Methods of Reducing Verbose Queries

Martha Enderby
University of Minnesota Morris
600 East 4th Street
Morris, MN 56267
ender034@morris.umn.edu

ABSTRACT

This paper analyzes three methods of reducing natural - language search queries in order to get more focused, useful search results. The first method, dependency parsing, involves using dependency parsing trees to identify key concepts in a query and weight terms accordingly; the second, query quality predictors, uses several descriptive features of each possible sub-query to predict the highest-quality sub-query for a given query; and the third, subset distribution, uses the average of several high-quality sub-queries.

In this paper, we will present brief outlines of each method, including any relevant calculations. Then, we will report the results the method obtained in the paper where it was presented.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Query Formulation

General Terms

Algorithms, Performance, Theory

Keywords

Verbose Queries, Long Queries, Query Quality, Query Reduction

1. INTRODUCTION

Most search engine queries are keyword queries, which have lengths of between two and six words and are therefore extremely focused on the query subject [3]. However, as much as 15% of all search queries are verbose [1]. Verbose queries are written in natural language rather than terse little phrases, and thus contain unnecessary or redundant words.

For instance, "verbose queries" is a short, keyword-focused query, whereas "describe the different methods one can use to get better search results from a verbose query" is a verbose query. Most of the words in it are not useful in deciding whether a document is relevant. Search engines demonstrate worse performance with verbose queries than with keyword queries [1], since they give equal weight to each word (excluding small, common words) in a query.

Therefore, in order to improve search engine performance on long natural language queries, the queries must be somehow reduced to their most essential components. According to Kumaran and Carvalho, A perfect reduction of a verbose query into a keyword query can improve search performance (here meaning the relevance of the documents returned by the query) by 30% [3].

This paper will evaluate three methods of reducing verbose queries: dependency parsing, subset distribution, and query quality prediction. Unfortunately, literature does not exist that compares these methods side-by-side in identical tests. The results presented for each method should therefore be taken as independent data. If a method has been tested against other methods, its specific section will indicate so and provide the compared results.

2. BACKGROUND

Recent research on verbose queries has focused on improving search engine performance with long queries. The two basic methods for reducing query length are selecting a small subset of an overall query to replace the original query and weighting individual words in a query to identify one or more key words that can be used for a more keyword-focused query. In either method, the challenge for a potential reduction algorithm is determining what words in a particular query are important, and that is where methods for query reduction differ beyond the subset-or-weighting division.

Verbose queries include "wh-" queries (queries that begin with the question words "who," "what," "where," "why," or "when"), but these queries have specific, widely-accepted techniques developed for answering them, because they tend to have a more fixed structure than other verbose queries do. As such, recent research has dealt exclusively with verbose queries that do not possess the mostly fixed form of "wh-" queries [2]. None of the methods summarized in this paper deal with "wh-" queries. Additionally, search engines generally strip common stop words (such as "the" or "in") automatically. Stripping out trivial words, therefore, is also not the focus of current research in verbose queries. There is no universal list of stop words, but all the methods described in this paper use the InQuery list of stop words [1], a 418-word list developed by the University of Massachusetts' Center

University of Minnesota Computer Science Seminar '11 Morris, Minnesota USA.

for Intelligent Information Retrieval. For the purposes of this paper, a "term" is either a single word or a small group of related words, such as noun phrases like "University of Minnesota."

The methods presented in this paper have all been tested against topics from a handful of collections from Text REtrieval Conferences, hereafter referred to as TREC topics. TREC is a series of workshops focusing on different areas of information retrieval. The three specific TREC collections that seem to be somewhat standard in verbose query reduction experiments are the Robust2004, a 2004 workshop on conversational analysis; the WT10g, an online dataset compiled for testing web-based data retrieval; the Gov2, another online dataset consisting of .gov pages; and the TREC123, which contains documents from TREC disks 1 and 2. Robust2004 contains about five hundred thousand documents, WT10g contains about 1.7 million documents, Gov2 contains about 25.2 million documents, and TREC123 contain 150 documents, giving a wide range of collection sizes [8, 3]. Note that not all of these collections relate specifically to the reduction of verbose queries. The specifics of their content are not important; what matters is that they are quality-controlled repositories with regularly-arranged data.

