Noname manuscript No.
(will be inserted by the editor)

Query Reformulation Mining: Models, Patterns, and Applications

Paolo Boldi · Francesco Bonchi
Carlos Castillo · Sebastiano Vigna

the date of receipt and acceptance should be inserted later

Abstract Understanding query reformulation patterns is a key task towards next generation web search engines. If we can do that, then we can build systems able to understand and possibly predict user intent, providing the needed assistance at the right time, and thus helping users locate information more effectively and improving their web-search experience.

As a step in this direction, we build a very accurate model for classifying user query reformulations into broad classes (generalization, specialization, error correction or parallel move), achieving 92% accuracy. We then apply the model to automatically label two very large query logs sampled from different geographic areas, and containing a total of approximately 17 million query reformulations. We study the resulting reformulation patterns, matching some results from previous studies performed on smaller manually annotated datasets, and discovering new interesting reformulation patterns, including connections between reformulation types and topical categories. We annotate two large query-flow graphs with reformulation type information, and run several graph-characterization experiments on these graphs, extracting new insights about the relationships between the different query reformulation types.

Finally we study query recommendations based on short random walks on the query-flow graphs. Our experiments show that these methods can match in precision, and often improve, recommendations based on query-click graphs, without the need of users' clicks. Our experiments also show that it is important to consider transition-type labels on edges for having recommendations of good quality.

CR Subject Classification H.2.8 [Database Management]: Database Applications - *Data Mining* · H.4.3 [Information Systems Applications]: Communications Applications

Boldi and Vigna
DSI, Università degli studi di Milano
Via Comelico 39/41, I-20135 Milan, Italy
E-mail: {boldi,vigna}@dsi.unimi.it

Bonchi and Castillo
Yahoo! Research
Diagonal 177, Barcelona, 080018, Spain
E-mail: {bonchi,chato}@yahoo-inc.com

Keywords Query Log Mining · Query Flow Graph · Session Segmentation · Query Recommendation

1 Introduction

Information retrieval is an interactive and iterative process: only in approximately half of the cases an information need is satisfied with just a single query [41,38]. In the other half of the cases, the user has to *reformulate* her initial query because it was over- or under-specified, or did not use terminology matching relevant documents, or simply contained errors or typos. The picture is made even more complex by the fact that, although queries are typically short [8], they usually form together chains of topically-related activities [36]. Users are increasingly relying on search to accomplish complex objectives, such as planning a holiday (from travel to lodging, to sightseeing, dining and nightlife) entirely online. Additional complexity is brought on by those search tasks that are so difficult and important for the user (e.g., deciding which car to buy, finding a new job, moving to another city), that she can go back to the same search mission again and again during a long period [17].

In order to assist the users in locating information more effectively, most large-scale Web search engines have started offering various supporting tools. As an example, *query recommendations* are a mechanism to help user reformulating their queries: these recommendations are typically queries similar to the original one, and they are obtained by analyzing the query logs, for instance, finding recommendations by clustering of queries [42], or by identifying frequent re-phrasings [2].

Query logs are in fact the main source of information for building search assisting systems. Web query logs contain a wealth of information about how users interact with the search engine. Extracting behavioral patterns from this abundance of information is a key step towards improving the service provided by search engines and towards developing innovative web-search paradigms. In particular, and this is the focus of this paper, *by mining query logs we can understand the dynamics underlying the query reformulation process*, and use this knowledge in applications aimed at improving the users' web-search experience. In this context we identify two main tasks:

1. Identifying *search mission* borders, by distinguishing query transitions that are *reformulations*, i.e., queries with a similar information need [36,29], from query transitions that represent a mission change. Search missions are also known as *chains* [36] and in the rest of the paper we use the two terms as synonymous.
2. After identifying the search missions, the query reformulations inside each chain must be classified into *query reformulation types* (abbreviated QRT). In this paper we focus on four query reformulation types: generalization, specialization, error correction, parallel move.

We tackled the first problem in our previous work [9], where we built a machine learning model for predicting the probability that two subsequent queries are part of the same search mission. Such model was then used to annotate the arcs of the *query-flow graph*—an aggregated representation of latent querying behavior which is contained in a query log. We then used the query-flow graph in applications such as query recommendation and segmentation of user sessions.

In this article instead, we focus on the tasks of modeling query reformulation types and characterizing query reformulation patterns, approaching these as data mining problems.

Our main contributions are:

Model. We show that accurate automatic classification of QRTs is possible. Learning automatically from a human-labeled query log sample, we build a model for automatic classification of QRTs (Section 4). Our model exhibits a very high accuracy, $\approx 92\%$ discriminating among 4 different reformulation types. The classifier is able to predict correctly even very difficult cases. We describe in detail the process followed to build the model, and then we inspect the model behavior. To the best of our knowledge this is the first work learning a model for the automatic classification of QRTs by mining a query log.

Patterns of reformulation strategies. Thanks to our automatic classifier, we are able to label very large query logs and to analyze them (Section 5). We divide users sessions into search missions, then we label each mission with our model and transform into a string of QRTs. Thus the query log is transformed into a bag of strings from which we can compute frequent sequential patterns which represent high-level search strategies. We analyze approximately 17 millions QRTs, and we compare our findings with the ones in the literature obtained mostly over small, manually-assessed collections. Moreover, we investigate the connections between the reformulation type and the topical categories of the queries in the reformulation.

Reformulation graphs. Using our model we can annotate the arcs of a *Query–Flow Graph* [9] with QRTs, obtaining what we call a *query transition graph* (Section 6). We present a study regarding the properties of this annotated graph, including the relationships between the various query reformulation types.

Recommendations. We propose a family of methods for query recommendation based on short random walks performed on different slices of the query-flow graph (Section 7). Our experiments show that these methods can match in precision, and often improve, query-click based recommendations without using clicks. Moreover our methods provide more diversity in the result sets. Our experiments also show that transition probabilities from one query to the next are not enough, and for obtaining good recommendations it is important to filter out queries that are not part of the same search mission, and to add QRT labels to edges.

Section 2 describes related work, whereas in Section 3 we discuss the taxonomy of QRTs that we adopt in this paper. Based on this taxonomy, in Sections 4 we build a classifier which we deeply characterize. Then in Section 5 we apply the classifier to label the query transitions in two query logs. From the labelled query logs we extract query reformulation patterns that we analyze from different perspective. Using the same classifier, in Section 6 we label a query-flow graph with QRTs, and we deeply characterize such graph, which is then used for query recommendation based on short random walks in Section 7. The last section presents our conclusions and outlines future work.

A preliminary version of this paper was presented in [11].

2 Related Work

2.1 Determining reformulation types

In one of the oldest papers on the subject, Lau and Horvitz [31] study a hand-tagged log from the Excite search engine, and propose a classification of QRTs. Their aim is to

build a Bayesian model of the user behavior exploiting also temporal information. Rie and Xie [38] consider in more detail reformulation patterns: also in their case, there is no automatic classification model; instead, they manually label and analyze 313 search missions. While defining the classes of query reformulation types (Section 3) we basically follow their taxonomy. The difference is that we adopt a coarser granularity for those reformulations that, while being part of the same search mission, are not simple direct reformulations of the previous query. Recently, Jansen *et al.* [26,27] analyzed a larger query log (1.5 millions query reformulations) using an automatic classifier. Their classifier is manually built following the concepts presented in the paper by He *et. al* [24]. By "manually built" we mean that the rule that identify a class of QRT and the definition of the class itself coincide perfectly, i.e., there's no automatic learning involved. The classification is built on 6 features all based on term differences between the two queries.

Beside the size of the dataset we use, the fundamental difference of our work with the studies mentioned previously is that we learn a model by mining a large query log, using 27 features. As an example, the classifier adopted by Jansen *et al.* [26,27] *defines* specialization as a query with additional terms; instead, by learning the model we obtain that specialization (as judged by humans) can be characterized by a combination of factors including query length and cosine similarity of n-grams. Similarly, a generalization was just a query composed of a subset of the original terms. Our classifier, instead, besides *finding* these expected rules is also able to *discover* unexpected things, e.g., that the reformulation from "`dango`" to "`japanese cakes`" is actually a generalization, even if the two queries have no terms in common and the second one is longer than the first one.

In the context of *image search*, [21] analyzed manually assessed reformulations of a group of users. In that case, of course, a number of reformulations involves interactions between text and images.

2.2 Query log analysis and applications

The importance of mining query logs to extract useful information about user behavior is clear since the seminal works [25,40]; such analysis has found fruitful application in many different contexts such as query recommendation [2,44] and document ranking [16].

Most of the work on query recommendation has focused on measures of query similarity [44,18] that can be used for query expansion [2] or query clustering [2,42]. A first attempt to model the user sequential search behavior is presented by Zhang and Nasraoui [44]: the arcs between consecutive queries in the same session are weighted by a damping factor d, whereas the similarity values for non consecutive queries are calculated by multiplying the values of arcs that join them. Instead, Fonseca *et al.* [18] discover related queries with a method based on association rules. Each session in the query log is seen as a transaction in which a single user submits a sequence of related queries in a time interval.

Baeza-Yates *et al.* [2] study the problem of suggesting related queries issued by other users and query expansion methods to construct artificial queries. The clustering developed is based on a term-weight vector representation of queries, obtained from the aggregation of the term-weight vectors of the URLs clicked after the query: the objective is to recommend queries that are related to the input query but may search

for different aspects of it. Wen *et al.* [42] also present a clustering method for query recommendation that is centered around four notions of query distance: keywords of the query; string matching of keywords; common clicked URLs; and distance of the clicked documents in some pre-defined hierarchy. Jones *et al.* introduced the notion of query substitution [30]. Similar queries can be obtained by replacing the query as a whole, or by substituting constituent phrases. Both similar queries and phrases are derived from user query sessions, and they proposed models for query re-ranking based on the similarity between the new query and the original one.

Particularly relevant for this paper is the application of query log analysis to the segmentation of sessions into user missions, a.k.a. chains [36]: successful examples of such an application were presented in [29,9]. Even if most research on query logs focused on single sessions, recent works [37] suggest their usefulness also to determine long-term interests of users.

Donato *et al.* [17] present a machine learning module, based on query log analysis, which is at the basis of Search Pad, a novel Yahoo! application that was launched in June 2009. Search Pad helps users keeping trace of the queries they have done, and results they have consulted. These automatically collected notes can be edited by the user that can add comments, additional information, move or delete notes, and save the note pad for later reuse. The novelty of Search Pad is that unlike previous notes-taking products, it is automatically triggered only when the system decides, with a fair level of confidence, that the user is undertaking a"complex research mission" and thus is in the right context for gathering notes. A complex research mission is a search task that requires the user to go back to the search engine again and again over a period of time with related questions. Example of such tasks could be: organizing a holiday, deciding which digital camera to buy, finding a job or gathering information on a health issue. Once Search Pad receives the triggering signal and is aware that the user is engaged in a research session, it prompts the user with a message asking if she wants to take notes related to this search.

