New Query Suggestion Framework and Algorithms: A Case Study for an Educational Search Engine

I. Bahattin Vidinli[1], Rifat Ozcan[2†]

[1,2]Computer Engineering Dept., Turgut Ozal University, Gazze cd. No:7 Etlik /Keçiören/Ankara, Turkey

[1]bahattin@vidinli.com, [2]rozcan@turgutozal.edu.tr

† Corresponding Author

Abstract

Query suggestion is generally an integrated part of web search engines. In this study, we first redefine and reduce the query suggestion problem as "comparison of queries". We then propose a general modular framework for query suggestion algorithm development. We also develop new query suggestion algorithms which are used in our proposed framework, exploiting query, session and user features. As a case study, we use query logs of a real educational search engine that targets K-12 students in Turkey. We also exploit educational features (course, grade) in our query suggestion algorithms. We test our framework and algorithms over a set of queries by an experiment and demonstrate a 66-90% statistically significant increase in relevance of query suggestions compared to a baseline method.

Keywords: Query suggestion, framework, educational search engine, query recommendation

1. Introduction

The number of Internet users in the world is estimated to be more than 2.89 billion, as of May 2014, which accounts for 42.3% of the human population (Internet Society, 2015). Advances in smart phone technology and network bandwidths also boosted Internet data traffic in the last few years. It has also enabled the very young population to access the Internet. People use search engines to find the relevant information in this immense environment. Large scale search engines try to cope with these demanding requirements coming from a very diverse population with differing cultures and ages. There is recent interest in literature for analyzing search behavior of different user groups, especially children.

The primary goal of a search engine is to retrieve relevant results of a query with high ranks. Even though this objective primarily depends on the search engine ranking algorithm, the quality of the submitted query is also important. To this end, query suggestion (recommendation) techniques help users to formulate or refine their queries. A search engine may provide suggestions while the user is typing the query (referred to in literature as "query auto-completion") or after a query submission. This paper focuses on the latter type of query suggestion techniques. Earlier studies addressing this problem mainly focus on large scale search engines that target the public with very diverse information needs. On the other hand, vertical search engines index resources on a specific subject and respond to queries coming from a specific group of people. An educational search engine, to give an example of a vertical engine, targets students (and instructors) and covers course-related resources. Query suggestion techniques for such systems are not well investigated in literature before. It is important to check whether algorithms proposed for large scale search engines still apply for vertical search engines. Query suggestion is more important for K-12 educational search engines since children have difficulty in formulating queries.

In this study, we first redefine and reduce the query suggestion problem as a series of query comparisons. We propose a general modular framework based on this definition. Our framework includes two major steps along with several minor steps. In the first major step, candidate queries for suggestion are selected. Next, these queries are sorted based on different query scoring algorithms exploiting various query, session and user features. Our framework is modular so that new query candidate selection and query scoring methods can easily be plugged in. It also enables several methods to be combined easily.

We evaluate the performance of query suggestion techniques proposed for large scale search engines on an educational search engine and try to improve query suggestion. Even though vertical search engines have smaller numbers of users and click-through data, they generally have more extensive and specific features related to their users and their content. In the case of an educational search engine, age, grade, school and city of a student could be recorded as extra features. In addition, indexed resources may have specific properties such as course, related grade(s), content type (subject description, questions, animation/game, etc.), related curriculum items. All these extra features could be exploited for more advanced ranking and query suggestion techniques. We propose new suggestion techniques based on some of these features and compare their effectiveness to former techniques in literature.

Our main contributions in this work are listed as follows:

- We redefine and reduce the "Query Suggestion (QS)" problem,
- We propose a modular, extendable query suggestion framework that enables new methods to be easily plugged in and may contain many QS algorithms.
- We evaluate the performance of a click-through data based QS technique proposed for general purpose search engines in literature, on a real educational search engine log.
- We propose new QS algorithms that exploit query features for general (query, session, user features) and educational search engines (course and grade features).
- We also propose hybrid algorithms that combine several QS techniques for higher effectiveness. These algorithms are integrated in the above mentioned framework.

This paper is organized as follows: Section 2 gives information on the related works on query suggestion and briefly mentions studies focusing on child users. We then introduce how we redefine and reduce the query suggestion problem. We present our query suggestion framework where the query suggestion problem is handled in a modular and pluggable architecture, in the subsequent section. We then mention query scoring and comparison algorithms exploiting various features such as query, session, user and educational properties. In Section 6, we mention our experimental setup and discuss the results. Finally, we conclude the paper and mention future research directions.

2. Related Work

Search engines appeared on the World Wide Web just after the Web became an important source of information in the '90s. Search engines started to offer basic query suggestions to users. It is important to analyze the general behavior of users on search engines before presenting the literature. Queries that are submitted by users are usually short, poorly built and mistyped (Cui, Wen, Nie, & Ma, 2002). Users may also not have enough information about the topic they are searching for. This is especially the case when developing a search system for child users (Torres, Hiemstra, Weber, & Serdyukov, 2012) because this group of users has difficulty in formulating their queries (Babuscu & Özcan, 2014).

Query suggestion literature works can be classified in two different groups (Bhatia, Majumdar, & Mitra, 2011). The majority of recent works (Arora & Duhan, 2013; Baeza-Yates, Hurtado, & Mendoza, 2005; Cao et al., 2008) mine query logs in order to give suggestions. The second line of studies (Bhatia et al., 2011; Bordogna, Campi, Psaila, & Ronchi, 2012) does not require a query log and performs “content analysis” of web pages or retrieved document snippets. There are several other studies that try to understand user context for suggesting new queries (Cao et al., 2008; Huang, Chien, & Oyang, 2003). User information (age, gender, username, IP, tools) and previous queries in a query session can be considered as context.

Query Suggestion was in the form of “Query Clustering” or “Term Clustering” in early studies (Lewis & Croft, 1990). Later works (Fonseca, Golgher, de Moura, & Ziviani, 2003; Wang & Zhai, 2008) try to identify association rules between queries by mining query logs. Several other studies (Baeza-Yates et al., 2005; Mei, Zhou, & Church, 2008; Wen, Nie, & Zhang, 2001) extract query-clicked URL/doc bipartite graphs using search logs and exploit the connections in order to figure out related queries and documents. These bipartite graphs consist of a group of nodes representing queries, another group of nodes representing URL's and edges connecting these two groups when there is a click between a query and a URL. We refer to this graph as “query click graph” in the rest of this paper. The “Hitting Time” algorithm (Mei et al., 2008) is an example of finding related queries using this graph. This algorithm computes the transition probabilities between initial query and candidate queries (candidates for suggestion) on the graph using frequencies as the weight between queries and clicked URL's. The details of this algorithm are presented in Section 5.1, since we adopt this algorithm in our study. As an alternative to the query click graph, Boldi et al., (2008) propose query-flow graph data structure by mining query logs. This graph contains edges between queries along with a weight component. Weight function is learned using textual, session and time-related features among pairs of queries. In a subsequent study (Boldi, Bonchi, Castillo, Donato, & Vigna, 2009), they propose random walk based method over this query-flow graph for query suggestion. (Wen et al., 2001) cluster queries using textual content of queries and their clicked documents and these clusters are used to determine frequently asked queries. (Baeza-Yates, Hurtado, & Mendoza, 2007) uses query clustering to both relevance ranking and query recommendation. In this work, clustering process is based on a term-weight vector representation of URLs clicked after queries. Actually, query logs may have more than just query-click information. There may also be features like query frequency, issue date/time, number of results etc. (Silvestri, 2010) which can be used in the query suggestion process. We also use many of these features in this work.

Another important method of query suggestion is content analysis. Methods mentioned in (Bhatia et al., 2011; Kraft & Zien, 2004) exploit the contents of queries, contents of documents, document snippets or anchor texts to identify similar queries. These are also used in situations where a query-log is not available.

One basic form of query suggestion is “query expansion”. Adding some relevant terms to the original query may be useful in many cases (Cui et al., 2002), which is relatively easy to implement. Query suggestion may be seen as a different form of “query reformulation which is usually done by users”. In other words, query suggestion can be seen as query reformulation which is done automatically by the search engine itself. In literature, query reformulations are classified into different categories such as generalization, specialization, error correction and parallel move.

(Jansen & Spink, 2006) show that 46% of users reformulate their queries based on analysis of a large query log. (Lau & Horvitz, 1999) presents a Bayesian network based probabilistic approach to predict how users refine their queries.

There is a recent interest in researching the search characteristics of young users (Torres, Weber, & Hiemstra, 2014; Usta, Altingovde, Vidinli, Ozcan, & Ulusoy, 2014) and developing new techniques for this group of users (Torres et al., 2012). As we noted earlier, child users have difficulty in formulating queries (Babuscu & Özcan, 2014) and query suggestion or auto-completion mechanisms have a higher necessity for them. In this study, we work with an educational search engine that is used primarily by students in grades 5 through 8. To the best of our knowledge, (Torres et al., 2012) presents the first work on query suggestion techniques specialized for children. Regarding the information needs of students, (Yılmazel, 2011) develop a query log based query suggestion component for a search engine indexing distance education textbooks for university students in Turkey. A collaborative filtering based approach is used (with Apache Mahout tool) such that similarities between queries (items) are computed based on the idea that how many times they are searched together in a session. His method only uses session information of queries and does not exploit any educational features. Our study is the first work proposing query suggestion techniques for an educational search engine targeting K-12 students, as far as we know. We introduce a modular framework and propose query suggestion algorithms exploiting query, session, user, and educational features.

3. Redefinition and Reduction of the Query Suggestion Problem

Query Suggestion (QS) is usually defined to be "finding some related queries given the original query which is issued by the user". For example, when the user issues the query "American airline", the search engine would suggest search terms such as "airline tickets", "online airline tickets", "American airline reservation" etc. In Section 2*,* we report several key papers proposing methods for finding these types of suggested queries. Majority of these works propose techniques that may operate on several data types such as query logs, document contents or user context and produce ordered list of suggested queries as a whole. To achieve a more modular, straightforward and practical approach, the query suggestion problem may be simplified as follows:

We redefine the query suggestion problem as follows: The QS problem should be simply thought of as "a series of 'comparison of two queries'". The first query in the comparison is the original query that was issued by the searcher (user). The second query is the "candidate query" that is to be suggested to the user, if selected at the end of suggestion process. The comparison of queries may depend on several features such as term similarity, query logs, etc. This approach of comparing queries may be practical in the sense that it simplifies the query suggestion problem, makes the process traceable, extendable and debuggable.

A set of candidate queries (for suggestion) are determined for a submitted (initial) query q_i and each candidate query (q_c) is compared to q_i. Finally, candidate queries can be sorted based on their ranks/scores and top-n candidate queries may be presented to the user as query suggestions.

This approach has these advantages:

- The QS problem is obviously reduced to a "comparison of two queries", the original and candidate query;
- Two queries may be compared with many straightforward methods;
- Multiple query comparison methods may be easily combined;
- It is easy to trace, debug and develop new methods with this approach.

With this reduction of the problem, one should only be concerned about the comparison of two queries, nothing more. However, the comparison does not necessarily refer to the similarity/relatedness of two queries but it may also measure different aspects of queries to be compared. For example, one can check for excessive closeness/similarity of the queries for diversification purposes.

4. Query Suggestion Framework

In this work, we suggest a straightforward QS framework which can be extended by plugging in new QS algorithms. By establishing a well-defined framework, the QS process and problem is simplified. Different methods and algorithms may be plugged into this framework, making it possible to compare and/or combine different methods.

