GEEK: Incremental Graph-based Entity Disambiguation

Alexios Mandalios
National Technical University of Athens
Athens, Greece
amandalios@islab.ntua.gr

Konstantinos Tzamaloukas
National Technical University of Athens
Athens, Greece
kot@dblab.ece.ntua.gr

Alexandros Chortaras
National Technical University of Athens
Athens, Greece
achort@cs.ntua.gr

Giorgos Stamou
National Technical University of Athens
Athens, Greece
gstam@cs.ntua.gr

ABSTRACT

A document may include mentions of people, locations, organizations, films, product brands and other kinds of entities. Such mentions are often ambiguous, with no obvious way for a machine to map them to real world entities, due to reasons like homonymy and polysemy. The process of recognizing such mentions in unstructured texts and disambiguating them by mapping them to entities stored in a knowledge base is known as Named Entity Recognition and Disambiguation (NERD) or Entity Linking.

In this paper, we introduce GEEK (Graphical Entity Extraction Kit), a NERD system that extracts named entities in text and links them to a knowledge base using a graph-based method, taking into account measures of entity commonness, relatedness, and contextual similarity. All relevant data is retrieved at runtime using public RESTful APIs. GEEK tries to push the performance limits of a straightforward disambiguation method, that doesn't require arduous training or a complex mathematical foundation.

CCS CONCEPTS

• **Information systems** → **Entity resolution**; **Information extraction**; • **Computing methodologies** → **Natural language processing**; • **Mathematics of computing** → *Approximation algorithms*;

KEYWORDS

Named Entity Recognition, NER, Named Entity Disambiguation, NED, NERD, Google Knowledge Graph, Wikipedia, k-partite graph, max weight k-clique, worst out heuristic

1 INTRODUCTION

Most pieces of text one can find online are to a large extent unstructured and unlabeled. Extracting the mapping between named entities appearing in text and real world objects stored in a knowledge base contributes a great deal towards understanding unprocessed written word. As an example, consider the sentence: "Arizona and Oklahoma are two of the ships that sank in Pearl Harbor during the events of World War II." A human reader, even one that is not familiar with 20th century history, can easily deduce the mapping of Figure 1. However, if we try to reproduce the same results using automated methods, then considerable effort is required in order to overcome the inherent ambiguity of natural language.

Even though NERD is a challenging NLP task, it has been extensively addressed in past research [22], as it is a crucial component for any system hoping to grasp the essence of natural language. The fact that NERD moderates (or ideally eliminates) the equivocal nature of natural language becomes evident even from our simple example of Figure 1. Knowing that a document contains references to two battleships, a military strike, and a war makes it easy to extract its topics and determine its position in an extended document collection, with minimal further processing.

This paper introduces GEEK (Graphical Entity Extraction Kit), a pipeline of tools and methods to perform NERD. It relies on Stanford CoreNLP for the named entity mention extraction task, and on Google's Knowledge Graph and Wikipedia APIs for collecting information on the extracted entities. The collected information is then mapped onto a graph, thus transforming the task of entity linking into a graph problem, which is easier to handle. The resulting graph problem is solved using a heuristic method, which incrementally refines the candidate entities. The entire pipeline is evaluated against established NERD systems on the GERBIL framework. GEEK is found to be competitive when compared against these systems in the NERD task, suggesting that it is a potent alternative to methods having a more complex analytical foundation.

The rest of the paper is structured as follows: Section 2 models the NERD process in general terms. In Section 3 we start the presentation of the proposed system by discussing how it performs the named entity recognition step, and in Section 4 we discuss the core of GEEK, the disambiguation step. Section 5 presents an extensive evaluation of the proposed system by comparing it to other state-of-the-art systems. Finally, Section 6 discusses related work, and Section 7 concludes the paper.

2 NERD MODELING

In this section we model NERD in a simple way that may serve as a general framework for NERD algorithms. We assume that we have a knowledge base $\mathcal{KB}$, which contains a mapping between entities of the world and Unique Resource Identifiers (URIs), which are concise and unambiguous. We also assume that there exists a set $\mathcal{N}$ of all names that can be used in natural language texts

LDOW2018, April 2018, Lyon, France

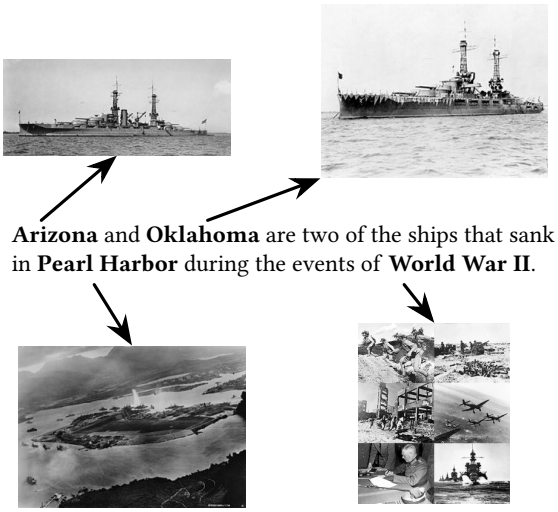

Figure 1: Named entity disambiguation in a short text.

to denote said entities in $\mathcal{KB}$. According to our natural language conventions, there is a mapping f between the entities in $\mathcal{KB}$ and those subsets of $\mathcal{N}$ that can be used to refer to them:

$$f : \mathcal{KB} \rightarrow 2^{\mathcal{N}}$$

This means that every entity in $\mathcal{KB}$ can be referred to in texts using only a specific set of names. If we model a text T as a simple string consisting of characters, then the appearance of a name $n \in \mathcal{N}$ in T means that there is a possible mention of the corresponding $\mathcal{KB}$ entity. The process of finding named entities in unstructured texts is called *named entity recognition* (NER), and can be described as finding substrings of T that map to any name $n \in \mathcal{N}$:

$$\text{NER} : T \mapsto M$$

where

$$M = \{m \mid m \text{ is a substring of } T \text{ and } \exists \text{URI} \in \mathcal{KB} \text{ s.t. } m \in f(\text{URI})\}$$