The information extracted from query logs can be summarized and suitably represented through query graphs [3], whose specific definition is geared on the application at hand. Some examples can be found in [20,15,9]. A recent application of query graphs to query-recommendation clustering is presented in [39], where a graph extracted from query logs is clustered to enhance the diversity in the set of query-refinement suggestions. The authors of [39] also model "off-topic drift" which corresponds to mission change in the nomenclature we adopt and to the terminal state in our query-flow graph.

2.3 Recommendations based on random walks

Craswell and Szummer [15] describe a method based on random walks on the query-click graph [6], that can be used to provide query recommendations as follows: given the input query, it computes the personalized PageRank [28] (with restart to the original query) of all the other queries, and then picks the top ones as recommendations. There are more details about this method in Section 7.2. Fuxman et al. [19] experiment with a similar approach in the context of finding related keywords for advertising.

Mei, Zhou and Church [34] instead use a computation of hitting time: assume that Q_0 is the input query: they start setting $h(Q_i, 0) = 0$ for all queries Q_i except for the original query Q_0 which has $h(Q_0, t) = 1\ \forall t \geq 0$, and then iterate the following for a

fixed number m of iterations:

$$h(Q_i, t) = \sum_{j \neq i} p_{ji} h(Q_j, t-1) \ ,$$

where p_{ji} is the probability of transition from Q_j to Q_i. For a query Q_i, what their process computes in $h(Q_i, t)$ is the probability that a random walk arrives to node Q_i within t steps or less.

Query recommendation systems can also be personalized by taking into account the user's history. Zhang and Nasraoui [44] bias recommendations exploiting user's history and introducing a "forgetting factor" which discounts older queries to favor more recent ones. A similar approach is used in [9] where a random walk with restart to the queries in the history of the user is done, preferring recent queries over older ones. As a general observation, recent works have shown that not only the previous query, but the long-term interests of users, are important for understanding his/her information need [33,37].

3 Query Transition Types

In this article we adopt a taxonomy of query transitions which is largely inspired by the similar classification in [38], with some differences that we summarize below. As depicted in Figure 1, there are basically two axes: a generalization-specialization axis, and a dissimilarity axis.

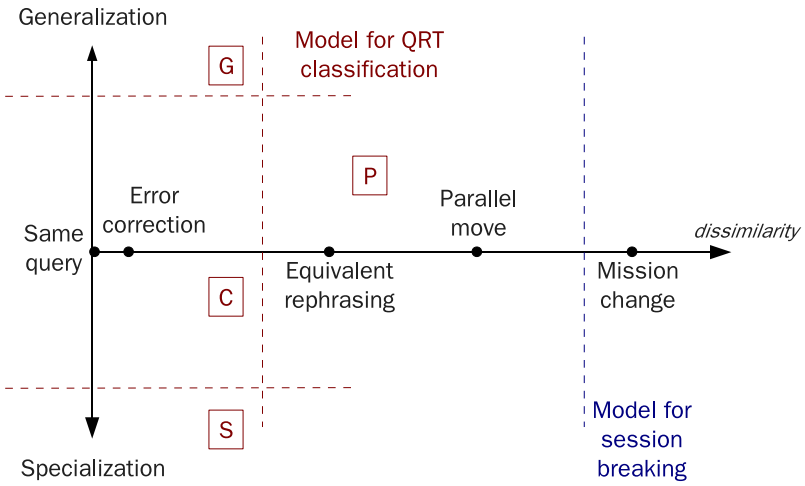


Fig. 1 Graphical depiction of transition types for pairs of consecutive queries. Transitions on the left of "Mission Change" are reformulations.

Along the dissimilarity axis (horizontal in Figure 1) we find a continuous variety of different types of query transition: as we move along the axis (from left to right, in the picture) the syntactic and semantic gap between the two queries, in terms of user's intent, gets larger and larger. We start with zero dissimilarity (*Same query*),

followed very closely by *Error correction*: the user is supposedly correcting an error (e.g., a typo) from her previous query, or trying a different spelling/capitalization of a query. Further along the dissimilarity axis we find *Equivalent rephrasing*: the user is re-phrasing, changing the wording of the query, but she has exactly the same goal (in the sense of [29]) as before: she just decided that the new formulation was more likely to return the results desired for. Then we find *Parallel move*: according to [38], this occurs when the "user modifies the queries from one aspect of an entity to another or from one thing to another, both of which share common characteristics"; the user is moving her focus to something related, but not equivalent—something that might have happened probably as a result of visiting some of the pages in the result set. Finally, we have *mission change*: the user is completely changing topic and she is looking for something else [36, 29].

Along the vertical axis instead we have *Generalization* and *Specialization*. Generalization occurs when the new query q' is more general than q (i.e., it should be satisfied by a superset of the results that are relevant for q'); in many cases (but not always) a generalization can be automatically identified because q' is a conjunction with a proper subset of the terms of q. There are other more difficult cases: for example, a user querying for the name of a specific soccer team and then querying to find a sports web site. In a specialization, instead, the new query q' is more specific than q (i.e., it should be satisfied by a subset of the results that are relevant for q); probably, the previous query returned too many results, few of them being of interest for the user. In a sense, generalization reflects the user's desire to increase recall, whereas specialization is the need to improve precision.

In our previous work [9], we developed a model for breaking sessions into chains or, in other terms, a model to detect mission changes. This model is represented in Figure 1 by the hyperplane separating *Mission change* from the rest. In this work we keep using that model for detecting mission changes, while we develop a new model for distinguishing QRTs. In particular on the dissimilarity axis we decided to cut in between what is a simple syntactical dissimilarity (we call this class `C` for *Correction*), and more substantial query reformulations which however remain in the same search mission (we call this class `P` from *Parallel move*). On the other axis we simply distinguish between class `G` (generalizations) and `S` (specializations). Some real examples for each kind of reformulation are shown in Table 1.

Table 1 Examples of reformulations

q	q'	QRT
cheapest phillips wacs7000	cheap stereos	`G`
sp tyres social club	sp tyres	`G`
cheapest phillips wacs7000	ebay	`G`
royal mail fdc albums	royal mail fdc albums spare	`S`
remortgage calculator	bbc remortgage calculator	`S`
foyles war screen caps	foyle's war screen caps	`C`
seaview riding school	ponies for sale	`P`
david murray actor	zonad film	`P`
videos koi carp fish farms..	videos koi carp ponds	`P`

Our classification of reformulations departs from the one proposed in [38] in the granularity used along the dissimilarity axis. Essentially they have the same three classes `G`, `S` and `C`, but instead of `P` they use a more fine-grained taxonomy—they

distinguish among parallel move, replacement with synonym, term variation, operator usage, type of resource and domain suffix.

Also the work in [26,27] presents a similar taxonomy, but they also consider *Content change* (when the current query is identical to the previous but executed on another content collection: e.g., web to images) and *Assistance* (the current query was generated by the user's selection of a query reformulation suggested by the search engine). Both scenarios are out of the scope of the present paper.

4 Automatic Classification

In this section we describe the process we followed in order to build a model for query-reformulation type classification.

4.1 The dataset construction

We started from a set of query pairs (q, q'), extracted from a query log of the Yahoo! UK search engine in early 2008. These query pairs were part of the training set that we used to build a model [9] for segmenting users sessions into *chains*, that is, topically coherent sequences of queries by one user. Every query pair (q, q') has the following two properties: (i) q and q' appeared in this order and consecutively at least once in the query log; (ii) q and q' belong to the same chain according to the labeling we did manually for the work in [9].

In order to create a training set for our QRT classification problem, a group of *editors* manually labelled the set of query pairs (q, q') with their QRT. It is worth noting that the same query reformulation (q, q') may be labelled by more than one editor: in cases of disagreement on the type of query reformulation by two or more editors, the query pair was removed by the training set. This left us with a set of 1 375 examples from which we used 2/3 for training and 1/3 for testing.

4.2 The features used

We used 27 features to build our model for QRT classification. The set of features is a superset of those used in our previous work [9], and some of them were shown to be effective for query segmentation also in other investigations [23,24,29]. The features are presented in Table 2, and can be divided into three groups:

- **Textual features.** We compute the textual similarity of queries q and q' using various similarity measures, including cosine similarity, Jaccard coefficient, and size of intersection. Those measures are computed on sets of stemmed terms and on character-level 3-grams. We also compute Levenshtein (edit) distance.
- **Session features.** We compute the number of sessions in which the pair (q, q') appears. We also compute other statistics of those sessions, such as, average session length, average position of the queries in the sessions, etc.
- **Time-related features.** We compute average time difference between q and q' in the sessions in which (q, q') appears, and the sum of reciprocals of time difference over all appearances of the pair (q, q').

Intuitively, the purpose of the session features and the time-related features, is to capture the relatedness of pairs of queries that appear *frequently* as reformulations in the query log. For instance, query pairs that appear with high frequency and with short time intervals between them, are expected to be more related. On the other hand, textual features are absolutely necessary for query pairs that appear *once*, which are the majority, and useful in general for query pairs that appear more than once, for instance to capture syntactic generalizations and specializations (which tend to respectively shorten and lengthen the query strings).

All features passed a features selection phase in which we evaluated each feature relevance w.r.t. our target variable (i.e., query reformulation type).

Discussion. There are at least two classes of features we are aware of that could have been useful to improve classification accuracy.

First, we refrained from using features that require access to extra information such as the resulting URLs or page snippets. For instance, we could have taken into account keywords in the documents returned by the search engine for each of these queries, or compute set intersection between the URLs returned for each query. Such features could in principle be helpful, as for instance generalization/specialization relationships should be reflected there as partial set inclusions. Although the latter data might be very powerful, and even decisive, to determine the query reformulation type, for efficiency reasons we wanted to limit ourselves to features that could be computed quickly without access to any extra information. In the particular case of an application such as query recommendations, search engines employ several techniques to reduce page loading time, including parallelizing some operations. Thus, for practical reasons we did not want to build a classifier that needed to wait for search results to be retrieved before being able of classifying an item.

Second, we used only features about the current query pair, and we did not consider features computed, for instance, from the previous query pair. More in general, we used a learning framework that classifies one pair of queries at a time, while for future work this could be modeled as a structured learning problem, in which the inputs and outputs are sequences of transitions. Learning schemes involving Hidden Markov Models or Conditional Random Fields could be promising for this task.

