Query Generation as Result Aggregation for Knowledge Representation

Matthew Mitsui
Department of Computer Science
Rutgers University
New Brunswick, NJ, USA, 08901
mmitsui@cs.rutgers.edu

Chirag Shah
School of Communication and Information
Rutgers University
New Brunswick, NJ, USA, 08901
chirags@rutgers.edu

Abstract

Knowledge representations have greatly enhanced the fundamental human problem of information search, profoundly changing representations of queries and database information for various retrieval tasks. Despite new technologies, little thought has been given in the field of query recommendation – recommending keyword queries to end users – to a holistic approach that recommends constructed queries from relevant snippets of information; pre-existing queries are used instead. Can we instead determine relevant information a user should see and aggregate it into a query? We construct a general framework leveraging various retrieval architectures to aggregate relevant information into a natural language query for recommendation. We test this framework in text retrieval, aggregating text snippets and comparing output queries to user generated queries. We show that an algorithm can generate queries more closely resembling the original and give effective retrieval results. Our simple approach shows promise for also leveraging knowledge structures to generate effective query recommendations.

1. Introduction

Knowledge mining has recently become an important component of text-based technologies and has shown us a new way to represent information on the Web. It has shown us how to extract entities – such as actors, presidents, or cities – from plain text, to quickly discover facts about those entities, and to compile or infer relationships about a single or multiple entities by operating on a knowledge graph. Knowledge mining has shown us how to enhance traditional Web search by supplementing queries and documents with additional latent structure.

In Web retrieval, the driving method of input has largely remained the same: a single text box for a natural language query. Yet now, knowledge graphs enhance and expand natural language queries into queries that can be issued to a knowledge base. The enhanced queries often return objects (e.g., actors in a movie) and relationships (e.g., people related to an actor) to supplement the "10 blue links" paradigm of traditional Web retrieval.

While knowledge representations have opened the discussion of backend representations for a variety of search tasks, query recommendation – the study of recommending relevant, related queries to an end user during a search task – has drawn little from this discussion. Query logs – i.e., logs of user search sessions – provide rich data for query recommendation algorithms. But recommender algorithms only recommend either past queries users have previously issued or reformulations. These recommended queries are a proxy to the information that is actually relevant to a user's information need; queries are not an information need and only express an aspect or approximate the need.

It would be better, then, to recommend information directly related to a user's need, rather than recommending past queries that are approximately related. One method is to recommend documents. While this can hasten the completion of a user's current interaction, it effectively "gives the user a fish" instead of "teaching a user to fish". To better inform future interactions, it is better to strike balance and teach a user how to query. Suppose we have a structure that models the "information" a user has observed so far, as well as what has been marked relevant. From this, we can perhaps determine related bits of information (for instance, words or relations) that are unexplored but relevant. From this breadth of unexplored information, we could hypothetically generate a new query. How well can we encapsulate this information within a succinct query? Furthermore, can an algorithmic method of extraction outperform humans, the source of queries from query log data?

URI: http://hdl.handle.net/10125/41690
ISBN: 978-0-9981331-0-2

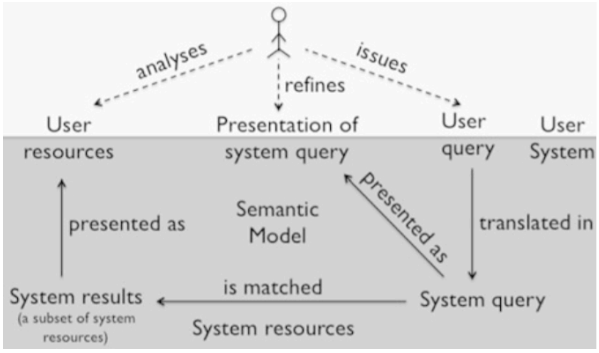

Figure 1. Overview of user and system interactions in Web search, as given by [5].

We describe a framework for mining relevant information and aggregating it into a natural language query to be used by humans in Web search. We show that queries generated from aggregation sometimes perform even better than humans in summarizing text into a query. In Section 2, we begin to describe how Web retrieval technology is related to the problem of query recommendation, discussing text retrieval and knowledge mining technologies. In Section 3, we propose a full pipeline of totally generative query recommendation, using motivating examples to show how it would be instantiated with knowledge management and information retrieval backend technologies, and we give our research question. In Sections 4 and 5, we detail our data collection process and algorithmic experiments. In Section 6, we give our technical results, and we conclude in Section 7 with our insights.

2. Background

While knowledge systems add much architecture and structure on top of text-based Web retrieval, many useful comparisons can be drawn between the two. We outline similarities and differences between both below, illustrating how query recommenders fit over various retrieval structures. Throughout Sections 2 and 3, we will use the running example query "things to do in Munich" to illustrate different concepts.

2.1 Web Search Architecture Overview

Figure 1, borrowed from Ceri et al. [5], illustrates a pipeline of what they call the "semantic search process". The user inputs a query to the system, which the system transforms into a more structured query. The structured query is matched to system resources, returning a ranked list of results that is possibly ordered by relevance. As we shall discuss, this model applies to both knowledge graph retrieval and text retrieval.

