Implementing Semantic Document Search Using a Bounded Random Walk in a Probabilistic Graph

Lubomir Stanchev
Computer Science Department
California Polytechnic State University
San Luis Obispo, CA, USA
stanchev@gmail.com

***Abstract*—Given a set of documents and an input query that is expressed using natural language, the problem of document search is retrieving all relevant documents ordered by the degree of relevance. Semantic document search fetches not only documents that contain words from the input query, but also documents that are semantically relevant. For example, the query "friendly pets" will consider documents that contain the words "dog" and "cat", among others. One way to implement semantic search is to use a probabilistic graph in which the input query is connected to the documents through paths that contain semantically similar words and phrases, where we use WordNet to initially populate the graph. Each edge in the graph is labeled with the conditional probability that the destination node is relevant given that the source node is relevant.**

Our semantic document search algorithm works in two phases. In the first phase, we find all documents in the graph that are close to the input query and create a bounded subgraph that includes the query, the found documents, and the paths that connect them. In the second phase, we simulate multiple random walks. Each random walk starts at the input query and continues until a document is reached, a jump outside the bounding subgraph is made, or the number of allowed jumps is exhausted. This allows us to rank the documents based on the number of random walks that terminated in them. We experimentally validated the algorithm on the Cranfield benchmark that contains 1400 documents and 225 natural language queries. We show that we achieve higher value for the *mean average precision* (MAP) measure than a keywords-based search algorithm and a previously published algorithm that relies on a variation of the probabilistic graph.

I. Introduction

Quite often, different words are used to describe the same concept in the query and in the documents. When this is the case, a simple keywords-based search algorithm will not retrieve all relevant document and recall will be low. For example, a keywords-based search for "furniture" will not fetch documents that do not contain the search term, but contain related words, such as "chair" and "couch". A *probabilistic graph* that contains words and phrases from the English language can help us address this problem. In our example, the input query will be connected to the word "furniture", which in turn will be connected to the words "chair" and "couch", which will be connected to documents that contain one or both words. In [40], we showed an efficient way for creating the probabilistic graph using information from WordNet and experimentally validated the quality of data in the graph. In this paper, we focus our attention on how the probabilistic graph can be used for semantic document search. Specifically, we present a novel algorithm that ranks the documents based on data from multiple random walks in a bounded subgraph of the probabilistic graph.

A document search engine needs to consider a medley of factors, such as which documents contain words from the input query, what is the frequency of these words in the documents, how rare are these words, are the words in the correct order and close to each other, how trust-worthy are the documents, and so on. In this paper, we present a document retrieval system that addresses some of these heuristics, including finding documents that contain rare words from the input query and finding documents that contain words that are similar to words from the input query. Our system does not consider the order or the proximity of the query words in each document and the trustworthiness of the documents. However, we believe that these features can be added to our algorithm and this remains a topic for future research. The hallmark of our system is that it can retrieve documents that contain words that are semantically similar to the words in the input query. This is an important characteristic because it allows us to find more relevant documents and increase the recall.

Our approach to semantic document search involves finding words and phrases that are similar to those in the input query. The most challenging part is building a precise probabilistic model that correctly captures the degree of semantic relevance between words. For example, our system should be able to determine whether a document that contains the word "chair" or a document that contains the phrase "coffee maker" is more relevant to a query that asks for "furniture". Note that computing the degree of semantic similarity between words is not trivial and it is sometimes challenging even for humans that are experts in the domain area. There is also a trade-off in our system between the precision of the answer and the efficiency of the algorithm, where tuning parameters allow us to balance between the two choices.

Most document retrieval algorithms are based on some version of the TF-IDF (stands for *term frequency – inverse document frequency* [20]) algorithm and the cosine document similarity metric and do not return documents that contain words and phrases that are similar to those in the input query. Those systems that do consider similar words (e.g., [21]) usually expand the input query by adding synonyms.

This approach differs from our system in two ways. First, we are capable of finding similar words that are not synonyms. Second, we measure the semantic similarity between the words and phrases in the query and in the documents. To summarize, our approach is more flexible and it allows us to find the probability that a document is relevant given that it contains words that are similar to words in the input query.

Our semantic document search system is based on a probabilistic graph [40]. The graph is initially built using information from WordNet, which contains about 150,000 of the most common words and phrases in the English language. WordNet uses the concepts of a word form and sense. A *word form* is a single word or a short phrases that represents a single entity, such as "sports utility vehicle". Every word form can represent one or more senses and each sense is represented by one or more word forms. For example "the season when leaves fall from the trees" and "a sudden drop from an upright position" are two of the senses of the word "fall". In the probabilistic graph, a node is created for every word form and every sense. Next, we use information from WordNet to create Horn clauses with probabilities. The logical formulas use only the unary predicate *rel*, which is true when the input concept is relevant. The Horn clauses with probabilities allow us to build a very precise Markov Logic Network (MLN) model [32] of our system and compute the weight of the edges in the graph. Note that the input query and documents are also added as nodes in the graph. The problem of finding and ranking relevant documents is translated into the problem of finding the probability that a document is relevant given that the input query is relevant. As [40] describes, computing the conditional probability of a node being relevant given that a different node in the probabilistic graph is relevant is straight forward when we consider a single path between the two nodes. However, when we consider multiple interweaving paths between two nodes in the graph, the problem becomes computationally expensive and we are not aware of any polynomial time solutions. This is why in this paper we present a Monte Carlo approximation algorithm that performs multiple random walks that start from the node for the input query.