The process of mapping the named entities in M to specific URIs of $\mathcal{KB}$ is called *named entity disambiguation* (NED). NED is required because the same name may be used for different entities, i.e. there may be distinct URIs e_1, e_2 s.t. $f(e_1) \cap f(e_2) \neq \emptyset$. The first step for disambiguating a mention m is to generate an appropriate candidate entity set for m, denoted as E_m. This is defined as the set of individuals in $\mathcal{KB}$ that can be referred to as m:

$$E_m = \{\text{URI} \in \mathcal{KB} \mid m \in f(\text{URI})\}$$

After we generate candidate entity sets for all named entities in a text T, NED selects the most appropriate entity from each of those sets:

$$\text{NED} : m \mapsto e \in E_m, \text{ for each } m \in M$$

To sum up, NER identifies individual names in a text T and NED maps those names to the most appropriate candidates for them in $\mathcal{KB}$. Clearly, to be more efficient, NED could disambiguate jointly the entire M. The process of *named entity recognition and disambiguation* (NERD) combines NER and NED:

$$\text{NERD} : T \xrightarrow{NER} M \xrightarrow{NED} E$$

where E is the set of URIs finally selected by NED.

Consider text T = "Arizona and Oklahoma are two of the ships that sank in Pearl Harbor during the events of World War II" of Figure 1. NER gives us the set of names $M = \{\text{Arizona, Oklahoma, Pearl Harbor, World War II}\}$. Next, we use Google Knowledge Graph (GKG) as knowledge base $\mathcal{KB}$ and apply NED to map these mentions to URIs in $\mathcal{KB}$. The problem is the ambiguity of mentions. For example:

$$E_{\text{Arizona}} = \{/\text{m}/0\text{vmt}, /\text{m}/019\text{r}32, /\text{m}/06\text{rxnl}, \ldots\}$$

In particular, "Arizona" could refer to the American state of Arizona[1], a Pennsylvania-class battleship named Arizona[2], and the beverage manufacturing company known as Arizona[3], among other candidates. NED needs to identify the correct choice for the specific text T, one that abides with human understanding of written word:

$$\begin{aligned} \text{Arizona} &\mapsto /\text{m}/019\text{r}32 \\ \text{Oklahoma} &\mapsto /\text{m}/01\text{b}8\text{zk} \\ \text{Pearl Harbor} &\mapsto /\text{m}/0\text{gc}1\_ \\ \text{World War II} &\mapsto /\text{m}/081\text{pw} \end{aligned}$$

This mapping is also illustrated in Figure 1.

3 NAMED ENTITY RECOGNITION

As the above-presented model suggests, the first step in any NERD pipeline is the identification of the named entity mentions set M in a given text T. This step is important, since the discovery of exactly those noun phrases that denote named entities directly affects the quality of the final disambiguation results. For the NER part of GEEK we use Stanford CoreNLP[4] [15], a constantly evolving NLP toolkit. In particular, we use its Entity Mentions Annotator, which analyzes a text T and outputs a list of named entity mentions in T, that can be used as an approximation of M. Stanford CoreNLP offers three named entity recogintion models, developed using different training data:

- 3 class model, which discovers entities of type Location, Person, or Organization.
- 4 class model, which discovers entities of type Location, Person, Organization, or Misc, so that there's a category for entities that don't match the first three (miscellaneous).
- 7 class model, which discovers entities of type Location, Person, Organization, Money, Percent, Date, or Time.

Experimentation with the above CoreNLP models revealed that, while the 3 class model is able to capture most relevant entities, in some cases it can be complemented by the 4 class and 7 class models. Moreover, the 3 class model is relatively better in detecting the full span of multi-word named entities. Thus, aiming for maximum named entity recognition coverage, we combine Stanford CoreNLP's models using the following three step procedure:

(1) We extract named entities using Stanford CoreNLP's 3 class model.

[1] https://g.co/kg/m/0vmt
[2] https://g.co/kg/m/019r32
[3] https://g.co/kg/m/06rxnl
[4] https://stanfordnlp.github.io/CoreNLP/

(2) We extract named entities using Stanford CoreNLP's 4 class model, but keep only those entities that don't overlap with the ones we got from the first step.
(3) We extract named entities using Stanford CoreNLP's 7 class model, but keep only those entities that don't overlap with the ones we got from the first two steps. Furthermore, we reject any entities that have types of Money, Percent, Date, or Time, as we aren't interested in quantitative or temporal entities.

We should note that the type Stanford CoreNLP provides for each detected named entity may not always be correct (people may be recognized as locations, locations as organizations, and so on). For this reason, we use Stanford CoreNLP only to identify the named entity mentions in the text, and we do not use the provided type information in the disambiguation process.

4 NAMED ENTITY DISAMBIGUATION

4.1 Candidate Entity Generation

After discovering the set M of named entities in text T, the next step is to find the best mapping between those named entities and the canonical entities that reside in $\mathcal{KB}$. To that end, for each $m \in M$ we generate the set of candidate entities E_m that contains only those entities that one could refer to by saying m, using Google's Knowledge Graph (GKG)[5] as our $\mathcal{KB}$. Technically, we achieve this by using Google's Knowledge Graph Search API (GKG API)[6], Google's replacement for their deprecated Freebase[7]. Using GKG API, one can get a ranked list of GKG entities related to a given query string. In particular, for each matching entity, GKG API returns its names, the corresponding Wikipedia[8] information, its schema.org[9] types, etc. In our case, we use the API to retrieve GKG entities that match a certain named entity mention. For example, querying for "Arizona" fetches a number of entities that match that string, like the state of Arizona, the Pennsylvania-class battleship, the beverage company, and so on.

Unfortunately, not all entities this method yields belong to the set of candidate entities E_m for a mention m. For example, it's not unusual that GKG API returns an entity that is missing fields critical for our disambiguation framework, such as the corresponding Wikipedia information. Entities missing such important response fields are deemed useless for our disambiguation pipeline, and are immediately rejected.