2.1.1. Connections to Text IR. In text-based retrieval, simple transformations are applied at the query translation level to the user's input query. Stopwords – such as "a", "and", "into" – are removed, and words are reduced to a basic form through stemming or lemmatization. Our running example may be transformed into "thing Munich". Likewise transformations are applied to the document collection in system resources. At the matching level, the simplest systems compare the transformed query to the transformed document with a simple term weighting scheme called the "term frequency inverse document frequency" (TFIDF). This weights terms in a document by their frequency and inversely weights them by their frequency across the corpus [22]. Matching and ranking simultaneously occur by scoring transformed queries and documents with a cosine measure. The scores determine a rank order. The transformations applied to queries and documents compose a "vector space model". Other document metadata such as PageRank scores and anchor text can be used as well in a more effective "learning to rank" model ordering documents [11]. The knowledge graph scenario, however, contains more than just a document collection in its system resources – it also contains a knowledge graph over entities and documents. Yet the same mode of interaction applies, as well as the recommendation problems that come with it.

2.1.2 Knowledge Systems Realizations. Users' queries to a web-based knowledge graph system – as with Google – are largely identical to naturalistic text retrieval queries, though Web-based knowledge systems also incorporate a knowledge graph. Knowledge graphs are important resources in several search-related applications like Web search, mobile search, and question answering. A common everyday example of a knowledge graph is Google's Knowledge Graph [23]. It supplements traditional search results with a "card" that summarizes entity information. Analogously to an abstract graph in Math or Computer Science, the basic units of a knowledge graph are nodes – i.e., entities like presidents, animals, or events – and edges – i.e., relationships between entities. Microformats and RDF, for instance, are two possible technologies to define the vocabulary for entities (e.g., "Munich") and pairwise or unary relationships (e.g., "In(EnglischerGarten,Munich)" or "City(Munich)"). These vocabularies are either defined through a central source such as schema.org or are mined in a decentralized way through web pages. Richer relationships can also be defined through ontology and inference frameworks such as OWL [3]. Knowledge graphs can be ultimately constructed by experts who

choose the relations, through a collaborative open-sourced approach, by combining rules and machine learning, or through totally automated learning. Much current research in relational learning is in the construction and curation of graphs [17].

Users' queries and a document corpus can be much more richly transformed with a knowledge base. One can, for instance, apply traditional vector space techniques after enriching queries and documents with simple entity and relationship information [7]. For example, the query "things to do in Munich" can be supplemented with the entities "Munich" and "City". A document about two tourist attractions in Munich – Englischer Garten and Marienplatz – can be supplemented with the entities "EnglisherGarten", "MarienPlatz", "Garden", "CitySquare" and "TouristAttraction", using relationship information to elaborate the nature of the two attractions. Then a cosine measure can be applied like in Section 2.1.1. Alternatively, user queries can be mapped to a system-specific semantic language [7]. One example is the query language of SPARQL, a database infrastructure often used to store and retrieve RDF-like data. An example SPARQL query, in plain English, would be "Select *x*'s such that *In(x,Munich)* and *TouristAttraction(x)*". The query gives constraints that can be matched against a graphical representation of entities and relations, known as graph pattern matching [5]. The result of the query is typically a list of entities (e.g., the tourist attractions mentioned before) and relationships (e.g., people related to a celebrity). In summary, knowledge graphs add structure to a user's search session, both in the queries and objects returned. As will be explained in Section 3, this can help us determine entities and relationships worth exploring for recommendation.

2.2. Query Recommendation

Now that we have explicated the semantic search process, we will describe query recommendation, a supplement to this infrastructure to assist Web users. Query recommendation is the process of recommending a query for a searching user to issue to a database, while the user is engaged in a task. In the language of our architecture, system resources often include a recommender that saves a user's search session data and recommends queries that the user should issue in the future (in our case, natural language queries to a Web search engine). While query recommendation in Web search engines is a fairly recent feature, it sees real use [21] and has made several advances since its inception. Yet the naturalistic queries recommended are typically either previously existing queries from a log or refinements of a user's previous queries. A simple approach, for instance, clusters similar queries from a log and recommends queries based on a static clustering [26]. Smarter approaches have incorporated more session context than this, for substantial gains. Such approaches, for instance, learn a Markov transition model of queries from logs, calculating the probability that pairs of queries co-occur within a portion of a search session [8,13,14]. Others generalize this approach to sequences of *n* queries [4,12] and have been shown to outperform pairwise methods. A less common approach is to refine or expand the most recent query, applying operations to a query like substituting, adding, or deleting words. One example uses topic modeling, like Latent Dirichlet Allocation (LDA), to mine latent query topics for refinement [2].