We experimentally validate our semantic search algorithm on the Cranfield benchmark that contains 1400 documents and 225 queries. Human subjects have determined the documents that are relevant for every query. We compare our algorithm with the TF-IDF algorithms that is implemented in Apache Lucene and the algorithm from [39], which looks only at disjoint paths when comparing two nodes. The experimental section shows that our semantic search algorithm produces higher value for the *mean average precision* (MAP) over all queries than the other two algorithms. The reason why our system has higher value for the MAP measure than the Apache Lucene system is because we find not only documents that contain words from the input query, but also documents that contain words and phrases that are semantically similar to those in the input query. In comparison with our previous work ([39]), the presented system performs better because our old algorithm considers only disjoint paths between the input query and each of the documents and it does not take into account the complex interweaving network of edges that can exist in the probabilistic graph.

In what follows, in Section II we review related research. Section III does a quick overview of the algorithm for creating logical formulas with weights from WordNet. The novelty here is that we also show how to create logical formulas that link documents and queries to words and phrases from WordNet. Section IV reviews how the logical formulas can be transformed into a probabilistic graph. The main contribution of the paper is in the next sections. Section V presents an approximation algorithm for finding the top-N relevant documents. Section VI shows how the algorithm compares with the Apache Lucene algorithm that is based on keywords-matching and our old algorithm from [39] on the Cranfiled benchmark [6], while concluding remarks and areas for future research are outlined in Section VII.

II. Related Research

In this section, we present a chronological overview of the major breakthroughs in semantic search research. In 1986, W. B. Croft proposed the use of a *thesaurus* of concepts for implementing semantic search [9]. The words in both the user query and the documents can be expanded using information from the thesaurus, such as the synonym relationship. Sequentially, there have been multiple papers on the use of a thesaurus to implement semantic search (e.g., [14], [17], [22], [35], [42]). This approach, although very progressive for the times, differs from our approach because we consider indirect relationships between words (i.e., relationships along paths of several words). We also do not apply query and document expansion. Instead, we use the probabilistic graph to find the documents that are semantically related to the input query. Similarly to the approach in [9], we use a probabilistic model to rank the documents in the result. Croft also proposed retrieving documents based on user interaction, where this direction has been further extended in the area of folksonomies [12]. Our system currently does not allow for user interaction when computing the list of relevant documents. However, we believe that allowing interactive mode during document search and implementing user profiling can improve our system and we identify this topic as an area for future research.

In later years, the research of Croft was extended by creating a graph that represents a semantic network [7], [31], [36] and graphs that contain the semantic relationships between words [3], [2], [8]. Later on, Ponzetto and Strube showed how to create a graph that only represents inheritance of words in WordNet [24], [37], while Jeh and Widom showed how to approximate the similarity between phrases based on information about the structure of the graph in which they appear [18]. All these approaches differ from our approach because they do not consider the strength of the relationship between the nodes in the graph. In other words, there are no weights that are associated with the edges in the graph.

The problem of semantic search is somewhat related to the problem of *question answering*. Instead of returning a

set of documents, question answering deals with the problem of finding the answer to a question inside the available documents. Natural language techniques are used to determine the type of expected answer [15], [28], [38]. For example, if the natural language analyzer determines that the answer to a question must be an animal, than words or concepts in the documents that can represent an animal are identified as potential query answers.

Since the early 1990s, research on LSA (stands for *latent semantic analysis* [11]) has been carried out. The approach has the advantage of not relying on external information. Instead, it considers the closeness of words in text documents as proof of their semantic similarity. For example, LSA can be used to detect words that are synonym [25]. This differs from our approach because we do not consider the closeness of the words in a document. We only consider the order of the words in the definition of a WordNet sense when we build the probabilistic graph, where we assume that the first words are more important. Although the LSA approach has its applications, we believe that WordNet provide higher quality of data than the input documents alone.

Since the late 1990s, ontologies have been examined as tools to improve the quality of the data that is returned by information retrieval systems [34]. However, ontologies use the Boolean search model. An ontology language, such as OWL, can be used to precisely annotate the input documents. Queries are expressed in a language that is based on mathematical logics, such as SPARQL, and a document is either part of the query result or it is not. Unlike the probabilistic model that is used in this paper, there is no notion of approximate query answering or ranking the documents based on their relevance to the input query. Therefore, this approach is better suited for query answering than for document search problems [27], [1], [5]. Research on automatic annotation of documents with OWL descriptions is also relevant [23], [29], [13].