Another inconvenience is that GKG API is a general purpose tool that has not been specifically designed to serve as generator of candidate entities to be used for NED. For example, when we query for "Arizona", GKG API returns, among other entities, Phoenix, the capital of the state of Arizona. It is pretty clear what happens here: GKG API returns not only those entities that could be referred to by saying "Arizona", but also entities closely related to "Arizona". That may make perfect sense for some applications, but in our case it poses a serious problem. Given that for a mention m we want E_m to be comprised of only those entities that we can refer to by saying m, we *shouldn't* allow Phoenix to be a candidate entity for "Arizona", as no one would use the name of a state to refer to its capital city. In order to mitigate the effect of this problem, we turn to Wikipedia for help. We try to make an educated decision about whether or not an entity can be referred to as m by querying Wikipedia's disambiguation pages and redirects. This is achieved by consulting Wikipedia's disambiguation pages for candidate entities and Wikipedia's redirect pages for possible aliases of entities, and giving priority to those entities returned by GKG API that are also returned by Wikipedia (as GKG API returns each entity's corresponding Wikipedia article). We fetch the required information using services provided by MediaWiki API[10]. We access the disambiguation pages and the redirects via the MediaWiki Links API[11] and the MediaWiki Redirects API[12] respectively.

By combining the aforementioned methods, we manage to construct an approximation for a mention's set of candidate entities. Building the sets of candidate entities properly is crucial for a successful disambiguation system [7], as, on one hand, including too few entities in a candidate entity set could lead to the exclusion of the appropriate entity, and, on the other hand, polluting a candidate entity set with too many entities could lead the disambiguation process astray.

4.2 NED measures

After building the set of candidate entities E_m for a named entity mention m, we need to select a concrete entity $e \in E_m$ to realize the mapping $m \mapsto e$. As stated above, our goal is try to make the same mapping a human reader would make. In order to do that, GEEK employs three measures akin to human thinking patterns.

4.2.1 GKG resultScore. When we query GKG API with a string m, then each returned entity is accompanied with a resultScore field. This is a positive real number that indicates how good a match is the returned entity for the given request. In our framework, this value is used as an alternative measure of an entity's *popularity prior* (also known as *prior probability* or *commonness*). Popularity prior is a measure of the conditional probability of one referring to an entity e, given the mention string m:

$$\text{Popularity Prior}[e|m] = \mathbb{P}[m \mapsto e|\text{mention } m \text{ appears in text}]$$

This conditional probability can be approximated in various ways, more prominently using Wikipedia and its links to deduce how often each surface form is used to refer to a specific entity in the encyclopedia. Popularity prior's usefulness as a disambiguation measure can be explained using what is seemingly a tautology: a mention m usually refers to the entity e that is referred to by m most of the time. However, it is one of the most widely used measures in NED systems [6, 12–14, 17, 21, 23–25]. Before we can use resultScore, we normalize it for all entities in each set of candidate entities E_m. As a result, in each E_m, the most popular entity will have a resultScore of 1, and all the rest will have $0 <$ resultScore < 1. For example, the most popular entity for $m =$ Arizona seems to be the state of Arizona, as expected.

[5] https://www.google.com/intl/es419/insidesearch/features/search/knowledge.html
[6] https://developers.google.com/knowledge-graph/
[7] https://developers.google.com/freebase/
[8] https://en.wikipedia.org/wiki/Main_Page
[9] http://schema.org/
[10] https://www.mediawiki.org/wiki/API:Main_page
[11] https://www.mediawiki.org/wiki/API:Links
[12] https://www.mediawiki.org/wiki/API:Redirects

4.2.2 Document Similarity. In prior work [13], the first few sentences of an entity's Wikipedia article have been used to extract the most relevant terms for the entity. We follow a similar approach. For every entity e returned by GKG API, the articleBody field contains the first few words of the corresponding Wikipedia article that describes the entity, denoted as T_e. Comparing the entire text T as a bag of words with those short descriptions can help us discover which entities are appropriate for the given context. The steps are:

(1) Tokenize T and T_e, remove stopwords, and stem the remaining tokens.
(2) Search for T's tokens in T_e using fuzzy string matching, for increased flexibility.
(3) Calculate a document similarity measure using the formula $\log(1 + |T \cap T_e|)/\log(1 + |T|)$. The logarithms serve to make sure we don't need high overlap of used words in T and T_e to achieve a high value. Also, the returned value is by definition normalized.

Simply comparing the words contained in two documents has the potential to guide the disambiguation process. For example, returning to Figure 1, the word "ship" would help a human understand that Arizona and Oklahoma are ships, and not states. The same goes for a system that uses this measure to calculate document similarity, as the word "ship", as well as several of its derivatives, appear multiple times in the respective articleBody fields.

4.2.3 Entity Relatedness. In the literature, it is common to assume that a text contains a few coherent topics [3, 4, 6, 8, 9, 12, 13, 21, 23–25, 27], so its entities are semantically related. This sort of "semantic locality" allows for *joint* or *collective* methods of disambiguation. These methods process all candidate entity sets of a document at the same time, and aim to select those entities (one from each set) that demonstrate maximum semantic relatedness. In most cases, Wikipedia's link structure is used as a way to calculate semantic relatedness between entities. Specifically, the more incoming links are shared between the Wikipedia articles describing two entities, the more semantically similar those entities are assumed to be. Most systems [4, 9, 11, 13, 14, 21, 24, 25] utilize an efficient derivative of the Normalized Google Distance [2] suggested by Milne and Witten [16], known as the Wikipedia Link-based Measure (WLM). One can easily gather an article's incoming links using the MediaWiki Linkshere API[13]. If IN_1 is the set of incoming links for e_1's Wikipedia article, IN_2 is the set of incoming links for e_2's Wikipedia article, and WP is the set of articles in Wikipedia, then:

$$\mathrm{WLM}(e_1, e_2) = 1 - \frac{\log(\max(|IN_1|, |IN_2|)) - \log(|IN_1 \cap IN_2|)}{\log(|WP|) - \log(\min(|IN_1|, |IN_2|))}$$

Returning to Figure 1, it is clear why relatedness is so important. It is much easier to disambiguate Arizona and Oklahoma as battleships, given that Pearl Harbor is the military strike, and, on the same note, it is pretty straightforward to disambiguate Pearl Harbor as the military strike, given the text talks about World War II.