The framework consists of two main steps: s*elect & sort.* Some additional relatively simple and small steps may also be included in the process to improve accuracy; so we include post-select (generic controls), post-sort (final controls) steps. Our framework (also shown in Figure 1) contains the following steps:

1. *Select/Find candidate queries (Major step)*
2. Generic controls (Optional, relatively minor step)
3. *Sort candidate queries with single/multiple algorithms (Major step)*
1. Final controls
   a) Generalization, diversification, (Optional, relatively minor step)
   a) Re-ordering, post-processing (Optional, relatively minor step)

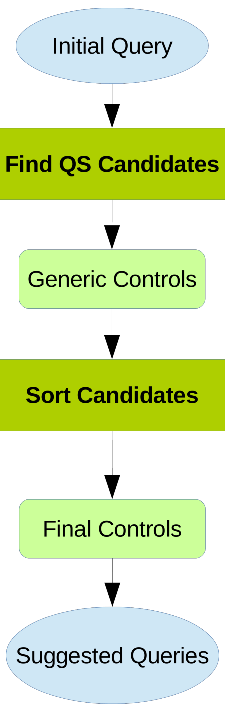


FIG. 1. Query suggestion framework

The basic ideas of our steps in the framework are described as follows:

- Selection of candidate queries can be done in a separate step, by means of different algorithms such as traversing the query click graph using Depth First Search (DFS) as in (Mei et al., 2008) or Breadth First Search (BFS) etc. This is a completely different and separate step to the other steps. The purpose is to "find, explore" possible query suggestions (hereafter referred to as Query Suggestion Candidates). In the most generic situation, all input queries may be candidate queries. If we have enough processing power, we may use this generic situation where all queries are considered as candidates for suggestion.
- Generic controls (although not a major step nor mandatory) may be used to eliminate some useless queries from candidate queries. Dropping very short (1-2 letter) queries, very long queries or mistyped queries are examples of such controls.
- Sorting the prepared candidate queries is the next major step. Existing QS algorithms that we mention in the Section 2, techniques we propose in this paper or any sorting algorithm that orders candidate queries can be used in this step. Our framework also enables

combining more than one sorting method, which we also employ in this work.

- Query generalization or diversification procedures (Non-major step, at least, at the moment) may be applied after the sorting phase in order to fine tune the suggestions before showing to the user. Query generalization selects more general forms of the initial query such as proposing "cell structure" or "cell" to the user submitting query "mitochondria". Diversification techniques can also be employed in this step so that very similar queries should not be shown to the user as suggestions. For instance, "multiplication in rational numbers", "multiplication of rational numbers" and "multiplication operation in rational numbers" queries should not be shown to the user together.
- One of the most important aspects of this framework is to break the query suggestion problem into pieces. This concept is also known as "Separation of Concerns" that is favored in several fields. Using this framework, one may contribute to the QS problem by proposing a new algorithm (i.e., query candidate selection algorithm) for a specific step without having to handle other steps.

The overall diagram of the framework is shown in Figure 1, with different colorings for major and minor steps. In the following subsections, we give further details of each step in our framework.

4.1. Selection Phase

Selection phase is the first major step of our proposed query suggestion framework. In this step, the purpose is to find candidate queries for suggestion. Candidate queries could either be selected from a set of previous queries by some means or generated in the absence of query logs. In this study, we focus on query suggestion methods using query logs. Candidate queries can be obtained from query logs by traversing query click graph using DFS or BFS. In the most generic situation, all input queries or all possible queries may be candidate queries, although this requires high processing power.

Our initial experiments show that candidate queries found using DFS seem straying away from the topic of the initial query as DFS traverses the query click graph in a depth first fashion (except for head queries). However, this method is used in the Hitting Time algorithm (Mei et al., 2008) that we mention in subsection 5.1.

Breadth First Search (BFS), on the other hand, seems more suitable for finding related queries since graph traversal is not straying away from the subject of initial query, in contrast to the DFS case. For this reason, we use and experiment with BFS as a "query selection algorithm" in this work and demonstrate in Section 6 that it is more suitable/useful for query suggestion, for at least our query log.

4.2. Generic Controls

These are some basic checks that can be done before proceeding to the sorting phase. Although this is not a major step, it may sometimes be quite useful to eliminate some erroneous or strange queries from the set of candidate queries. We find that the following basic controls are useful in general QS procedure:

i. Long queries having too many words/letters removed;
ii. Too short queries are removed;
iii. Queries that are a subset of the initial query are removed;
iv. Too generic queries (such as science, math, course, questions etc.) are removed;
v. If the course of the initial query is known (or can be predicted), queries related to different courses are removed from the candidate set.

After we eliminate candidate queries based on these criteria, we finally apply a click frequency threshold such that candidate queries with very low number of clicks are also filtered.

All filtering mechanisms apply for general purpose search engines. However, the last item exploits an educational feature and only applies for vertical search engine focusing on education materials, which is our case in this work. Our basic intuition behind this step is that query suggestion system should suggest queries related to the same course as the course of the initial query. Note that this requires course information of initial query and candidate queries to be known or predicted. The list of controls can be extended by experimenting with QS framework/algorithms or based on the specific domain we are searching. For future work, spell checking and correction can be implemented in this stage (we did not implement/use spell checking feature).

4.3. Sorting Phase

In this subsection, we describe the sorting phase of our framework. The sole purpose of this major step is to order candidate queries based on some means such as similarity to the initial query or co-occurrence with the initial query in the same query sessions etc. Candidate queries could be sorted based on such various aspects. Our framework provides a modular mechanism so that various orderings of candidate queries could be combined for higher accuracy in query suggestion. In Section 5 we present various query scoring and comparison techniques by exploiting query features for general (query, session, user features) and educational search engines (course and grade features). Note that our framework allows new candidate query ordering mechanisms to be employed in a straightforward manner.

We now give the details of our sorting step in our framework. Let q_i denotes the initial query submitted by the user. In the following example our initial query is “American airline”. Let Cq_i denotes the set of candidate queries for q_i in a vector form as shown below (for simplicity, only 4 candidates are shown). Assume that there are N different candidate sorting methods that are available in the framework and each of them produces an ordering of candidate queries by computing scores. We denote this score vector as V_j (j ranging from 1 to N) and show an example below.

$$\text{Candidate Queries}\left(Cq_i\right)=\begin{bmatrix}\text{american airlines}\\ \text{airline tickets}\\ \text{airline phone}\\ \text{airline reservation}\end{bmatrix}=\begin{bmatrix}q_1\\ q_2\\ q_3\\ q_4\end{bmatrix}$$

$$V_1 = \begin{bmatrix} \text{Score of } q_1 \text{ wrt } q_i \\ \text{Score of } q_2 \text{ wrt } q_i \\ \text{Score of } q_3 \text{ wrt } q_i \\ \text{Score of } q_4 \text{ wrt } q_i \end{bmatrix} = \begin{bmatrix} Sq_1 \\ Sq_2 \\ Sq_3 \\ Sq_4 \end{bmatrix}$$

Our framework combines various candidate ordering algorithms. This can be done by any aggregation method. We worked on the aggregation methods that are mentioned in Section 4.3.1. The aggregation of sorting algorithms may be seen similar to the aggregation of search engine results in a metasearch engine as described in (Aslam & Montague, 2001). We tried to improve the query suggestion performance by combining multiple sorting algorithms.

4.3.1. Aggregation Methods

Aggregation methods may be classified at least in two categories, score based and rank based methods (Renda & Straccia, 2003). Other approaches may also be suggested. We currently work with these two types of aggregation methods.

4.3.1.1. Weighted Score Based Aggregation

We suggest that, aggregation of different sorting/ordering algorithms may be done by weighted summation of score vectors, as shown in the following formula:

$$\text{Final Score Vector} = k_1 \cdot Norm(V_1) + k_2 \cdot Norm(V_2) + \ldots + k_n \cdot Norm(V_n) = \sum_{i=1}^{\textit{number of algorithms}} k_i \cdot Norm(V_i)$$

where normalization is computed as:

$$Norm(V) = \{ \frac{x}{max(V)} \mid \forall x \in V \}$$

Each ordering is weighted by coefficients k_1 through k_n. Note that each score vector V_j can have scores in different ranges so they need to be normalized (re-scaled into 0-1 range) before combination. Finally, a score vector is obtained. By re-scaling the individual vectors, the *final score is not over-affected by a single algorithm* (even if it has high/erroneous values), instead, single algorithm affects the final score up to a certain reasonable amount. This is also useful to suppress errors, mistakes or biases caused by some algorithms. As we experienced in Section 6, different algorithms achieve different success rates on different input queries. Just like metasearch engine case, input methods does/may not achieve same success rates (Aslam & Montague, 2001), therefore, weighted combination of algorithms may show better performance.

Next, we observe that some candidate sorting algorithms (especially the ones depending on query frequency, number of results) produces a few very high scores with lots of very low scores. We revise our formula as below by applying log transformation for such skewed distributions. Note that logarithmic transformation is not applied to all score vectors in our works but here we show as such for the sake of simplicity.

$$\text{Final Score Vector} = k_1 \cdot Norm(\log(V_1)) + k_2 \cdot Norm(\log(V_2)) + \ldots + k_n \cdot Norm(\log(V_n))$$

$$= \sum_{i=1}^{\textit{number of algorithms}} k_i \cdot Norm(\log(V_i))$$

The coefficients k_1, k_2.. are used to promote algorithms which are known/observed to have better effects in final results. For testing purposes, coefficients of some algorithms may be taken as zero. It may also be useful to try different coefficients (parameter tuning) to test overall QS performance. This type of calculation is quite modular and allows us to use different sorting algorithms developed by different researchers and/or developers.

In our experiments we determine a set of *k*-coefficients. We also notice that some algorithms (hence coefficients) may behave better in some sets of initial queries. To overcome and use this case, we may use a set of *k*-coefficients to determine the first 8-10 query suggestions while we use another set of *k*-coefficients for the remaining 2-6 suggestions. This way, it might be possible to increase the overall accuracy of Top10-15 query suggestions. We revise our formula as follows (assume that logarithmic scaling is performed in the *Norm* function):

$$\text{Final Score Vector for 1st group of QS} = FSV_1 = \sum_{i=1}^{\textit{number of algorithms}} k_{1i} \cdot Norm(V_i)$$

$$\text{Final Score Vector for 2nd group of QS} = FSV_2 = \sum_{i=1}^{\textit{number of algorithms}} k_{2i} \cdot Norm(V_i)$$

where FSV_1 will be used for selecting the first 8-10 query suggestions and FSV_2 will be used for selecting the remaining 2-6 suggestions (depending on the number of final query suggestions, which may be a total of 10 or 15). If there are overlapping query suggestions, they should be unified in the final result set. This may be seen as a combined method of two different hybrid methods, where they contain different *k*-coefficients for each algorithm. In this paper, we experiment with the previous version of our formula and leave the latter as a future work.

4.3.1.2. Rank Based Aggregation

In this line of aggregation methods, the ranks of a result set of an algorithm (in our case query suggestion candidates) are used to weight the overall ordering, when multiple algorithms are combined.

4.3.1.2.1. Borda Count Method

The Borda Count method (Borda, 1781; Dwork, Kumar, Naor, & Sivakumar, 2001) combines the results of different methods by weighting each item proportionally to the position of that item in the result set of each sub method. The formula and points that are provided by a sub method is illustrated in Table 1. The points for each method are computed using this table, then all points of candidate queries are summed and final ranking is obtained by sorting candidate queries with decreasing average points. This method is noted as Hybrid-2-Borda Count or similar in other figures and tables.

TABLE 1. Borda Count ranking of candidates for a method.