Note that there are papers that consider a hybrid approach to information retrieval using both an ontology and keywords matching. For example, [33] examines how queries can be expanded based on the information from an OWL knowledgebase. Alternatively, [41] proposes a ranking function that depends on the length of the logical derivation of the result, where the assumption is that shorter derivations will produce more relevant documents. Unfortunately, these approaches are only useful in the presence of an ontology. However, research on automatic annotation of documents with OWL descriptions is still in its early stages of development.

Lastly, note that recent research has examined using a random walk over a version of the probabilistic graph for computing similarity between words ([16]) and between texts [30]. However, unlike our approach, they use the random walks to create stationary distributions for words (or texts) and compare two stationary distributions to compute the similarity between the respective words (or texts).

III. Creating the Horn Clauses

A. About WordNet

In our study, we use WordNet 3.0, which contains approximately 150,000 different word forms. Recall that a word form is a single word or short phrase, such as "United Nations". Throughout the paper, we will refer to both words and word forms as *terms*. WordNet also contains information about the senses of each term, where a term can have many senses and a sense can be represented by many terms. When a term has multiple senses, WordNet specifies the frequency (i.e., the popularity) of each sense. For each sense, WordNet gives us its definition and example use of a term that represents the sense in a sentence.

WordNet also contains information about the relationship between senses. The senses in WordNet are divided into four categories: nouns, verbs, adjectives, and adverbs. For example, WordNet stores information about the hypernym and hyponym relationships between nouns. The *hypernym* relationship corresponds to the super class relationship. For example, "canine" is a hypernym of "dog". The *hyponym* relationship is the reverse (i.e., it corresponds to the "kind-of" relationship). For example, "dog" is a hyponym of canine. WordNet also provides information about the meronym and holonym relationship between noun senses. The *meronym* relationship corresponds to the "part-of" relationship. Note that WordNet provides three types of meronyms: *part*, *member*, and *substance*. The three types of meronyms can be explained with the following examples: a "tire" is part of a "car", "car" is a member of "traffic jam", and a "wheel" is made from "rubber", respectively. The *holonym* relationship is the reverse of the meronym relationship. For example, "building" is a holonym of "window". For verbs, WordNet defines the *hypernym* and *troponym* relationships. X is a hypernym of Y if performing X is one way of performing Y. For example, "to perceive" is a hypernym of "to listen". The verb Y is a troponym of the verb X if the activity Y is doing X in some manner. For example, "to lisp" is a troponym of "to talk". Lastly, WordNet defines the *related to* and *similar to* relationship between adjective senses, which are self-explanatory. WordNet does not define any relationships for adverbs.

B. The Probability Model

We create a random variable for each sense and each term in WordNet. We use the single predicate *rel* in our model that tells us if the concept is relevant. We will model WordNet as a set of formulas of the form $rel(X) \Rightarrow rel(Y)$. Following the MLN model [32], the weight of a formula is computed as the natural logarithm of the odds of the formula being true, that is, $ln(\frac{p}{1-p})$. However, since we want all available evidence to have positive influence on the formulas, we first apply a transformation from the interval [0,1] to the interval [0.5,1] for each probability value before computing the weight of the formula. Note that, in the MLN model, events with probabilities below 0.5 are considered as negative events, that is, events that will most likely not happen. Equation 1 and 2 are

shown next, where we use $P^e(Y|X)$ to denote our confidence of the formula being true and refer to this value as the *evidence probability*.

$$weight(rel(X) \Rightarrow rel(Y)) = ln(\frac{P^+(Y|X)}{1 - P^+(Y|X)}) \quad (1)$$

$$P^+(Y|X) = 0.5 + \frac{P^e(Y|X)}{2} \quad (2)$$

As an example, suppose that the probability of someone who is interested in the word "chair" is also interested in the word "table" is 10%. Then $P^+(table|chair) = 0.55$ and the weight of the formula $rel(chair) \Rightarrow rel(table)$ will be equal to $ln(0.55/0.45) = 0.2$.

C. Creating Horn Clauses from the Documents and Queries

Consider the query q_1 "Furniture of the European Renaissance". There are three non-noise words in the query. We will create the following Horn clauses to represent this relationship.

$$rel(q_1) \Rightarrow rel(furniture), (minMax(0, 0.9, \frac{1}{3}))$$
$$rel(q_1) \Rightarrow rel(european), (minMax(0, 0.9, \frac{1}{3}))$$
$$rel(q_1) \Rightarrow rel(renaissance), (minMax(0, 0.9, \frac{1}{3}))$$

Note that we put the evidence probabilities in parenthesis, where Equations 1 and 2 will be used to translate them into weights. The *minMax* function is defined as follows and is used to smoothen the input.

$$minMax(minValue, maxValue, ratio) =$$
$$minValue + (maxValue - minValue) \cdot \frac{-1}{log_2(ratio)}$$

In almost all cases, the function returns a number between the first two arguments. The only exception is that, in order to avoid division by 0, we define $minMax(minValue, maxValue, 1) = 1.2 \cdot maxValue$. In general, the evidence probability for linking queries to terms is computed as $minMax(0, 0.9, ratio)$, where *ratio* is the number of times the term appears in the query divided by the total number of words in the query. The special case is when there is a single non-noise word in the query and then we use the second version of the *minMax* equation. The number 0.9 represents the probability that if we are interested in the query, then we are also interested in one of the terms in the query. The number is relatively high because almost always when we are interested in a query, we are also interested in one of the words in it.