4.3 Building the candidate entity graph

Previously, we touched on the positive correlation that exists between the proposed NED measures and a disambiguation that seems natural to a human reader. However, we need to find a way to combine those measures in a sensible way, as, more often than not, each of them favors a different disambiguation decision. For example, in the text shown in Figure 1, the GKG resultScore measure indicates that Arizona and Oklahoma are states, while document similarity and entity relatedness indicate that they are, in fact, ships.

We combine those three measures on a graph G, the cornerstone of GEEK, which we call *candidate entity graph*. Given a text T that contains k named entity mentions $M = \{m_1, m_2, \ldots, m_k\}$, we generate candidate entity sets $E_1, E_2, \ldots, E_k$. Then, for each candidate entity set $E_i = \{e_{i1}, \ldots e_{in_i}\}$, where $n_i = |E_i|$, we add to G nodes $e_{i1}, \ldots e_{in_i}$, where each node e_{ij} corresponds to the j-th candidate entity for mention m_i. We complete the construction of G, by connecting each node e_{ij} to each node e_{uv}, where $i \neq u$, with an undirected weighted edge. The edge's weight is calculated as a linear combination of the three NED measures introduced above, applied for both of the candidate entities e_{ij} (j-th candidate entity for mention m_i) and e_{uv} (v-th candidate for mention m_u):

$$weight(e_{ij}, e_{uv}) = a \cdot \left(\frac{b \cdot \mathrm{rs}(e_{ij}) + (1-b) \cdot \mathrm{sim}(T, T_{e_{ij}})}{2} + \frac{b \cdot \mathrm{rs}(e_{uv}) + (1-b) \cdot \mathrm{sim}(T, T_{e_{uv}})}{2} \right) + (1-a) \cdot \mathrm{WLM}(e_{ij}, e_{uv})$$

where $rs \equiv$ normalized GKG resultScore
$sim \equiv$ binary document similarity
$0 \leq a, b \leq 1$

The candidate entity graph G is undirected, weighted, complete, and k-partite. That means G's nodes are distributed among k independent sets. If two nodes belong to the same independent set, then they are not adjacent. If they belong to different independent sets, then they are adjacent and connected by an undirected weighted edge. The idea behind this is simple: there is no point in connecting two entities that are candidates for the same named entity mention m_i, as they are mutually exclusive, that is, if $e_{ij}, e_{il} \in E_i$, then $(m_i \mapsto e_{ij}) \implies \neg(m_i \mapsto e_{il})$ and $(m_i \mapsto e_{il}) \implies \neg(m_i \mapsto e_{ij})$ The parameters a and b serve to add two degrees of freedom to the way we prioritize NED measures. In particular:

- a determines how much we value the matching between a mention's string (calculated as GKG resultScore) as well as the complete text's string (calculated by document similarity) and the candidate entity's attributes, versus how much we value WLM.
- b determines how much we value GKG's resultScore as a degree of entity commonness, versus how much we value term-overlap document similarity.

Given that all NED measures are normalized (their values range from 0 to 1), it follows from the way we calculate the weights of G's edges that those weights are also normalized.

4.4 Solving the candidate entity graph

Building the candidate entity graph G is an important step towards disambiguating the named entities found in text T. That is because we transform an unstructured, hard to process piece of text

[13] https://www.mediawiki.org/wiki/API:Linkshere

into a well-defined, abstract data structure we can work with. Of course, building G is not the end of the story. We need to find a way to map G to a disambiguation for T's entities. This mapping represents GEEK's main contribution when compared to the literature's graph-based NERD solutions: a straightforward subgraph extraction method that incrementally eliminates unsuitable candidate entities. This leads to a sequence of candidate entity graphs $G^{(1)}, G^{(2)}, G^{(3)}, \ldots$, where each $G^{(x+1)}$ better approximates the correct disambiguation result compared to $G^{(x)}$.

Given that G is comprised of k independent sets of nodes, each set E_i containing candidate entities for named entity mention m_i, it is clear that we need to select exactly one node e_{ij} from each independent set E_i. This process is equivalent to mapping each named entity mention to exactly one of its candidate entities. However, what is not clear is the optimal way to perform said mapping. In our framework, we *assume* that the correct disambiguation lies in G's maximum weight k-clique, denoted as G^*. G^* is the induced subgraph of G that contains exactly one node from each independent set E_i, such that the total weight of the connecting edges is maximized. Hence, we face the problem of finding a maximum weight k-clique in an undirected, weighted, complete, and k-partite graph. Such combinatorial optimization problems have been tackled by NERD frameworks in the past, and have been proved to be NP-hard [12–14, 23]. This is pretty clear on an intuitive level, as we have to choose one entity from each of k candidate entity sets E_i, $1 \leq i \leq k$. If each E_i contains n candidate entities on average, then our choices for the disambiguation are exponentially many: n^k. Consequently, we cannot hope to create an exact algorithm to find G^*, given reasonable runtime and resource consumption constraints.

In this context, we need to come up with an approximation for G^*. To this end, we formulate a heuristic method that is tailored to our problem specifications. Our goal isn't to find a general purpose solution for the maximum weight k-clique optimization problem, but only a way to solve the problem, given that it arises from a semantically coherent text and its candidate entities. The method we suggest is based on the fact that a lot of candidate entities seem outlandish for a specific textual context. Thus, finding these out of place entities is the key to our disambiguation algorithm. For example, in the text of Figure 1, the candidate entity Arizona Beverage Company seems to be a bad choice for the disambiguation of mention "Arizona", as suggested by all three NED measures. Indeed, the Arizona Beverage Company isn't usually what one refers to when one says "Arizona" (low commonness), the text doesn't contain any terms that would suggest it's talking about this company (low document similarity), and, finally, the beverage company isn't related to the other entities in the text (low entity relatedness).