The query recommendation literature commonly compares a user's queries to candidate ones by examining clicks, the surface form of a query, and the snippets from search engine result pages (SERPs) – the URL's, snippets, and titles returned in a linear list in engines like Google and Bing. Yet query recommendation could also compare the lists of relations and entities extracted from the queries, as in the vector space model in Section 2.1.2. Even with this, the query recommendation approaches cited thus far are variations of content filtering and collaborative filtering at the level of queries. Content-based recommender systems recommend items based on a user's previous item history (in this case, queries). Collaborative filters recommend items based on other users' interests, like by combining multiple users' querying behavior into a transition model [1]. A hybrid approach such as Song et al. [24], which combines clickthrough data and a topical representation of queries, also operates purely at the level of existing queries. This is analogous to a movie recommendation framework, but in this recommendation domain, item features cannot be used to generate a new item. A movie recommender cannot recommend a new combination of actors and directors if such a movie does not exist. In query recommendation, however, the basic units of analysis are not actors, genres, and directors; they are words, and new combinations of words can always be built into queries.

We must also note that Web keyword queries only express an approximation or aspect of an information need. In the end, a searcher is looking for information that is relevant to a need. The above literature either recommends previous queries or iterations of them. If an existing query recommends exactly what a user wants, then it should be returned, but such recommenders cannot return something new but more desirable. Therefore gains can be made in determining what a user wants first and then generating a query

with this new knowledge. Prior work has shown that there are potential gains to be made in a search session by injecting part of another user's search into the current user [9]. Nevertheless, it has been shown that merely interleaving results is not enough [19] and that performance gains can be made with algorithmic intervention. Starting with information to generate new recommendations for queries, then, would be the next approach in recommendation.

3. The Generative Query Recommendation Approach

In the previous section, we laid out our working definitions and terminology for knowledge systems and text retrieval systems, which have very analogous infrastructure. We advocate for a generative query recommendation approach that can apply to both types of infrastructure. Our algorithm for generative query recommendation decomposes into 4 steps:

1. Transform interaction data (e.g., query and document text) according to the system's translation method.
2. Map this data to a subset of textual or graphical space (e.g., a subgraph in the general knowledge graph or a textual topic model).
3. Determine a next set of information that should be retrieved, based on some predetermined criteria (e.g., relevance to the task or diversification of results).
4. Aggregate this set into a query.

No modification of the Figure 1 framework is necessary. In Section 2.1, we abstractly outlined how steps 1 and 2 are currently implemented. We now give illustrative, running examples of steps 3 and 4 in hypothetical systems to detail a full pipeline.

3.1. Determining Next Steps

In query recommendation, we are provided interaction data such as user queries, their respective SERPs, and interactions with the results (e.g., clicks, dwell time on pages, and other browser interactions). We can use this data to determine things that the user deemed relevant and not relevant, to determine the next actions that a user should take. In text-based search, we can use LDA similarly to [2] to model the underlying topics of both sets of words. Topics are weighted sets of words, so we can compute pairwise topic similarity to determine the next best topic, such as a relevant but novel topic (e.g., the "Vienna tourism" topic for our example). We do not further discuss this aspect in detail but assume an algorithm exists to give us the text to aggregate in our algorithms. In our experiments, we focus on aggregation.

With a knowledge graph, the relevant and irrelevant interactions can be represented as subgraphs within the general knowledge graph, with weight given to more recent interactions or more relevant interactions (determined from longer dwell times, page bookmarking, etc.), giving more weight to more relevant parts of the subgraph. Determining the next step would be equivalent to determining the next set of hops in the graph, or the next hops from the strongest weighted parts of the subgraphs.

3.2. Aggregation

After determining some structure of information that a user should explore, the next necessary step – to give the user an action item to execute – is to generate the query for recommendation, since in Web search, querying is the primary mode of action. The knowledge graph offers some challenging complications. The structure added by relations simplifies the process of expanding them into queries. If a recommender only produced the relation "In(EnglischGarten, Munich)", it can recommend "Englisch Garten Munich" as a query. But even a single node can have many unexplored relations, many of which are extremely dissimilar (e.g., "Population(Munich)", "In(EnglishGarten, Munich)", and "MayorOf(DieterRieter,Munich)") and cannot be formulated in a concise query. Our data from Figure 2, for instance, shows most queries in our external dataset are of length 3. The challenge, then, is to aggregate dissimilar relations into a query, perhaps defined by an ontology or automatically learned. We do not further explore this here but discuss it in the Future Work.

Our text-based scenario with LDA offers simpler aggregation with a similar tradeoff. After determining the "Vienna tourism" topic for recommendation, for instance, one could sample from the probability distribution of words, or even get the most likely words, to form the query "trains to Vienna". The challenge here is making a coherent query, rather than simply sampling words like "opera palace trains".

In the technical component that follows, we test a simple mapping component. We convert query results (snippets of text) into pieces of information (a list of words) and aggregate this representation into a short query. A SERP approximates an information need; it is the information a search engine deems relevant to a query. Conversely, can an ordered list of relevant pieces of information be captured in a single query, by either users or algorithms? If so, how do they compare?