Next, suppose that the word "chair" appears a total of ten times in the titles of documents. Consider the document d_1 with title "chair ergonomics and accessories". We can represent the relationship between the word "chair" and d_1 using the following formula.

$$rel(chair) \Rightarrow rel(d_1), (minMax(0, 0.6, \frac{1}{10}))$$

In general, the evidence probability for linking a term to a document that contains it in its title is computed as $minMax(0, 0.6, ratio)$, where *ratio* is the number of times the term appears in the title of the document divided by the total number of times the term appears in the title of any document. The parameter 0.6 denotes the probability that given that we are interested in a term, we are also interested in one of the documents that contains the term in its title. Note that the formula rewards terms that are rare because we want to give greater weight to documents that contain rare query words in their title.

Lastly, suppose that the word "chair" appears a total of 1,000 times in the bodies of documents. Consider the document d_1 in which the word appears ten times. We can represent this knowledge using the following formula.

$$rel(chair) \Rightarrow rel(d_1), (minMax(0, 0.3, \frac{10}{100}))$$

In general, the evidence probability for linking a term to a document that contains it in its body is computed as $minMax(0, 0.3, ratio)$, where *ratio* is the number of times the term appears in the body of the document divided by the total number of times the term appears in the body of any document. The parameter 0.3 denotes the probability that given that we are interested in a term, we are also interested in one of the documents that contains the term in its body. Note that the number is half of the size of the number for document titles because we believe that a term that appears in the title of a document is twice as significant as a term that appears in the body of a document. Again, we reward rare terms by giving higher value for the evidence probability of such terms.

D. Creating the Horn Clauses for WordNet

A previous paper [40] contains a detailed overview of the algorithm that models WordNet as a set of Horn clauses with weights. Here, we only show a previously unpublished example that demonstrate how the algorithm works.

In the previous subsection, we showed how the query "Furniture of the European Renaissance" can be connected to the word "furniture" and how the word "chair" can be connected to a document that contains the word in its title and body. Here, we show how the words "furniture" and "chair" can be connected, which will allow us to connect the query "Furniture of the European Renaissance" to documents that contain the word "chair".

First, note that the word furniture has a single sense: "furnishings that make a room or other area ready for occupancy". Accordingly, we will create the following formula.

$$rel(furniture) \Rightarrow, rel(furnishings\ that\ make\ a\ room \ldots), 10$$

Note that since the number 10 is a weight and not an evidence probability, it is not put in parenthesis. This translates to evidence probability of 99.99%. The idea is that since the word *furniture* has a single sense, someone who is interested in the word must be also interested in this sense.

Second, note that this sense has 25 hyponyms, including the senses for words bed, cabinet, seat, and table. For our example,

we are interested in the sense for the word seat: "furniture that is designed for sitting on". Accordingly, we will create the following formula.

$$rel(\textit{furnishings that make a room} \ldots) \Rightarrow$$
$$rel(\textit{furniture that is designed for sitting on}),\ (0.9 * \frac{10}{1000})$$

In order to understand how the evidence probability was computed, suppose that size of our example sense for "seat" is 10 and the sum of the sizes of all 25 hyponyms of the sense for the word "furniture" is 1000. In general, we compute weight for hyponym formulas as 0.9 multiplied by the size of the sense and divided by the sum of the sizes of all the hyponym senses of the initial sense. The number 0.9 represents the probability that given that we are interested in a sense, we are also interested in one of its hyponyms. We use information from the British National Corpus (BNC) [4] to estimate the size of the sense. Roughly, the equation looks at the frequency in textbooks, as recorded in BNC, of the different words that represent a sense and takes the weighted average of these frequencies.

Next, note that the sense for the word seat "furniture that is designed for sitting on" has as a hyponym that is the sense "a seat for one person" for the word "chair". Accordingly, we will create the following formula.

$$rel(\textit{furniture that is designed for sitting on}) \Rightarrow$$
$$rel(\textit{a seat for one person}),\ (0.9 * \frac{10}{100})$$

Here, we assumed that the size of the sense for "a seat for one person" is 10 and the sum of the sizes of all 9 hyponyms of the sense "furniture that is designed for sitting on" is 100.

Next, given that someone is interested in a sense, they must also be interested in all the words that represent the sense. Specifically, we have the following formula.

$$rel(\textit{a seat for one person}) \Rightarrow rel(chair), 10$$

In the next section, we will create an edge in the graph for each logical formula. In other words, we have shown one way the words "furniture" and "chair" will be connected in the probabilistic graph.