In order to identify those alien entities in the candidate entity space, we consider for each candidate entity e_{ij} the disambiguation in which e_{ij} has the maximum weight contribution. For each candidate entity e_{ij}, we define its *maximum contribution graph* $G_{e_{ij}}$ as the G^* candidate in which node e_{ij} has the maximum possible weight of incident edges. Calculating the $G_{e_{ij}}$ graphs for all candidate entities is straightforward, as it only requires visiting e_{ij}'s neighbors in G. Each $G_{e_{ij}}$ graph suggests a disambiguation for all entities in T, and those graphs can be used to identify the alien entities discussed above. To elaborate further, when we construct $G_{e_{ij}}$ for a candidate entity e_{ij}, it's like we are forcing e_{ij} in the given context, in order to see what happens with our objective function, which is the resulting graph's total weight. We hypothesize that there are two cases:

- e_{ij} is the correct disambiguation choice for mention m_i, which will be reflected on $G_{e_{ij}}$'s increased total weight, as the NED measures' values will be high.
- e_{ij} is not the correct disambiguation choice for mention m_i, which will be reflected on $G_{e_{ij}}$'s decreased total weight, as the NED measures' values will be low.

Aiming for an incremental disambiguation process that eliminates said alien entities, we developed a worst out heuristic method, which removes the entity that seems most unfitting in each step, until we achieve disambiguation of all entities. This is more effective than a best in heuristic, as it is easier to identify the many entities that don't fit, rather than those few entities that are correct. An outline of this method is presented by Algorithm 1. An example of building the maximum contribution graph for a candidate entity is presented in Figure 2. It's important to note that after each removal step, we need to recalculate the maximum contribution graphs for our candidate entities, as demonstrated in Figure 3. Solving the candidate entity graph is the final step in the disambiguation process, and the entire pipeline is illustrated by a flow diagram in Figure 4.

```
1:  function WorstOutHeuristic(G)
2:      if |E_i| ≤ 1 ∀i = 1, 2, ..., k then
3:          return G                          ▷ disambiguation complete
4:      end if
5:      for each e_ij ∈ G do
6:          calculate G_{e_ij}
7:      end for
8:      tied_nodes = {e_ij ∈ G : |E_i| > 1 ∧ ∄e_uv with |E_u| > 1
                      such that weight(G_{e_ij}) > weight(G_{e_uv})}
9:      find e ∈ tied_nodes with minimum incident edges weight
10:     return WorstOutHeuristic(G \ e)
11: end function
```

Algorithm 1: Outline of the worst out heuristic method used to disambiguate named entities in text. Every function call isolates the "most outlandish" entity, which is then removed from graph G, until disambiguation is achieved. Note that the total weight of the maximum contribution graphs is the first thing we take into account. However, we use the nodes' incident edges weights to resolve ties.

5 EXPERIMENTAL EVALUATION

5.1 Datasets

We evaluated GEEK using three kinds of datasets: a small texts dataset we manually generated, a medium-sized texts dataset comprised of Reuters news articles, and a selection of two other datasets provided by GERBIL [28] (again a dataset containing smaller texts

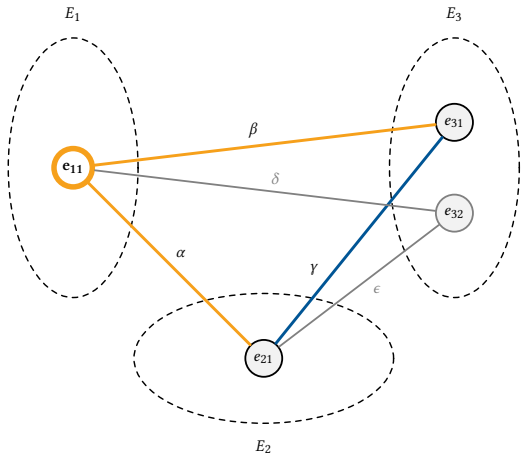

Figure 2: Building the maximum contribution graph for node e_{11} of independent set E_1 in the case of a three-mention text T. Node e_{11} greedily chooses nodes e_{21} and e_{31} for $G_{e_{11}}$. This means $\beta \geq \delta$. Edges e_{11}–e_{21} and e_{11}–e_{31} are chosen because they are the maximum weight connectors between e_{11} and the respective node sets E_2 and E_3. Edge e_{21}–e_{31} completes $G_{e_{11}}$'s edges. We have $weight(G_{e_{11}}) = \alpha + \beta + \gamma$ and $contribution(G_{e_{11}}, e_{11}) = \alpha + \beta$. These two criteria, in that order, applied for all nodes in G, are used to determine the node to be eliminated. In this illustration, the number of nodes in sets E_1, E_2, and E_3 is limited only for the sake of simplicity.

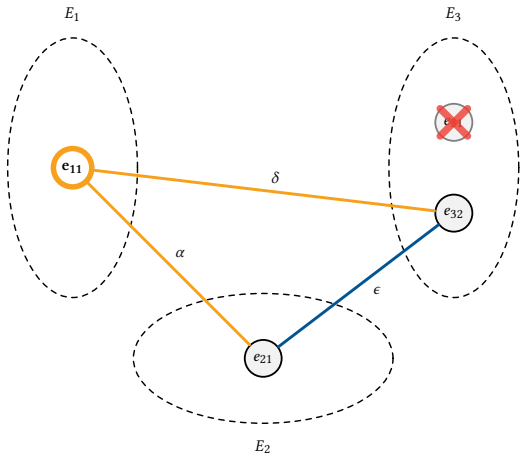

Figure 3: Continuing on Figure 2, we assume that node e_{31} has been eliminated. Among other nodes, e_{11} needs to update its maximum contribution graph $G_{e_{11}}$. Now e_{11} is connected to e_{32}, as it offers the next highest weight connection to set E_3. Edges e_{11}–e_{21} and e_{11}–e_{32} are chosen because they are the maximum weight connectors between e_{11} and the respective node sets E_2 and E_3. Edge e_{21}–e_{32} completes $G_{e_{11}}$'s edges. Now we have $weight(G_{e_{11}}) = \alpha + \delta + \epsilon$ and $contribution(G_{e_{11}}, e_{11}) = \alpha + \delta$. We conclude that both elimination criteria have been altered, a fact that demonstrates why we need to update the maximum contribution graphs after every node removal.

and another comprised of larger documents), on its web-based platform that facilitates the comparison of several NERD systems[14]. We note that we didn't experiment on annotators that have been already outperformed by the NERD tools provided by GERBIL, such as Milne and Witten's Wikipedia Miner system [16].

