Journal of Artificial Intelligence Research 1 (2015) 1-15 Submitted 10/15; published IN PROGRESS

Classical Search Optimizations for Entity-Linking Systems

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

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

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

Abstract

This paper describes a simple heuristic approach to solving large-scale constraint satisfaction and scheduling problems. In this approach one starts with an inconsistent assignment for a set of variables and searches through the space of possible repairs. The search can be guided by a value-ordering heuristic, the *min-conflicts heuristic*, that attempts to minimize the number of constraint violations after each step. We will solve this problem through local search and validating our results using the relevance function using some of the methods of artificial intelligence such as simulated annealing. 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 speech based agents to discover what a user is referring to based on the wikipedia knowledge base.

1. Introduction

Named entity recognition is a *explain*. This 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. On a narrower panorama, the task of entity linking based on wikipedia is a task we describe as assigning the most-related. This paper will focus on defining a robust measure of relatedness to achieve a classical search-based, artificial intelligence solution to the task of named-entity recognition to accomplishing effective entity linking by optimizing various methods of quantifying this most-related method. Being able to use a structured KB as a way of quickly identifying entities from context has a myriad of applications ranging from speech recognition, to intelligent personal assistants, which is especially relevant to recent systems such as Facebook‘s “M”.

We will follow measures of semantic relatedness known from recent natural language processing research as ways of obtaining robust measures that determining whether or not an entity and link assignment is accurate enough.

1.1 A New Abstraction Based on Existing Methods

Our new approach formulates named-entity linking (NEL) as a search problem through a construction known as a relevance function. This is an abstraction that relies on existing algorithms, such as TF-IDF, Explicit Semantic Analysis, and the Bag of Words (BOW) model (Hachey 40) to determine if a link is the most likely candidate for a given keyword. 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 the ones mentioned before. Our formulation abstracts most of the problem-specific details of NEL into simply searching a state space for the best possible pairing of entities to candidate links above a certain relevance score threshold. The key to our formulation is that every assignment of a keyword to a candidate link is crucial to determining the next possible assignment effectively. That is, every action is *deterministic* and there are multiple approaches our implementation follows to structure NEL as a problem that can be solved through algorithms such as Depth First Search or Breadth First Search. 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

As a search problem, every state and action pair in a given search path will be important to the next state and action pair we determine. As previously mentioned, our solution will require that each previous link-keyword assignment provide crucial information for future assignments. One of the problems we encounter is how to determine the relevance between assignments as we traverse the state space of possible links a keyword can be paired to. In many cases, keywords in an input are highly related and one keyword can be immediately helpful in constraining the possible candidate state space of another keword in the input. For example,

This is what we refer to as *linking-dependence*. 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. For example, assume that our input is "Steve Jobs was fantastic at Apple" and that we have successfully linked "Steve Jobs" to his wikipedia page with a high confidence (relevance score) value. Suppose now that we are assigning "Apple" to its relevant link and that its candidate link state space contains.

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

However, taking into account that we have just assigned "Steve Jobs" before will restrict $Candidates(Apple)$ to

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

We will refer to this as linking-dependence. We need to talk more about the specific problems that arise with determining relevance effectively.

2. Classical Search Formulation

Our system 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 (None, 0) where the the second element of this tuple is a "relevance score" for each particular link-keyword pair.

For example, our initial input could be "An airplane has a heavy wing", and our pre-processed initial state will end up being a dictionary of the form

$$\{`airplane' : (None,\ 0),\ `wing' : (None,\ 0)\}$$

This input is then passed into a search agent that uses optimal and suboptimal algorithms such as Breadth First Search, Depth First Search, Uniform Cost Search, and A^* Search with problem specific heuristics and returns a terminal state that has every word assigned a link with a relevance score above a certain threshold based on the relevance function of choice, such as

$$\{`airplane' : (wikipedia.org/Airplane,\ 0.98),\ `wing' : (wikipedia.org/Wing,\ 0.87)\}$$

There are additional considerations in our search formulation we have to take into account when formulating our state space this way. We will discuss these by going into detail of how a specific Algorithm such as Depth First Search in our system finds a highly relevant terminal state in the following section.