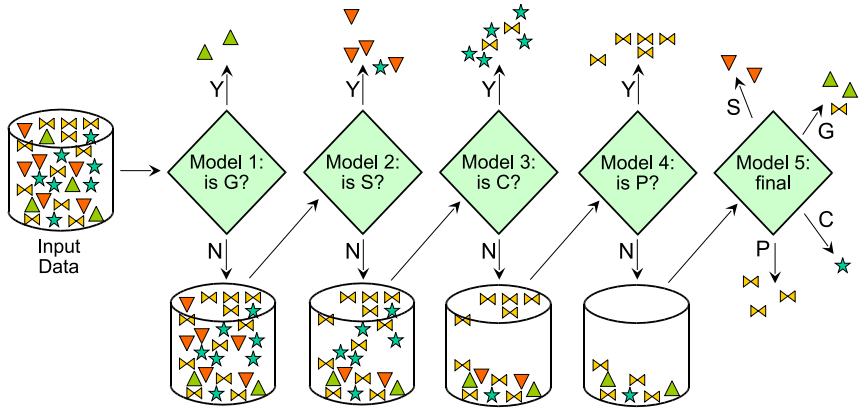
4.3 Building the model

We tried many different classifiers induction methods for our classification problem. Standard methods such as a boosted decision tree showed an accuracy of approximately 85% in predicting query reformulation types. The model that we finally selected exhibits an accuracy of 92% on a test set of unseen cases. In the following we describe how we obtained such a model.

Instead of directly tackling the 4-classes problem, we built four distinct binary classification problems, where in each problem the target variable is being or not a certain QRT (e.g., *is_G*?, *is_S*?, etc.). Then we attacked all the four problems concurrently and we built four different classifiers, one for each problem. Among the four classifiers built we choose the best performing one to be our first classifier. In particular, the selection was based not on *accuracy*, but on *precision* (i.e., the number of true positives divided by the total number of elements labeled as belonging to the class). The rationale for this is that at this stage we do not care much about *false negatives*: we just want to

Table 2 Description of the features extracted for each query reformulation (q, q').

Session-related features		
$f1$	*COUNT*	Number of sessions in which reformulation (q, q') occurs;
$f2$	*PROBABILITY_FORWARD*	$f1$ divided by number of sessions in which (q, x) occurs $(\forall x)$;
$f3$	*PROBABILITY_REVERSE*	$f1$ divided by number of sessions in which (x, q') occurs $(\forall x)$.
$f4$	*CLICKS_SINCE_SESSION_BEGIN*	Among all sessions containing (q, q'): average number of clicks since session begin,
$f5$	*CLICKS_SINCE_LAST_QUERY*	and since the query preceding this pair.
$f6$	*TOTAL_EVENTS_AVG*	Average session size;
$f7$	*QUERIES_SINCE_SESSION_BEGIN_AVG*	average position in session (i.e, number of queries before q),
$f8$	*FRACTION_QUERIES_SINCE_SESSION_BEGIN_AVG*	*QUERIES_SINCE_SESSION_BEGIN_AVG/ TOTAL_EVENTS_AVG*
$f9$	*IS_FIRST_QUERY_PAIR_FRACTION*	Fraction of occurrences in which (q, q') is the first pair in the session.
$f10$	*IS_LAST_QUERY_PAIR_FRACTION*	Fraction of occurrences in which (q, q') is the last pair in the session.
Temporal features		
$f11$	*TIME_INTEREVENT_AVG*	Average time elapsed between q and q' in each session in which both occur.
$f12$	*SUM_RECIPROCAL_TIME*	Sum of $1/t_i$ where t_i is the elapsed time between a query i and the previous event in the session.
Textual features		
$f13$	*EDITDISTANCE*	Levenshtein distance (a.k.a. edit distance).
$f14$ and $f15$	*LENGTH_1*, *LENGTH_2*	Length in characters of q and q'.
$f16$	*LENGTH_DIFF*	*LENGTH_2- LENGTH_1*,
$f17$	*LENGTH_DIFF_RATIO*	(*LENGTH_2- LENGTH_1*)/(*LENGTH_1*) .
$f18$	*TRIGRAMS_COSINE*	Each query is turned into a bag of character tri-grams: cosine similarity,
$f19$	*TRIGRAMS_JACCARD*	Jaccard coefficient,
$f20$	*TRIGRAMS_INTERSECTION*	size of the intersection between the two bags.
$f21$	*TERMS_COSINE*	Each query is turned into a bag of stemmed terms: cosine similarity,
$f22$	*TERMS_JACCARD*	Jaccard coefficient,
$f23$	*TERMS_INTERSECTION*	size of the intersection between the two bags.
$f24$ and $f25$	*TERMS_LENGTH_1*, *TERMS_LENGTH_2*	Number of terms in q and q'.
$f26$	*TERMS_LENGTH_DIFF*	*TERMS_LENGTH_DIFF* = *TERMS_LENGTH_2* − *TERMS_LENGTH_1*.
$f27$	*TERMS_LENGTH_DIFF_RATIO*	*TERMS_LENGTH_DIFF_RATIO* = (*TERMS_LENGTH_2* − *TERMS_LENGTH_1*)/*TERMS_LENGTH_1*



(a)

Rule 1 of model 1: *is_G?*	**Rule 1 of model 2:** *is_S?*
if *TERMS_COSINE* > 0.47 **and** *LENGTH_DIFF_RATIO* ≤ -0.37 **then** *is_G?* $= Y$	**if** *TRIGRAMS_COSINE* > 0.42 **and** *TERMS_LENGTH_DIFF* > 1 **then** *is_S?* $= Y$

Rule 1 of model 3: *is_C?*	**Rule 1 of model 4:** *is_P?*
if *QUERIES_SINCE_SESSION_BEGIN_AVG* ≤ 1.91 **and** *EDITDISTANCE* ≤ 3 **then** *is_C?* $= Y$	**if** *FRACTION_QUERIES_SINCE_SESSION_BEGIN_AVG* > 0.65 **and** *TERMS_JACCARD* ≤ 0.25 **and** *LENGTH_DIFF* ≤ 5 **and** *TERMS_LENGTH_DIFF* > 0 **then** *is_P?* $= Y$

(b)

Fig. 2 (a): high-level depiction of our QRT classification model. (b): the first rule (most representative) from each of the binary classifiers.

make some decisions with very high confidence and put those cases aside. False negatives do not represent a problem: they are not definitively errors, as they still have the chance to be classified correctly later. In fact the process continues greedily this way:

1. select the classifier (and the associated classification problem) that exhibits the highest precision;
2. remove from the training set the examples classified as positive form the selected classifier;
3. on the remaining examples train new models for the remaining classification problems and go to point 1;
4. when we have a model for each QRT, train the final 4-classes model on the remaining of the data.

Therefore false negative errors made by the first four classifiers can be saved by the last classifier. The whole process is depicted in Figure 2(a). In our case the first classifier

is the one for the target variable *is_G*?, then *is_S*?, then *is_C*? and finally *is_P*?. This order somehow represents also the easiness in distinguishing a class of QRT from the others: that is, class P is the hardest to be detected.

Another important thing to highlight is that as examples pass trough a classifier, not only the training set is reduced in number of examples, but it is also enriched in features. In fact with each example that is predicted as negative, the confidence is attached with which the classifier has done such a prediction. So the fifth classifier will actually receive in input 31 features: the 27 described in Table 2, plus the confidence with which all the four previous classifiers have predicted the example to be negative.

Each of the five models is a rule-based classifier built with C5.0, the successor of the well-known C4.5 decision tree induction algorithm [35]. While building the first four classifiers, in order to boost precision (i.e., achieving very low number of false positives, while paying in terms of recall) we used the possibility of defining different *misclassification costs* for different kind of errors: e.g., telling to the classifier induction algorithm to weight a false positive the double of a false negative. Finally, for the fifth model (the 4-classes one) we used *boosting* with 15 decision trees.

Reducing the multi-class scenario to binary classification is most usually solved with the so-called *one-against-all technique*, where many binary classifiers are used in parallel and the positive answer with highest accuracy is selected. This technique, however, is well-behaved when the underlying binary classification problems have all the same level of difficulty, which is far from being our case; our solution has the advantage of exploiting the lack of symmetry inherent in our problem to get rid of the easiest cases as soon as possible so to obtain better accuracy on the more difficult query reformulations.

4.4 Further insight in the model.

In Figure 2(b) we report the most representative rule (i.e., the one with highest precision) for each of the first four classifiers.

We can observe that the rule for generalization (`G`) asks for a high similarity of terms, and as expected it also requires the second query to be shorter than the first one, as forced by the negative value of *LENGTH_DIFF_RATIO*. The most representative rule for specialization (`S`) instead requires high similarity of n-grams and that the second query has at least one term more than the first query: thus the second query must be longer then the first one as intuitively expected, and the opposite of what happens with `G`.

The rule of the third model, for error correction (`C`) requires a small edit distance and that the query reformulation is generally close to the session begin. Finally, the most representative rule for parallel move (`P`) requires to appear late in the session and to have small similarity.

The fifth model is more complex to be inspected as it contains 15 classifiers each one made by several rules. It is worth highlighting that this model makes large use of the four additional features which are the confidence with which all the four previous classifiers have predicted an example to be negative.

For instance, the following is a rule that we can find in one of the 15 classifiers of the boosting model:

if *confidence*($is\_C? = N$) > 0.99
and *confidence*($is\_G? = N$) ≤ 0.94
and $PROBABILITY\_FORWARD > 0.5$
and $PROBABILITY\_REVERSE \leq 0.5$
then $is\_G? = Y$

The rule says that for a given example (q, q'), if the confidence with which the third model decided that it is not a generalization is not that high, while the confidence of not being an error correction is very high, and if more than half of the times q appears in the query log is followed by q', while less than half of the times q' appears in the query log, it is preceded by q, than (q, q') is a generalization. This example also shows how false negative errors of the first four classifiers may be "corrected" by the fifth classifier.

Our model is able to achieve a high accuracy also thanks to some very difficult prediction that it is able to do correctly. In Table 3 we report some of these difficult predictions.

Table 3 Some example of difficult cases predicted correctly by our classifier

q	q'	QRT
`dango`	`japanese cakes`	G
`cars for sale south hams`	`auto trader`	G
`Find samebody in Germany`	`Find my friend in berlin`	S
`Nutrition`	`Vegetarian Society`	S
`ikea`	`corner vanity units`	S
`sport`	`PSV Eindhoven v Tottenham`	S

Consider the example on the first row: our classifier is able to correctly determine that the reformulation from query `dango` to query `japanese cakes` is a generalization. Another nice example can be found in the last row where the query `sport` is specialized into `PSV Eindhoven v Tottenham`: also in this case the guess was not straightforward due to the lack of textual similarity.