Small texts dataset. In the interest of our system's evaluation, we collected and compiled a dataset of small texts, which are dense in highly ambiguous named entities. The entities in this dataset were manually extracted and annotated. Each named entity was mapped to its best match in GKG and Wikipedia. This dataset was inspired from the KORE 50 NIF NER Corpus[15]. Its texts were gathered from online sources, such as encyclopedia articles, news feeds, blog posts, social media, and so on. It aims to test our system's disambiguation capabilities given a limited context and a well defined semantic core. From now on, it will be denoted as SAT-300 (300 Short Ambiguous Texts). The SAT-300 dataset is available as part of this paper[16].

Medium-sized texts dataset. The second part of our experimental evaluation was performed using a standard benchmark dataset in the field of NERD. This is a dataset of 231 Reuters news-wire articles used by Hoffart [11] to compare his own NERD system with other disambiguation systems found in the literature. Much like SAT-300's texts, these were processed by hand, and their named entities were mapped to a number of knowledge bases, including Freebase (GKG's predecessor) and Wikipedia. This dataset aims to test our system's disambiguation capacity when it comes to longer texts, that may include a larger number of possibly unrelated topics. From now on, it will be denoted as Reuters-231.

Other datasets. To the end of further experimental evaluation of GEEK, we turn to GERBIL and use the KORE-50 dataset in order to assess its performance when it comes to short texts, as well as the complete CoNLL dataset to test our system on larger texts.

5.2 Evaluation Measures

In order to assess the disambiguation that a system produces for a single text T, the precision, recall, F_1 score, and accuracy measures may be used. In the context of NERD, precision is the fraction of correctly disambiguated named entity mentions that are generated by the system:

$$\text{precision} = \frac{|\text{correctly disambiguated entities}|}{|\text{disambiguated entities produced by the system}|}$$

Recall is the fraction of correctly disambiguated named entity mentions that should be disambiguated:

$$\text{recall} = \frac{|\text{correctly disambiguated entities}|}{|\text{entities that should be disambiguated}|}$$

The F_1 score is the harmonic mean of precision and recall:

$$F_1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

[14] http://gerbil.aksw.org/gerbil/
[15] https://datahub.io/dataset/kore-50-nif-ner-corpus
[16] https://github.com/WWW-2018-submission-SAT-300-dataset/SAT-300-dataset

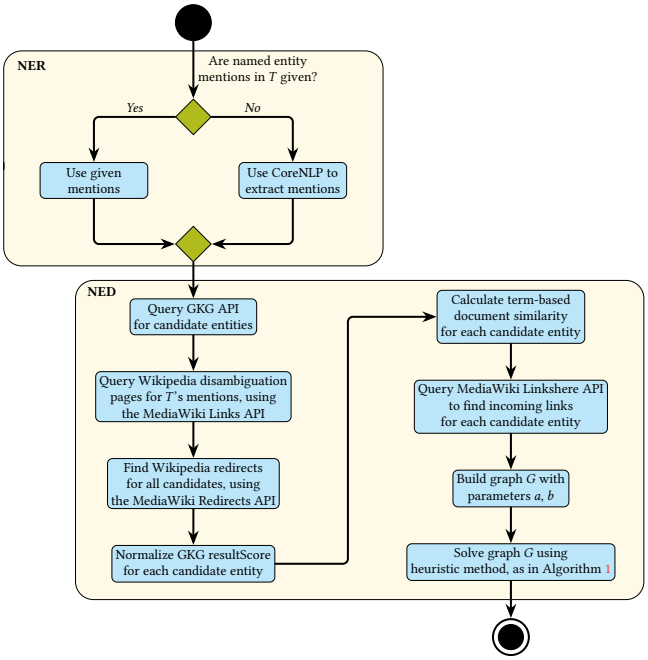

Figure 4: GEEK's named entity recognition and disambiguation pipeline. First we have the NER step, followed by the NED step. In the disambiguation step, which is the most critical, we start by defining a crude estimate of the candidate entity set by using the GKG API, which is refined with the use of Wikipedia's APIs. Next we collect the information we need to build the candidate entity graph, which is solved in a heuristic manner.

The evaluation measures of precision and recall are used when the NERD system is responsible for the step of NER. Oftentimes, that is avoided, when the named entities to be disambiguated have already been marked and are provided to the disambiguation system straightaway. In this case, only the NED part of the system is evaluated, and accuracy is used as the only evaluation measure:

$$\text{accuracy} = \frac{|\text{correctly disambiguated entities}|}{|\text{entities marked in text}|}$$

In the case of a collection of texts, one could use precision and recall to evaluate a system's NER + NED performance, and accuracy to evaluate its NED performance, for each and every one of the texts separately. However, this would turn the experimental results into a long, difficult to interpret list of numbers. That is why we use the Micro Average and Macro Average aggregation approach:

- *Micro Average*: We assume that the dataset's texts are combined to form a large text, on which we are able to use the above-provided definitions of precision, recall, and accuracy.
- *Macro Average*: We evaluate the system's performance on the dataset by averaging precision, recall, and accuracy calculated on each of the collection's texts.

For the sake of space preservation, we use the abbreviations Micro Average F_1 score $\equiv \mu AF_1$, Macro Average F_1 score $\equiv MAF_1$, Micro Average Accuracy $\equiv \mu AA$, and Macro Average Accuracy $\equiv MAA$.

