Search Optimizations for Entity-Linking Systems

Raul E. Jordan RAULJORDAN@COLLEGE.HARVARD.EDU
A.B. Candidate in Computer Science.
Harvard University.

Melissa C. Lee LEE05@COLLEGE.HARVARD.EDU
A.B. Candidate in Computer Science.
Harvard University

Sameer N. Mehra SMEHRA@COLLEGE.HARVARD.EDU
A.B. Candidate in Computer Science.
Harvard University

Abstract

This paper describes a search-based approach to linking keyword entities in text to their corresponding wikipedia entries through optimizations of semantic-relatedness and a novel abstraction using existing methods. We will solve this problem for a given input through the optimization of certain parameters and the confidence value of a keyword-link pair, which we will refer to as a "relevance score", using methods such as Explicit Semantic Analysis and a Vector Space Model. Most of this project will explore different methods of defining this relevance function and its results. We present an approach that has wide applicability to understanding natural language and making it easier for computer agents to discover what a user is referring to with high confidence based on a structured knowledge base.

1. Introduction

Named-entity recognition is a complex task that has yielded powerful results in recent years, and the strength of using a structured corpora of Wikipedia has given rise to a wide-variety of techniques in the fields of machine learning and artificial intelligence as part of the greater scope of natural language processing that seeks to identify entities effectively. At a broad level, this means being able to identify entities, also known as "keywords", within a corpus and identify them based on some prior structured knowledge. On a narrower panorama, the task of entity linking based on wikipedia is a task we will tackle by creating a system that is able to process a passage of text and assign to each keyword its corresponding wikipedia link. While it may be easy for humans to do this by identifying the entire context of a corpus and being able to gauge what the best link could be for an entity, computer systems need a way of quantifying how the relatedness of a potential link could change given the other keywords in the text.

In particular, if the input text is *Lindsay was a fantastic actress*, we want to link *Lindsay to Lindsay Lohan* knowing that the keyword *actress* is in the same context rather than *John Lindsay of Balcarres*, a secretary of state of Scotland. We need an abstraction that is able to effectively capture this notion of context and how "good" of a fit a particular link could be to a keyword. This paper will focus on defining a robust measure of link-relatedness, which we will refer to as "relevance", as an abstraction to achieve a search-based solution to the task of named-entity recognition. Being able to use a structured knowledge base as a way of quickly identifying entities from context has a myriad of applications ranging from speech recognition to intelligent personal assistants, which are especially relevant to recent systems such as Facebook‘s “M” that rely heavily on natural language processing and context to detect what a user truly wishes to communicate.

1.1 A New Abstraction Based on Existing Methods

Our new approach formulates named-entity linking (NEL) as a search problem by assigning each keyword a wikipedia link and taking into account previous link assignments to effectively and accurately determine the following ones through a method known as a “relevance function”. This is an abstraction that relies on existing algorithms, such as TF-IDF and Explicit Semantic Analysis (ESA) (Hachey 40) to determine if a link is the most likely candidate for a given keyword based on the set of all possible candidate links that word can take, as well as all the other keyword-link assignments around it. This function will map a keyword-link pairing to a score between 0 and 1 known as a *relevance score* based on the existing *semantic relatedness* algorithms such as ESA. Our formulation abstracts most of the problem-specific details of NEL by first assigning each keyword in a corpus a wikipedia link individually with a high confidence, and then using a variant of local search to iterate over each assignment and use the assignments around it to potentially modify its current link into a potentially better one. The key to our formulation is that every assignment of a keyword to a candidate link is crucial to improve the accuracy of the following assignments on each iteration of our system until we reach a satisfying state where the relevance scores of every assignment do not vary as much as before. Much of our discussion focuses on the problem of assessing an effective relevance function mapping and taking into account previous link-keyword assignments to achieve a terminal state of our formulation.

1.2 Range of Problems Addressed

It is crucial that in our system, every assignment of a wikipedia link to a keyword can effectively increase the accuracy of the next assignments our algorithm chooses. By accuracy, we refer to the number of “correct” keyword-link pairings in an input that we know are correct based on a testing set.