2.1 High Level Example: Depth First Search

We implement the Depth First Search algorithm specified in AIMA (citation needed) but with key differences in the how each node is expanded into all of its possible child nodes. This will be the main driver of our implementation that obtains the associated actions of these child nodes to append to the frontier of the search algorithm, which in this case is represented using a stack as a data structure. This is where relevance scores are computed and assigned to each possible keyword-link pairing.

Obtaining the child states for a given state will be as follows

1. In order from left to right on the input, pick a keyword to assign a link to.

2. Query the wikipedia API for every possible candidate link that keyword can take. Other literature on entity-linking mentions that candidate generation is critical for successful named-entity linking (Hachey 40), but our implementation already does this by abstracting this through the wikipedia API

3. Compute the relevance score between every possible link and the given keyword

4. Return a list of triples of the form

   $$[(keyword, link, score),\ ...\ ]$$

   that includes all possible candidate links and their associated score for a given keyword.

Afterwards, we build dictionaries for all of the possible resulting states from this list of successor states and we can push them to the search algorithms frontier.

Our algorithm ends when we find a state that has every keyword assigned with a relevance score of at least a certain value we decide between 0 and 1. This logic is implemented in the isGoalState function which checks for a terminal state in our implementation as in the AIMA algorithmic description. This is a score cutoff we have decided and we will discuss its limitations and the reasoning behind it in the following section.

3. The Relevance Function

Given a link, a keyword, and a current state, how do we figure out 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 somehow 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.

Formally, we define the relevance function as

$$R : \mathcal{S} \times \ \mathcal{L} \times \ \mathcal{W} \rightarrow [0, 1]$$

Where $\mathcal{S}$ is the state space of our search formalism
$\mathcal{L}$ is the space of candidate links a particular keyword can take
$\mathcal{W}$ is the space of keywords in our given state, which are simply the keys of our dictionary in our implementation.

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

$$\eta = \{w \in \mathcal{W} - \{c\} \mid c = \textit{current keyword}\}$$

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

Where η we define as the *context* of a given keyword and *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 certain terms 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 meaan it is most likely not a unique keyword

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

Where k is the current keyword, 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 wikipedia pages of the other keywords in the state, and $\{d \in \mathcal{D} \mid k \in d\}$ is the number of wikipedia pages corresponding to each of the other keywords that contain k (Chisholm 149).

The key to an effective relevance function will involve taking into account the similarity of the context of a given keyword to that keywords wikipedia entry using cosine similarity and encoding terms with TF-IDF weights. There are variations of this approach that have been successful, such as the ESA method which we will implement and is summarized as

The input text T is represented as a TF-IDF term vector. For each word $w_i \in T$, the method uses an inverted index to retrieve Wikipedia articles $c_1, ..., c_n$ containing w_i. The semantic relatedness of the word w_i with concept c_j is computed such that the strength of association between w_i and c_j is multiplied with the TF-IDF weight of w_i in T. The relatedness score for any two documents is determined by computing the cosine similarity between the vectors of document-concept semantic relatedness. (Dojchinovski 3)

4. Algorithmic Analysis & Evaluation

One of the most promising general approaches for solving combinatorial search

4.1 Implementation Critique

problems is to generate an initial, suboptimal solution and then to apply

5. Algorithmic Limitations

n-grams in paragraphs. Currently our searching formulation only works for single-word entity linking.

6. System Feature Testing

Testing

7. Key Takeaways From Implementation

Takeaways

8. Empirical & Graphical Results

9. Appendix I

A trace of the program showing how it handles key examples or some other demonstration of the program in action.

10. Appendix II

A clear description of how to use your system and how to generate the output you discussed in the write-up and the example transcript in Appendix 1. N.B.: The teaching staff must be able to run your system.

11. Appendix III

A list of each project participant and that participants contributions to the project. If the division of work varies significantly from the project proposal, provide a brief explanation.