Ranking	**Candidates**	**Formula**	**Points**
1st	query 1	n	5
2nd	query 2	n-1	4
3rd	query 3	n-2	3
4th	query 4	n-3	2
5th	query 5	n-4	1

4.3.1.2.2. Weighted Borda Count Method (Weighted Borda Fuse)

This is a slightly modified version of previous Borda Count Method, where each method is separately weighted by a coefficient (Aslam & Montague, 2001), just like the weighted score based aggregation that is mentioned in Section 4.3.1.1. We also tested this method to measure the effect of weighting each submethod. This method is noted as Hybrid-3-Weighted Borda Count or similar in other figures and tables.

4.3.1.2.3. Weighted Voting Method

This is a rather simpler method where a candidate query is given a point/vote when it exists in the top-n results of a sub-method. For example, if a query exists in top-n results of 4 different methods, it is given 4 point. We used top-30 results of sub-methods to implement this simple rank aggregation method.

4.3.1.3. Selection of score and rank aggregation methods

The purpose of this work is not to compare/measure rank aggregation methods. But we try to select and use the most appropriate one in our framework for the query suggestion field. We experience the performance of score based rank aggregation in our initial experiments. The advantage of score based aggregation is also noted in literature; according to (Renda & Straccia, 2003), score based methods outperform rank based methods in a metasearch engine setting.

We suggest the score based aggregation over rank based aggregation. We think (and see in our experiment results) that, our method of using “scores” of sorted lists, instead of rank positions are better, where scores are available/computable. Using “ranks, position of suggestions” may be used successfully when there is no “scores” of suggested queries. If there are scores, it should be better to use them, as they will achieve more sensitive computations, especially when we try to combine results of different methods. Scores of suggested queries may be think of another dimension of information, another vector. Let's assume the ranked list as in Table 2,

TABLE 2: Sample Query Suggestions for Rank Aggregation

Rank	Query Suggestion	Scores (using some method)
1	American airline	0.9
2	Airline tickets	0.2
3	Airline bus	0.1

If we only use ranks for this suggestion, the ordering value of second item is close to the first one. However, if we can use scores (if they exists/computable), then the ordering value of second item is much lower then the first one. This situation has no affect when we have only one method, since the ordering is not changed whether we use ranks or scores; however, if we have more than one method to combine/aggregate (especially when we have many methods/features) then the score values of query suggestions will become very important, very effective in the final sorting. That's why we introduced the "comparison of queries" idea and introduced/used scores of suggestions. As shown in Section 6 (Experiments), we also achieve higher success rates using score based rank aggregation.

5. Candidate Query Scoring and Comparison Techniques Exploiting Query Features

In this section, we describe several candidate query scoring and comparison techniques that we employed in our proposed framework. Each of these techniques computes a score for a candidate query and different orderings of candidates are obtained by this way. Our framework produces a hybrid ordering by a combination of scores for each ordering as we mention in the previous section. Candidate query scores can be computed in two different ways. We give the list of techniques/features that is used in this work below. The first group of techniques (1 through 5) computes a score by comparing candidate query to the initial query (q_i), such as number of sessions candidate query and initial query occurs together. On the other hand, second group of techniques (6 through 10) assigns scores only based on features of candidate queries such as its frequency, number of results, number of clicks, etc.

1. Hitting Time scores of queries (Mei et al., 2008)
2. Session count of queries
3. Session proximity of queries
4. Path frequency scores (4 different algorithm on this feature)
5. Grade similarity of candidate and initial query
6. Number of clicks for a candidate query
7. Frequency of a candidate query (as a measure of the popularity)
8. Number of search results for a candidate query (e.g., a very rare query or a meaningless query may return few or no results)
9. The number of users who submits a candidate query
10. Average dwell time for query results of a candidate query (How much time users staying on a result document)

Some features/properties we introduce/use are detailed in the following subsections. Other features such as number of clicks, number of search results, frequency etc. are self-explanatory. We remind that all these candidate query scoring and comparison techniques are employed in our query suggestion framework that is described in the previous section. There are a total of 13 algorithms exploiting these features.

5.1. Hitting Time Scores

We implemented and used Hitting Time algorithm as a baseline method for comparing to our new methods. We also used it in our hybrid algorithms as a sub-method. Apart from the hybrid algorithms, we did not use Hitting Time in our algorithms, that is, our other algorithms are not based on it.

Hitting Time scores are obtained using the algorithm defined in (Mei et al., 2008). In that work, a graph *G* (basically, query click graph) is constructed with set of queries (V_1), set of URL's (V_2) and edges (E) that connect these sets. If there is a click from a query i in V_1 to a URL k in V_2, this means that there is an edge from i to k and the weight of this edge is assigned as the number of clicks and is denoted as $w(i, k)$.

Given the initial (named as starting query in Mei et al. (2008)) query (q_s) that is submitted by a user, a subgraph is constructed by starting from the q_s and traversing the graph using depth first search (DFS) algorithm. The walk on the graph is ended when a predetermined number of (candidate) queries are discovered. In Section 6, we stop traversing the graph after reaching 300 queries. Note that, the original article in (Mei et al., 2008) uses the DFS as a graph traversal method. We test both the DFS method and the BFS method in the candidate selection and evaluate their performance in Section 6.

The probability of reaching from query i to query j (transition probability) in this subgraph is shown by the following equation:

$$p_{ij}=\sum_{k\in V_2}\left(\frac{w(i,k)}{d_i}\frac{w(k,j)}{d_k}\right)$$

where d_i and d_k are computed as:

$d_i=\sum_{j\in V_2} w(i,j)$ that is, sum of all clicks (frequencies) originating from the i'th query

$d_k=\sum_{i\in V_1} w(i,k)$ that is, sum of all clicks (frequencies) originating from all queries to k'th URL/document.

V_1: set of queries

V_2: set of URLs/documents

$w(i, k)$: click frequency from i'th query to k'th URL, $i\in V1, k\in V2$

For each candidate query, an h-score, which is initially zero, is computed in an iterative manner as follows:

$$h_i(t+1)=\left(\sum_{j\neq s} p_{ij}\, h_j(t)\right)+1$$

s: starting (initial) query, for which we are going to find suggestions.

After a predefined number of iterations, candidate queries having the smallest K h-scores are selected as query suggestions.

5.2. Algorithms Using Session Information

In this subsection, we describe two candidate query scoring algorithms exploiting sessions in query logs. We follow the general practice in the literature and form sessions using successive queries of a user in a 30 minutes time window. We give details of two algorithms as follows:

5.2.1. Session Count:

The number of sessions having both initial query (q_i) and candidate queries are counted for each candidate query. The basic intuition behind this method is that two queries in the same session are related to each other similar to (Yılmazel, 2011). The frequency of such sessions can be used as the score for candidate queries. Recall that we define score vectors (V) in Section 4.3, for each different candidate scoring algorithm. Score vector for session count method is shown below (assume 4 candidate queries for simplicity):

$$\text{Candidate Queries} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}$$

$$\text{Score Vector} = V = \begin{bmatrix} \text{Number of sessions having both } q_1 \wedge q_i \\ \text{Number of sessions having both } q_2 \wedge q_i \\ \text{Number of sessions having both } q_3 \wedge q_i \\ \text{Number of sessions having both } q_4 \wedge q_i \end{bmatrix}$$

5.2.2. Session Proximity

This is a similar and improved version of the session count method. The sensitivity of the previous algorithm is improved by calculating the proximity of two queries (initial query and candidate query) where both of them exist in the same sessions. The proximity of queries is defined as the number of query submissions between initial and candidate query (need to add 1 to prevent divide by zero). This method assigns higher score to candidate queries that are nearby to initial query in the sessions. This is inspired by the assumption that nearby queries in a session may be more related to each other than queries that are very far away in the session. Session proximity score for each candidate query q can be formulated as follows:

$$\text{Session Proximity Score}(q) = \sum_{\substack{k=1 \\ q_i \neq q \\ q_i \in S_k \wedge q \in S_k}}^{\text{number of sessions}} \frac{1}{|pos(S_k, q_i) - pos(S_k, q)|}$$

where *pos*($_{S_k,\ q_i}$) and *pos(Sk, q) are* positions of initial query (q_i) and candidate query (q) in session S_k, respectively.

We observe that this algorithm works reasonably well when there is sufficient session information. Although this algorithm generally works as expected, it may fail in certain situations such as if the user fixes an erroneous query, where both (erroneous and correct) queries reside in the same session. In that case, this method could give high score for the mistyped query. This “false positive” should be handled (by spelling correction or suitable techniques) separately and we leave this for future works.

5.3. Path Frequency Scores/Algorithms

In this subsection, we propose new candidate query scoring algorithms that exploit paths between initial query and candidate queries on the query click graph. In the following sections, we first mention how we define and construct paths and how we compute frequencies over these paths. Later, we describe our algorithms exploiting these “path frequencies”.

5.3.1. Path in Query Click Graph

We use query click graph to represent the clicking behavior of users. In the context of educational search engine log, each document is called a "learning object" (LO) that contains information for a learning objective. Hereafter, we refer to each clicked document as an LO. When a user searches for a query q_1 and clicks on LO_1, an edge from q_1 to LO_1 is constructed. If some other users search for q_2 and click on LO_1 then another edge from q_2 to LO_1 is also established. Assume that f_1 denotes the number of times users search for q_1 and click LO_1. Similarly, let f_2 denotes the frequency of clicks from query q_2 to LO_1. This constitutes a relation from q_1 to q_2, because two different queries have received a click on the same document/LO. We can denote this clicking path as follows:

$$q_1 -(f_1)\rightarrow LO_1 -(f_2)\rightarrow q_2$$

The path may be longer when multiple LO jump points exist in a path as following:

$$q_1 -(f_1)\rightarrow LO_1 -(f_2)\rightarrow q_2 -(f_3)\rightarrow LO_2 -(f_4)\rightarrow q_3$$

We omit LO information and denote the path as the following notation:

$$q_1 -(f_{mean1})\rightarrow q_2 -(f_{mean2})\rightarrow q_3$$

$$\textit{where } f_{mean1}=(f_1+f_2)/2$$

$$\text{and } f_{mean2}=(f_3+f_4)/2$$

We call this a click-path. We have three nodes, two edges (segments) in this click path. With this information, q_1 has a mutual relationship with q_3 to some extent. The basic motivation behind defining these click paths is to find paths between initial query (q_i) and candidate queries (q_c) on the query click graph (i.e., $q_i\rightarrow q_1 \rightarrow \ldots \rightarrow q_n \rightarrow q_c$), so that we can measure the relationship between initial and candidate queries. In the next subsection, we describe four different candidate query scoring algorithms exploiting frequencies and lengths of such paths.

5.3.2. Using Path Frequencies

In this subsection, we define different metrics/algorithms to measure the relationship between initial and candidate queries using click paths. The basic motivation behind these metrics is to exploit path frequencies and path lengths. First, we explain how frequencies affect the relationship with the following example. It is obvious that *q4* and *q6* is much more related than *q1* and *q3*, because number of clicks for common documents is much higher than the latter case. So, sum of frequencies is proportional to the degree of relationship.

path 1: $q_1 \xrightarrow{(2)} q_2 \xrightarrow{(3)} q_3$

path 2: $q_4 \xrightarrow{(12)} q_5 \xrightarrow{(15)} q_6$

Secondly, we investigate the path lengths for our metrics. Suppose that we have the following paths:

path 1: $q_1 \xrightarrow{(10)} q_2 \xrightarrow{(10)} q_3$

path 2: $q_4 \xrightarrow{(10)} q_5 \xrightarrow{(10)} q_6 \xrightarrow{(10)} q_7$

It can be thought that the relationship between q_1 and q_3 is stronger than the relationship between q_4 and q_7, since path 2 is longer than path 1 (assuming similar frequencies). Therefore, it is assumed that the degree of relationship is inversely proportional to the path length.