As an example, consider when the input text to our system is the sentence “Jobs was fantastic at Apple”. Our system will take in the input and preprocess it and find an initial, complete assignment of links for each keyword. Say we have assigned [Steve] Jobs to his respective wikipedia page and that we have assigned Apple to the wikipedia page for the fruit. When our system iterates over the assignments of our input after this initial assignment and reaches Apple, it will consider that [Steve] Jobs was assigned to the wikipedia page for the CEO of Apple Inc, and now the possible candidate link space of the keyword Apple will change from

$$\textsc{Candidates}(Apple) = \{Apple(Fruit),\ AppleInc., ...\}$$

to simply contain

$$\textsc{Candidates}(Apple) = \{Apple\ Inc.\}$$

Which is the correct assignment for Apple in this context.

One of the problems we encounter above is the task of semantic-relatedness - that is, how can we quantify a keyword-link pairing as correct or incorrect and how can we use the corpus of the wikipedia pages corresponding to the other link assignments to make a current assignment better. However, it might not be the case that a particular future assignment has any dependence on the previous ones or the other entities around it. We will discuss possible ways of solving this problem by using our problem formulation while at the same time identifying the inherent limitations of capturing qualitative notions of relatedness through specific numerical assignments, both in this approach and in the greater field of natural language processing.

2. Problem Formulation

We will begin by formulating our problem as we do in class

S	State Space	The set of all possible dictionaries of word/link assignments created from our input words. The keys are the keywords, and the values are a tuple of an assigned link and its relevance score to the keyword. 0 means no relevance and a value close to 1 means highly relevant. This is found through measures of Cosine Similarity and TF-IDF Weighting as discussed in the following section.
A	Action Space	All possible wikipedia pages we could assign to a link. An action will involve assigning one wikipedia page to one link and finding its relevance score.
res(s, a) = s'	Transition Model	A new resulting dictionary after assigning or changing a wikipedia page assignment to one word in our current dictionary
s_0	Initial State	A dictionary of keywords where the keys are the words and the values are all initially assigned independently to their highest relevance score candidate link.
Terminal(s)	Terminal State	We reach a terminal state after many steps of iteration until the variance between relevance scores of each assignment to the next for a keyword does not change greatly. This can simply be the the state we obtain after N iterations and N can be changed at runtime

Our algorithm first takes in a block of text and preprocesses it to obtain the keywords and build up our initial state of our search problem as a dictionary where the keys are the keywords and the values are their respective wikipedia page assignments which are initially set to be the highest relevance scoring candidate link for that word and its corresponding relevance score, such as $\{keyword: (link, score), ...\}$. Our initial input could be "An airplane has a heavy wing”, and our preprocessed initial state will be a dictionary of the form

$$\{airplane:\ (candidateLink_1,\ 0.7553),\ wing:\ (candidateLink_2,\ 0.8483)\}$$

which will be what our search formulation will begin with. The next step is using a variant of local search that can modify this complete and potentially inconsistent assignment into a complete assignment that can be highly accurate after many iterations. Now we describe how an initial state is obtained and how its context is then processed to improve every link assignment.

2.1 High Level Description: Obtaining an Initial State

Our system first identifies the key terms in an input using NLTK. Now for each keyword, we obtain the set of its candidate links using the Wikipedia python API. For every candidate link, obtain its relevance score to the input text. At the end, we obtain the link that has the maximum relevance score and assign it to its keyword. We return an initial state as described in our formulation, where it is a dictionary with each keyword being a key and the values being their corresponding maximum relevance scoring link and the associated score. Obtaining the maximum link relevance score is described below.

```
def getMaxRelevance(inputText, keyword):
    linkRelevanceScores = []
    for candidateLink in candidateLinks(keyword):
        score = relevance(inputText, candidateLink)
        linkRelevanceScores.append((candidateLink, score))

        return tuple of (candidateLink, score) with the maximum score in
            linkRelevanceScores
```

2.2 The Relevance Function

Given a link, a keyword, and a current state, how do we determine if that link is the best fit for that keyword effectively using the context given to us? In our implementation, there is a crucial consideration: we must be able to use previous assigned keywords as factors in our score. This means that given the content of a wikipedia link, we must use the current keyword and the keywords around it to gauge how good of a match that link is to our current keyword. To accomplish this task, we will discuss several implementations of different measures of semantic-relatedness and approaches that have been used in literature to take advantage of the structured corpora of wikipedia.