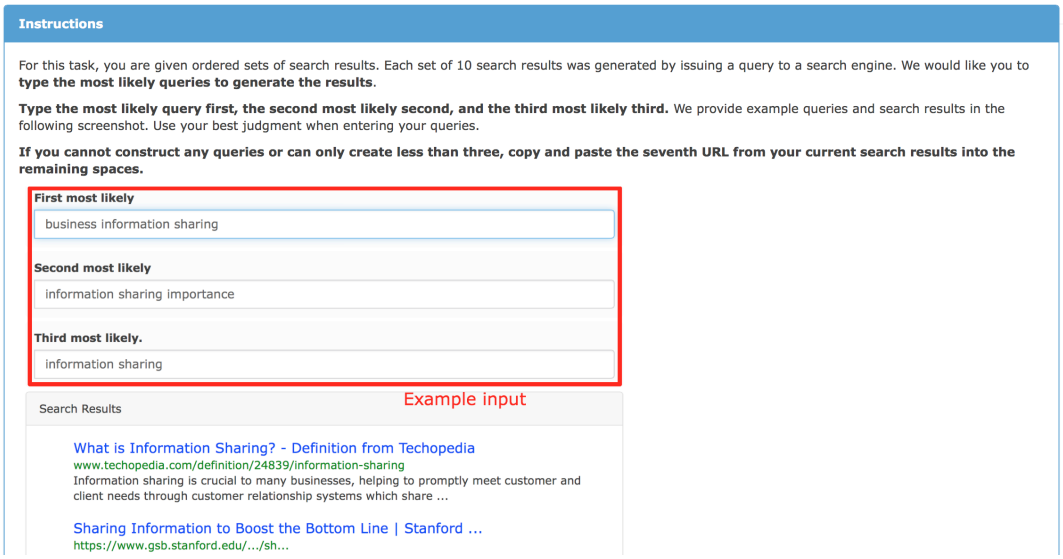

Figure 2. Layout of the Mechanical Turk task.

3.3. Research Question

The duration of our paper focuses on Step 4 of the algorithm – the aggregation. For our research question: Given a set of results marked as relevant, can a system accurately capture this information in a usable query? We illustrate an answer to this question in the positive in two experiments. Namely, we show that a system can outperform humans in such aggregation and that with reasonable training, even a simple text-based system (so presumably a knowledge-based system) can give accurate recommendations, given a set of data.

4. Datasets

In this section, we explain the data collected to run our experiments that address our research question.

4.1. User Data in Mechanical Turk

To compare users to algorithms, we must first provide users with some snippets of relevant information and ask them to generate some queries. In accordance with our framework, the set of results contain snippets that represent our "relevant information" that an aggregation algorithm should aggregate into a query. Specifically, we ask anonymous Web users to guess the underlying query that could have generated (and in fact, did generate) a set of results. We recruited them through crowdsourcing, a method of retrieving data by utilizing the labor of a large number of people. In recent years, this has emerged as a major method of retrieving large labeled data sets inexpensively, particularly through Amazon Mechanical Turk[1]. Mechanical Turk is a service in which *requesters* can publish microtasks, called *human intelligence tasks* (HITs), for *workers* to complete. Workers search Mechanical Turk for tasks and complete them to receive payment; requesters accept submissions and pay workers depending on their performance. While workers use this as a source of money, researchers have used this as an inexpensive, quick source of large sets of annotated data when recruiting large numbers of people for a lab study (e.g., thousands) becomes infeasible. Mechanical Turk and other crowdsourcing tools have been used for several natural language tasks, such as determining the political bias in segments of texts [25] as well as for machine translation [18]. A task was posted on Mechanical Turk by [10], in which users were asked to mark pages with relevance to a query; our task is the reverse of this one.

Table 1. Mechanical Turk statistics.

Number of HITs	4
Workers per HIT	30
Result pages per HIT	11
Queries per result page	3
Inputs per HIT	33
Total Inputs	3951
Average Time per HIT	26m9s
Average words per query	3.05
Number of blank inputs	9
Estimated hourly rate	$1.721
Total money spent	$99

Mechanical Turk workers (i.e., our "users") were asked to annotate sets of 10 search results given as <page title, URL, snippet> triples, simulating the first

[1] https://www.mturk.com/

page of commercial search engine results (the first SERP). We told users that each set of search results was generated by a query and asked them to give the top 3 queries that were most likely to generate the set of results, ordering them from most to least likely. If users could not produce 3, we asked them to fill the remaining entries with a specific URL, using this as a filter for bad workers. We presented workers with search results from 44 topics. We distributed these among 4 HITs, yielding 11 topics and 33 annotations per HIT. See Table 1 for a breakdown of HIT statistics and Figure 2 for a sample screenshot of the task. We also accepted only users with at least a 95% approval rating located in the United States to work on our HITs. We manually verified the validity of each Turker input and rejected only 3 out of 123 submissions. As can also be seen from Figure 1, we did not boldface any of the snippet words that were contained in any of the original queries, as is common with commercial search engines today, because we did not want to artificially prime users to include boldfaced words in their queries.

4.2. TREC Session Data

4 of the above topics were our own, but the other 40 were from the TREC 2014 Session Track[2]. The TREC 2014 Session Track is a collection of user sessions on 60 text retrieval topics, where users input queries to a text database. It is often used in IR experiments to evaluate the performance of ranking algorithms and query suggestions. While there are 60 TREC topics, we removed duplicate TREC topics that were rewordings of the same genre, and used a subset of 40 topics: numbers 2-7, 9-11, 14, 16-18, 20-25, 28, 31-35, 37-38, 41-42, 44-46, 48, 52-53, 55-58, and 60. These topics are exploratory search topics - topics that are complex enough to require multiple queries and potentially multiple search sessions. Exploratory queries have been estimated to comprise 10% of search sessions and 25% of overall queries [6,15]. We chose exploratory topics because exploratory search represents the ideal scenario (multiple queries in a long session) for our generative query recommendation. In total, there were 2460 queries from these topics. We used a sample of 40 queries (1 randomly chosen from each topic) for the Mechanical Turk task. We used all 2460, however, to examine algorithmic performance in depth, over various parameters.