A TREC topic consists of a title, which is usually one to four terms long; a description that is between three and thirty terms long; and the narrative, which contains the full text of the topic [3]. Query reduction work usually focuses on TREC topic descriptions, since they have the most in common with verbose search queries.

"Training" is machine learning technique used to allow a program to evolve based on previous data. In order to successfully reduce a test query, a method must first be trained on other data, called training data. The methods presented in this paper were trained using RankSVM, a learning-to-rank algorithm that, given two results, ranks one as better than the other.

3. DEPENDENCY PARSING

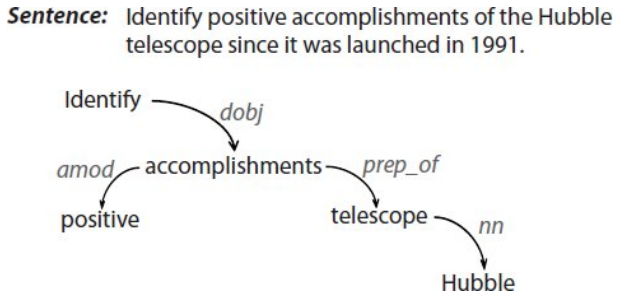

Figure 1: In this dependency parsing tree, an arc represents a dependency between two words. The label on the arc indicates the type of dependency. "Accomplishments" and "telescope" are related by the preposition "of," "Hubble" is part of the same noun phrase as "telescope," "accomplishments" is a dependent object of the verb "identify," and "positive" is an adjective modifying "accomplishments." Note that this tree is incomplete for the given sentence: it does not map any dependencies that cannot be traced back to "accomplishments."

3.1 Dependency Parsing Trees

One way to weight words in a verbose query is by using a dependency parsing trees, as shown by Park and Croft [6]. A parse tree is a collection of nodes and arcs that divides words in a sentence based on their part of speech and relation to one another. A dependency parsing tree, in particular, relates words based on what other words they modify, i.e. an arc from word A to word B means that B modifies A. The label between two connected words indicates the kind of connection, as shown in figure 1.

Not every relationship gleaned from a dependency parsing tree is useful, and to limit the number of syntactic features to look at, Park and Croft limited each feature to two arcs [6]. They then used those arcs as input features for RankSVM.

3.2 Ranking Terms

After getting the syntactic features of a query from a dependency parsing tree, in order to get labels for training data, this method weights each term's relative importance in a query. The equation Park and Croft use to find a term t's importance is as follows:

$$E(t) = \frac{1}{N_m} \cdot \sum_{c \epsilon C_m} (\varphi(c,t) - \varphi(c)) \quad (1)$$

In equation 1, m is the number of terms in a given query, C_m is all possible combinations of m terms excluding t, c is a single combination, N_m is the number of terms in C_m, $\varphi(c)$ is the search performance (in terms of the relevancy of retrieved documents) of c, and $\varphi(c,t)$ is the search performance of c when combined with t. This equation estimates the usefulness of term t by summing the impact that it has on all the terms in C_m [6]. A higher *E(t)* indicates that a term connects strongly to the rest of the query and thus is likely important to the meaning of the query as a whole. Conversely, a low *E(t)* indicates that a term does not connect strongly to the rest of the query, or relates to only a few other words in the query. The low *E(t)* term is more likely to be a modifier or secondary idea rather than a primary one.

3.3 Results

Park and Croft tested their query term ranking algorithm on topics taken from Robust2004 and WT10g. The average number of terms per question was 8.7 in the Robust2004 and 6.5 in WT10g [6]. The query ranking algorithm was tested against a method described by Bendersky and Croft [1]. A major difference between the two methods is that Bendersky and Croft's algorithm identifies a single key concept, while Park and Croft's algorithm can yield multiple key concepts.

Results from the tests indicated that Park and Croft's query term ranking was more effective for the longer (8.7-word average) queries of Robust2004, but that Bendersky and Croft's method was more effective for the shorter (6.5-word) queries of WT10g. Bendersky and Croft noted that Robust2004 contained a higher percentage of queries with multiple key concepts [1], which likely accounts for why Park and Croft's method was more effective on that set of queries.