IV. Creating the Probabilistic Graph

We start by creating a node in the graph for each term and each sense, that is, for each random variable. Next, we convert the evidence probabilities of the formulas to weights using Equations 1 and 2. Note that there can be several identical formulas with possibly different weights that are generated. When this is the case, we will merge all such formulas into a single formula. The weight of the new formula is equal to the sum of the weights of the old formulas. Note that adding the weights is consistent with the MLN model [32] because the probability of a world W being true is computed using the following equation.

$$P(W) = \frac{1}{total} \cdot e^{\sum\limits_{F} weight(F) * |F(W)|} \tag{3}$$

In the equation, *total* is a normalizing constant that is used to make sure that the probabilities over all worlds add up to one. The sum is over all formulas F in our knowledgebase. The expression $weight(F)$ is used to denote the weight of the formula F and $|F(W)|$ is equal to one when the formula F is true in the world W and is equal to 0 otherwise. Obviously, merging identical formulas by adding up their weights follows the above formula.

Next, we add an edge between X and Y in the graph for each logical formula of the following type.

$$rel(X) \Rightarrow rel(Y),\ w$$

The weight of the edge will be converted to a probability and will be computed using the following equations.

$$p = \frac{1}{1 + e^{-w}} \qquad edge\,weight = \frac{2 * p - 1}{sum\,of\,edge\,weights}$$

The first equation converts the weight to a probability. The second equation maps the probability from the interval [0.5,1] back to the interval [0,1] and divides the result by the sum of the weights of all edges that leave the source node X. This guarantees that the sum of the weights of all the edges that leave a node will be equal to one, which will be important when we perform our random walk algorithm. Note that in the probabilistic graph that was constructed, the weight of each edge is equal to the probability that a user is interested in the destination concept given that they are interested in the source concept, where we assume that the user is interested in only one of the destination concepts.

V. Finding Relevant Documents

In this section, we will describe three algorithms for finding relevant documents. In the next section, we will compare how they work on the Cranfiled benchmark [6]. The first two algorithms use a scoring function to compute the distance between a query and a document, where the resulting documents are ranked relative to the value of the scoring function in descending order.

First, let us examine the scoring function that is used by Apache Lucene [10], which is popular software that contains a toolkit of routines for information retrieval. Given a document d and a query q, the scoring function is defined as follows.

$$score(q,d) = \sum_{t\ in\ q} (\sqrt{tf(t\ in\ d)} * (1 + log_2(\tfrac{numDocs}{docFreq(t)+1}))^2)$$

In the equation, $tf(t\ in\ d)$ denotes the number of appearances of the term t in the document d, *numDocs* refers to the total number of documents, and $docFreq(t)$ refers to the number of documents in which the term t appears. This follows the TF-IDF equation because the second part of the equation is one way of computing the inverse document frequency. The scoring function can be multiplied by boosting and normalizing parameters, which are skipped because they are optional parameters and require user tuning. The problem

with this approach is that it does not compare similar terms from the query and the documents.

Second, let us quickly examine the algorithm from [39]. Given a document d and a query q, the scoring function is defined as shown in Equations 4 and 5.

$$score(q,d) = \sum_{Pt \text{ is a cycleless path from node q to node d}} P_{Pt}(d|q) \quad (4)$$

$$P_{Pt}(d|q) = \prod_{(n_1,n_2) \text{ is an edge in the path } Pt} P(n_2|n_1) \quad (5)$$

In Equation 5, $P(n_2|n_1)$ is used to denote the weight of the edge from the node n_1 to the node n_2. Informally, we compute the directional similarity between two nodes in the graph as the sum of the non-overlapping paths between the two nodes, where we eliminate cycles from the paths. We compute the similarity between two nodes along a path as the product of the weights of the edges along the path, which follows the Markov chain model. The problem with this approach is that it is not deterministic because there can be multiple way of enumerating the non-overlapping paths between two nodes. Depending on our choice, we can produce results that are significantly different.

Lastly, we presented our bounded random walk algorithm that consists of two phases. In the first phase, we start at the query node and find all documents that can be potentially relevant without worrying about ranking. The algorithm is shown in Fig. 1. Initially, *boundingSet* is the empty set. As we find more nodes, we keep adding them to the set. The initial call to the method is *depthFirst*$(query, 1, 0)$. The algorithm performs a depth-first search starting at the query node and finds all documents that can be potentially relevant. Note that if we do not reach a document at the end of the path, then we remove the visited nodes on the current path from the bounding set because they do not lead to a document. In the experimental results, we used $maxDepth = 10$. This means that paths of length more than 10 edges are of no interest because the strength of the evidence weakens as the length of the path grows. We also set $minDistance = 1/(1000 * num\_documents)$, where *num_documents* is the total number of documents. The idea is that the more documents we have, the lower the probability that we will reach a particular document. To summarize, the algorithm for the first phase is built on the hypothesis that if a document is relevant, then there must exist at least one relatively short path to the document where the product of the weights of the edges along the path is bigger than our cutoff value.