As abstractions in our implementation, we define the LinkRelevance as the relevance score between a wikipedia link’s page content and the input text to our system and the DocumentRelevance as the relevance score between two different wikipedia pages’ content.

Formally, we define the functions above as

$$\textsc{LinkRelevance} : \mathcal{L} \times\ \mathcal{W} \rightarrow [0, 1]$$

$$\textsc{DocumentRelevance} : \mathcal{L} \times \ \mathcal{L} \rightarrow [0, 1]$$

Where $\mathcal{L}$ is the space of candidate links a particular keyword can take and $\mathcal{W}$ is the space of words in our input text.

Our relevance function formally abstracts the notion of semantic relatedness by capturing the best match between a keyword and a potential link with a score assigned to it. To begin to implement it, we define the following key terms

$$\textsc{Context} = \{w \in \mathcal{W} - \{c\} \mid c = \text{current keyword}\}$$

$$\textsc{Similarity} = \cos(A, B) = \frac{A \cdot B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Where Similarity is the cosine similarity between two vectors, which is chosen to enforce a relevance score between 0 and 1. We will also use Term-Frequency Inverse-Document Frequency (TF-IDF) - a statistic that scores the importance of terms in a corpus based on two metrics:

1. weighing a term up if it appears frequently in a document
2. weighing a term down if it appears frequently in multiple documents. This will mean it is most likely not a unique keyword

$$\text{TF-IDF} = \sqrt{f(k, p)} \cdot \log\left(\frac{|\mathcal{D}|}{\{d \in \mathcal{D} \mid k \in \mathcal{D}\}}\right)$$

Where k is a current term, p is its candidate link's wikipedia page content, $f(k, p)$ is the term frequency of k in p, $|\mathcal{D}|$ is the total number of documents, and $\{d \in \mathcal{D} \mid k \in d\}$ is the number of documents that contain a current term, k (Chisholm 149).

The key to an effective relevance function will involve encoding terms with TF-IDF weights. There are variations of this approach that have been successful in literature, but one that stands out over others (Yano 9) in terms of time and space complexity is known as Explicit Semantic Analysis. Dojchinovski provides a summary of the method, which we will work into our formulation and modify to implement our final relevance function through the following high-level description:

A selected keyword is preprocessed and represented as a TF-IDF term vector. For each keyword $w_i \in$ Context, the method retrieves the possible Wikipedia articles from the Wikipedia API $c_1, ..., c_n$ containing w_i. The semantic relatedness of the word w_i with candidate link c_j is computed such that the similarity of association between w_i and c_j is multiplied with the TF-IDF weight of w_i. The relatedness score for any two documents is determined by computing the cosine similarity between the vectors of this candidate link and keyword semantic relatedness. (Dojchinovski 3).

The relevance computed in the previous section is the LinkRelevance, which only takes into account the input text and a candidate link. However, the DocumentRelevance will be useful when using previous link assignments to determine a next assignment in our formulation.

2.3 High Level Description: Using Context to Improve Link Assignments

We now formulate a new algorithm based on the local search formulation we have previously seen in the course. We begin with an initial complete assignments of links to keywords from the previous section. Then, we iteratively implement the following process: for every keyword in the state in some order such as left to right, we compare each of its candidate links to all the other assigned links in the state using DocumentRelevance and obtain the sum of these document relevances. We do a convex combination of both LinkRelevance and this sum of DocumentRelevance for every candidate link and the currently assigned link. If we find that this combined value for a candidate link is less than the current assignment, we replace that assignment. At the end, we return the modified initial state. This is similar to every action taken in a local search formulation, where the neighbors of a given state are obtained and modified until a consistent assignment is reached. In this formulation, there is no definite formalization of a consistent assignment, so we will simply repeat this process for N iterations, where N is large enough to improve the scoring of each keyword-link assignment over time in our terminal state.