Note that it is possible to have multiple paths between initial and candidate queries through different documents or intermediate queries. Our metrics should also consider these multiple paths when computing a relationship score between candidate query and initial query. We propose four different path based algorithms as shown in Table 3, considering all these aspects. Algorithms 1 and 2 depend on a single path but Algorithms 3 and 4 support multiple paths. Single path algorithms use the first path that is found during graph traversal. Algorithms 1 and 2 give the same fixed weight of "one" to each path segment so they simply compute the sum of frequencies over the path. In the score formula, path(j) represents jth portion of path (jump point from one query to another or edge, segment). The frequency value (number of clicks) over the path(j) is denoted as Fr(path(j)) in the formula. However, Algorithms 3 and 4 assign different weights to each segment in the path which is inversely (exponentially) proportional to the position of path segment. The motivation behind this is that path segments which are closer to the initial query should have higher impact on the score compared to segments that are far away. The only difference between Algorithms 1 and 2 (also for Algorithms 3 and 4), is the path length component. The former algorithm divides the sum of frequency values to the path length, while the latter one uses the square of the path length.

TABLE 3. Path Frequency algorithms.

Path Frequency Algorithm	Uses Multi Path	Path Segment Weight	Formula
Algorithm 1	No	Fixed=1	$Score_1 = \frac{\left(\sum_{j=0}^{len(path)} Fr(path(j))\right)}{len(path)}$
Algorithm 2	No	Fixed=1	$Score_2 = \frac{\left(\sum_{j=0}^{len(path)} Fr(path(j))\right)}{(len(path))^2}$
Algorithm 3	Yes	Inversely proportional to the position of segment=2^{-j}	$Score_3 = \sum_{i=1}^{numberofpaths} \frac{\left(\sum_{j=0}^{len(path_i)} Fr(path_i(j)) \cdot 2^{-j}\right)}{len(path_i)}$
Algorithm 4	Yes	Inversely proportional to the position of segment=2^{-j}	$Score_4 = \sum_{i=1}^{numberofpaths} \frac{\left(\sum_{j=0}^{len(path_i)} Fr(path_i(j)) \cdot 2^{-j}\right)}{(len(path_i))^2}$

Here, we demonstrate computations for our Path Frequency based algorithms with examples. Assume that we have the following click path extracted from our Turkish educational search engine log:

açılarına göre üçgenler —(4.5)→ üçgen çizimi —(23.5)→ üçgen çeşitleri —(5.0)→ matematik noktaların birbirine göre uyumu —(3.5)→ paralel iki doğru *(triangles by angles —(4.5)→ drawing triangle —(23.5)→ types of triangles —(5.0)→ math harmony of points —(3.5)→ two parallel lines)*

Here path frequency values in the array format are [4.5, 23.5, 5.0, 3.5]. Algorithms 1 and 2 compute scores of 12.17 and 4.06 respectively. As seen in Table 3, Algorithm 2 gives more importance to the path length.

Secondly, we give two click paths from initial query "açılarına göre üçgenler" *(triangles by angles)* to the candidate query "geniş açı" *(wide angle)*, in order to demonstrate multi path algorithms (Algorihtms 3 and 4). We present each path and their scores for two different algorithms. The total scores for Algorithms 3 and 4 are computed as 8.62 and 3.33, respectively.

Path: 0

açılarına göre üçgenler —(4.5)→ üçgen çizimi —(23.5)→ üçgen çeşitleri —(5.5)→ geniş açı

(triangles by angles —(4.5)→ drawing triangle —(23.5)→ types of triangles —(5.5)→ wide angle)

Path frequencies: [4.5, 23.5, 5.5]

Algorithm 3 path value: 5.87

Algorithm 4 path value: 1.95

Path: 1

açılarına göre üçgenler —(4.5)→ üçgen çizimi —(2.0)→ geniş açı

(triangles by angles —(4.5)→ drawing triangle —(2.0)→ wide angle)

Path frequencies: [4.5, 2.0]

Algorithm 3 path value: 2.75

Algorithm 4 path value: 1.38

We note that multi path algorithms detect and use paths that are not circular. Our path detection step detects and eliminates circular paths. However, paths containing other paths (paths sharing common segments) are preserved.

5.3.3. The Difference of Path Frequency Algorithm to the Hitting Time Method

Both Path Frequency and Hitting Time algorithms uses query-URL click frequencies. To avoid confusion, we explain how they are operating different on the same data. The Hitting Time algorithm that we tested as a baseline method uses "transition probabilities" from one query to another. For example, let's assume we are traveling from query i (q_i) to URL k (url_k) (as in Section 5.1). In this case, the transition probability from query i to url k is computed like:

$$\text{Transition Probability} = \frac{w(i,k)}{d_i}$$

where d_i is computed as:

$d_i = \sum_{j \in V_2} w(i,j)$ that is, sum of all clicks (frequencies) originating from the *i*'th query,

and

w(i, k) = Click count (frequency) from query i to URL k.

That is, all clicks originating from q_i are taken into account, not surprisingly, as "probability" is computed. In our Path Frequency method, however, probabilities are not used, just click frequencies from q_i to url_k are used (considering multiple paths into account). We also weight click frequencies based on the distance to the initial query.

5.4. Educational Features Exploited for Query Suggestion

As we noted earlier, we experiment our query suggestion algorithms using an educational search engine log. Using educational features while developing these algorithms may improve the overall accuracy. We exploit grade and course features of queries. In the following subsections, we explain how we extract and use these features in our query suggestion algorithms. We note that other educational features such as school, age, curriculum etc. may also be utilized for the same purpose but we leave those features as the future work.

5.4.1. Grade Similarity

Educational search engine used in this study contains learning objects (documents) and each learning object is associated with one or more grades in K12 level. We try to predict the grade information of a query by analyzing its results (learning objects). The distribution of learning objects in the top-k result of the query for each grade is computed. This distribution shows that which grades are more likely for the query. We compute this grade distributions for the initial query (q_i) and candidate queries. Our motivation behind this feature is to promote candidate queries that have similar grade distribution with the initial query. In other words, we simply want candidate queries targeting the same grade as the initial query. We do not want to suggest queries for a different grade.

Grade similarity score for each candidate query q is computed by the below formula. This formula basically computes the dot product of grade distribution vectors of search results for candidate query q and initial query q_i. Outer summation accounts for each grade and inner summations compute the counts of search results associated for the k^{th} grade, both for candidate and initial queries. Our formula produces higher scores for queries having more "common" grades. It is expected to eliminate candidate queries that are of different student grade than the initial query, for an educational search engine.

$$\text{Grade Similarity Score}(q,q_i)=\sum_{k=1}^{ng}\left(\sum_{j=1}^{nr}F(q,k,j)\cdot\sum_{j=1}^{nr}F(q_i,k,j)\right)$$

where,

$$F(q,k,j)=\begin{cases}1 & \text{If search result document at rank j for query q is associated with grade k}\\ 0 & \text{Otherwise}\end{cases}$$

ng is the number of grades and nr denotes the number of search results (whichever is larger, q or q_i)

This algorithm may not be sufficient standalone for query suggestion, however, accuracy will improve when used in conjunction with other algorithms, as described in Section 4 and 4.3.

5.4.2. Course Matching

Educational search engines provide learning resources related to different courses such as math, science, social sciences and history. Our motivation here is to find a way to predict the target course for a query. If we have such a predictor, we can compare target courses of initial query and candidate queries, so we can enforce them to be the same course.

We follow a similar idea as in grade similarity. Each search result (learning object) is associated with only one course. We compute support count for each course based on search results of a query by below formula (support count for course *c*). Each document for a course contributes to its support count based on its rank in the search result list. Documents at high ranks have higher impact on support counts compared to the ones at low ranks.

$$Support_c(q)=\sum_{j=1}^{nr} G(q,c,j)$$

where,

$$G(q,c,j)=\begin{cases} \frac{1}{j} & \text{If the search result document at rank j for query q is associated with course c} \\ 0 & \text{Otherwise} \end{cases}$$

nr denotes the number of search results

Next, we normalize support counts for each course by computing the support ratios by the below formula. Finally, we predict the target course of a query if the maximum support ratio is higher than a predetermined threshold value (In our experiments, we use 0.8 as the threshold). If any of the support ratios does not satisfy this threshold, we could not identify the course.

$$\text{Support Ratio}_c(q)=\frac{Support_c(q)}{\sum_{c=1}^{nc} Support_c(q)}$$

where, nc denotes the number of courses

If we can identify the target course of the initial query (q_i) we can select candidate queries from the same course. Note that queries targeting different courses could be eliminated in different steps of our framework such as in query click graph construction, generic controls or final controls steps. It may be useful to eliminate such queries at the beginning while we traverse query click graph in order to construct a better candidate query set. The aim here is to give a chance to other candidates referring to the same course as the initial query instead of investing processing power for candidates for different courses.

Please note that if we cannot determine the target course for the initial query (if no course satisfies the support ratio threshold), we skip course matching step and does not filter candidate queries while traversing the query click graph.

We also evaluate the performance of QS algorithms without course filtering but achieved even lower success rates (these results are not given in Experiments section).

6. Experiments and Discussions

The framework and algorithms mentioned in previous sections are tested by generating suggestions for a set of queries randomly selected from our query log. The quality of suggestions is then evaluated by four human assessors. Experimental results are evaluated using two different metrics. We give the details of our dataset, experimental setup and experimental results in the following subsections.

6.1. Dataset

We use a sample query log of a Turkish educational search engine. Vitamin^{TM} is a commercial web-based educational framework that provides interactive educational content for K12 students in Turkey. Vitamin has more than 1.2 million registered users. Users can search for educational content on the web interface. Query log contains issued queries, clicked results and their ranks. (Usta et al., 2014) analyze query, session, user and click characteristics of this query log.

Our sample in this study consists of 2,028,395 query submissions of 52,713 users received for 6 months between December 2013 and May 2014. This sample contains 325,241 query submissions with at least one click. When we construct query click graph, queries with no clicks are not used.

However, session related features (session count and session proximity) are extracted from the whole set including queries with no clicks. We have 52,635 unique queries in our sample.

Even though our query suggestion framework may also work with algorithms that does not require a query log, in this study we use query log based query suggestion algorithms. Therefore quality and size of the query log can be crucial for performance of these algorithms. We estimate that the success rate of our algorithms will rise as the dataset size increases.

6.2. Experiment Procedure

In this subsection, we give details of our experiment procedure. Each step of our experiment is summarized as follows:

- Session reconstruction
- Random selection of frequently used, long tail and torso queries
- Query suggestions on these random queries, using eight different algorithms (We actually tested many more algorithms with different parameters, but included eight of them here)
- Evaluation of top-k query suggestions by four human assessors (k=10 currently)
- Calculating quality of suggestions according to algorithms, types of queries (head, torso, tail)
- Statistical significance tests

The following subsections describe each of these steps.

6.2.1. Reconstruction of query sessions

We reconstruct query sessions since session information is used in our algorithms. Successive queries of a user within a 30-minute time window are considered part of the same session, like previous approaches (Torres & Weber, 2011). If the time interval between new query and previous query for a specific user is longer than 30 minute, then it is considered as a new query session. Note that our session features (session count and session proximity) require initial query and candidate query to exist in the same session. Therefore we only focus on sessions with at least two queries. In our log, we extract 150,214 sessions having more than one query.