The second phase of the algorithm is shown in Figures 2 and 3. Initially, the *findRelevantDocuments* is called with the query node as input. The algorithm populates the *frequency* array, which tells us how many times each document was visited. The relevant documents are the ones with the highest frequency, where the result will be sorted by the value of the frequency in descending order. The *findRelevantDocument* method simply calls the *randomWalk* method a bunch of times, where the number of times the method is called depends on the number of documents in the bounding set. The idea is that if the bounding set is small, then there is no reason to perform too many random walks.

Algorithm 1 $depthFirst(currentNode, distance, depth)$

```
if currentNode is a document then
    add currentNode to boundingSet
    return
end if
if depth > maxDepth or distance < minDistance or
currentNode is in boundingSet then
    remove nodes that belong only to current path from
    boundingSet
    return
end if
add currentNode to boundingSet
for all neighbors neighbor of currentNode do
    depthFirst(neighbor, distance *
    edgeWeigth(currentNode, neighbor) , depth + 1)
end for
```

Fig. 1. First phases of the Bounded Random Walk Algorithm.

Algorithm 2 $findRelevantDocuments(queryNode)$

```
for i ← 1 to |bounding_set| * 1000 do
    documentID = randomWalk(queryNode)
    if documentID = −1 then
        continue
    end if
    frequency[documentID] ← frequency[documentID] + 1
end for
```

Fig. 2. Second phases of the Bounded Random Walk Algorithm.

The *randomWalk* function starts at the input node and keeps hoping until it finds a document, it jumps outside the bounding set, or the number of allowed hops is exhausted. Note that it is possible that we reach a node that is a dead end (each subsequent hop leads to a node that is already visited). When this is the case, we just return -1, which means that we were unable to find a document.

VI. Experimental Results

The Cranfield benchmark [6] contains 1400 short documents about the physics of aviation. Each document contains a title and a short body that is usually around 10 lines. As part of the benchmark, 225 natural language queries were created. The documents and queries were examined by experts in the area and the documents that are relevant to each query were identified. The relevant documents were clustered in four groups. Highly relevant documents were given relevance score

Algorithm 3 $randomWalk(currentNode)$

```
for i ← 0 to maxDepth do
  if currentNode is a document then
    return currentNode //found document
  end if
  if currentNode not in boudingSet then
    return -1
  end if
  repeat
    nextNode ← getRandomNextNode(currentNode)
  until nextNode is not already visited or loop has run for
  maxDept * 10 times
  if above loop ran maxDept * 10 times then
    return -1 // dead end
  end if
  curentNode ← nextNode
end for
return -1 // no document found and hop limit reached
```

Fig. 3. The method takes a random walk starting at *currentNode*. It returns a document if it finds one and -1 otherwise.

of 1, less relevant documents were given a relevance score of 2, and even less relevant documents were given a relevance score of 3, while documents of minimum interest were given a relevance score of 4.

Table I shows the Mean Average Precision (MAP) score for our algorithm, the algorithm that only considers disjoint paths [39], and the Apache Lucene algorithm. Note that for the Apache Lucene algorithm we doubled the frequency of the words in document titles to make it comparable to the other two algorithms. For each algorithm, we ran four experiments. In the first experiment, we only considered the documents with relevance score of 1 to be relevant. In the second experiment, we only considered the documents with relevance scores of 1 and 2 to be relevant and so on. Each of the experiments took about 10 minute to complete on a typical laptop with an Intel Core i7 processor and 4GB of main memory. When implementing our algorithm, we set $maxDept = 10$ and $minDistance = 1/(1000 * 1400)$. If we increase the maximum length of a path that we consider, then the value of the MAP score decreases slightly because we start discovering associations between words that are not true. The same applies for the *minDistance* parameter value. If we decrease it significantly, for example by a factor of 1000, we start discovering semantic relationships between words that are not related and the value for the MAP score decreases.

For each query, we computed the *mean average precision* score, which is also known as the MAP score. Consider the query Q. Let $\{D_i\}_{i=1}^{d}$ be the relevant documents. Let R_i be the set of documents that are retrieved by the algorithm until document D_i is returned. Then the MAP score for the query Q is defined as the average precision of R_i over all values, or

	Rel. 1	*Rel. 1-2*	*Rel. 1-3*	*Rel. 1-4*
Bounded random walk	**0.31**	**0.32**	**0.32**	**0.37**
Disjoint paths [39]	0.29	0.29	0.30	0.35
Lucene algorithm	0.25	0.25	0.27	0.29

TABLE I
MAP VALUES FOR DIFFERENT ALGORITHMS AND DEGREES OF RELEVANCE FOR THE CRANFIELD BENCHMARK.

formally as shown in Equation 6.