5. Experiments

In Experiment 1, we directly compared user and algorithmic performance. We extracted the queries given by users in our Mechanical Turk task and compared them to queries that our algorithms generated when given the same sets of titles, snippets, or both. To maximize replicability, we omitted the 4 extra topics from Mechanical Turk's 44 from analysis, only analyzing user guesses and algorithmic outputs from the 40 TREC Session Track topics.

We found (as will be explained in the results) that algorithms could more closely guess the queries than users in Experiment 1. Hence in Experiment 2, we examined parameter settings to explore what would make an algorithm perform well. Namely, we examined different evaluation metrics and scaled the size of input to the algorithms. How does algorithmic performance change, for instance, when 100 relevant snippets are used instead of 10? Is it always better to use both titles and snippets in generating a query? We examined these types of questions in Experiment 2, using all 2460 queries from our 40 TREC topics.

In both experiments, we used 4 simple algorithms. For our first two algorithms, we assumed we knew the underlying query length (given as input, call it n) and extracted n query words that maximized the query's cosine score (i.e., the top n words) from the titles, snippets, or both (depending on what feature was used). SERP text preprocessing included lemmatization and stopword removal. For the second algorithm, we chose the top skipgrams of length n instead. We counted only skipgrams that also contained the most frequent word, as we found that the most frequent word was almost always in the gold standard query. Skip-grams generalize n-grams; words are allowed to occur within a window of k words in order to be counted in the model. For smaller data, skip-grams can cope with data sparsity that can affect n-gram models, and they can be an effective smoothing method for language model estimation [20]. Our third and fourth algorithms are copies of the prior two, with the output query length sampled probabilistically instead from a distribution over training data. We re-ran our algorithms several times, always holding out a random 70% of the queries as training data to develop a model of the probability of a given query length. See Figure 2 for an example query length distribution, over the 2640 TREC queries. To generate query lengths for the third and fourth algorithms, we randomly drew the length from a similar distribution on our held out data.

For all experiments and data collection, inputs are URLs, titles, and snippets associated with the true, underlying *gold standard query* (hidden from the algorithm for evaluation). Lists of results for eachquery

[2] http://ir.cis.udel.edu/sessions/index.html

are generated through the Google Search API[3]. In separate lab and live studies, we found that over 95% of queries were issued to Google. Similarly, search results Mechanical Turk users' input queries and algorithm output queries were extracted with the Google Search API, the time of collection from Mechanical Turk. This allowed us to not only compare generated surface strings but also meaningfully compare lists of search results (all returned by the same search engine).

6. Results and Analysis

6.1. User vs. Algorithmic Performance

When comparing user and algorithm results from Experiment 1, we measured the Jaccard distance between the guessed query terms (i.e., generated by users or algorithms) and the true underlying *gold standard query terms* (i.e., those that were hidden). Jaccard distance is given as: $Jaccard(q_1,q_2)= Intersection(q_1,q_2)/Union(q_1,q_2)$, where q_1 and q_2 are lists of query terms. This score, the set intersection of two sets of words over the set union, represents the amount of overlap between two queries, with scores ranging from 0 to 1 and 1 representing $q_1=q_2$. For instance, *Jaccard({hello,world},{hello,mother})= 1/3*.

For simplicity, we lemmatized words and removed stopwords from our *gold standard query* before analysis, and did the same for all inputs, algorithmically generated queries, and user queries. This assumes stopwords and morphological differences in words should not affect retrieval results much. Since the number of tasks and queries for both experiments were small, we could manually do entity recognition and disambiguation for all query terms, result snippets, and output queries for better evaluation. We manually normalized multiple names for the same entity - e.g., normalizing "united kingdom" and "uk" into just "uk". Lastly, in the Mechanical Turk task, users were asked to give their 3 best guesses for each set of results they were given. We took each user's best guesses and only presented those results for our final analyses.

Table 2. Experiment 1 - Algorithmic vs. user performance, in Jaccard distance on 40 queries. User scores are independent of input columns. (P) stands for the probabilistic model of generating query length.

	Titles	*Snippets*	*Both*
Users	0.66	-	-
Cosine	0.76	0.86	0.87
Cosine (P)	0.53	0.60	0.59
Skipgram	0.66	0.73	0.78
Skipgram (P)	0.47	0.48	0.50

See Table 2 for our results from Experiment 1, on 40 queries. Both users and algorithms examined 10 search results per query. The results strongly suggest that algorithms can be more effective at guessing a query that can guide a user to a certain set of relevant information. This strongly suggests the need for algorithmic intervention; given a set of information that has been marked as important, algorithms can more accurately guess a query that users would need, as the cosine and skipgram greatly outperform users in Jaccard distance – i.e., they get more words correct. These generative queries can ultimately be better than log queries – which are always supplied by users. However, this is only in the condition where the number of query terms is known. Humans still outperform our algorithms when combined with our model for query length, but this only suggests the need for a smarter model for generating query length.