6.2.2. Query selection

The evaluation of query suggestions requires relevance judgments of human assessors. Since manual labeling is a time consuming process, we can measure the performance of various algorithms for a limited amount of queries. We randomly select 20 rarely used (long tail), 20 frequent queries (head) and 20 queries between them (torso) from the query log. Our aim here is to analyze performance of algorithms for queries with varying frequency. Queries are considered rare, torso and frequent according to the criteria shown in Table 4. That is, queries with frequency between 5 and 20 are considered “rare” etc. Queries with frequency less than 5 are observed and assumed to be erroneous or useless and are not taken into account.

TABLE 4. Query categorization/selection criteria.

Query Type	Minimum frequency	Maximum frequency
Rare (long tail)	5	20
Torso	21	500
Frequent (Head)	501	∞

6.2.3. Algorithms used in experiments

In this work, we suggest/work on 10 different query scoring and suggestion algorithms (or 13 including different types of Path Frequency algorithms) along with 4 different aggregation methods of those query suggestion algorithms. One can use any combination of these query suggestion algorithms and aggregation methods. In this work, the following eight algorithms/hybrid combinations are experienced and shown in our evaluation results:

1. Non-Hybrid Methods:
   a) Hitting Time (5.1) with Depth First Search (DFS) graph traversal method. Note that, in addition to the original method, we apply generic controls (4.2), (HT-DFS)

b) Hitting Time (5.1) with Breadth First Search (BFS) graph traversal method, (HT-BFS)

c) Path Frequency-3 Algorithm, mentioned in Section 5.3, (PF3)

d) Path Frequency-4 Algorithm, mentioned in Section 5.3, (PF4)

2. Hybrid methods:

   a) Hybrid-1-BFS method (4.3.1.1),

   b) Hybrid-2-Borda Count method (4.3.1.2.1),

   c) Hybrid-3-Weighted Borda Count method (4.3.1.2.2),

   d) Hybrid-4-Weighted Voting method (4.3.1.2.3),

Hitting Time algorithm is proposed by (Mei et al., 2008), but we propose to use BFS as the graph traversal method while searching for candidate queries. We show that BFS traversal produce better suggestions compared to DFS. The remaining algorithms are proposed in this study. For all algorithms, we use the generic controls procedure to eliminate some meaningless or erroneous queries, as described in Section 4.2. It is also applied to the Hitting Time algorithm, which makes it slightly different than the original method in the article (Mei et al., 2008).

We produce top-10 query suggestions for each query in our sample of 60 queries (20 rare, 20 torso and 20 frequent) using each of query suggestion algorithms. Therefore, we have different query suggestions for the same set of 60 queries, for each algorithm we tested. The relevance scores of each query is computed separately for each algorithm that is tested. These suggestions are evaluated by four human assessors. The assessors are 2 Computer Science PhD students, a high school teacher and a faculty member with a PhD degree, their ages are between 30-40 years. All assessors are familiar with search engines.

6.2.4. Parameter tuning for use in hybrid method(s)

All of our hybrid methods except the Borda Count method need coefficient (weight) parameters to be set. We experiment with the following parameter tuning methods:

6.2.4.1. Parameter tuning using cross validation

We used cross validation technique in order to determine parameters. We had 60 queries in the evaluation data set. We split the dataset into 4 pieces, each containing 15 queries. We evaluated and measured the performance of candidate query scoring methods (sub-methods that are mentioned in Section 5) on each piece of 15-queries dataset and used average relevance scores of methods as their coefficients in the hybrid method. We then evaluated the performance of hybrid method(s) with the determined coefficients on the remaining queries (3 pieces of 15-queries dataset, total of 45). We repeat this for all pieces and computed the average performance for hybrid methods. However, we do not obtain better average relevance values than the success rate of Path Frequency-3 (an underlying method in Hybrid methods) using cross validation, hence we conduct manual parameter tuning for these 3 hybrid methods.

6.2.4.2. Manual parameter tuning

We try different parameters by trial and error, based on our experiences and observations on the produced query suggestions. Since there are 13 different sub-algorithms (hence parameters) to be set, it is infeasible to test all combinations of parameter values. Therefore, we apply the following strategy:

We have the success rates of 13 different sub-algorithms obtained during cross validation parameter tuning method (by means of 15-query sets). Using this information, we test primarily the parameters of best sub-algorithms in a reasonable range for top-4 (Path Frequency-3, Hitting Time, Session Proximity, Click Counts) sub-algorithms, determine best parameters for each of Hybrid-1, Hybrid-3 Weighted Borda Count and Hybrid-4 Weighted Voting methods (Hybrid-2 Borda Count has static parameters of all 1).

It is possible to use/develop other parameter tuning methods to achieve higher success rates, however this is not in the scope of this work and left as a future work.

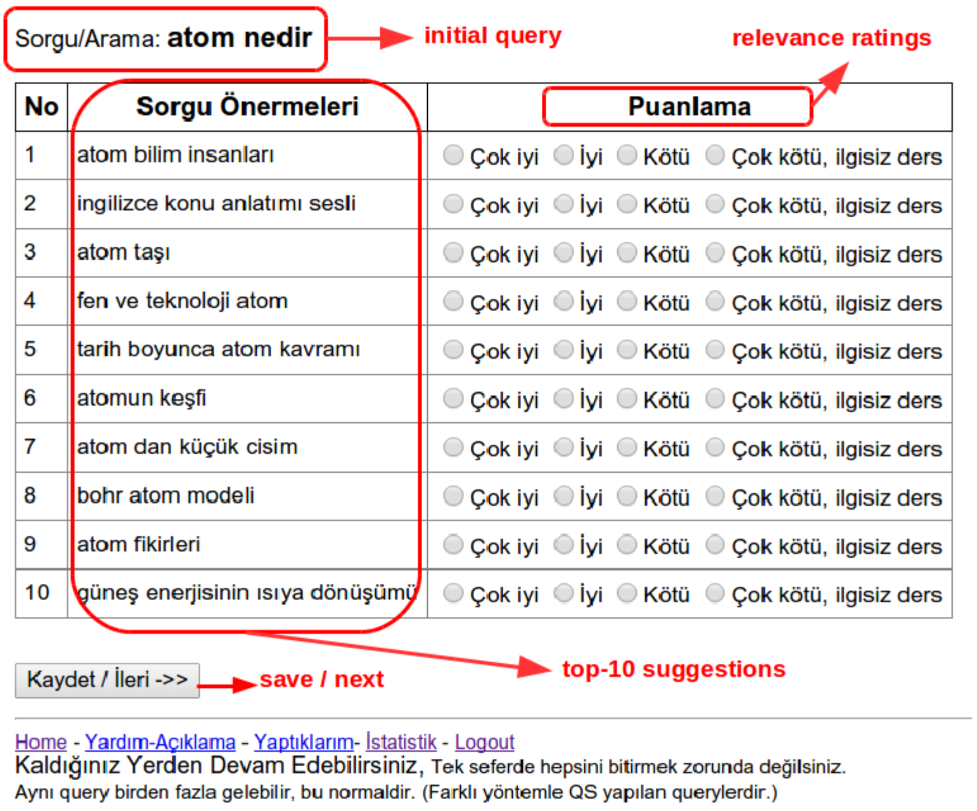
6.2.5. Evaluation setup

We prepare a simple web interface as shown in Figure 2 (Queries and relevance grades are written in Turkish) for evaluating each query suggestion algorithm. Assessors have to login to the web interface with a user name. We have four human assessors and each of them judges the relevance of top-10 suggestions of queries produced by algorithms. The assessors were unaware of the query suggestion algorithm while assessing the suggestions. This leads to a more accurate assessment, as assessors were neutral to the query, knowing nothing about the query type (head, torso and longtail) and algorithm used.

As shown in Figure 2, we use a four point assessment such that each suggestion is judged as "Highly Relevant" *(Çok iyi)*, "Fairly Relevant" *(İyi)*, "Marginally Relevant" *(Kötü)* and "Irrelevant at all, wrong course" *(Çok kötü, ilgisiz ders)*. Relevance levels for each judgment are listed in Table 5.

TABLE 5. Relevance levels of each assessment for the evaluation.

Relevance level	Assessment
0	Irrelevant at all, wrong course
1	Marginally relevant, at least same course
2	Fairly relevant
3	Highly relevant


Sorgu/Arama: **atom nedir**

No	Sorgu Önermeleri	Puanlama
1	atom bilim insanları	○ Çok iyi ○ İyi ○ Kötü ○ Çok kötü, ilgisiz ders
2	ingilizce konu anlatımı sesli	○ Çok iyi ○ İyi ○ Kötü ○ Çok kötü, ilgisiz ders
3	atom taşı	○ Çok iyi ○ İyi ○ Kötü ○ Çok kötü, ilgisiz ders
4	fen ve teknoloji atom	○ Çok iyi ○ İyi ○ Kötü ○ Çok kötü, ilgisiz ders
5	tarih boyunca atom kavramı	○ Çok iyi ○ İyi ○ Kötü ○ Çok kötü, ilgisiz ders
6	atomun keşfi	○ Çok iyi ○ İyi ○ Kötü ○ Çok kötü, ilgisiz ders
7	atom dan küçük cisim	○ Çok iyi ○ İyi ○ Kötü ○ Çok kötü, ilgisiz ders
8	bohr atom modeli	○ Çok iyi ○ İyi ○ Kötü ○ Çok kötü, ilgisiz ders
9	atom fikirleri	○ Çok iyi ○ İyi ○ Kötü ○ Çok kötü, ilgisiz ders
10	güneş enerjisinin ısıya dönüşümü	○ Çok iyi ○ İyi ○ Kötü ○ Çok kötü, ilgisiz ders

Kaydet / İleri ->>

Home - Yardım-Açıklama - Yaptıklarım- İstatistik - Logout
Kaldığınız Yerden Devam Edebilirsiniz, Tek seferde hepsini bitirmek zorunda değilsiniz.
Aynı query birden fazla gelebilir, bu normaldir. (Farklı yöntemle QS yapılan querylerdir.)

FIG. 2: Web-based GUI of the query suggestion evaluation system

6.3. Experimental Results

In this subsection, we present our experimental results. We first analyze the agreement among four human assessors using Cohen's Kappa metric. Next, we investigate the performance of query suggestion algorithms using two different metrics such as average relevance level and normalized discounted cumulative gain (NDCG). Finally, we perform statistical significance tests over our results.

6.3.1. Assessor agreement

Cohen's weighted Kappa statistic (Cohen, 1968) measures the agreement among two raters in a user study. We calculate the weighted Kappa for each pair of our assessors (for all eight methods we worked on) and then take the average. The average pairwise Cohen's Kappa value is 0.37 which is considered to be a fair agreement (Landis & Koch, 1977). In this test, results below zero are considered as no agreement, above zero is considered some agreement and 1 is the ideal situation which presents the perfect agreement (If the agreement is by chance then Kappa value is zero). Our test shows some degree of agreement among our raters. The interpretation of kappa agreement is shown in Table 6 (Landis & Koch, 1977). According to this table, the kappa value of 0.37 is close to the upper boundary of the kappa classification, which is close to "moderate agreement".

