | Entertainment | 11 |
| shapely figures | 5 | Entertaniment | 9 |
| video eyewear | 3 | Business | 11 |
| ana visi | 2 | Entertainment | 1 |
| make money | 6 | Entertainment | 10 |
| spanked cutie | 5 | Recreation | 10 |
| suger babes | 4 | Adult | 1 |
| pin up girls | 4 | Art/Shopping | 11 |

spamicity of the selected queries. For each query, we manually evaluated its ability of attracting spam results by resubmitting it to the Yahoo! search engine and counting the number of spam pages obtained within the top ten results. We marked as spam every query containing at least one spam result in the top answers.

Given that no agreement on a univocal definition of *spam* page has been reached so far, we enforced the strength of our judgements following the labeling guidelines used in the construction of the WEBSPAM-UK2007[4] collection, which is a Web spam dataset built through the collaborative effort of a team of volunteers to advance research on Web spam detection.

We matched the top results obtained for each query selected by our algorithm against the Yahoo! Directory. Table 4(a) shows the most frequent categories associated with the results returned for the queries. Only the top levels are considered: Level 1 is the top category, Level 2 the second highest category. The main categories obtained include Shopping, Business, Movies, Sex and Adult galleries.

The results obtained in the assessment phase are really encouraging: two thirds of the evaluated queries collected at least one spam result, and thus were marked as spam queries. A few examples are shown in Table 4(b), which reports, for each query, the number of spam results, the main category assigned by Yahoo! Directory and the number of different categories associated.

The inspection of the categories obtained for our sample set makes us notice that most of the queries analyzed, even those that are not classified as spam, are associated with (top) results related to adult, celebrities, video or shopping. We believe this is a confirmation of the effectiveness of our strategy for selecting queries that are characterized by a significant degree of spamicity, or, if not spam, by the covering a broad set of topics.

The peculiar nature of the queries discovered by our heuristic suggests an immediate application of the tool to developing parental filters.

[4] `http://www.yr-bcn.es/webspam/datasets/uk2007/guidelines/`

### 6.2 Conclusions

In this work we have introduced the idea of combining different graph representations of query logs coming from different relevance signals, to be able to tackle different problems related to queries in a completely unsupervised manner. This is very important in cases where we do not have enough labeled data such as in non-popular languages.

The combination of the graph is done with very simple operations over graphs. In spite of the simplicity of the approach we show that we can obtain very good results for two different applications as detecting the type of a query transition or query spam. The former can be used to adapt the ranking of new queries, whereas the results obtained in the latter case could be used directly in the design of parental filters.

Further work includes scalability issues of the methods proposed, testing our implementation in an iterative map/reduce scenario as well as apply these ideas to other applications. In fact, the same ideas could be applied to the identification of polysemic queries, the recognition of logical sessions (missions) or for query recommendations. We believe that the approach introduced in this paper will be a valid support for the analysis of the huge amount of data stored in the logs of search engines. Our methodology can provide efficient methods for combining different aspects of the same multi-faceted scenario and acquiring novel knowledge that might not be obtained through other methods, in particular if we have the restriction of having to use unsupervised methods.

## References

1. Architecture and Performance of Runtime Environments for Data Intensive Scalable Computing, Portland, OR, 11/09 (2009)
2. Baeza-Yates, R.: Graphs from search engine queries. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 1–8. Springer, Heidelberg (2007)
3. Baeza-Yates, R., Hurtado, C., Mendoza, M.: Query Clustering for Boosting Web Page Ranking. In: Favela, J., Menasalvas, E., Chávez, E. (eds.) AWIC 2004. LNCS (LNAI), vol. 3034, pp. 164–175. Springer, Heidelberg (2004)
4. Baeza-Yates, R., Hurtado, C., Mendoza, M.: Query Recommendation Using Query Logs in Search Engines. In: Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004. LNCS, vol. 3268, pp. 588–596. Springer, Heidelberg (2004)
5. Baeza-Yates, R., Tiberi, A.: Extracting semantic relations from query logs. In: KDD 2007, pp. 76–85. ACM, New York (2007)
6. Beeferman, D., Berger, A.: Agglomerative clustering of a search engine query log. In: KDD 2000, pp. 407–416. ACM Press, New York (2000)
7. Boldi, P., Bonchi, F., Castillo, C., Donato, D., Gionis, A., Vigna, S.: The query-flow graph: model and applications. In: CIKM 2008, pp. 10+ (October 2008)
8. Boldi, P., Bonchi, F., Castillo, C., Vigna, S.: From 'dango' to 'japanese cakes': Query reformulation models and patterns. In: WI 2009, IEEE, Los Alamitos (September 2009)

9. Boldi, P., Vigna, S.: The webgraph framework: Compression techniques. In: WWW 2004. ACM Press, New York (2004)
10. Castillo, C., Corsi, C., Donato, D., Ferragina, P., Gionis, A.: Query-log mining for detecting spam. In: AIRWeb 2008 (2008)
11. Cohen, J.: Graph twiddling in a mapreduce world. Computing in Science and Engg. 11(4), 29–41 (2009)
12. Craswell, N., Zoeter, O., Taylor, M., Ramsey, B.: An experimental comparison of click position-bias models. In: WSDM 2008, pp. 87–94. ACM Press, New York (2008)
13. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: OSDI 2004: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, p. 10. USENIX Association, Berkeley (2004)
14. Diemert, E., Vandelle, G.: Unsupervised query categorization using automatically-built concept graphs. In: WWW 2009, pp. 461–470. ACM, New York (2009)
15. Fonseca, B.M., Golgher, P.B., de Moura, E.S., Ziviani, N.: Using association rules to discover search engines related queries. In: LA-WEB 2003. IEEE Computer Society, Washington (2003)
16. He, H., Singh, A.K.: Graphs-at-a-time: query language and access methods for graph databases. In: SIGMOD 2008, pp. 405–418. ACM, New York (2008)
17. Jones, R., Klinkner, K.L.: Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In: CIKM 2008 (2008)
18. Nardini, F.M., Perego, R., Silvestri, F., Castillo, C., Donato, D., Baraglia, R.: Aging effects on query flow graph for query suggestion. In: CIKM 2009 (2009)
19. Qiu, G., Liu, K., Bu, J., Chen, C., Kang, Z.: Quantify query ambiguity using odp metadata. In: SIGIR 2007, pp. 697–698. ACM, New York (2007)
20. Radlinski, F., Joachims, T.: Query chains: learning to rank from implicit feedback. In: KDD (2005)
21. Raghavan, V.V., Sever, H.: On the reuse of past optimal queries. In: SIGIR 2008, pp. 344–350. ACM Press, New York (1995)
22. Rieh, S.Y., Xie, H.: Analysis of multiple query reformulations on the web: the interactive information retrieval context. Inf. Process. Manage. 42(3), 751–768 (2006)
23. Song, R., Luo, Z., Wen, J.-R., Yu, Y., Hon, H.-W.: Identifying ambiguous queries in web search. In: WWW 2007, pp. 1169–1170. ACM Press, New York (2007)
24. Wen, J.-R., Nie, J.-Y., Zhang, H.-J.: Clustering user queries of a search engine. In: WWW 2001, pp. 162–168. ACM, New York (2001)