Since DocumentRelevance and LinkRelevance measure two different quantities, we can do a comparison between them through a convex combination that introduces a new parameter, $\alpha \in [0,1]$ that can weigh both against each other. We can obtain a measure that captures both into a single value, ψ. We change the assigned link when the candidate has a smaller psi value because it is a convex combination.

$$\psi = (1-\alpha)\cdot \textsc{LinkRelevance} + \alpha \cdot \textsc{DocumentRelevance}$$

Where document relevance is calculated between a link and every other assigned link in a state. We can use this as follows: when considering a candidate link for a keyword, we can compute its value of ψ and also compute the value of ψ for the currently assigned link. If the candidate link has a lower resulting ψ value, we can replace the current assignment with this candidate link and move on to the next keyword. It could be the case that $\alpha = 1$, the case where only document-relevance is considered, could give a decently accurate set of assignments in our system. However, we want to explore values for $0 < \alpha \leq 1$ that can give us an increase in accuracy by considering the context of a keyword to determine the best candidate links.

We implement these procedures as shown below

```
def runLocalSearch(self, state, alpha, iterations):
    for i in range(iterations):
        for keyword in state:
            for candidateLink in candidateLinks(keyword):
                candidateDocumentRelevances = []
                currentAssignmentDocumentRelevances = []
                for otherAssignedLink in context(keyword):
                    # Obtain the Document Relevances as a list
                    candidateDocumentRelevances.append(documentRelevance(candidateLink,
                        otherAssignedLink)
                    currentLinkDocumentRelevances.append(documentRelevance(currentLink,
                        assignedLink)
                # Obtain the LinkRelevances
                currentLinkRelevance = linkRelevance(keyword, currentLink)
                candidateLinkRelevance = linkRelevance(keyword, candidateLink)

                # Obtain a convex combination of the link relevances and the
                # sum of the document relevances
                candidatePsi = (1 - alpha)*candidateLinkRelevance +
                    alpha*sum(candidateDocumentRelevances)
                currentPsi = (1 - alpha)*currentLinkRelevance +
                    alpha*sum(currentLinkDocumentRelevances)


                # If the candidate link's convex combination is less than the
                    current
                # link's convex combination, we replace that assignment
                if candidatePsi < currentPsi:
                    replaceAssignment(keyword, candidateLink)

    return initialState
```

3. Algorithmic Optimizations

One way we optimized our algorithm to account for the high number of iterations over the wikipedia search results from the API calls in our system was to cache the entire candidate link set for a keyword in memory. This way we can have constant access to the corpus of wikipedia pages every time we need to perform a relevance score computation or even when assigning a link to a keyword. Our system takes a long time to cache the results at first, but the following computations will be entirely dependent on our implementation afterwards.

Another problem we optimized was the ordering heuristic within our algorithms. As we have seen before in Constraint Satisfaction Problems that used local search to arrive at complete and consistent assignments, iterating over keywords from left to right when modifying assignments may not necessarily be the best or most efficient ordering heuristic. We would instead want to change a keyword-link assignment that has the smallest possible candidate link space in our algorithm and proceed in that order.

3.1 Smallest Candidate Link Space Heuristic

Given a state, how does the algorithm determine the keyword to assign based on the other assignments? Ideally, we want to restrict the state space of every unassigned keyword based on the other assignments to find the one that has the smallest space of candidate links as in the *Jobs* and *Apple* example of section 1.2. This is necessary for our algorithm to make sense as a search problem. We want to be able to use current assignments to help determine future ones, and using this abstraction will allow us to implement our solution effectively.

```
def restrictCandidateLinkSpaces(keywords):
    # For each keyword, we keep track of its space of candidate links
    candidateLinkSpaces = {}
    # We initialize the state spaces for the
    # unassigned keywords
    for keyword in keywords
        # Obtains the list of candidate links for a keyword
        candidateLInkSpaces[keyword] = wikipedia.search(keyword)

    # Now for each keyword, we restrict its state
    # space based on all the other assigned keywords
    for keyword in keywords:
        # Determine how related the keyword is to the context.
        # For each link in the state space, calculate the relevance
        # to the context of the keyword and restrict state space to eliminate
        # low scoring links
        for link in candidateLinkSpaces[keyword]:
           candidateLinkSpaces[keyword] =
               findRelevanceAndRestrictStateSpace(keyword, link)
```