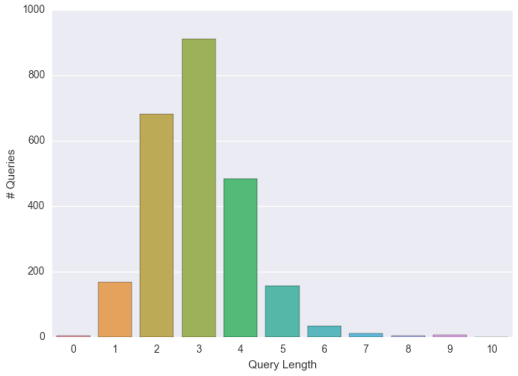

Figure 3. Count of query terms per query, over our 2460 TREC queries.

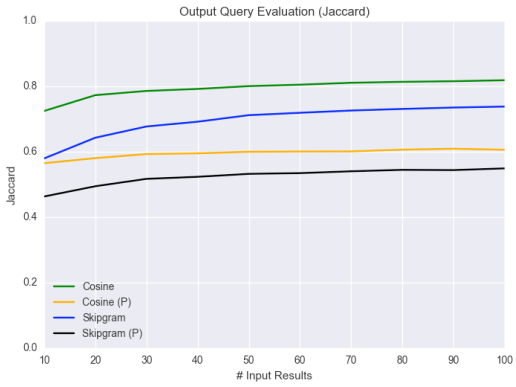

Figure 4. Experiment 2 - Jaccard scores for algorithms with increasing input size.

[3] https://www.google.com/cse/

6.2. Algorithmic Performance In Depth

Table 2 suggests that both titles and snippets are important, but Table 3 compares algorithmic performance on all 2460 queries, comparing algorithmic models built on the first 10 results versus the first 100. While performance improves overall when using more results for generating models, titles become less important and even hurt performance. Indeed we found this to be the case with several alterations of the number of results.

For subsequent analyses, our algorithms hence only used snippets and the top 100 original results, but we show in Figure 4 what happens when only snippets are used but the number of input results increases. First, Figure 4 further demonstrates the need for a stronger method for determining recommended query length, as there is a substantial difference between the versions with and without the probabilistic query model. Second, it shows performance increasing for all algorithms as the number of input results increases. Third, although skipgrams perform more poorly overall, the gains in performance are much more drastic performance improvements (a difference of about 0.16 versus 0.09), possibly suggesting that skipgrams may eventually equal or outperform our cosine-based method with more data.

We lastly give URL-based results in Figures 5-8. This means we compare the list of returned URLs from our algorithms to the original list of results, across the full 2460 queries. Even if queries may not exactly resemble queries that a user may issue, they may still be useful for recommendation if they return effective results. We present 4 different URL-based scores: Mean Average Precision (MAP), Discounted Cumulative Gain (DCG), the normalized version of DCG (NDCG), and Mean Reciprocal Rank. Each score is a function that accepts an ordered list of items (in our case, URLs), some relevant and some nonrelevant; it outputs a numeric score. Greater weight is given when more items are relevant or when relevant items are located closely to the top of the list. One example we provide here is $DCG@K = relevance(r1)+relevance(r_2)/log(2)+relevance(r_3)/log(3)+...+relevance(r_k)/log(k)$ where r_i is the relevance score for the ith item. K is a variable on the list length, common to many of the metrics we use. We provide [16] as a more complete reference.

Our definition of relevance varies with K. We mark a URL in a list of output results as relevant if and only if it is in the top K original results we used for modeling. We first note that for URL-based scores, the performance increases but does so more slowly as K increases. This is due to the decay of the functions; added value diminishes as more relevant results are added further down the list. As the cosine algorithm outperforms skipgrams in Jaccard, it is expected that cosine should outperform here as well, and it does in most metrics. The only exception is MAP. According to our definition of K, this means that more relevant outputs are marked near the top in the skipgram algorithm, but these are located in the lower part of the list in the inputs. This means that some of the top retrieved URL's in the skipgram approach are from the bottom of the original list to the top. This can be important in serendipitous retrieval, or in retrieval tasks where rare information is desirable.

Table 3. Experiment 2 - Jaccard distance scores, when training on the top K results, for the full TREC Session Track.

Algorithm (K)	*Titles*	*Snippets*	*Both*
Cosine (10)	0.65	0.72	0.75
Cosine (100)	0.68	0.82	0.79
Skipgram(10)	0.56	0.58	0.62
Skipgram (100)	0.58	0.74	0.73

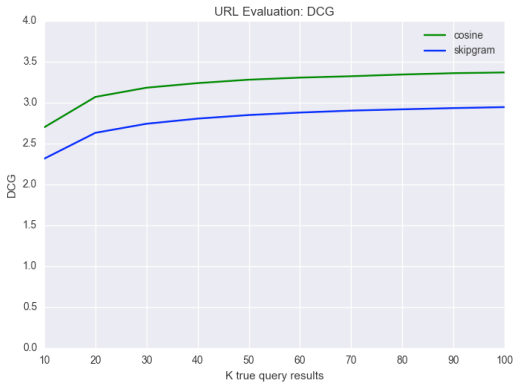

Figure 5. Experiment 2 - DCG for our algorithms, given a known query length. K is the number of original URL's considered for relevance.