Similarly to the case of a single text, we use the aggregated version of accuracy in cases where the named entities to be disambiguated are given to the disambiguation system as part of its input.

5.3 System Parameterization

Our system includes several parameters that affect the behaviour and performance of the disambiguation process. The most crucial elements we need to decide on are:

- The maximum number of entities requested by GKG API. Reducing this value means we have fewer candidates for each named entity mention, which makes the disambiguation process easier. On the other hand, increasing this value means we have richer, more inclusive candidate entity sets, at the expense of disambiguation efficiency.
- b, which indicates how much we value GKG resultScore versus document similarity as NED measures when we build our candidate entity graph G.
- a, which indicates how much we value GKG resultScore and document similarity versus WLM when we build our candidate entity graph G.

We tuned these parameters using about a hundred short ambiguous texts similar to those contained in SAT-300, and we settled on the following values for our experiments:

- We request a hundred candidate entities for each mention, which covers most cases without excluding the correct entities from the candidate entity sets or making the candidate entity graph unnecessarily complex.
- $b = 0.85$, which means we value GKG resultScore much more than document similarity. The fact that term-based document similarity is a rather naive NED measure was obvious from the early stages of our analysis. Our testing concurs with this, as the system seems to perform better when the other NED measures undertake the heaving lifting.
- $a \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$, meaning we make a transition from a system that only cares about WLM ($a = 0.0$) to a system that only cares about the combination of GKG resultScore and document similarity ($a = 1.0$). We didn't choose only one value for a, as the best value of a seems to vary depending on the text in hand. Also, observing the system's behaviour with different values of a can give us insight into its optimal configuration.

5.4 Experimental Results

SAT-300 dataset. The performance of our system on the SAT-300 dataset in terms of the μAF_1 and MAF_1 measures can be seen in the SAT-300 relevant parts of Table 1 and Figure 5. We used the F_1 score, because in this experiment we fed the texts to the system as plain strings, and it was the system's responsibility to recognize and then disambiguate the named entities.

For comparison, the results obtained from GERBIL for a sample of its available annotators on the same dataset using the A2KB task[17] are displayed in Table 2. We note that our system outperforms other systems on small ambiguous texts. Based on this experimental analysis, we can draw several conclusions about our system:

[17] Full SAT-300 results: http://gerbil.aksw.org/gerbil/experiment?id=201801240021

Table 1: GEEK scores on SAT-300 and Reuters-231 datasets for different values of a (all values in %).

a	SAT-300		Reuters-231	
	μAF_1	MAF_1	μAA	MAA
0.0	72.48	71.91	57.53	51.83
0.1	77.65	76.60	61.42	57.78
0.2	83.25	82.55	69.83	67.35
0.3	88.01	87.48	79.37	76.67
0.4	92.13	92.21	85.29	84.03
0.5	**92.87**	**93.58**	**87.66**	**87.84**
0.6	88.64	89.54	84.21	85.70
0.7	82.41	82.99	82.26	84.31
0.8	74.48	74.90	81.35	83.51
0.9	68.57	68.29	78.65	81.36
1.0	65.08	64.60	76.86	80.27

Table 2: Scores of GERBIL's annotators when faced with the A2KB task of finding named entities in SAT-300's texts and then linking them to a knowledge base. Best GEEK configuration is appended for comparison (all values in %).

Annotator	μAF_1	MAF_1
AIDA	68.28	63.66
Babelfy	57.29	56.08
DBpedia Spotlight	58.62	54.43
Dexter	46.81	41.35
Entityclassifier.eu NER	35.26	33.39
FOX	39.18	34.97
Kea	2.64	1.39
WAT	60.55	51.92
GEEK	***92.87***	***93.58***

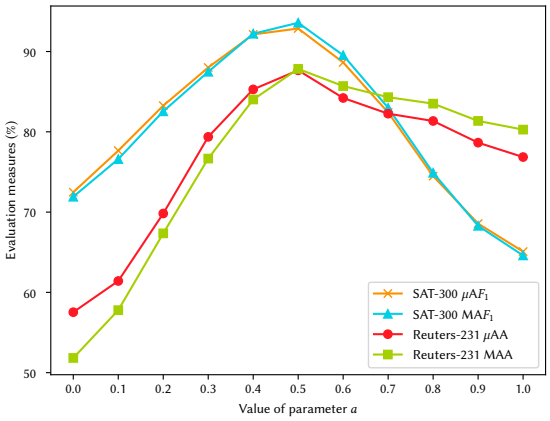

Figure 5: Illustration of our system's performance on the SAT-300 and Reuters-231 datasets, as the parameter a increases from 0 to 1.

- The system's best configuration is $a = 0.5$. This means that balancing between GKG resultScore and document similarity versus entity relatedness yields the best results.
- We notice from Table 1 and Figure 5 that $a = 0.0$ gives us better results than $a = 1.0$. This means that the most important measure is entity relatedness. This reflects the nature of our dataset. Indeed, we anticipate small, semantically coherent texts to contain highly related entities. We also observe that GEEK performs better on SAT-300 compared with Reuters-231. That difference is due to the entities found in the two datasets: SAT-300 only contains entities stored both in GKG and Wikipedia, so there are rarely out-of-knowledge-base entities; this is not true for Reuters-231, as well as the larger CoNLL dataset below, where exist many entities that don't exist in GKG. Those inevitably lead to false positives in the context of an annotator that always tries to map all mentions to entities.
- Both Micro Average and Macro Average aggregation measures give us similar results.