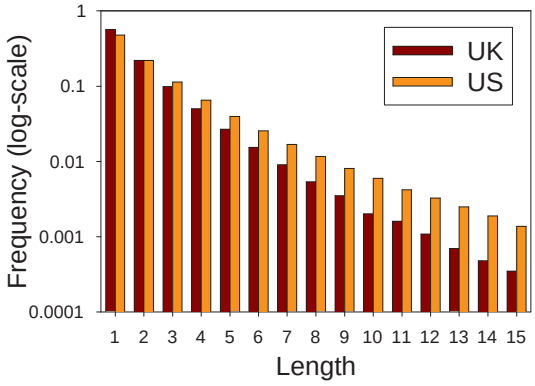
5 Query Reformulation Patterns

Using our model we can automatically label query transitions in very large query logs to analyze typical patterns. In this section we report some results of this analysis.

5.1 Datasets

We used two datasets from Yahoo!'s in-house query logs. The first one corresponds to the UK dataset from which the training data were extracted in the previous part. The second one corresponds to a completely different dataset from searches in the Yahoo! US search engine in early 2008. Single-session queries are not considered in these data.

We first segmented all user activity into *chains* through the model we developed in [9]. Then, we extracted from each query log the features listed in Table 2 and labeled



(a)

	UK	US
G	4.4%	9.5%
S	37.5%	30.1%
C	10.4%	5.0%
P	47.7%	55.5%
	$n = 6M$	$n = 10M$

(b)

	US≥ 5	Rie & Xie[38]
G	11.0%	12.7%
S	26.5%	23.4%
C	4.0%	5.2%
P	58.5%	56.2%
	$n = 4M$	$n = 2K$

(c)

Fig. 3 (a) distribution of chain length in the two datasets, without counting the special symbol X. (b) and (c) QRT distributions.

each query reformulation in each chain with the model we described in the previous section.

The classification of query reformulations transforms each chain into a string of QRTs, and the query log is transformed into a bag of strings. Each string is started and ended by a special character X, representing the border of a search mission. Thus our query log looks like: {XPSX, XPSPX, XCCCPPX, XSSPX, XPPSPPPPPX, XPSGPSSSX, XPPPPSX, XSGSX, XSX, XSPPCX,...} (given that single-queries are not present, the string XX does not occur in the data).

The UK dataset contains 3 376 775 chains for a total of 6 578 275 QRTs without considering mission changes. The US dataset contains 4 087 898 chains for a total of 10 496 317 QRTs. We remark that the size of the dataset we analyze is much larger than those reported in the literature for this problem: for instance in [38] analyze 313 chains, all containing at least 6 queries (i.e., 5 query reformulations), for a total of 2 109 QRTs, while Jansen *et. al* [26,27] analyze approximately 1.5 millions of query reformulations. Even if we focused only on chains of length at least 5, we would still have 222 727 chains for a total of 1 529 539 QRTs in the UK dataset, and 527 420 chains containing 4 316 676 QRTs in the US dataset. In the following we denote "**UK**≥ 5" and "**US**≥ 5" the two datasets when we only consider long chains.

5.2 Query reformulation distribution

In Figure 3(a) we report the distribution of chain length on the two datasets (without counting the special symbol X), while in Figure 3(b) we report the distribution of reformulation types. In Figure 3(c) we show the distribution of reformulation types from the work of Rie and Xie [38] (merging in the class P the different categories that they consider: parallel move, replacement with synonym, term variation etc.), and we compare it with the US dataset limited to chains of length 5 or more (to mimic what Rie and Xie do on their own data). The reader can appreciate a substantial agreement between the findings obtained here and in [38].

Table 4 Ratio of the conditional probability $P(Current = a|Prev = b)$ with respect to the prior probability $P(Current = a)$. Deviations of more than 50% (i.e., a ratio ≤ 0.5 or ≥ 1.5) are shown in boldface.

	UK dataset					**US dataset**				
	Previous					Previous				
Current	G	S	C	P	X	G	S	C	P	X
G	0.8	**1.7**	**0.3**	**0.4**	1.2	0.6	**2.0**	0.6	0.6	0.9
S	1.3	0.7	**0.5**	0.7	**1.6**	1.4	0.6	0.6	0.7	**1.6**
C	**0.3**	**0.4**	1.2	0.6	**1.8**	**0.5**	**0.5**	**4.0**	0.7	**1.6**
P	**0.5**	0.9	0.6	0.8	1.4	0.6	0.8	0.7	1 .0	1.3
X	1.4	1.4	**1.7**	**1.5**	**0.0**	1.3	1.4	**1.5**	1.4	**0.0**

As reported by Rie and Xie [38], the class P is largely the most populated (47%-58%). It is worth noting that this is slightly overestimated, as it is partially due to some *false negative* errors of the model used to segment sessions into chains [9]. In fact, we have observed that mission changes that are not detected as such by that first model are typically recognized as P by the model for QRT classification. This is quite natural if we think that parallel move is the class that is semantically closer to mission change, as depicted in Figure 1.

The widespread presence of P would call for a more fine-grained categorization of this kind of reformulations, like the one adopted by [38]; to distinguish between "real" parallel moves (in the sense of Rie and Xie) and other kinds of reformulations, it would be probably helpful to know if the user clicked on at least one result before reformulating the query or not. This would be a departure from our decision of considering only information that can be directly deduced from the queries themselves (either from their textual content, or from their temporal position in the user's query flow): therefore we decided not to pursue this path any further, leaving this kind of fine-grained analysis as an object for future work.

On the generalization-specialization axis, as expected, specializations (30%-38%) are much more frequent than generalizations (4%-10%). This difference is however largely reduced when focusing on chains of length 5 or more, as reported in Figure 3(c).

5.3 Conditional reformulation probability

For deeper inspection in Table 4 we report conditional probabilities depending on the previous QRT, that is,

$$P(Current = a|Prev = b).$$

From this table we can make some important observations: (*i*) generalizations probability is boosted after a specialization; (*ii*) specializations are very likely to occur at the beginning of a chain, or after a generalization; (*iii*) error corrections are common at the beginning or end of a chain, or after another error correction. What is interesting is that all the above observations are confirmed on both datasets.

5.4 Interesting frequent reformulation pattern

We also counted the frequency of patterns (i.e., substrings of any length) in the datasets. Frequency of a pattern is defined not as the total number of occurrences, but as the

Table 5 Interesting patterns.

Pattern	Frequency			
	UK	**US**	**UK**≥ 5	**US**≥ 5
XC	12.7%	5.6%	7.8%	4.5%
SG	2.8%	7.6%	16.4%	30.6%
GS	2.5%	6.1%	17.7%	30.3%
CX	11.3%	4.6%	6.1%	3.1%
XS	38.2%	35.5%	44.5%	34.5%
CC	1.4%	1.3%	5.1%	4.8%
SGS	0.9%	2.5%	8.6%	14.6%
CCC	0.3%	0.2%	1.5%	1.4%
GSG	0.2%	1.0%	2.5%	7.1%
SSG	0.7%	1.8%	7.6%	10.9%
XSG	1.7%	4.0%	4.1%	6.9%
SGX	1.3%	3.1%	2.2%	4.8%

number of strings in the database that contain the given pattern. We selected some patterns by means of an *interestingness* measure defined as the ratio between the real frequency, and the *expected* frequency which is computed assuming independence of QRTs. Table 5 lists a few of the interesting patterns we found; they confirm and complement the findings in Table 4: error corrections are more frequent at the beginning of a chain (XC), they also tend to appear contiguously (CC, CCC, ...), and sequences of alternating specialization-generalization are more frequent than expected (SG, GS, ...).

5.5 Topic Patterns

In this section we report a preliminary experiment that we conducted in order to check how query reformulations and mission changes relate to query *topics*. In principle, belonging to the same mission is not the same as belonging to the same topic. For instance, a person looking for information about a country may start by looking at governmental sites, then look for information about art and culture, then check economic indicators, etc. Queries in the same mission may belong to different topics. Also queries in the same broad topic may be part of different missions.

Query topical classification. There are many approaches to topical query classification, e.g. [32]. In this experiment we issued[1] each query to the Yahoo! search engine, obtained the top 20 documents, and used an in-house automatic document classifier to obtain the most likely Yahoo! directory (dir.yahoo.com) topic for each document returned. Next we did a majority voting among the topics of the documents associated to the query to determine the query topic. To increase precision at the expense of coverage, if the main topic was not at least twice as prevalent as the second topic we considered the query topic as "unknown". This is a slow yet very simple query classification method that is nevertheless quite precise. We used it to classify by topic 100K queries from the UK data and 100K queries from the US data.

Results. For each query transition, we compared the top-level topic of the two queries involved in the transition: this is usually something very broad such as "science →

[1] We used programmatic access to the search engine that bypasses automatic error correction of misspellings.

health", etc. If the two topics coincide, we count this as a top-level topic match in Table 6. As before we denote mission changes with the transition type `X`.

Table 6 Fraction of transitions where the top-level topic remains the same, and example salient topic pairs, on both datasets.

QRT	Topic match		Salient top-level topic pairs
G	UK	64%	reference→reference government→government
	US	64%	reference→government reference→reference
S	UK	59%	reference→reference government→ government
	US	71%	reference→reference government→ government
C	UK	54%	reference→computers and internet news and media→news and media
	US	53%	reference→health science→social science
P	UK	46%	arts→reference reference→government
	US	48%	reference→education social science→government
X	UK	22%	computers and internet→recreation entertainment→education
	US	23%	recreation→health soc. and culture→computers and internet

From a user's perspective, we can see that whenever our classifier detects a mission change, the user is more likely to change the broad topic than to stay in the same broad topic. The opposite occurs in the case of generalization, specializations, and error corrections, in which the user is more likely to stay in the same broad topic. As expected, parallel moves are more ambiguous from the perspective of broad topics.

Next, we verified if some broad topics are more likely to motivate certain transition types than others. Table 6 shows some top-level topic pairs with the highest ratio of their probability conditioned to each transition type with respect to their prior probability.

For generalization (`G`) and specialization (`S`), it is frequent to observe pairs of queries that are both reference search (dictionary/encyclopedia) or searching for some government-related topics. In the case of parallel moves (`P`), switches to and from reference search are common. As for mission change (`X`), we observe an interesting fact: there are frequent changes from and to recreation/entertainment topics which may signal alternating between work/study related activities and leisure.

6 Transition Graph

The *query-flow graph*, that we introduced in [9], is an aggregated representation of the interesting knowledge about latent querying behavior which is contained in a query log. It is a directed graph, where nodes are queries, and there exists an edge between

queries (q, q') if the two queries appear consecutively in some session in the query log. Moreover edges may hold application-dependent information of various types.