$$MAP(Q) = \frac{1}{d}\sum_{i=1}^{d} Precision(R_i) \tag{6}$$

The precision for R_i is defined as the fraction of retrieved documents that are relevant, or formally as shown in Equation 7.

$$Precision(R_i) = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} \tag{7}$$

Next, let us examine Table I in more details. The MAP score is the average MAP value over all 225 queries. The top algorithm is the algorithm that is described in the paper. As the table suggests, it produces higher value for the MAP metric than the Apache Lucene algorithm and the disjoint paths algorithm [39]. The reason we get better results than the Apache Lucene algorithm is because the later algorithm performs simple keywords matching and does not consider the semantic relationship between the terms in queries and documents. It is clear from the table that our algorithm produces especially good results when we consider documents with relevance score from 1 to 4 to be relevant. The reason is that our algorithm is strong at identifying documents that are weakly related with the input query. Conversely, the Apache Lucene algorithm fails to discriminate between documents that do not contain the query words. Our algorithm produces better results than the disjoint path algorithm because we created a very precise mathematical model of the words in the English language. Our bounded random walk algorithm is not only fast, but it also allows us to rank the documents very precisely based on the probability that each document is relevant given that the input query is relevant.

VII. CONCLUSION AND FUTURE RESEARCH

In this paper, we presented a bounded random walk algorithm for finding documents that are relevant to an input query. It turns out that creating a bounding subgraph is very efficient. The reason is that most paths are quickly pruned because the product of the weights of the edges quickly drops below our threshold. For example, for most queries the bounded subgraph is relatively small. Once the bounded subgraph is found, performing multiple random walks in it is also very efficient. The major advantage of using the bounded subgraph is that we eliminate paths that do not lead to a document quickly. This makes the random walks very efficient. In the experimental section of this paper, we showed that our

algorithm produces good results on the Cranfield benchmark because it is not only efficient, but it also computes the probability of a document being relevant in a very precise manner. Note that our algorithm is also tunable. We can change the $maxDept$ and $minDistance$ parameters of the algorithm that creates the bounded subgraph and the number of random walks in order to trade accuracy for better performance.

We have identified two areas for future research. First, we plan on extending the probabilistic graph with information from DBpedia [26]. We believe this will allow us to capture more semantic relationships between phrases and will increase the accuracy of our algorithm. Second, we want to incorporate term ordering and proximity in our algorithm. For example, [19] shows how to create graphs that represent the term ordering and we believe that we can incorporate such graphs in our algorithm. Again, the hope is that this will increase the accuracy of the algorithm.

REFERENCES

[1] A. A. Bernstein and E. Kaufmann. Gino - A Guided Input Natural Language Ontology Editor. *Fifth International Semantic Web Conference*, 2006.

[2] M. Agosti and F. Crestani. Automatic Authoring and Construction of Hypertext for Information Retrieval. *ACM Multimedia Systems*, 15(24), 1995.

[3] M. Agosti, F. Crestani, G. Gradenigo, and P. Mattiello. An Approach to Conceptual Modeling of IR Auxiliary Data. *IEEE International Conference on Computer and Communications*, 1990.

[4] L. Burnard. Reference Guide for the British National Corpus (XML Edition). *http://www.natcorp.ox.ac.uk*, 2007.

[5] P. Cimiano, P. Haase, and J. Heizmann. Porting Natural Language Interfaces between Domains – An Experimental User Study with the ORAKEL System. *International Conference on Intelligent User Interfaces*, 2007.

[6] C. W. Cleverdon. The Significance of the Cranfield Tests on Index Languages. *In Proceedings of Special Interest Group on Information Retrieval*, pages 3–12, 1991.

[7] P. Cohen and R. Kjeldsen. Information Retrieval by Constrained Spreading Activation on Sematic Networks. *Information Processing and Management*, pages 255–268, 1987.

[8] F. Crestani. Application of Spreading Activation Techniques in Information Retrieval. *Artificial Intelligence Review*, 11(6):453–482, 1997.

[9] Croft. User-specified Domain Knowledge for Document Retrieval. *Nineth Annual International ACM Conference on Research and Development in Information Retrieval*, pages 201–206, 1986.

[10] D. Cutting. Apache Lucene. *http://lucene.apache.org*, 2014.

[11] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by Latent Semantic Analysis. *Journal of the Society for Information Science*, 41(6):391–407, 1990.

[12] T. Gruber. Collective knowledge systems: Where the social web meets the semantic. *Web Journal of Web Semantics*, 2008.

[13] R. V. Guha, R. McCool, and E. Miller. Semantic Search. *Twelfth International World Wide Web Conference (WWW 2003)*, pages 700–709, 2003.

[14] A. M. Harbourt, E. Syed, W. T. Hole, and L. C. Kingsland. The Ranking Algorithm of the Coach Browser for the UMLS Metathesaurus. *Seventeenth Annual Symposium on Computer Applications in Medical Care*, pages 720–724, 1993.