4. Algorithmic Analysis & Evaluation

As a way of assessing algorithmic efficiency, we begin by determining the time complexity of the approaches we followed. Our choice of encoding states as dictionaries of keyword-link mappings allows many of our subroutines to have constant time access to viewing and modifying these states, reducing the overhead from the problem formulation. However, there are additional factors in our implementation that can significantly slow down its run-time. The relevance function subroutine utilizes TF-IDF in its implementation, an algorithm known to have a worst-case running time of $O(|N| \cdot |V|)$ where N is the corpus of the wikipedia page of a keyword and V is the space of all candidate links for that keyword. Using an inverted index, however, can improve TF-IDF to $O(|B| \cdot |V|)$ where B is the average over all terms in the query document of the number of other documents in which a keyword appears (Salakhutdinov 2).

Reducing the overhead of all these subroutines will be implementation-dependent and will only be practically efficient for a set of input text with small size. Our large amount of API calls to obtain the text content of a wikipedia link are inefficient only at the start of our algorithm where we fetch every possible page content for the candidate links in our input and cache them in local memory. Once the cache is built, our algorithm will be purely dependent on the python runtime and on our implementation. The calculations involved in finding the relevance and cosine similarity are built upon the Natural Language Toolkit (NLTK) and Scikit-Learn as well as NumPy, which all have very optimized runtimes built upon lower-level C bindings, so they are the best approach to the more computation-intensive tasks of our system.

5. Algorithmic Limitations

Currently, our formulation simply iterates for a set number of iterations to improve the accuracy of the keyword-link assignments, but there is no set number of iterations that will guarantee us the best solution to human levels. Also, we are highly limited by underlying methods of our relevance function, namely TF-IDF and the cosine similarity of the vector space model. In particular, this approach does not take into account the ordering of words within a wikipedia page content, and assumes that all the terms are independent statistically, which is not the case in practice. Also, our improvement of linking accuracy relies on the choices of α through convex combinations of LinkRelevance and DocumentRelevance, which is a very problem-specific heuristic that would be difficult to expand outside of this problem domain. TF-IDF weighting also causes the problem of poorly representing pages with long content because of the high dimensionality of the weighted vector, which is the opposite of what we would intuitively need. For example, a longer page usually means that there is a lot of data and knowledge on the topic at hand and might be one of the most possible links to a keyword in an input.

6. Testing of Major System Features

Our system testing will rely on measuring the linking accuracy of our search optimizations. We define linking accuracy as the percentage of correctly linked entities in a given input text. We will begin with a testing set of sample raw text we have created along with a solution set corresponding to each of those texts with their entities correctly linked to what we know are their right wikipedia pages. We will run every text in this testing set on our system, and measure the number of correctly named entities per input when compared to the solution set with a different number of iterations. All of our test sets are inside of the test/ folder in our project. Sample tests can be ran by running python test.py.

We will perform the same test, but by changing the parameter α in our convex combination of LinkRelevance and DocumentRelevance at a different range of values and analyze how the accuracy of our system changes based on an increase in choice of α. We provide this testing set along with the solution set inside of the test subfolder within our github code, which can be run as explained within the appendix of this report.

We run our testing data for 1 iteration on 3 different choices of alpha. We will measure accuracy as the average accuracy for a given alpha by running the entire testing data set.

Alpha	**Average Accuracy Over All Test Data**
0	15.87%
0.25	80.56%
0.5	91.67%
0.75	91.67%
1	91.67%

7. Key Takeaways from Algorithmic Implementation

Our implementation has taught us that there is still a lot of work to be done in effective and fast entity linking systems. As an active area of research in NLP, even using the best python packages for the task, we still encountered many difficulties with achieving human-level effectiveness in this problem, and also discovered how difficult it is to create effective formalisms for systems dealing with language in this manner.