In this section we report the results of an investigation on the query-flow graph, where the edges have been annotated with transition types (obtained with our classification model presented in Section 4) and counts (number of times the query pair was observed in the log). Figure 4 shows a small sub-graph of the query-flow graph with edges labeled with QRT from the UK dataset. In the following we refer to the query-flow graph with edges labeled with QRT simply as *transition graph*.

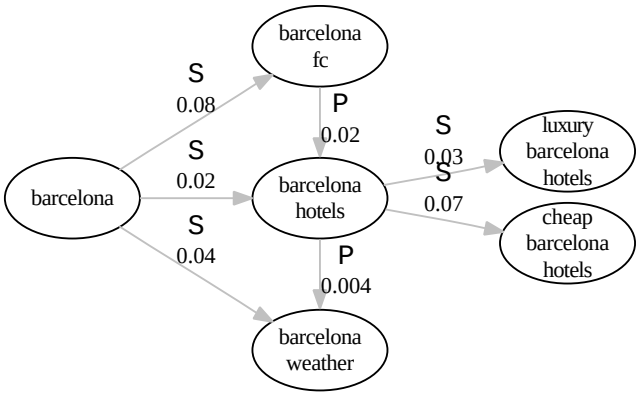


Fig. 4 Example of some reformulations around the query "`barcelona hotels`" extracted from the UK dataset. The feature *PROBABILITY_FORWARD* is also included in the figure.

The investigation that we report in this section has a twofold aim: on one hand, we would like to have at least some indirect proof that our classification does not contain major inconsistencies, and while doing this we will also be able to understand which parts of the process are probably more error-prone. On the other hand, through this inspection we will gain some deeper insight in the QRT classification task itself.

We used entire sessions to build the graph, not only missions, so mission changes are also included as transitions. For the UK dataset, we used all transitions to construct the graph, whereas for the US dataset, we discarded all hapax transitions (those that only appear once). The resulting transition graphs have the following sizes:

- UK dataset: 21 247 414 nodes, 21 216 958 arcs (0.99 arcs/node);
- US dataset: 58 312 610 nodes, 53 960 925 arcs (0.93 arcs/node).

Most properties were studied by filtering the transition graph according to the transition type; this way, each transition graph gave rise to five "slices" of the graph, one for each transition type.

Table 7 Basic properties of the transition graphs

		Gen.	Spec.	Corr.	Par.	X
Density (arcs per node)	UK	0.04	0.31	0.07	0.41	0.17
	US	0.06	0.26	0.05	0.25	0.39
Size of largest strongly	UK	0.00%	0.26%	0.14%	2.51%	1.10%
connected component	US	0.00%	0.07%	0.20%	2.41%	1.45%
Number of nontrivial nodes	UK	0.37%	4.31%	1.80%	12.26%	3.20%
	US	1.49%	1.20%	1.52%	3.38%	1.34%
Average weighted reciprocity%	UK	0.0%	0.2%	1.7%	1.6%	3.1%
$\rho(q, q', -)$	US	0.0%	0.8%	12.1%	14.8%	26.3%

6.1 Overall properties

Table 7 presents some data about the overall structure of the transition graphs; notice that the majority transitions are either parallel moves (P), or correspond to mission change (X): this is a consequence of the fact that the majority of chains are very short[2].

Most of the remaining transitions are specializations (a concrete evidence that users of search engines reformulate their queries mostly seeking to improve precision, whereas recall is usually not an issue), immediately followed by parallel moves. Generalizations are rare, and so are error-corrections; the latter datum, though, largely depends on the fact that the engine itself performs some error correction, so the user rarely needs to actually correct the query.

Table 7 also presents an analysis of strongly connected components[3], showing that all graphs are extremely sparse and essentially acyclic. If we delete from the graph all isolated nodes and isolated arcs (an arc (q, q') is isolated iff q has outdegree 1 and q' has outdegree 0), the number of remaining nodes (called "nontrivial" in Table 7) is extremely small.

6.2 Anti-symmetry and correlations

Some of the transition types should exhibit some natural properties; for example, both G and S are conceptually partial orders, so they should be transitive and anti-symmetric. Of course, we cannot expect these properties to hold deterministically, both because of the presence of noise and because we should take into account the frequency of each observed transition.

We measure symmetry using a **weighted reciprocity**. This metric takes a value close to 0 if an arc in one direction has a much smaller or larger weight than the arc in the opposite direction, and a value close to 1 if both arcs have similar weights. We define the weighted reciprocity as follows: let $c(q, q', t)$ be the count associated to arc (q, q') in a given graph t (in our setting this corresponds to a graph containing only transitions of type t), or zero if (q, q') is not an arc in t, and define

$$\rho(q, q', t) = \min(c(q, q', t), c(q', q, t)) / \max(c(q, q', t), c(q', q, t)).$$

[2] An even larger fraction of mission changes would be observed if we considered also single-query physical sessions.

[3] A strongly connected component in this graph is a maximal subset of queries such that any two queries are connected by directed reformulation paths (in both directions).

Table 8 Jaccard coefficients (per mille) between the set of arcs of each graph and the transpose of each graph

UK dataset

	$\texttt{G}^T$	$\texttt{S}^T$	$\texttt{C}^T$	$\texttt{P}^T$	$\texttt{X}^T$
G	0.0	**6.3**	0.0	0.1	0.0
S	**6.3**	2.0	0.4	3.2	1.7
C	0.0	0.4	**13.1**	1.2	0.9
P	0.1	3.2	1.2	**10.1**	6.7
X	0.0	1.7	0.9	6.7	**21.6**

US dataset

	$\texttt{G}^T$	$\texttt{S}^T$	$\texttt{C}^T$	$\texttt{P}^T$	$\texttt{X}^T$
G	0.0	**124.7**	0.7	2.1	0.4
S	**124.7**	9.7	2.2	23.6	1.7
C	0.7	2.2	**117.9**	3.1	1.0
P	2.1	23.6	3.1	**128.5**	26.4
X	0.4	1.7	1.0	26.4	**236.6**

In the ideal case, if t defines a perfectly anti-symmetric relation this quantity should be 0 for all arcs in t, whereas it should be 1 for perfectly symmetric relations.

The average $\rho(q, q', -)$ for all arcs (q, q') is shown in Table 7: notice that the values are all very small, due to the sparsity of all graphs, but they are significantly closer to zero (or even exactly zero) for G and S, whereas they are significantly larger for the other transition types.

Another measure of symmetry can be obtained disregarding the counts, and simply measuring the Jaccard coefficient between the set of arcs of each transition graph and its transpose (i.e., the graph obtained transposing every arc): again, in the absence of noise this measure should ideally be 0 for asymmetric relations, and 1 for symmetric relations. This measure, although less fine-grained than the previous because it does not take frequency into account, can be used also to compare different graphs. Table 8 reports the results for every transition graph and every transpose (for the sake of readability, we highlighted the largest entry in every row/column): as before, all values are small, but the reader can verify that the largest values are found on the diagonal for C, P, and X (witnessing that they are somehow symmetric), whereas for G and S we have the largest values when each is compared with the transpose of the other.

Indeed, in the absence of classification errors, S and G should converge to be mutually transpose as the number of observations grows. Every specialization reformulation of one user can be done, in the opposite direction, as a generalization reformulation, and viceversa.

6.3 Entropy of query reformulations

The purpose of this experiment is to measure to which extent the reformulation type is determined by the query. We defined the **reformulation-type entropy** of a query as the entropy of the distribution with probabilities

$$p_q(t) = \sum_{q'} c(q, q', t) / \sum_{q', t} c(q, q', t) \quad ,$$

where as before $c(q, q', t)$ is the count of reformulations from q to q' having reformulation type t. Here we ignore the transition type X. To consider only queries for which we have

Table 9 Entropy measures

	UK data	US data
Reformulation-type entropy	1.1	1.0
Next-query entropy:		
Generalization (`G`)	1.0	1.3
Specialization (`S`)	5.4	2.6
Correction (`C`)	1.1	1.3
Parallel move (`P`)	6.5	4.0

enough information, we averaged the entropy over all queries q having $\sum c(q, q', t) \geq 100$.

An average value close to 0 would mean that the query determines almost completely the reformulation type (for instance, that certain queries almost always are followed by a correction, while other queries almost always are followed by a parallel move, and so on). An average value close to 2 (there are four categories here: `G`, `S`, `C`, `P`) would mean that any reformulation type is possible. Indeed this value is close to 1, as shown in Table 9, meaning that when writing a reformulation for a query, the user will decide mostly between two reformulation types on average.

Next we measured to which extent a certain reformulation type is more predictable than another reformulation type. For instance, if a given query is followed by an error correction, we would expect that the particular error correction chosen is more determined by the query than if the user were doing a reformulation of type "parallel move" where there is a broader range of choices.

To measure this we examined the **next-query entropy** for a query q and a reformulation type t, this is the entropy of the distribution with probabilities

$$p_{t,q}(q') = c(q, q', t) / \sum_i c(q, i, t) \ .$$

We averaged this over the same queries as with the reformulation-type entropy. The results are shown in Table 9. The next-query entropy is small for generalizations and error corrections, but closer to 1 than to 0, meaning that there is still some variability when the user decides to use this type of reformulation. The next-query entropy for specialization and parallel moves is substantially higher, from 3 to 6 bits, meaning that the users pick between several choices on average (the entropy may be lower in our US graph probably due to the removal of pairs with count equal to one).

7 Query Recommendation

In this section we demonstrate that the automatic QRT classifier can be applied to a key task for search engines: the generation of query suggestions. This section extends results presented in [10].

7.1 Experimental framework

Our experiments for query recommendation are based on the "Spring 2006 Data Asset" distributed by Microsoft Research[4]. The data consists of a query log excerpt with 15

[4] `http://research.microsoft.com/users/nickcr/wscd09/`

million queries, most of them in English, sampled over one month and including a query and query-id, an anonymous session-id, a timestamp, and the results (for each result, the position on the result page and a timestamp is also provided). Part of the adult queries was extracted and provided separately: we did not use them in our experiments, though.

We encoded the data using the WebGraph framework [12] (the framework has been originally built to represent web graphs, but it turns out to be useful to represent succinctly large graphs in general) and also the high-performance hashing classes from the Sux4J project [7].

For creating the Query Flow Graph, we used the model that we trained on a different dataset—a set of query pairs (q, q'), extracted from a query log of the Yahoo! UK search engine in early 2008. These query pairs were first used to build a model [9] for segmenting users sessions into *chains*, that is, topically coherent sequences of queries by one user.