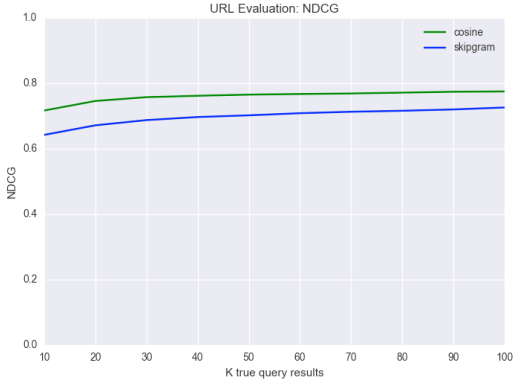

Figure 6. Experiment 2 - NDCG scores.

7. Conclusions and Future Work

We see much promise in a completely generative approach to query recommendation in Web-based search. In our simple approach, our algorithm shows that intelligently determining the length of a query is key, but solving that problem can potentially improve the user experience. Taking a simple counting approach to determining important items – as with our counting-based approach – can be sufficient, but with enough data, longer distance relationships between items may bring performance gains and even get lesser known relevant items – as with our skipgram approach.

We acknowledge that our experimentation is in a text-based retrieval environment, using text snippets as the relevant items to be aggregated into a query. However, in Section 3, we made a case for generalizing our approach to a knowledge-based framework for generating queries, aggregating subgraphs from a general knowledge graph. The weighted subgraph represents what a user should search for next. Our weighting was at the word level. A concrete implementation of this aggregation process is beyond our current approach. We show that a simple aggregation process, where the aggregates (i.e., text snippets) are largely unstructured, can lead to relevant retrieval results. Also, the SERP snippets we used in our experiment were generated by real queries, so they represented genuine information needs by real users.

Our aggregation process and evaluation also makes simplifying assumptions about the structure of queries, omitting stopwords – as in "museums *in* NYC". Knowledge systems add layers of structure on top of relevant information, providing names for entities, disambiguation, and relationships. Relations can even resemble natural language – such as *In(MOMA,NewYorkCity)* – and can offer a much more straightforward transformation process into a natural language query. Our general approach is agnostic to the specifics of the general retrieval architecture outlined in Figure 1.

Our work shows an optimistic step towards a bottom-up knowledge-based system for search recommendation. We found that such a system can greatly outperform human efforts to consolidate an information need into a query. Future work is required to make a complete recommender system to compare to current non-generative recommenders. Namely, step 3 and 4 need to be concretely defined and implemented for a knowledge base, or in our case step 3 could be integrated with our text-based aggregation. The final test is an A/B test that compares user satisfaction and performance on two systems: A) one using the traditional query recommendation approaches outlined in Section 2.3, and B) another using the generative approach. This paper shows an optimistic step towards a complete system that recommends information, instead of just someone else's queries.

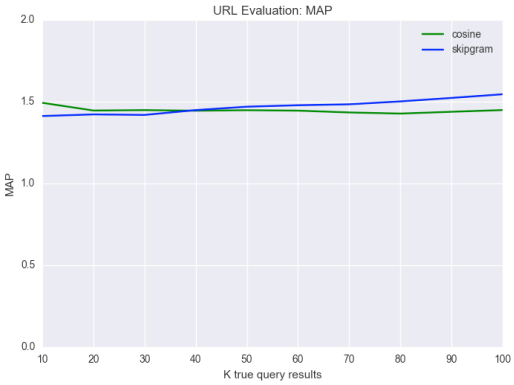

Figure 7. Experiment 2 - MAP scores.

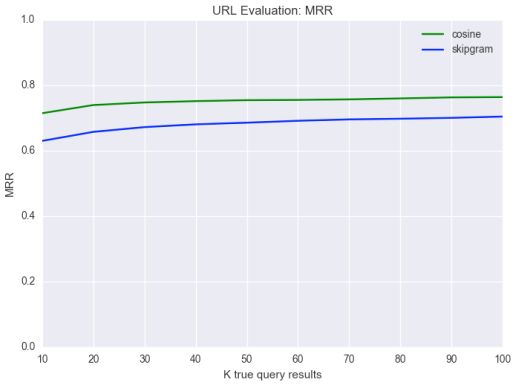

Figure 8. Experiment 2 - MRR scores.

References

[1] Adomavicius, G., and Tuzhilin, A. 2005. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Trans. On Know. and Data Eng. 17, 6 (June 2005), 734-749.

[2] Bing, L., and Lam, W. 2011. Investigation of web query refinement via topic analysis and learning with personalization. Proceedings of the 2nd SIGIR Workshop on Query Representation and Understanding (SIGIR-QRU).