[15] E. H. Hovy, L. Gerber, U. Hermjakob, M. Junk, and C. Y. Lin. Question Answering in Webclopedia. *TREC-9 Conference*, 2000.

[16] T. Hughes and D. Ramage. Lexical Semantic Relatedness with Random Graph Walks. *Computational Linguistics*, 7(June):581–589, 2007.

[17] K. Jarvelin, J. Keklinen, and T. Niemi. ExpansionTool: Concept-based Query Expansion and Construction. *Springer Netherlands*, pages 231–255, 2001.

[18] G. Jeh and J. Widom. SimRank: A Measure of Structural-context Similarity. *Proceedings of the Eight ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 538–543, 2002.

[19] John Violos and Konstantinos Tserpes and Evangelos Psomakelis and Konstantinos Psychas and Theodora Varvarigou. Sentiment analysis using word-graphs. *International Conference on Web-Intelligence, Mining, and Semantics*, 2016.

[20] K. Jones. "a statistical interpretation of term specificity and its application in retrieval". *Journal of Documentation*, 28(1):11–21, 1972.

[21] R. Jones, B. Rey, O. Madani, and W.Greiner. Generating Query Substitutions. *International Conference on World Wide Web*, 2006.

[22] S. Jones. Thesaurus Data Model for an Intelligent Retrieval System. *Journal of Information Science*, 19(1):167–178, 1993.

[23] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff. Semantic Annotation, Indexing, and Retrieval. *Journal of Web Semantics*, 2(1):49–79, 2004.

[24] R. Knappe, H. Bulskov, and T. Andreasen. Similarity Graphs. *Fourteenth International Symposium on Foundations of Intelligent Systems*, 2003.

[25] T. K. Landauer, P. Foltz, and D. Laham. Introduction to Latent Semantic Analysis. *Discourse Processes*, pages 259–284, 1998.

[26] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia ? A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[27] V. Lopez, M. Sabou, and E. Motta. PowerMap: Mapping the Real Semantic Web on the Fly. *Fifth International Semantic Web Conference (ISWC2006)*, 2006.

[28] D. Moldovan, S. Harabagiu, M. Pasca, R. Mihalcea, R. Goodrum, and R. Girju. LASSO: A Tool for Surfing the Answer Net. *Text Retrieval Conference (TREC-8)*, 1999.

[29] B. Popov, A. Kiryakov, D. D. Ognyanoff, D. Manov, and A. Kirilov. KIM A Semantic Platform for Information Extraction and Retrieval. *Journal of Natural Language Engineering*, 10(3):375–392, 2004.

[30] D. Ramage, A. N. Rafferty, and C. D. Manning. Random Walks for Text Semantic Similarity. *Workshop on Graph-based Methods for Natural Language Processing*, pages 23–31, 2009.

[31] L. Rau. Knowledge Organization and Access in a Conceptual Information System. *Information Processing and Management*, 23(4):269–283, 1987.

[32] M. Richardson and P. Domingos. Markov Logic Networks. *Machine Learning*, 62(1-2):107–136, 2006.

[33] C. Rocha, D. Schwabe, and M. Aragao. A Hybrid Approach for Searching in the Semantic Web. *Thirteenth International World Wide Web Conference (WWW 2004)*, pages 374–383, 2004.

[34] S. S. Luke, L. Spector, and D. Rager. Ontology-Based Knowledge Discovery on the World Wide Web. *Internet-Based Information Systems: Papers from the AAAI Workshop*, pages 96–102, 1996.

[35] M. Sanderson. Word Sense Disambiguation and Information Retrieval. *Seventeenth annual international ACM SIGIR conference on Research and development in information retrieval*, 1994.

[36] P. Shoval. Expert consultation system for a retrieval database with semantic network of concepts. *Fourth Annual International ACM SIGIR Conference on Information Storage and Retrieval: Theoretical Issues in Information Retrieval*, pages 145–149, 1981.

[37] Simone Paolo Ponzetto and Michael Strube. Deriving a Large Scale Taxonomy from Wikipedia. *22nd International Conference on Artificial Intelligence*, 2007.

[38] K. Srihari, W. Li, and X. Li. Information Extraction Supported Question Answering. *In Advances in Open Domain Question Answering*, 2004.

[39] L. Stanchev. Creating a Phrase Similarity Graph from Wikipedia. *Eight IEEE International Conference on Semantic Computing*, 2014.

[40] L. Stanchev. Creating a Probabilistic Graph for WordNet using Markov Logic Network. *Proceedings of the Sixth International Conference on Web Intelligence, Mining and Semantics*, pages 1–12, 2016.

[41] N. Stojanovic. On Analyzing Query Ambiguity for Query Refinement: The Librarian Agent Approach. *Twenty Second International Conference on Conceptual Modeling*, pages 490–505, 2003.

[42] Y. Yang and C. G.Chute. Words or Concepts: The Features of Indexing Units and their Optimal use in Information Retrieval. *Seventeenth Annual Symposium on Computer Applications in Medical Care*, pages 685–68, 1993.