The query recommendation methods are based on the probability of being at a certain node after performing a random walk over a query graph. This random walk starts in the node corresponding to the input query. At each step, the random walker either remains in the same node with probability 0.9, or follows one of the out-links with probability equal to 0.1; in the latter case, the links are followed proportionally to $w(i, j)$. The weights $w(i, j)$ can be arbitrary and are used to bias the random walk towards highly-relevant items, we describe several concrete weighting schemes below. For the random walk, we either do a single step, or repeat this for 5 or 10 iterations.[5]

We compare two different scoring methods. In the first case the queries to present to the user are chosen based on the personalized PageRank values obtained by the random walk described above: this is the "absolute" scoring method in Tables 13 and 14. An alternative scoring method ranks the results based on the ratio between the values obtained in the previous case and the PageRank values obtained by using no personalization (i.e., restarting at a random node), setting the random jump value to 0.15 and letting the algorithm run until convergence: this is referred to as the "relative" scoring method in the same tables.

7.2 Baseline for query recommendation

For comparison, we also implemented a query-recommendation system based on the method by Crasswell and Szummer [15], which uses a bipartite query-document graph. This query-document graph is defined as $G' = (Q \cup D, E')$, $E' \subseteq Q \times D$ with Q the set of documents and D the set of pages. The edges are symmetric, $(i, j) \in E' \Rightarrow (j, i) \in E'$. Let $c' : E' \to \mathbb{N}$ be the number of clicks with $c'(i, j) = c'(j, i)$ describing the number of clicks obtained by document j when shown as a result of query i.

Although there are several alternatives for the transition probabilities, we used the two different weighting schemes described in [15]. The "forward' weighting scheme corresponds to following edges proportionally to the number of clicks associated to them, using weights

$$w_f(i, j) = \frac{c'(i, j)}{\sum_{k:(i,k)\in E'} c'(i, k)} \ .$$

[5] We observed that performing 10 iterations or more does not improve the results and we omit those results. We also tested a "random jump" probability of 0.1 or 0.2 which actually worsened the results so we did not include it in the analysis of results.

The "backwards" weighting scheme uses different weights

$$w_b(i,j) = \frac{w_f(j,i)}{\sum_{k:(j,k)\in E'} w_f(j,k)} \ .$$

In the paper introducing these weights, they observe that the "backwards" weighting scheme provides better results than the "forward" weighting scheme for their task of finding relevant images for an input query. In our experimental results we observe the same, with an even greater advantage for the "backwards" weighting scheme as will be presented below.

For generating the recommendation we proceed as above, except that we used 6 or 12 iterations to do an even number of steps and end the random walk in a query and not in a document.[6]

7.3 Assessment method

The evaluation of the recommendations produced by the different systems was done in the following way. A set of 114 input queries having frequencies between 700 and 15 000 was selected at random; we used these frequencies limit to avoid very frequent queries (which are often navigational and for which query recommendations are not useful) or very infrequent queries (for which in this dataset there will be no recommendations). Queries were very varied in nature, e.g., "`grey's anatomy`", "`juno`", "`Maggie Gyllenhaal`', "`cnn news`", and "`guitar tabs`". We discarded all the queries containing a domain name.

Next, we generated the top 5 recommendations for each query using each system, and pooled the results together; this yielded on average 53.4 different recommendations per system. Then, a group of 5 assessors entered a simple assessment interface where each assessor was presented a random query and then in sequence all the different recommendations for that query in random order, without knowing which system(s) produced the recommendation.

The assessor was also able to see the search engine results for the original query and the recommended query that was being evaluated. The assessor was asked if the recommendation was **useful**, **somewhat useful** or **not useful**, considering the original query. A very broad instruction was given: a useful recommendation is a query such that, if the user submits it to the search engine, it provides new results that were not available using the original query, and that agree with the inferred user intent of the original query. Of course there is a great deal of subjectivity in this assessment as the original intent is not known for sure by the assessor.

Table 10 Example assessments for query "`cnn news`"

Useful	Somewhat useful	Not useful
cnn world news	abc7chicagonews	CNN
msnbc news	nba scores	cnn.com
fox news	cnnfyi	verizon netmail

[6] We also did experiments with 24 iterations that did not yield improvements over 12 iterations and are omitted in the experimental section.

Table 10 shows a sample assessment for the input query "`cnn news`". In practice, recommendations that are considered useful are typically either specializations of parallel moves in the sense of [38], while recommendations that are considered not useful tend to be either trivial variants of the original query, or completely unrelated queries.

In total, we received 6 093 assessments distributed as per Table 11.

Table 11 Distribution of assessments, $n = 6,093$

Assessment	Probability
Useful	25.1%
Somewhat useful	11.6%
Not useful	62.1%
Can not assess	1.2%

The assessment task was described as difficult by the assessors. We measured inter-assessor agreement on 560 overlapping query-recommendation pairs that were judged by two different assessors. We considered three scenarios: A. each label is a different category; B. labels "somewhat useful" and "not useful" are together in a category; C. labels "useful" and "somewhat useful" are together in a category. Next we measured the observed agreement P_a and Cohen's Kappa statistic which compares the agreement expected by chance P_c with the observed agreement using the formula $\kappa = \frac{P_a - P_c}{1 - P_c}$.

Table 12 Inter-assessor agreement as a probability P_a and in terms of Cohen's Kappa κ, $n = 560$

	Scenario	P_a	κ
A.	Useful vs Sw.useful vs Not useful	68%	0.43
B.	Useful vs (Sw.useful or Not useful)	86%	0.46
C.	(Useful or Sw.useful) vs Not useful	77%	0.59

As shown in Table 12, the scenario C. is the best of the three and shows a moderate amount of agreement between the assessors ($\kappa = 0.59$). The relatively small level of agreement can be compared with other similarly subjective web evaluation tasks such as $\kappa = 0.85$ for web page type classification [22], $\kappa = 0.72$ for query type classification [43], $\kappa = 0.61$ for link type classification [22], $\kappa = 0.63$ for web spam classification [14], etc.

7.4 Results

Usefulness score. The *Uscore* column in Table 13 is the probability that a recommendation issued by a system is labeled as "useful" or "somewhat useful", in accordance to the scenario that maximizes the inter-assessor agreement as explained above. The column concerning significance (p-value, omitted when over 0.1) contains the probability of observing a score of *Uscore* or less by chance, assuming that all the systems have the same accuracy as the top one.

Small differences in p-value for systems having the same *Uscore* depend on the fact that the significance is computed considering the number of valid assessments for each system among the 114 queries evaluated, excluding the "Can not assess" label in Table 11. Lines are drawn in the table at $p = 0.1, 0.05, 0.01$. Notice that we

Table 13 Usefulness score for each system: probability that a recommendation issued by the system is useful or somewhat useful

Uscore	p-value	System	Iter.	Scoring
0.58		Queryflow-S	10	Abs.
0.58		Queryflow-S	5	Abs.
0.57		Queryflow-SP	1	Abs.
0.56		Queryflow-SP	10	Abs.
0.56		Queryflow-SP	5	Abs.
0.55	0.10	Queryflow-SPC	1	Abs.
0.55	0.06	Queryflow-GSPC	1	Abs.
0.55	0.06	Queryflow-S	1	Abs.
0.55	0.07	Queryflow-SPC	5	Abs.
0.55	0.06	Queryflow-SPC	10	Abs.
0.55	0.07	QueryDocument-Bwd	6	Rel.
0.55	0.10	QueryDocument-Bwd	24	Rel.
0.55	0.06	QueryDocument-Bwd	12	Rel.
0.54	0.03	Queryflow-S	10	Rel.
0.54	0.02	Queryflow-SC	5	Abs.
0.54	0.02	Queryflow-S	5	Rel.
0.54	0.02	QueryDocument-Bwd	2	Rel.
0.54	0.04	QueryDocument-Bwd	12	Abs.
0.54	0.02	QueryDocument-Bwd	6	Abs.
0.54	0.03	QueryDocument-Bwd	24	Abs.
0.53	0.01	Queryflow	1	Abs.
0.53	0.01	Queryflow-GSPC	5	Abs.
0.53	0.01	Queryflow-GSPC	10	Abs.
0.53	0.01	Queryflow-SC	10	Abs.
0.52	$< .01$	Queryflow	5	Abs.
0.52	$< .01$	QueryDocument-Bwd	2	Abs.
0.52	$< .01$	Queryflow-SC	1	Abs.
0.52	$< .01$	Queryflow-SC	1	Rel.
0.51	$< .01$	Queryflow	10	Abs.
0.51	$< .01$	Queryflow-SC	10	Rel.
0.51	$< .01$	Queryflow-SC	5	Rel.
0.47	$< .01$	Queryflow-SP	1	Rel.
0.47	$< .01$	Queryflow-SP	10	Rel.
0.47	$< .01$	Queryflow-SP	5	Rel.
0.45	$< .01$	Queryflow-SPC	10	Rel.
0.45	$< .01$	Queryflow-SPC	1	Rel.
0.45	$< .01$	Queryflow-SPC	5	Rel.
0.44	$< .01$	Queryflow	10	Rel.
0.44	$< .01$	Queryflow	1	Rel.
0.44	$< .01$	Queryflow	5	Rel.
0.44	$< .01$	Queryflow-GSPC	10	Rel.
0.43	$< .01$	Queryflow-GSPC	1	Rel.
0.43	$< .01$	Queryflow-GSPC	5	Rel.
0.39	$< .01$	Queryflow-(S^2)	1	Rel.
0.39	$< .01$	Queryflow-(S^2)	10	Rel.
0.39	$< .01$	Queryflow-(S^2)	1	Abs.
0.38	$< .01$	Queryflow-(S^2)	10	Abs.
0.32	$< .01$	QueryDocument-Fwd	24	Abs.
0.32	$< .01$	QueryDocument-Fwd	12	Abs.
0.32	$< .01$	QueryDocument-Fwd	6	Abs.
0.30	$< .01$	QueryDocument-Fwd	2	Abs.
0.29	$< .01$	QueryDocument-Fwd	24	Rel.
0.28	$< .01$	Queryflow-(SG)	10	Rel.
0.28	$< .01$	QueryDocument-Fwd	6	Rel.
0.28	$< .01$	QueryDocument-Fwd	12	Rel.
0.28	$< .01$	Queryflow-(SG)	10	Abs.
0.27	$< .01$	QueryDocument-Fwd	2	Rel.
0.23	$< .01$	Queryflow-(SS^T)	10	Abs.
0.23	$< .01$	Queryflow-(SS^T)	10	Rel.

are here testing our systems against a very strong null hypothesis, because only the top 5 recommendations are being considered, and many of them are correct; so the probability of guessing *among them* is very high, even at random.