TABLE 6. Inter-rater Agreement for Interpretation of Kappa

Kappa value (k)	**Agreement**
< 0	Poor
0 – 0.20	Slight
0.21 – 0.40	Fair
0.41 – 0.60	Moderate
0.61 – 0.80	Substantial
0.81 – 1.00	Almost perfect

6.3.2. Performance of algorithms

We compare the performance of query suggestion algorithms using two different metrics. The first metric simply computes the average relevance level (ranges between 0 and 3) for each algorithm based on assessor ratings. We also analyze the success rates for queries with varying frequencies (head, torso, tail). As a second metric, we employ normalized discounted cumulative gain (NDCG) measure in order to take the ranking of query suggestions into account.

6.3.2.1. Performance of algorithms using average relevance

We compute the average relevance of each query of each algorithm, then we compute the average of those relevance scores for each algorithm. The list of abbreviations of algorithms that are used in figures and other tables are shown in Table 7.

Figure 3 shows average relevance scores of query suggestion algorithms for 60 queries in our experiments. It is seen that Hitting Time-DFS has the lowest success rate. Our path frequency based algorithms and hybrid algorithms achieve the highest average relevance levels. Path Frequency-3 is the best algorithm among non-hybrid methods, while Hybrid-1 (with weighted score based aggregation) is the best algorithm among all.

We plot the performance gain, as percent of new algorithms compared to the baseline Hitting Time-DFS method in Figure 4. It is seen that using BFS graph traversal in candidate selection step improves the Hitting Time method by 66%. Path frequency based algorithm that we propose in this paper considerably increases the average relevance level of the baseline, by 81%. Finally, our hybrid algorithm that combines several features and methods (Hybrid-1 BFS) achieves the highest improvement with 90% relative increase. This shows that Hybrid methods that rely on score based aggregation achieves higher relevance levels than Hybrid Methods that rely on rank based aggregation, hence we primarily suggest using score based aggregation as in Section 4.3.1.1.

In our experiments, we observe that new algorithms eliminate most of the “meaningless/erroneous” queries that were suggested by the baseline method, even when the generic controls routine (Section 4.2) was not used.

The statistical significance tests of average relevance scores that are shown in Figure 3 are given in Section 6.3.3.

TABLE 7. List of Abbreviations of Algorithms Used

Abbreviation	Algorithm
HT-DFS	Hitting Time Algorithm using Depth First Search (DFS) as graph traversal method (original Hitting Time method)
HT-BFS	Hitting Time Algorithm using Breath First Search (BFS) as graph traversal method (modified Hitting Time method)
PF3-BFS	Path Frequency-3 method (5.3)
PF4-BFS	Path Frequency-4 method (5.3)
Hybrid-1	Hybrid algorithm using BFS and weighted score based aggregation (4.3.1.1)
Hybrid-2-Bo	Hybrid algorithm using BFS and Borda Count method as rank aggregation (4.3.1.2.1)
Hybrid-3-WBo	Hybrid algorithm using BFS and Weighted Borda Count method as rank aggregation (4.3.1.2.2)
Hybrid-4-WV	Hybrid algorithm using BFS and Weighted Voting method as rank aggregation (4.3.1.2.3)

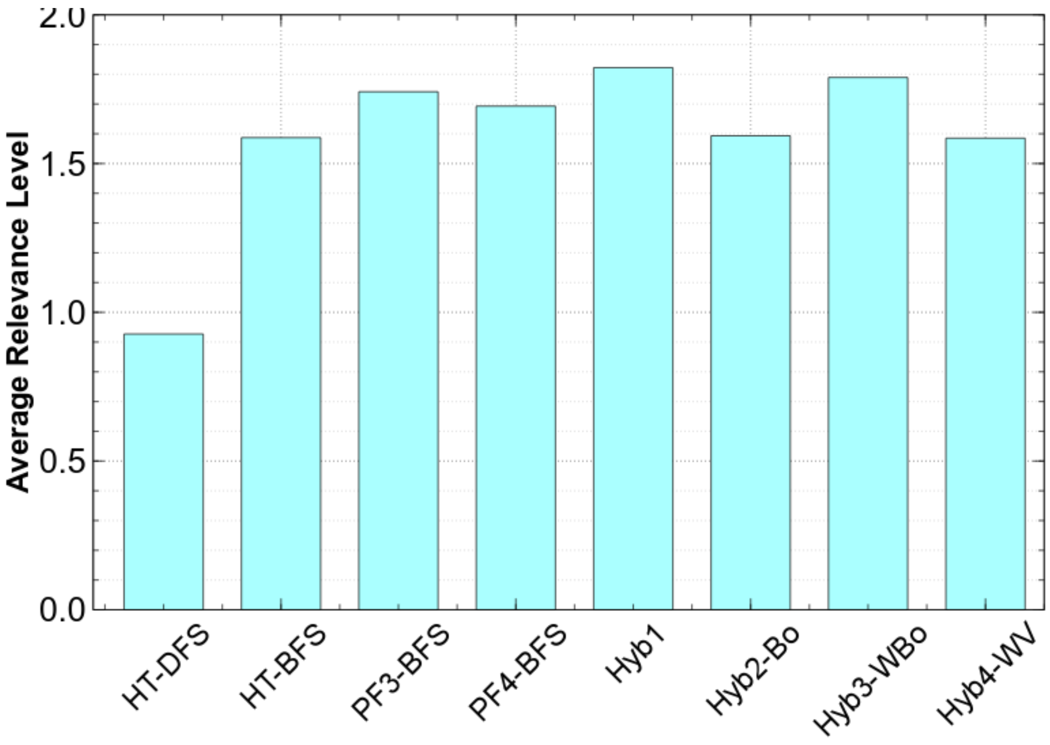


FIG. 3. Average relevance levels for query suggestion algorithms

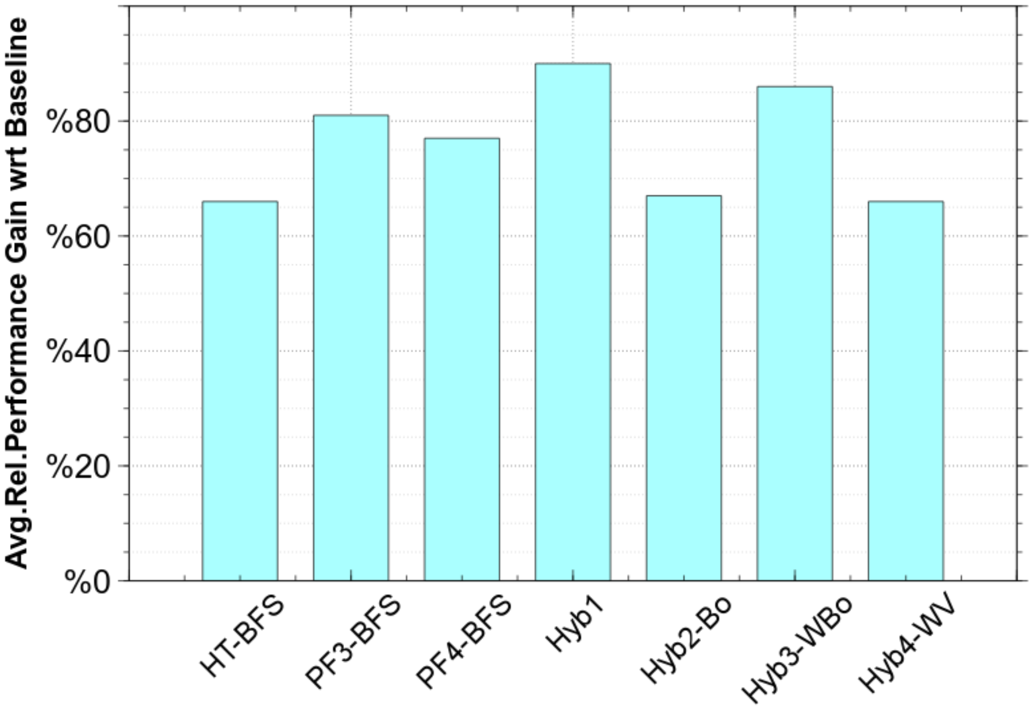


FIG. 4. Percentage performance gain of new algorithms (in average relevance level) with respect to Hitting Time-DFS baseline.

Next, we evaluate performance of query suggestion algorithms for head, torso and tail queries.

Figure 5 shows average relevance scores of algorithms for these types of queries, separately. As the first finding, query suggestions for head (frequent) queries have the highest relevance scores for all methods. This can be attributed to the size of available click data for these queries compared to torso and tail queries. All new method's success rates are higher than the baseline method. However, Hitting Time-BFS seems very close to the Hybrid-1 for frequently used (head) queries. Hitting Time-BFS is the slightly modified version of the original method, which just uses Breadth First Search as graph traversal method. Regarding this, the remaining new methods (other than Hybrid-1) have no advantage for head queries, with respect to the Hitting Time-BFS method. Torso and tail queries show similar (close to each other) performances within each algorithm. However, it is interesting to see that, for Path Frequency and Hybrid methods, the success of tail queries are higher than torso queries (within that method), while tail queries have lowest success for both Hitting Time (DFS and BFS) methods. This might be attributed to the fact that combining multiple methods increases the success for the the tail queries, as we suggest in our framework.

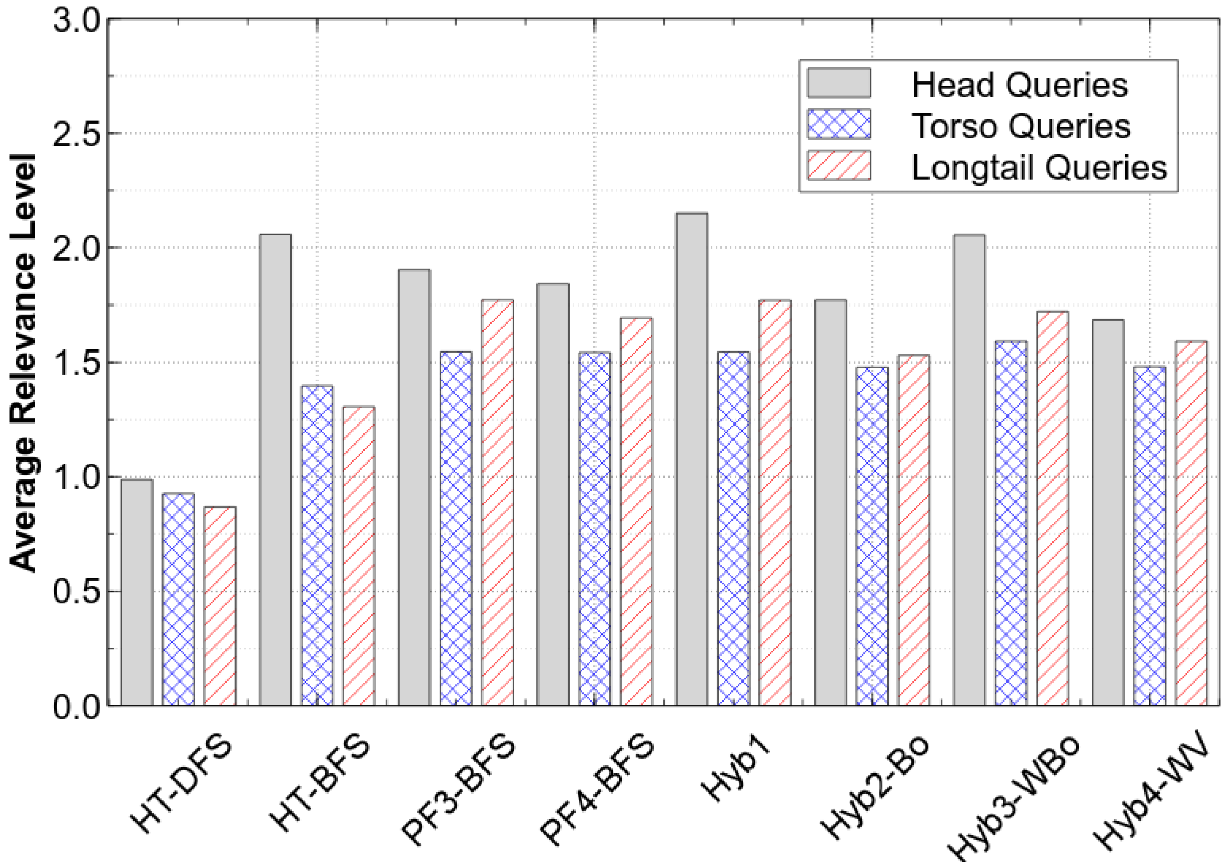


FIG. 5. Average relevance levels of query suggestion algorithms for head, torso and tail queries