4. QUERY QUALITY PREDICTORS

Park and Croft's query reduction method involves weighting each individual term in a query based on how that term

connects to other terms. The method presented in this section, by contrast, does away with extraneous search terms altogether, chopping the original verbose query into sub-queries. To do this, each word in a query is assigned a label of "keep" or "do not keep." Their work uses methods also used in document retrieval. As such, this query reduction method is much more dependent on the contents of the collection from which a query is taken than the methods summarized in sections 3 and 5.

4.1 Quality Predictors

In order to decide which terms within a query to keep, Kumaran and Carvalho utilize several different quality predictors [3]. The quality predictors Kumaran and Carvalho used are referred to as features, which means that they are individual measurable heuristic properties of a query. Some of these features are pre-retrieval, which means they are derived directly from statistics about the query and collection of documents on which the query is run. Others, such as query clarity (see section 4.1.3), are post-retrieval, which means they can only be derived after a query is run. Post-retrieval features are more expensive to compute than pre-retrieval features. All of the equations used in sub-sections 4.1.1 to 4.1.8 come from Kumaran and Carvalho [3].

4.1.1 Mutual Information

Mutual information (MI) is a method of measuring the dependency between two terms. A term refers to either a single word or a short phrase like "University of Minnesota." All the terms in a sub-query are represented as vertices in a graph, with the MI of two terms as edge weights. MI is calculated in equation 2. By connecting all the graph's vertices in a way that maximizes MI without producing any loops, we can find the graph's maximum spanning tree. The maximum spanning tree is then used as a predictor of the sub-query's quality, with more weight indicating a higher-quality query.

$$I(x,y) = \log_{10} \frac{n(x,y))}{n(x) * n(y)} \tag{2}$$

where *n(x,y)* is the number of times the terms x and y occur within a term window of 100 terms across the collection, and *n(x)* and *n(y)* are the frequencies of x and y in the collection.

4.1.2 Sub-Query Length

Sub-query length (SQLen) is the number of terms in a sub-query. The optimal length of a sub-query has been found to be between two and six terms.

4.1.3 Query Clarity

Query Clarity (QC) represents the dissimilarity between the documents returned by a query (also called the query model) and the collection as a whole. A high query clarity indicates that the query is narrowly-focused.

$$QC = \sum_{w \epsilon Q} P_{ml}(w|Q) * \log_2 \frac{P_{ml}(w|Q)}{P_c(w)} \tag{3}$$

where $P_{ml}(w|Q)$ is the conditional probability of the occurrence of the word w in the query model, and $P_c(w)$ is the probability of the occurrence of w in the collection. Query clarity is a post-retrieval feature.

4.1.4 Simplified Clarity Score

Calculation of QC is expensive. Simplified Clarity Score (SCS) is a comparable method for determining clarity without having to generate a query model.

$$SCS = \sum_{w \epsilon Q} P_q(w|Q) * \log_2 \frac{P_q(w|Q)}{P_c(w)} \tag{4}$$

where $P_q(w|Q)$ is the probability of the occurrence of the word w in the query.

4.1.5 Inverse Document Frequency-Based Features

Inverse Document Frequency (IDF) measures the relative importance of a term within a collection. The IDF of a word w is calculated as follows:

$$IDF_w = \frac{log_2 \frac{N+0.5}{N_w}}{log_2(N+1)} \tag{5}$$

where N is the number of documents in a collection and N_w is the document frequency of w.

For each sub-query, Kumaran and Carvalho calculated the sum, standard deviation, maximum/minimum, maximum, arithmetic mean, geometric mean, harmonic mean, and coefficient variation of the IDFs of that query's terms to use as additional query quality predictors.

4.1.6 Query Scope

Query Scope (QS) is a measure of the number of documents returned by a query relative to the size of the collection as a whole. If QS is too large, the query is probably not useful since it retrieves too many documents. QS of 0 is also not useful, since a query that returns no results is not a good query for a particular collection.

$$QS = -log_{10} \frac{N_Q}{N} \tag{6}$$

where N_Q is the number of documents containing at least one term from the query.