In the recommendations generated using the query-flow graph, the score decreases as we introduce more transition types: specialization transitions seem to produce the most useful recommendations (Queryflow-S), whereas adding parallel moves (Queryflow-SP), corrections (Queryflow-SPC), and eventually generalization (Queryflow-GSPC, different at $p = 0.06$) results in less useful recommendations.

The "absolute" scoring method works better than the "relative" scoring method for the queryflow-based recommendations at a significance of $p = 0.04$, and doing multiple iterations instead of only one (which corresponds to taking the maximum) is better at $p = 0.06$.

We also added a system named just "Queryflow" in Table 13, without including any slice name: in this system the weights are computed over all transitions, independently of whether they were part of the same mission or not. This is worse than the systems that selects only specializations and counts only over transitions in the same mission at $p = 0.01$.

The recommendations based on the baseline (query-document graph) have either the same performance as recommendations using Queryflow-S, or a lower performance at a significance of $p = 0.07$. In this case, the "backwards" weighting scheme performs much better than the "forwards" weighting scheme at $p < 0.01$. This was already noticed in [15]: the gap, in our case, is even larger. Finally, Figure 5 is a chart of the best performing variant of each system.

Diversity score. Next we computed a measure of diversity in the resulting set. This is done by taking each sampled query, and each recommendation labeled as useful or somewhat useful, and issuing that recommended query to a search engine. Given that we are taking the top-5 recommendations per system, this generates a maximum of 25 URLs. The average *Dscore* in Table 14 is the average number of distinct URLs in this multiset across the 114 queries evaluated which were not present in the result set for the original query.

Significance is computed using the individual score (0 to 25) obtained by each system for each of the 114 assessed queries; we assume scores have a normal distribution and compute the probability of observing the scores we get or less, assuming that all systems have the same performance as the top system (using a one-sided t-test). Lines are drawn in the table at $p = 0.1, 0.05, 0.01$. We observe a change in the relative position of different systems in the top half of the table with respect to Table 13, indicating that this measure is different from the measure based purely on the labels associated to the recommended queries.

8 Conclusions

This section synthesizes our main findings and research directions for future work.

Main findings. During the course of this research, we have found that it is possible to automatically determine the type of a query reformulation, if the appropriate features are used. We have achieved 92% accuracy in distinguishing among four broad classes of query reformulations, noticing that the learning scheme is important as it can exploit the fact that some class boundaries are more fuzzy than others.

Table 14 Diversity score of recommended queries: distinct documents among the top-5 results for the top-5 useful or somewhat useful recommendations

Dscore	p-value	System	Iter.	Scoring
13.49		Queryflow-S	10	Abs.
13.44		Queryflow-S	5	Abs.
13.20		Queryflow-SP	1	Abs.
13.04		Queryflow-SP	10	Abs.
12.99		Queryflow-SP	5	Abs.
12.84		Queryflow-SCP	1	Abs.
12.73		Queryflow-SCP	5	Abs.
12.70		Queryflow-GSPC	1	Abs.
12.70		Queryflow-SCP	10	Abs.
12.52		Queryflow-S	1	Rel.
12.42		Queryflow-S	1	Abs.
12.40		Queryflow-S	10	Rel.
12.38		Queryflow	1	Abs.
12.38		Queryflow-GSPC	5	Abs.
12.37		Queryflow-S	5	Rel.
12.33		Queryflow-SC	5	Abs.
12.33		Queryflow-GSPC	10	Abs.
12.28	0.10	QueryDocument-Bwd	6	Rel.
12.25	0.10	Queryflow-SC	10	Abs.
12.21	0.08	QueryDocument-Bwd	12	Rel.
12.21	0.08	QueryDocument-Bwd	12	Abs.
12.16	0.08	QueryDocument-Bwd	2	Rel.
12.11	0.06	QueryDocument-Bwd	6	Abs.
12.01	0.08	Queryflow	5	Abs.
11.97	0.05	QueryDocument-Bwd	2	Abs.
11.92	0.05	Queryflow-SC	1	Rel.
11.89	0.07	Queryflow	10	Abs.
11.77	0.04	Queryflow-SC	1	Abs.
11.64	0.03	Queryflow-SC	10	Rel.
11.60	0.03	Queryflow-SC	5	Rel.
11.13	0.01	Queryflow-SP	1	Rel.
11.04	0.01	Queryflow-SP	5	Rel.
10.94	0.01	Queryflow-SP	10	Rel.
10.62	$< .01$	Queryflow-SPC	1	Rel.
10.61	$< .01$	Queryflow-SPC	5	Rel.
10.56	$< .01$	Queryflow-SPC	10	Rel.
10.43	$< .01$	Queryflow	10	Rel.
10.39	$< .01$	Queryflow	1	Rel.
10.36	$< .01$	Queryflow	5	Rel.
10.28	$< .01$	Queryflow-GSPC	10	Rel.
10.25	$< .01$	Queryflow-GSPC	1	Rel.
10.08	$< .01$	Queryflow-GSPC	5	Rel.
9.25	$< .01$	Queryflow-(S^2)	1	Rel.
9.21	$< .01$	Queryflow-(S^2)	10	Rel.
9.17	$< .01$	Queryflow-(S^2)	1	Abs.
9.00	$< .01$	Queryflow-(S^2)	10	Abs.
6.68	$< .01$	Queryflow-(SG)	10	Abs.
6.63	$< .01$	Queryflow-(SG)	10	Rel.
5.75	$< .01$	QueryDocument-Fwd	12	Abs.
5.71	$< .01$	QueryDocument-Fwd	6	Abs.
5.69	$< .01$	Queryflow-(SS^T)	10	Abs.
5.61	$< .01$	Queryflow-(SS^T)	10	Rel.
5.49	$< .01$	QueryDocument-Fwd	2	Abs.
5.05	$< .01$	QueryDocument-Fwd	6	Rel.
5.04	$< .01$	QueryDocument-Fwd	12	Rel.
4.83	$< .01$	QueryDocument-Fwd	2	Rel.

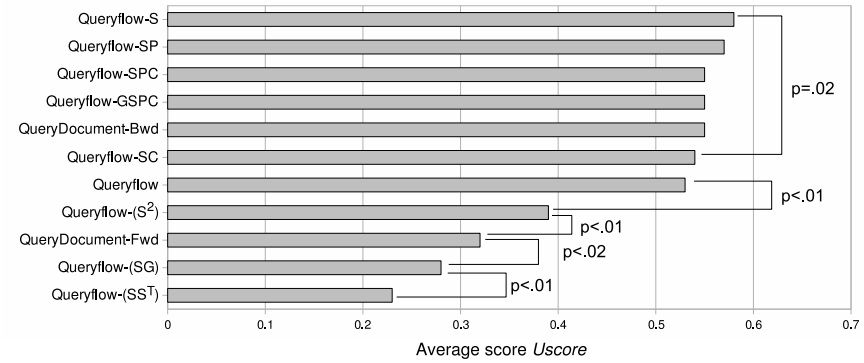


Fig. 5 Usefulness scores, best variant per system

We applied the classifier to a large query log and studied reformulation paths that are the sequence of reformulations that a user does in the course of a search mission. This allowed us to study query reformulation patterns, matching some results of previous studies done over much smaller data set using manual assessments, and extracting new patterns which are discoverable giving that our automatic classifier enables the processing of a much larger set of data than when using manual annotations. From some of the patterns we extracted, we can see for instance that generalization and specializations appear frequently together in alternating order, and that error corrections are more frequent either at the beginning of a search mission or after another error correction. When mapping query transitions to topical categories we see that reference search is a typical context for generalizations and specializations, and that many mission changes are associated to switches from or to entertainment/recreation sites.

We annotated a large query-flow graph with transition types, and noticed the anti-symmetry of generalization and specialization there. We also observed that given a query, the distribution of possible generalizations and error corrections tend to be more concentrated than the distributions of specializations or parallel moves.

Follow-up work. Since our initial formulation in [9] and follow-up papers [10,11], other aspects of query-flow graphs have been studied.

Baraglia et al. [5,4] show that the transition probabilities in the query-flow graph change over time. The changes in the graph may reduce the quality of the recommendations if an old query-flow graph is used.

Anagnostopoulos et al. [1] propose a method for generating query recommendations based on optimizing the expected path a user will take on the query-flow graph. This can lead to a better user experience in terms of issuing several interesting queries in sequence, while keeping the relevance of query recommendations high.

Bordino et al. [13] embed the query-flow graph (or sub-graphs of it) into a low-dimensional space. The authors show that this projection preserves semantic distances between queries while allowing a fast computation of query similarity.

Future work. In this paper we focus mainly on characterization and pattern mining, but the next natural step is to use these results as building blocks for several applications. In particular, the query transition graph can be used to build new query recommendation systems, or to improve existing ones.

Table 15 Example showing the possibilities of composing query reformulation graphs

P-path	SG-path	GS-path
	bike trader	
ebay	mopeds	mopeds
auto trader	used motorbikes	bike insurance
mcn	two wheels	mini motos
	disney channel	
youtube	disney	playhouse disney
cbbc	disney games	disney store
you tube	games	disney
	movie downloads	
free music downloads	free movies	free music downloads
music downloads	movies	movie trailers
bebo	free downloads	free mp3 downloads
	prestwick airport	
ryanair	glasgow airport	glasgow airport
glasgow airport	glasgow	restaurants in prestwick
edinburgh airport	watson car parks	prestwick tourist information
	sony ericsson	
nokia	sony ericssons	sony center
o2	k800i software	sony psp
carphone warehouse	w880 pc suite	sony vaio

One important feature in recommendations is *diversity*: we may achieve diverse recommendations by exploring the transition graph to find an appropriate combination of specializations, generalizations, and parallel moves. Another issue is to be able to take user context and history of previous queries into consideration (i.e., *recommendation with history* [9]): we may provide recommendations that do not depend only on the last query, but on the last 3-4 queries, and are in the QRT class that is the most likely to occur next. Using the the frequency of query reformulation patterns mined from large query logs, as reported in Section 5, we can define a stochastic process that tell us which is the next most probable QRT: then we can use this information to decide which paths to follow from the current node in the query graph (i.e, the last user query). Another possible application, is *lookahead recommendation*: based on the observation that query recommendations are mostly useful when are specializations, we can visit the specialization transition graph and recommend queries that are specializations of specializations of the current query. This may provide some unexpected, yet interesting recommendations, and in some cases anticipate the user in her own research mission.