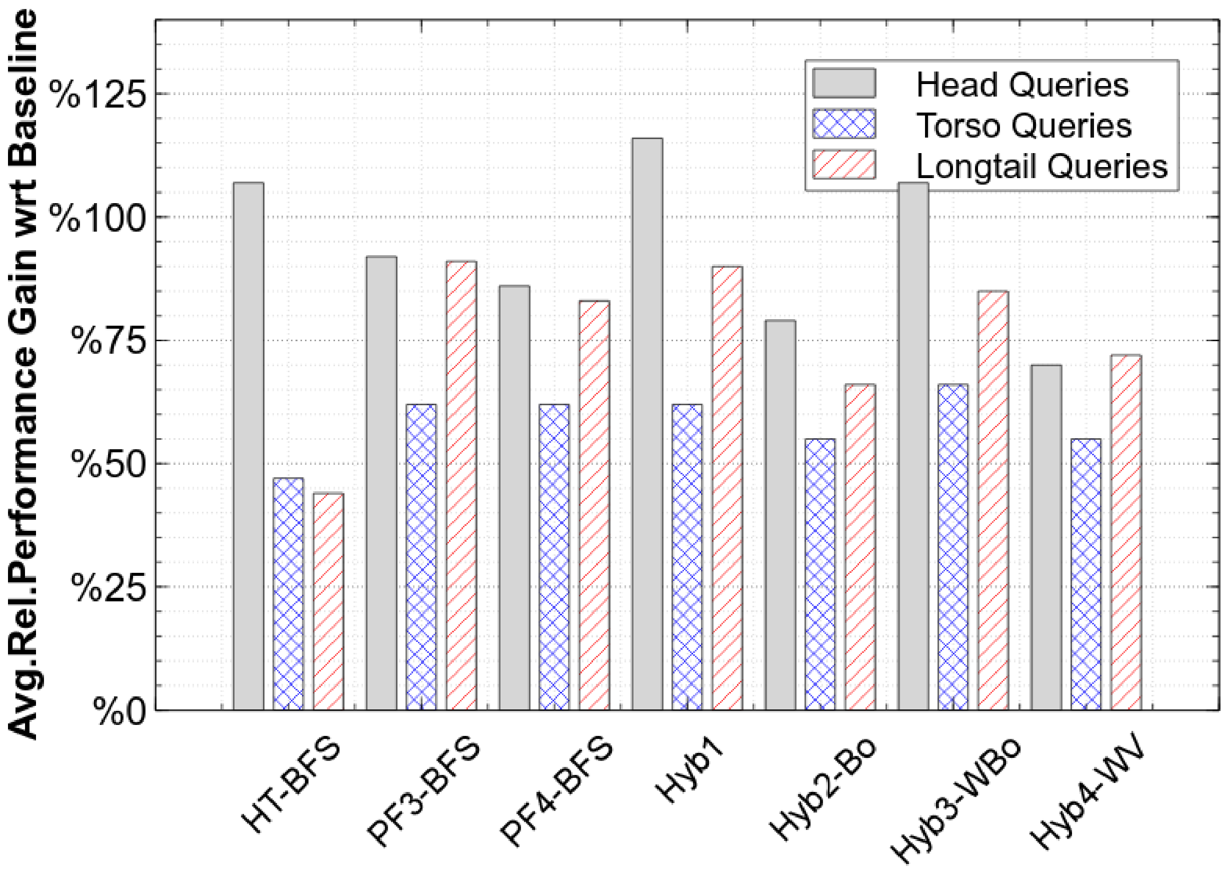


FIG. 6. Percentage performance gain of new algorithms (in average relevance level) for head, torso and tail queries, separately.

We plot the performance gain of new algorithms compared to the baseline for head, torso and tail queries in Figure 6. As we see, the highest performance gain is observed for head queries mostly and the highest performance gain for head queries is observed for Hybrid-1-BFS method. It may also notable to see the performance gain for head and longtail queries for PF3-BFS algorithm is very similar, which shows the success of this algorithm.

Hitting Time-BFS method increase average relevance level by around 107% for head and 47% for torso queries. The improvements for Path Frequency and Hybrid algorithms show similar trend such that they increase relevance level for torso queries by around 50-60%. All Path Frequency and Hybrid algorithms have higher improvements for torso and tail queries, while only Hybrid-1 is higher for head queries, with respect to the performance improvement of Hitting Time-BFS.

Please note that, in all methods (including the original and modified Hitting Time-BFS), we run "generic controls" routine (which includes course checking) to further improve QS performance and eliminate obvious errors.

6.3.2.2. Performance of algorithms using NDCG

As the second evaluation metric, we use NDCG to compare performance of query suggestion algorithms. This well-known metric in web search literature, also accounts for ranks of results in addition to the relevance level. Figure 7 shows performance of algorithms in NDCG metric for top-10 query suggestions. We compute NDCG values for a query using relevance levels (between 0-3) of four human assessors, separately. Then we compute the average of four NDCG values.

We can see the higher success rates for all new suggested algorithms for NDCG metric too. The Path Frequency-3 algorithm achieves success rates very similar to highest one, Hybrid-1-BFS, as like average relevance metric. Figure 8 shows the increase of performance of new algorithms using NDCG metric. We can see that Path Frequency and Hybrid-1 algorithms achieve high performance gain. However, Hybrid Borda Count and Hybrid Weighted Voting algorithms have very similar performance gain with respect to HT-BFS. This may show us that, rank aggregation methods of Borda Count and Weighted Voting may not achieve much performance gain with respect to score based rank aggregation, unless other modifications are made.

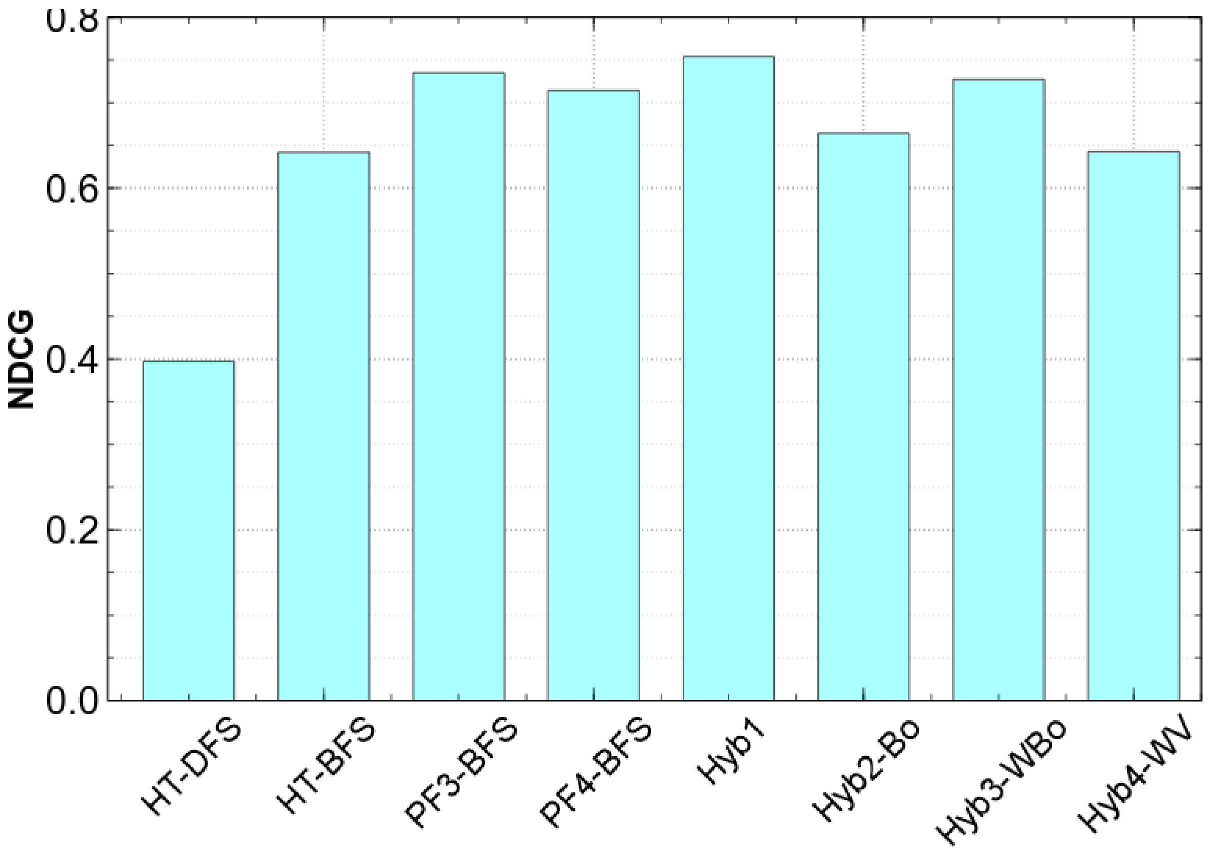


FIG. 7. NDCG performance of algorithms

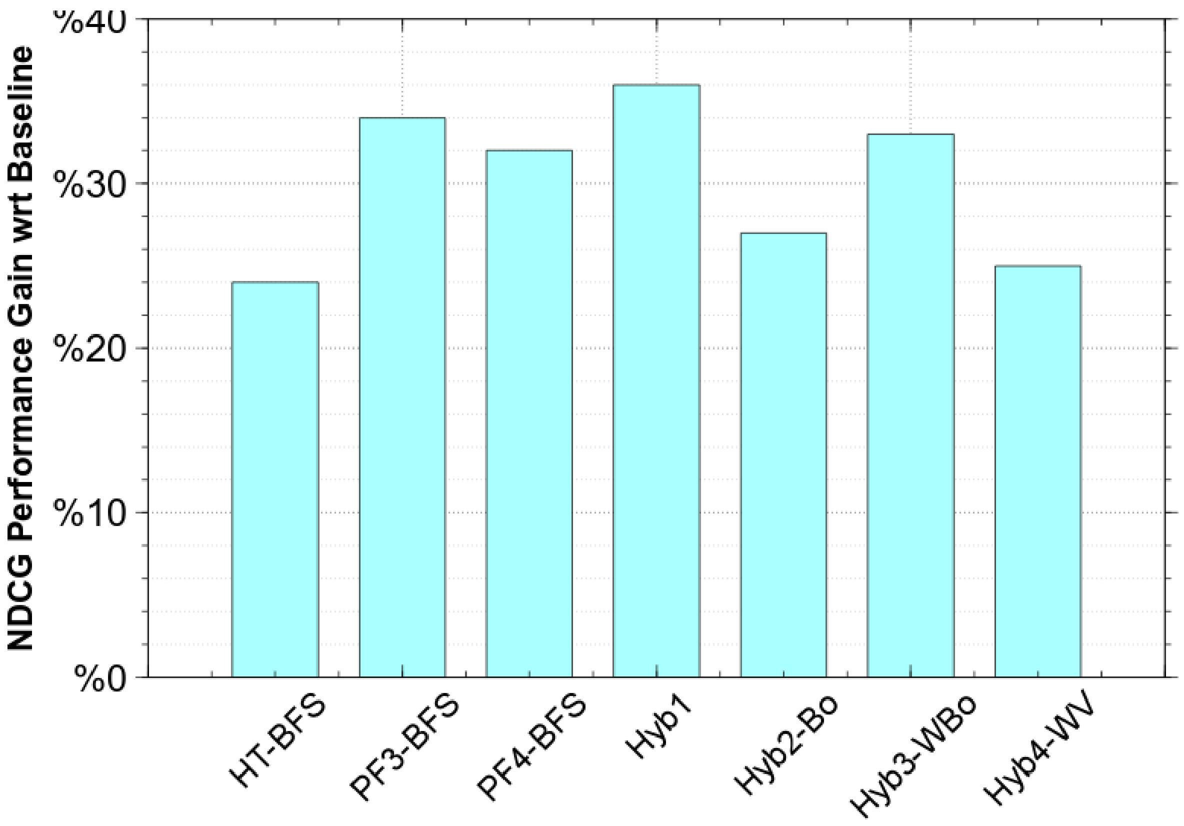


FIG. 8. Percentage performance gain of new algorithms (in NDCG metric) with respect to Hitting Time-DFS baseline.