4.1.7 Similarity Collection/Query-based features

Similarity Collection/Query (SCQ)-based features measure how similar a query is to the collection as a whole. Unlike QS, a high SCQ is considered an indication of good query quality.

$$SCQ_w = \left(1 + ln(\frac{n(w)}{N})\right) * ln\left(1 + \frac{N}{N_w}\right) \tag{7}$$

where *n(w)* is the frequency of w in the collection.

As with IDF (section 4.1.5), Kumaran and Carvalho calculated several values based on SCQ.

4.1.8 Inverse Collection Term Frequency-based features

Inverse Collection Term Frequencey (ICTF)-based features are similar to IDF-based features (section 4.1.5).

$$ICTF_w = log_2 \frac{n(w)}{T} \tag{8}$$

where T is the number of term occurrences in the collection.

Again, Kumaran and Carvalho used ICTF to calculate aggregate statistics similar to IDF and SCQ.

4.1.9 Similarity Original Query

Similarity Original Query (SOQ) measures how much of the original verbose query's information a given sub-query retained. Kumaran and Carvalho calculated SOQ by finding the cosine similarity between the term frequency- IDF vectors of the original query and the sub-query.

4.2 Results

Each original query Q of length n has $O(2^n)$ possible sub-queries, denoted by the set $SQ = \{sq_1, sq_2, ..., sq_{2^n}\}$. Each sub-query was also represented by a vector of 31 query quality predictors $sq_i = [x_{i1}, x_{i2}, ..., x_{i31}]$.

In addition to selecting only sub-queries with lengths of between three and six words as per section 4.1.2, Kumaran and Carvalho also limited the sub-queries they used to those sub-queries which (a) were among the top 25 sub-queries ranked by the MI feature (section 4.1.1) for their original query, and (b) contained at least one of the proper nouns or temporal expressions contained in the original query.

Kumaran and Carvalho tested their query reduction method on Robust2004, TREC123, and a combination of the two collections. Against unreduced queries, they saw a 10% improvement in mean average precision (MAP) in TREC123, a 6.8% improvement in MAP in Robust2004, and a 5.8% improvement in MAP in the two collections combined. Recall that a perfect reduction can improve a query by 30%.

In all three cases, the most important feature (the one that was most useful in choosing a sub-query) was query clarity. Other important features were mutual information, total ICTF, max ICTF, and total IDF. [3].

5. SUBSET DISTRIBUTION

The final method that this paper will be summarizing is another way to choose sub-queries of a verbose query. Kumaran and Carvalho [3] focused on using query reduction to find the single best search query. Xue et al., by contrast, looked at the average performance of all sub-queries of a given query.

5.1 Conditional Random Fields

A conditional random field, or CRF, is a framework for labeling and segmenting data input such as natural language text [4]. CRFs are visualized as undirected graphs, with each vertex representing, in the case of natural language processing, a word. Edges represent dependencies between two words.

CRFs are used to generate the conditional probability $P(\mathbf{y}|\mathbf{x})$ where the variables $\mathbf{x}$ represent observed knowledge about the entities one wishes to predict, and the variables $\mathbf{y}$ represent those entities' attributes. A verbose query Q can be divided into a sequence of words $\mathbf{x} = \{x_1, x_2 ... x_n\}$ where n is the number of words in Q. $\mathbf{y} = \{y_1, y_2 ... y_n\}$ is a series of labels where y_i takes a value of 1 or 0, which denote "keep" and "do not keep" x_i, respectively. Given $\mathbf{x}$ and $\mathbf{y}$, a sub-query can be generated.

Using a standard CRF, $P(\mathbf{y}|\mathbf{x})$ is calculated as follows:

$$P(y|x) = \frac{\exp(\sum_{k=1}^{K} \lambda_k f_k(\mathbf{x}, \mathbf{y}))}{Z(x)} \tag{9}$$

$$Z(x) = \sum_{y} \exp\left(\sum_{k=1}^{K} \lambda_k f_k(\mathbf{x}, \mathbf{y})\right) \tag{10}$$

In equations 9 and 10, exp(X) means e^X, f_k represent features extracted from the input series $\mathbf{x}$ and the label sequence $\mathbf{y}$, K is the total number of features, λ_k is the weight of feature f_k. *Z(x)* is a normalizer which ensures the conditional probability $\sum_y P(\mathbf{y}|\mathbf{x}) = 1$.