Also, *composed query reformulation graphs* could be a fruitful source of query recommendations. To show what is the evidence we have found for this, we composed the `G` and `S` graphs to obtain a graph in which each edge indicates a two-step reformulation (specializing and then generalizing the original query, or viceversa). We weighted the edges in these graphs by multiplying the probabilities of following each link (these probabilities are the feature *count_norm1* in Table 2). The result of the top `SG` and `GS` paths from a set of example queries in the UK dataset is shown in Table 15, along with the top parallel moves (`P`) by count from each example query. In the examples we reviewed, the `SG` and `GS` paths yield interesting recommendations. Comparing them with other types of path (including, e.g., $\mathtt{SS}^T$ and $\mathtt{S}^T\mathtt{S}$) is one of our projects for extending the current work.

Finally, simultaneously learning both the query reformulation types and how to segment a session into chains (the two tasks that we identified and separated in the Introduction) might be a way of achieving a non-trivial improvement in accuracy. This would mean formulating our task in similar terms as, for instance, the task of *part-of-speech* and *bracketing* in Natural Language Processing. Also the insights obtained from the analysis of the graph can be used, by imposing an asymmetry constraint between specialization and generalization during the learning process.

Acknowledgments: The authors wish to thank Massimiliano Ciaramita, Debora Donato and Aristides Gionis for their help.

References

1. Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A.: An optimization framework for query recommendation. In: WSDM '10: Proceedings of the third ACM international conference on Web search and data mining, pp. 161–170. ACM, New York, NY, USA (2010). DOI http://doi.acm.org/10.1145/1718487.1718508
2. Baeza-yates, R., Hurtado, C., Mendoza, M.: Query recommendation using query logs in search engines. In: In International Workshop on Clustering Information over the Web (ClustWeb, in conjunction with EDBT), Creete, pp. 588–596. Springer (2004)
3. Baeza-Yates, R., Tiberi, A.: Extracting semantic relations from query logs. In: KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 76–85. ACM Press, New York, NY, USA (2007). DOI 10.1145/1281192.1281204
4. Baraglia, R., Castillo, C., Donato, D., Nardini, F.M., Perego, R.: The effects of time on query flow graph-based models for query suggestion. In: Proceedings of RIAO (2010)
5. Baraglia, R., Castillo, C., Donato, D., Nardini, F.M., Perego, R., Silvestri, F.: Aging effects on query flow graphs for query suggestion. In: CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management, pp. 1947–1950. ACM, New York, NY, USA (2009). DOI http://doi.acm.org/10.1145/1645953.1646272
6. Beeferman, D., Berger, A.: Agglomerative clustering of a search engine query log. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 407–416. ACM Press (2000). DOI http://dx.doi.org/10.1145/347090.347176
7. Belazzougui, D., Boldi, P., Pagh, R., Vigna, S.: Theory and practise of monotone minimal perfect hashing. In: Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments (ALENEX), pp. 132–144. SIAM (2009)
8. Belkin, N.J.: The human element: helping people find what they don't know. Commun. ACM **43**(8), 58–61 (2000)
9. Boldi, P., Bonchi, F., Castillo, C., Donato, D., Gionis, A., Vigna, S.: The query-flow graph: model and applications. In: Proc. of ACM 17th Conference on Information and Knowledge Management (CIKM), pp. 609–618. ACM Press, Napa Valley, CA, USA (2008)
10. Boldi, P., Bonchi, F., Castillo, C., Donato, D., Vigna, S.: Query suggestions using query-flow graphs. In: WSCD '09: Proceedings of the 2009 workshop on Web Search Click Data, pp. 56–63. ACM, New York, NY, USA (2009). DOI http://doi.acm.org/10.1145/1507509.1507518
11. Boldi, P., Bonchi, F., Castillo, C., Vigna, S.: From "dango" to "japanese cakes": Query reformulation models and patterns. In: WI-IAT '09: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, pp. 183–190. IEEE Computer Society, Washington, DC, USA (2009). DOI http://dx.doi.org/10.1109/WI-IAT.2009.34
12. Boldi, P., Vigna, S.: The WebGraph framework I: Compression techniques. In: Proc. of the Thirteenth International World Wide Web Conference (WWW 2004), pp. 595–601. ACM Press, Manhattan, USA (2004)
13. Bordino, I., Castillo, C., Donato, D., Gionis, A.: Query similarity by projecting the query-flow graph. In: SIGIR '10: Proceedings of the 33rd annual international ACM SIGIR conference on Research and development in information retrieval. ACM Press (2010)

14. Castillo, C., Donato, D., Becchetti, L., Boldi, P., Leonardi, S., Santini, M., Vigna, S.: A reference collection for web spam. SIGIR Forum **40**(2), 11–24 (2006). DOI http://dx.doi.org/10.1145/1189702.1189703
15. Craswell, N., Szummer, M.: Random walks on the click graph. In: SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 239–246. ACM Press, New York, NY, USA (2007). DOI 10.1145/1277741.1277784
16. Craswell, N., Zoeter, O., Taylor, M., Ramsey, B.: An experimental comparison of click position-bias models. In: WSDM '08: Proceedings of the international conference on Web search and web data mining, pp. 87–94. ACM (2008)
17. Donato, D., Bonchi, F., Chi, T., Maarek, Y.: Do you want to take notes? Identifying research missions in Yahoo! Search Pad. In: Proceedings of the 19th International Conference on World Wide Web (WWW 2010) (2010)
18. Fonseca, B.M., Golgher, P.B., de Moura, E.S., Ziviani, N.: Using association rules to discover search engines related queries. In: LA-WEB '03: Proceedings of the First Latin American Web Congress. IEEE Computer Society, Washington, DC, USA (2003)
19. Fuxman, A., Tsaparas, P., Achan, K., Agrawal, R.: Using the wisdom of the crowds for keyword generation. In: WWW '08: Proceeding of the 17th international conference on World Wide Web, pp. 61–70. ACM, New York, NY, USA (2008). DOI http://dx.doi.org/10.1145/1367497.1367506
20. Glance, N.S.: Community search assistant. In: In Artificial Intelligence for Web Search, pp. 91–96 (2001)
21. Goodrum, A.A., Bejune, M.M., Siochi, A.C.: A state transition analysis of image search patterns on the web. In: Image and Video Retrieval, *Lecture Notes in Computer Science*, vol. 2728, pp. 193–197. Springer (2003)
22. Haas, S.W., Grams, E.S.: Page and link classifications: connecting diverse resources. In: DL '98: Proceedings of the third ACM conference on Digital libraries, pp. 99–107. ACM, New York, NY, USA (1998). DOI http://doi.acm.org/10.1145/276675.276686
23. He, D., Göker, A.: Detecting session boundaries from web user logs. In: Proceedings of the BCS-IRSG 22nd annual colloquium on information retrieval research, pp. 57–66. Cambridge, UK (2000)
24. He, D., Göker, A., Harper, D.J.: Combining evidence for automatic web session identification. Inf. Process. Manage. **38**(5), 727–742 (2002). DOI http://dx.doi.org/10.1016/S0306-4573(01)00060-7
25. Jansen, B.J., Spink, A., Bateman, J., Saracevic, T.: Real life information retrieval: a study of user queries on the web. SIGIR Forum **32**(1), 5–17 (1998)
26. Jansen, B.J., Spink, A., Narayan, B.: Query modifications patterns during web searching. In: Information Technology, 2007. ITNG '07. Fourth International Conference on, pp. 439–444 (2007). DOI http://dx.doi.org/10.1109/ITNG.2007.164
27. Jansen, B.J., Zhang, M., Spink, A.: Patterns and transitions of query reformulation during web searching. International Journal of Web Information Systems **3**(4), 328–340 (2007). DOI http://dx.doi.org/10.1108/17440080710848116
28. Jeh, G., Widom, J.: Scaling personalized web search. In: WWW '03: Proceedings of the 12th international conference on World Wide Web, pp. 271–279. ACM Press, New York, NY, USA (2003). DOI http://dx.doi.org/10.1145/775152.775191
29. Jones, R., Klinkner, K.L.: Beyond the session timeout: Automatic hierarchical segmentation of search topics in query logs. In: CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management, pp. 699–708. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1458082.1458176
30. Jones, R., Rey, B., Madani, O., Greiner, W.: Generating query substitutions. In: Proceedings of the 15th international conference on World Wide Web, WWW 2006, pp. 387–396. Edinburgh, Scotland, UK (2006)
31. Lau, T., Horvitz, E.: Patterns of search: analyzing and modeling web query refinement. In: UM '99: Proceedings of the seventh international conference on User modeling, pp. 119–128. Springer-Verlag New York, Inc. (1999)
32. Li, Y., Zheng, Z., Dai, H..: Kdd cup-2005 report: facing a great challenge. SIGKDD Explor. Newsl. **7**(2), 91–99 (2005). DOI http://dx.doi.org/10.1145/1117454.1117466
33. Luxenburger, J., Elbassuoni, S., Weikum, G.: Matching task profiles and user needs in personalized web search. In: CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge mining, pp. 689–698. ACM, New York, NY, USA (2008). DOI http://dx.doi.org/10.1145/1458082.1458175

34. Mei, Q., Zhou, D., Church, K.: Query suggestion using hitting time. In: CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge mining, pp. 469–478. ACM, New York, NY, USA (2008). DOI http://dx.doi.org/10.1145/1458082.1458145
35. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
36. Radlinski, F., Joachims, T.: Query chains: learning to rank from implicit feedback. In: KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, pp. 239–248. ACM Press, New York, NY, USA (2005). DOI http://dx.doi.org/10.1145/1081870.1081899
37. Richardson, M.: Learning about the world through long-term query logs. ACM Trans. Web **2**(4), 1–27 (2008). DOI http://dx.doi.org/10.1145/1409220.1409224
38. 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). DOI http://dx.doi.org/10.1016/j.ipm.2005.05.005
39. Sadikov, E., Madhavan, J., Wang, L., Halevy, A.: Clustering query refinements by user intent. In: WWW '10: Proceedings of the 19th international conference on World Wide Web. Raleigh, North Carolina, USA (2010)
40. Silverstein, C., Henzinger, M., Marais, H., Moricz, M.: Analysis of a very large altavista query log. Tech. rep., Digital SRC (1998)
41. Spink, A., Jansen, B.J., Wolfram, D., Saracevic, T.: From e-sex to e-commerce: Web search changes. IEEE Computer **35**(3), 107–109 (2002)
42. Wen, J., Nie, J., Zhang, H.: Clustering user queries of a search engine. In: Proceedings of the 10th international conference on World Wide Web, pp. 162–168. ACM (2001)
43. White, R.W., Clarke, C.L.A., Cucerzan, S.: Comparing query logs and pseudo-relevance feedbackfor web-search query refinement. In: SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 831–832. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1277741.1277931
44. Zhang, Z., Nasraoui, O.: Mining search engine query logs for query recommendation. In: WWW '06: Proceedings of the 15th international conference on World Wide Web, pp. 1039–1040. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1135777.1136004