8. Appendix I

We can run our program for $\alpha = 1$ and for one iteration as follows on an input of related entities like

```
python core.py “Thermodynamics is the study of heat” --iterations 1 --alpha 1
```

we obtain Thermodynamics: (en.wikipedia.org/wiki/Thermodynamics, 0.44943641652398209)
Study: (en.wikipedia.org/wiki/Study, 0.44943641652398209)
Heat: (en.wikipedia.org/wiki/Heat, 0.44943641652398209)

If we run the same example for 5 iterations and an alpha of 1 we obtain the same result

Now for an alpha value of 0.5
we obtain Thermodynamics: (en.wikipedia.org/wiki/Thermodynamics, 0.44943641652398209)
Study: (en.wikipedia.org/wiki/Study, 0.44943641652398209)
Heat: (en.wikipedia.org/wiki/Heat, 0.44943641652398209)

Now for an alpha value of 0.1
we obtain Thermodynamics: (en.wikipedia.org/wiki/Third law of thermodynamics, 0.11145825029743928)
Study: (en.wikipedia.org/wiki/Study, 0.44943641652398209)
Heat: (en.wikipedia.org/wiki/Enthalpy of vaporization, 0.12731567199023894)

Now for an alpha value of 0.0
we obtain Thermodynamics: (en.wikipedia.org/wiki/Third law of thermodynamics, 0.11145825029743928)
Study: (en.wikipedia.org/wiki/Study, 0.44943641652398209)
Heat: (en.wikipedia.org/wiki/Enthalpy of vaporization, 0.12731567199023894)

9. Appendix II

To run our system, download the code in our Github Repository `https://github.com/rauljordan/OptimizedEntityLinking` Download all the python package requirements by using the command env/bin/pip install -r requirements.txt and navigate to the OptimizedEntityLinking subfolder within it. From there, run python core.py with the following options

```
python core.py “Airplanes are aircraft” --iterations 10 --alpha 0.1
```

If any errors occur related to NLTK, make sure you go to a python shell and write import nltk and then nltk.download() to download the necessary NLTK packages (tokenizers and stemmers packages). In the input above, the argument after core.py is the text you wish to provide, iterations is the number of iterations that our search will perform before returning a final state, and alpha is the convex combination parameter described in section 2.3 that can take values in the range 0 to 1. The default value of iterations is 1 and the default alpha is 0. The system will then print out to the console the different stages of the algorithms and their runtime, and will end by printing out a terminal state where all the keywords are linked to their corresponding wikipedia pages. A sample output of a terminal state is

$$\{airplanes : (wikipedia.org/Airplane, 0.983), aircraft : (wikipedia.org/Aircraft, 0.983)\}$$

10. Appendix III

Raul focused on developing the report for the project as well as formulating the problem in terms of concepts used in class. Melissa focused on the design and presentation of our project. Sameer focused on developing the code base as well as the algorithmic development of our project.

11. References

Yano, Kang. 2015. Taking advantage of Wikipedia in Natural Language Processing.

Chisholm, Hachey. 2015. Entity disambiguation with web links. Transactions of the Association for Computational Linguistics, 3:145156.

Dojchinovski, Kliegr, Laek, and Zamazal. “Wikipedia search as effective entity linking algorithm”. In Text Analysis Conference (TAC) 2013 Proceedings. NIST, 2013.

Hachey, Ben, Will Radford, Joel Nothman, Matthew Honnibal, and James R. Curran. “Evaluating Entity Linking with Wikipedia.” Artificial Intelligence: 130-50.

Luo, Gang, et al. “Joint Named Entity Recognition and Disambiguation.”

Milne and Witten. An effective, low-cost measure of semantic relatedness obtained from Wikipedia links. Proc. AAAI08 Workshop on Wikipedia and Artificial Intelligence, Chicago, IL.