CRFs are usually trained on test input where the optimal labeling sequence is known. Such an optimal sequence is, however, generally not available when looking at verbose query parsing. Therefore, instead of using the "gold standard" sub-query, the modified version of CRFs utilized by Sutton et al. looks at the retrieval performance of all possible sub-queries [7]. This version of CRFs is referred to as CRF-*perf*, because it is based on the performance of the possible sub-queries. A standard CRF will optimize label accuracy, CRF-*perf* optimizes projected search performance. CRF-perf is defined as follows:

$$P_m(y|x) = \frac{\exp(\sum_{k=1}^{K} \lambda_k f_k(\mathbf{x}, \mathbf{y}))m(\mathbf{y})}{Z_m(\mathbf{x})} \tag{11}$$

$$Z_m(x) = \sum_{y} \exp\left(\sum_{k=1}^{K} \lambda_k f_k(\mathbf{x}, \mathbf{y})\right)m(\mathbf{y},\mathbf{M}) \tag{12}$$

where $m(\mathbf{y})$ is the retrieval performance of a sub-query $\mathbf{y}$, m is a specific performance measure function, such as average precision, and M is the retrieval method used. Retrieval methods will be discussed further in section 5.2.2.

5.2 Types of Features

Xue et al. use three types of features to represent different levels of dependencies within a query [8]. Those types are as follows:

Independency Features consist of a single word each, which allows the general function $f_k(\mathbf{x}, \mathbf{y})$ to be specialized as $f_k(x_i, y_i)$, since x_i is the only word looked at. An example of an independency feature is unigram term frequency, the number of times a single word (or "unigram")appears, independent of all other words.

Local Dependency Features illustrate dependencies between words in a query. The general function $f_k(\mathbf{x}, \mathbf{y})$ can be specialized as $f_k(x_i x_j x_k ..., y_i y_j y_k)$. An example of a local dependency feature is an organization name like "University of Minnesota," which represents a single entity even though it contains more than one word.

Global Dependency Features describe properties of a selected sub-query, which is generated from all query words and their labels. Since Global Dependency Features use all the words in a query, $f_k(\mathbf{x}, \mathbf{y})$ can be used unaltered. An example of a global dependency feature is sub-query length, which depends on all the words in a sub-query.

5.3 Retrieval Methods

Recall that CRF-perf describes a general framework for different retrieval models and performance measures, indicated by $m(\mathbf{y},M)$ where m is the performance measure function and M is the retrieval model. Xue et. al demonstrate four different retrieval models, but these are not the only types that can be used within the CRF-perf framework [8]. Those methods are the query likelihood model using the sub-query (SubQL), the query likelihood model using both the original query and the sub-query (QL+SubQL), the sequential dependency model using the sub-query (SubDM),

and the combination of the sequential dependency model on the original query with the query likelihood model on the sub-query (DM+SubQL), which will be described in greater detail in sections 5.3.1 to 5.3.4.

5.3.1 SubQL

The query likelihood model is the probability that a document is relevant to a given query. SubQL, the query likelihood model using a subquery Q_s, can be calculated as follows:

$$score_{QL}(D, Q_s) = \sum_{t \epsilon T(Q_s)} log(P(t|D)) \quad (13)$$

where $T(Q_s)$ is a set of query terms of Q_s, t is a term in $T(Q_s)$, and D is a document. $P(t|D)$ is the probability of finding t in D.

5.3.2 SubDM

The sequential dependency model is the likelihood that two adjacent terms in a query are related [5]. SubDM, the sequential dependency model using the sub-query, can be calculated as follows:

$$\begin{aligned} score_{DM}(D, Q_s) = \lambda_T & \sum_{t \epsilon T(Q_s)} log_{10}(P(t|D)) \\ +\lambda_O & \sum_{o \epsilon O(Q_s)} log_{10}(P(o|D)) \\ +\lambda_U & \sum_{u \epsilon U(Q_s)} log_{10}(P(u|D)) \end{aligned} \quad (14)$$

where $O(Q_s)$ denotes a set of ordered bigrams (two-word sequences) extracted from Q_s and $U(Q_s)$ denotes a set of unordered bigrams extracted from Q_s. λ_T, λ_O, and λ_U are weight parameters usually set, respectively, at 0.85, 0.1, and 0.05, according to Metzler and Croft [5].