[3] Bizer, C., Mendes, P.N., and Jentzsch, A. Topology of the Web of Data. In Semantic Search over the Web. Virgilio, R.D., Guerra, F., and Velegrakis, Y. (Eds.). 2012, pages 3-29, New York, NY, USA. Springer.

[4] Cao, H., Jiang, D., Pei, J., Chen, E., and Li, H. Towards context-aware search by learning a very large variable length hidden markov model from search logs. In Proceedings of the

18th International Conference on World Wide Web, WWW '09, pages 191-200, New York, NY, USA, 2009. ACM.

[5] Ceri, S., Bozzon, A., Brambilla, M., Valle, E.G., Piero, F., and Quateroni, S (Eds.). Topology of the Web of Data. In Web Information Retrieval. 2013, pages 181-206, New York, NY, USA. Springer.

[6] Donato, D., Bonchi, G., Chi, T., and 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 '10, pages 321-330, New York, NY, USA, 2010. ACM.

[7] Fernandez, M., Cantador, I., Lopez, V., Vallet, D., Castells, P., and Motta, E, Semantically enhanced information retrieval: an ontology-based approach. J. Web Semant. 9(4), 434-452 (2011).

[8] Fonesca, B.M., Golgher, P., Pôssas, B., Ribeiro-Neto, B., and Ziviani, N. Concept-based interactive query expansion. In Proceedings of the 19th International Conference on the World Wide Web, WWW '10, pages 321-330, New York, NY, USA, 2010. ACM.

[9] González-Ibáñez, R., Shah, C., and White, R. Pseudo-collaboration as a method to perform selective algorithmic mediation in collaborative IR systems. In Proc. ASIST (2012).

[10] Grady, C., and Lease, M. 2010. Crowdsourcing document relevance assessment with mechanical turk. In Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk, CSLDAMT '10, 172-179. Stroudsburg, PA, USA: Association for Computational Linguistics

[11] Hang, L. 2011. A short Introduction to learning to rank. IEICE TRANSACTIONS on Information and Systems 94)10): 1854-1862.

[12] He, Q., Jiang, D., Liao, Z., Hoi, S. C. H., Chang, K., Lim, E.P., and Li, H. Web query recommendation via sequential query prediction. In Proceedings of the 2009 IEEE International Conference on Data Engineering, ICDE '09, Washington, DC, USA, 2009. IEEE Computer Society.

[13] Huang, C.K., Chien, L.F., and Oyang, Y.J. Relevant term suggestion in interactive web search based on contextual information in query session logs. J. Am. Soc. Inf. Sci. Technol., 54(7):638-649, May 2003.

[14] Jones, R., Rey, B., Madani, O., and Greiner, W. Generating query substitutions. In Proceedings of the 15th International Conference on World Wide Web, WWW '06, pages 387-396, New York, NY, USA, 2006. ACM.

[15] Kotov, A., Bennett, P.N., White, R.W., Dumais, S.T., and Teevan, J. Modeling and analysis of cross-session search tasks. In Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '11, New York, NY, USA, 2011. ACM.

[16] Manning, C., and Nayak, P. (24 March 2016) Introduction to Information Retrieval - Evaluation. https://web.stanford.edu/class/cs276/handouts/EvaluationNew-handout-6-per.pdf

[17] Nickel, M., Murphy, K., Tresp, V., and Gabrilovich, E. A Review of Relational Machine Learning for Knowledge Graphs. Proceedings of the IEEE 104(1): 11-33 (2016)

[18] Novotney, S., and Callison-Burch, C. 2010. Shared task: Crowdsourced accessibility elicitation of Wikipedia articles. In Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk, CSLDAMT '10, 41-44, Stroudsburg, PA, USA: Association for Computational Linguistics.

[19] Pickens, J., Golovchinsky, G., Shah, C., Qvarfordt, P., and Back, M., Algorithmic mediation for collaborative exploratory search. In Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval (SIGIR '08). ACM, New York, NY, USA, 315-322.

[20] Shazeer, N., Pelemans, J., and Chelba, C. Skip-gram language modeling using sparse non-negative matrix probability estimation. CoRR, abs/1412.1454, 2014.

[21] Shiri, A., and Zvyagintseva, L. 2014. Dynamic query suggestion in web search engines: A comparative examination. Proceedings of the Annual Conference of CAIS.

[22] Singhal, A. Modern Information Retrieval: An Overview. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Vol. 24, No. 4. (2001), pp. 35-42.

[23] Singhal, A. "Introducing the Knowledge Graph: things, not strings," May 2012 [Online]. Available: https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html

[24] Song, W., Liang, J. Z., Cao, X. L., and Park, S. C. An effective query recommendation approach using semantic strategies for intelligent information retrieval. Expert Syst. Appl. 41(2):366-372, Feb. 2014.

[25] Yano, T., Resnik, P., and Smith, N.A. 2010. Shedding (a thousand points of) light on a biased language. In Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk, CSLDAMT '10, 152-158. Stroudsburg, PA, USA: Association for Computational Linguistics.

[26] Zahera, H.M., El-Hady, G.F., and El-Wahed, G.F., Query recommendation for improving search engine results. Int. J. Inf. Retr. Res. 1(1):45-52, Jan. 2011.