6.3.3. Statistical Significance Tests

To assess and validate the results of the experiment, we run statistical significance tests. We have eight different query suggestion algorithms. We run paired t-tests to compare our algorithms to the baseline method, Hitting Time-DFS, using average relevance scores of 60 queries as our sample.

Our null hypothesis here is that there is no statistically significant difference between relevance levels or NDCG values of two algorithms. We present the results of these tests for 8 pairs of algorithms in Table 8.

It is seen that all new suggested methods obtain statistically significant difference in relevance level compared to the baseline method (Hitting Time-DFS). On the other hand, when we compare two leading methods (Path Frequency-3 and Hybrid-1-BFS) with Hitting Time-BFS to measure their improvement than the slightly modified version of the baseline method (Hitting Time-BFS), we can see that, there are still significant improvement in the new methods. It is seen than there is no statistical significant difference between HT-BFS and our PF3 algorithm based on average relevance level.

TABLE 8. Results of paired t-tests for some of pairs of query suggestion algorithms.

Algorithm 1	Algorithm 2	p-value for NDCG	p-value for Average Rel.	Meaning for NDCG	Meaning for Average Rel.
Hitting Time-DFS	Path Freq-3 BFS	5.58×10^{-13}	3.85×10^{-12}	Very Significant improvement	Very Significant improvement
Hitting Time-DFS	Path Freq-4 BFS	1.30×10^{-12}	3.07×10^{-13}	Very significant improvement	Very significant improvement
Hitting Time-DFS	Hybrid-1 BFS	8.22×10^{-14}	7.68×10^{-13}	Very Significant improvement	Very Significant improvement
Hitting Time-DFS	Hybrid-2 Borda Count	4.09×10^{-11}	5.90×10^{-13}	Very Significant improvement	Very Significant improvement
Hitting Time-DFS	Hybrid-3 Weighted Borda Count	3.84×10^{-12}	1.39×10^{-14}	Very Significant improvement	Very Significant improvement
Hitting Time-DFS	Hybrid-4 Weighted Voting	1.92×10^{-10}	2.75×10^{-12}	Very Significant improvement	Very Significant improvement
Hitting Time-BFS	Path Freq-3 BFS	0.007	0.123	Significant improvement	No Significant improvement
Hitting Time-BFS	Hybrid-1 BFS	0.001	0.0381	Significant improvement	Significant improvement

7. Conclusion

In this study, we show that query suggestion problem can be reduced (simplified) to a query comparison problem. We establish a modular, well-defined framework that may contain many other query suggestion/comparison algorithms. Our framework is also important for practical purposes such that one can contribute to the query suggestion field by developing a superior method for a specific step, without having to deal with all aspects. We also design and implement 13 different algorithms utilizing educational and other features of queries, using our framework. Our experiments use a real life educational search engine log. We conduct an experiment in order to measure the relevance of query suggestions and show that new algorithms possess higher success rates (of about 66-90%) than the baseline method. This work also demonstrates that reduction and simplification of the query suggestion problem makes developing and combining query suggestion algorithms easy and practical.

As a future work, it is possible to exploit additional educational features such as related curriculum items, user demographic features etc. Including a spell checker would increase the accuracy of the suggestions. We also plan to integrate content-based query suggestion methods that do not rely on query logs into our framework.

8. Acknowledgments

This work is supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under grant no. 113E065.

9. References

Arora, M., & Duhan, N. (2013). Design of query suggestion system using search logs and query semantics. *International Journal of Application or Innovation in Engineering & Management (IJAIEM)*, *2*(6). Retrieved from http://www.ijaiem.org/Volume2Issue6/IJAIEM-2013-06-25-075.pdf

Aslam, J. a, & Montague, M. (2001). Models for Metasearch. *Acm Sigir*, 276–284. http://doi.org/10.1145/383952.384007

Babuscu, F., & Özcan, R. (2014). How students use search engines for school informational tasks, In Proceedings of International Conference on e-Education, (pp. 97-107).

Baeza-Yates, R., Hurtado, C., & Mendoza, M. (2005). Query recommendation using query logs in search engines (pp. 588–596). Springer.

Baeza-Yates, R., Hurtado, C., & Mendoza, M. (2007). Improving search engines by query clustering. *Journal of the American Society for Information Science and Technology*, *58*(12), 1793–1804. http://doi.org/10.1002/asi.20627

Bhatia, S., Majumdar, D., & Mitra, P. (2011). Query suggestions in the absence of query logs (pp. 795–804). ACM.

Boldi, P., Bonchi, F., Castillo, C., Donato, D., & Vigna, S. (2009). Query suggestions using query-flow graphs (pp. 56–63). ACM. Retrieved from http://dl.acm.org/citation.cfm?id=1507518

Borda, J. C. (1781). Memoire sur les elections au scrutin. *Histoire de l'Academie Royale Des Sciences*.

Bordogna, G., Campi, A., Psaila, G., & Ronchi, S. (2012). Disambiguated query suggestions and personalized content-similarity and novelty ranking of clustered results to optimize web searches. *Information Processing & Management*, *48*(3), 419–437. http://doi.org/10.1016/j.ipm.2011.03.008

Cao, H., Jiang, D., Pei, J., He, Q., Liao, Z., Chen, E., & Li, H. (2008). Context-aware query suggestion by mining click-through and session data (pp. 875–883). ACM.

Cui, H., Wen, J.-R., Nie, J.-Y., & Ma, W.-Y. (2002). Query expansion for short queries by mining user logs. *IEEE Trans. Knowl. Data Eng*, *15*(4), 829–839. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.5627&rep=rep1&type=pdf

Dwork, C., Kumar, R., Naor, M., & Sivakumar, D. (2001). Rank aggregation methods for the web (pp. 613–622). ACM. Retrieved from http://dl.acm.org/citation.cfm?id=372165

Fonseca, B. M., Golgher, P. B., de Moura, E. S., & Ziviani, N. (2003). Using association rules to discover search engines related queries (pp. 66–71). IEEE.

Huang, C.-K., Chien, L.-F., & Oyang, Y.-J. (2003). Relevant term suggestion in interactive web search based on contextual information in query session logs. *Journal of the American Society for Information Science and Technology*, *54*(7), 638–649. Retrieved from http://onlinelibrary.wiley.com/doi/10.1002/asi.10256/full

Internet Society. (2015). *Internet society global internet report 2014*. Retrieved from https://www.internetsociety.org/sites/default/files/Global_Internet_Report_2014_0.pdf

Jansen, B. J., & Spink, A. (2006). How are we searching the world wide web? A comparison of nine search engine transaction logs. *Information Processing & Management*, *42*(1), 248–263. http://doi.org/10.1016/j.ipm.2004.10.007

Kraft, R., & Zien, J. (2004). Mining anchor text for query refinement (pp. 666–674). ACM.

Landis, J. R., & Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, *33*(1), 159–174.

Lau, T., & Horvitz, E. (1999). Patterns of search: analyzing and modeling web query refinement. *Courses and Lectures-International Centre for Mechanical Sciences*, 119–128. Retrieved from ftp://131.107.65.22/pub/ejh/queryrefine.pdf

Lewis, D. D., & Croft, W. B. (1990). Term clustering of syntactic phrases. *Proceedings of ACM SIGIR-90*.

Mei, Q., Zhou, D., & Church, K. (2008). Query suggestion using hitting time (pp. 469–478). ACM.

Renda, M. E., & Straccia, U. (2003). Web Metasearch: Rank vs. Score Based Rank Aggregation Methods. *Proceedings of the 2003 ACM Symposium on Applied Computing - SAC '03*, 841–846. http://doi.org/10.1145/952532.952698

Silvestri, F. (2010). Mining Query Logs: Turning Search Usage Data into Knowledge. *Foundations and Trends® in Information Retrieval*, *4*(1-2), 1–174. http://doi.org/10.1561/1500000013

Torres, S. D., Hiemstra, D., Weber, I., & Serdyukov, P. (2012). Query recommendation for children (pp. 2010–2014). ACM. Retrieved from http://doc.utwente.nl/84325/1/cikm12children.pdf

Torres, S. D., & Weber, I. (2011). What and how children search on the web (pp. 393–402). ACM.

Torres, S. D., Weber, I., & Hiemstra, D. (2014). Analysis of search and browsing behavior of young users on the web. Retrieved from http://ingmarweber.de/wp-content/uploads/2014/02/Analysis-of-Search-and-Browsing-Behavior-of-Young-Users-on-the-Web.pdf

Usta, A., Altingovde, I. S., Vidinli, İ. B., Ozcan, R., & Ulusoy, Ö. (2014). How k-12 students search for learning? Analysis of an educational search engine log (pp. 1151–1154). ACM Press. http://doi.org/10.1145/2600428.2609532

Wang, X., & Zhai, C. (2008). Mining term association patterns from search logs for effective query reformulation (pp. 479–488). ACM.

Wen, J.-R., Nie, J.-Y., & Zhang, H.-J. (2001). Clustering user queries of a search engine (pp. 162–168). ACM. Retrieved from https://research.microsoft.com/en-us/people/jrwen/qc-www10.pdf

Yılmazel, Ö. (2011). Guiding students to answers: query recommendation. *Turkish Online Journal of Distance Education*, *12*(3), 85–94.

10. Vitae

10.1. İ. Bahattin Vidinli

İ.Bahattin Vidinli is a PhD candidate in the Computer Engineering Department of Turgut Ozal University in Ankara, Turkey. He received MS in Information Systems in Gazi University, Ankara, Turkey. Worked in the Information and Computer Technologies in private and public sector as project manager, software developer and systems designer; worked in the fields of information processing, cyber security, system and web based software development. Contact him at the Department of Computer Engineering, Turgut Ozal University, Ankara, Turkey; bahattin@vidinli.com.

10.2. Rıfat Özcan

Rıfat Özcan is an assistant professor in the Computer Engineering Department of Turgut Özal University in Ankara, Turkey. His research interests include Web search engines, caching, and information retrieval. He received his MS in University of Texas at Arlington, PhD degree in Bilkent University, Ankara, Turkey. Contact him at the Department of Computer Engineering, Turgut Ozal University, Ankara, Turkey; rozcan@turgutozal.edu.tr.