5.3.3 QL+SubQL

QL+SubQL, a combination of the original query Q and a sub-query, where both parts use the query likelihood model. It can be calculated as follows:

$$\begin{aligned} score(D, Q, Q_s) = \alpha * score_{QL}(D, Q) \\ +(1 - \alpha) * score_{QL}(D, Q_s) \end{aligned} \quad (15)$$

where α is a weighting parameter that Xue et. al set as 0.8 [8].

5.4 DM+SubQL

DM+SubQL is another query and sub-query combination, where the sequential dependency model is used for the original query and the query likelihood model is used for the sub-query. It can be calculated as follows:

$$\begin{aligned} score(D, Q, Q_s) = \alpha score_{DM}(D, Q) \\ +(1 - \alpha) score_{QL}(D, Q_s) \end{aligned} \quad (16)$$

where α is the same weighting parameter it was in section 5.3.3.

5.5 Results

Xue et al. tested their query reduction method on topics from Robust2004, WT10g, and Gov2. They only considered sub-queries with lengths between three and six words. The baseline methods they tested against included the sequential dependency model on the original query (DM), Bendersky and Croft's method (KeyConcept) [1], and Kumaran and Carvalho's method (SRank) [3]. SubQL, QL+SubQL, SubDM, and DM+SubQL were tested first using the single top- performing sub-query and the top 10 sub-queries [8]. The performance measures used were the mean average precision (MAP), which measures the relevance of returned documents with an emphasis on ranking relevant documents higher, and the precision at ten (P@10), which is the fraction of relevant documents among the first ten returned documents.

SubQL, QL+SubQL, and SubDM all performed a little better than KeyConcept, with an overall average of 0.52% improvement across the three collections. DM+SubQL showed the most significant improvement over the baseline methods using both the top one and top ten sub-queries. It showed an average improvement of 7.42% over KeyConcept. Standard DM was also a strong method, even garnering the best results on WT10g when using P@10 as the performance measure. Therefore, we can conclude that combining sub-queries with the original query yields better performance, and that DM is the optimal method to use on the original query. On average, Xue et. al's optimal methods for each collection and performance measure performed about 11% better than Kumaran and Carvalho's method, which itself showed an average improvement of about 8% over unreduced verbose queries [8, 3].

6. CONCLUSION

In this paper, we presented three different methods for reducing verbose queries: dependency parsing, query quality predictors, and subset distribution. Dependency parsing was tested on WT10g and Robust2004; query quality predictors were tested on Robust2004, TREC123, and a combination of those two collections; and subset distribution was tested on WT10, Robust2004, and Gov2.

All three methods improved search performance from unreduced queries. Additionally, both dependency parsing and subset distribution performed better than the Key Concept method described by Bendersky and Croft [1], and subset distribution performed better than query quality predictors method.

Acknowledgments

Special thanks to Elena Machkasova, my advisor for this paper, and to Elijah Mayfield, who made some invaluable suggestions.

7. REFERENCES

[1] M. Bendersky and W. B. Croft. Discovering key concepts in verbose queries. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 491–498, New York, NY, USA, 2008. ACM.

[2] S. Huston and W. B. Croft. Evaluating verbose query processing techniques. In *Proceeding of the 33rd*

international ACM SIGIR conference on Research and development in information retrieval, SIGIR '10, pages 291–298, New York, NY, USA, 2010. ACM.

[3] G. Kumaran and V. R. Carvalho. Reducing long queries using query quality predictors. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 564–571, New York, NY, USA, 2009. ACM.

[4] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[5] D. Metzler and W. B. Croft. A markov random field model for term dependencies. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 472–479, New York, NY, USA, 2005. ACM.

[6] J. H. Park and W. B. Croft. Query term ranking based on dependency parsing of verbose queries. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 829–830, New York, NY, USA, 2010. ACM.

[7] C. Sutton and A. Mccallum. *Introduction to Conditional Random Fields for Relational Learning*. MIT Press, 2006.

[8] X. Xue, S. Huston, and W. B. Croft. Improving verbose queries using subset distribution. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 1059–1068, New York, NY, USA, 2010. ACM.