Reuters-231 dataset. The results of the experimental evaluation of our system using the Reuters-231 dataset can be seen in the Reuters-231 relevant parts of Table 1 and Figure 5. For this dataset, we only calculate accuracy because this is the measure used by Hoffart [11] in the original article. This means that in this case the system performed only the NED task: we fed the system with the texts to be disambiguated, along with the named entity mentions they contain. Our conclusions are:

- The system's best configuration still is a balanced value of $a = 0.5$. In that case, accuracy exceeds 87%. This represents a 5% improvement compared against Hoffart's AIDA system.
- We notice from Table 1 and Figure 5 that $a = 1.0$ gives us better results than $a = 0.0$. This means that the most important measures are GKG resultScore and document similarity. That's no surprise, as the Reuters-231 dataset is comprised of news articles, which contain a fair amount of more or less unambiguous mentions, which can be resolved without resorting to entity relatedness.
- In contrast to SAT-300's experimental results, we see that the Micro Average and Macro Average approaches behave somewhat differently in the case of Reuters-231. More specifically, we have $\mu AA > MAA$ for $a < 0.5$ and $\mu AA < MAA$ for $a > 0.5$. This variance offers us insight into the way our NED measures function:
  - For small values of a, we prioritize coherence in our texts, expressed by the relatedness between entities. This decision works in our favor in the case of texts that contain a large number of entities, as those are expected to be semantically correlated, thus easy to disambiguate using WLM. On the other hand, texts that contain few, possibly unrelated entities will be harder to process. This means that the majority of errors is expected to happen in entity-sparse texts, lowering their individual accuracy measure. For example, if we have a text T that contains two named entities and we get one of them wrong, then its accuracy immediately drops to 50%. Returning to the definition

of Micro Average and Macro Average aggregation measures, we conclude that our MAA will be affected, as we average significantly smaller numbers. Of course, μAA is not affected in any way, as this measure doesn't care about the dataset's texts individually. That's why we get $\mu AA > MAA$ for $a < 0.5$.

– For larger values of a, we prioritize commonness and document similarity in our texts, expressed by the measures of GKG resultScore and term overlap. This decision works in our favor in the case of texts that contain few unrelated entities, so coherence wouldn't help. On the other hand, texts that contain many semantically similar entities are harder to disambiguate. This means that the majority of errors is expected to happen in texts containing a high number of entities, which doesn't affect their individual accuracy measure as much. For example, if we have a text T that contains a hundred named entities and we get one of them wrong, then its accuracy barely drops to 99%. In contrast to what we noticed above, we conclude that our MAA will not be drastically affected. Again, μAA is not affected in any way. This analysis explains why we get $\mu AA < MAA$ for $a > 0.5$.

KORE-50 and CoNLL datasets. Setting $a = 0.5$, which from the experimental evaluation turned out to be the best overall performing value, we further tested GEEK by comparing its performance on GERBIL's A2KB task against the service's annotators on the KORE-50 dataset[18], a dataset similar to SAT-300, but with a higher level of ambiguity, as well as the complete CoNLL collection of documents[19]. The results, in terms of the μAF_1 and MAF_1 scores, can be found in Table 3. We conclude that GEEK outperforms other annotators on the small documents of KORE-50, but it does not achieve top performance when it comes to the larger texts of CoNLL. This is due to the nature of the disambiguation algorithm applied by GEEK, where in-document coherence is crucial, and this attribute is less prominent in larger documents.

To sum up, GEEK seems to outperform the state-of-the-art on short documents, while being competitive on longer documents.

Table 3: Comparison of GEEK with GERBIL's annotators when faced with the A2KB task of finding named entities in KORE-50 and CoNLL texts and then linking them to a knowledge base (all values in %).

Annotator	KORE-50		CoNLL	
	μAF_1	MAF_1	μAF_1	MAF_1
AIDA	58.40	52.58	**67.35**	**64.23**
Babelfy	56.45	52.63	44.81	39.66
DBpedia Spotlight	35.24	28.50	53.92	51.27
Dexter	23.28	17.00	47.47	43.40
Entityclassifier.eu NER	29.97	26.97	44.92	42.03
FOX	28.02	25.31	57.23	57.26
Kea	50.31	46.27	39.81	36.10
WAT	51.95	39.63	67.22	64.21
GEEK	***62.90***	***61.50***	*53.69*	*51.16*

6 RELATED WORK

6.1 NER

As mentioned above, the subtask of NER is critical for a NERD pipeline. When it comes to NER, two main approaches have been followed in the literature:

- Building a dedicated tool. This could mean anything from using a statistical approach [3] to training a classifier on Wikipedia's links [16]. This approach is the most flexible, as the developers of the NERD system can tailor the NER component to their needs, or even blur the boundaries between NER and NED [6, 20, 26].
- Using an off-the-shelf tool. This limits the flexibility of the NER component, but offers the upside of having an established and well-tuned framework for the preprocessing step of entity recognition, which decouples the tasks of NER and NED. One such tool is Stanford NER[20], used by Hoffart [11] in his NERD system. An evolution of this tool, in the form of Stanford CoreNLP, is also used in our framework.

6.2 NED

Even though the NER options are pretty straightforward, the NED methods vary wildly across the literature. A non-exhaustive list of NED systems that stand out for the novel ideas they introduced follows: Bunescu and Pasca [1] were the first to appreciate the value of Wikipedia's structured information, like articles, redirects, and disambiguation pages, to the end of entity disambiguation. They introduced *Wikification*, the idea of mapping textual mentions to Wikipedia articles. After that point, all successful NERD systems used Wikipedia as a resource in one way or the other. Cucerzan [3] created the first system that executes NERD in an end-to-end process, getting unstructured text as input, recognizing entities, and mapping them to Wikipedia. Also, his work introduces the potential value of joint or collective disambiguation. Milne and Witten [16] introduced a novel way to recognize mentions by training a classifier on Wikipedia data and stressed the importance of prior probability or commonness in the NED process. However, their greatest contribution was the repackaging of Normalized Google Distance into the Wikipedia Link-based Measure, an efficient and effective way of calculating entity relatedness, which was used by the majority of NERD frameworks that followed. Kulkarni et al. [13] implemented the first full-fledged collective disambiguation system. They also made significant contributions to the algorithmic side of collective disambiguation, recognizing its high complexity and resorting to heuristic methods. Graph-based collective inference methods are among the most successful in the field of entity linking. As is the case with GEEK, these methods always reduce the text to an appropriate graph, which can guide the disambiguation process. Han et al. [10] and Hoffart [11] proposed graph based structures that combine the disambiguation features of the respective systems and model the entity disambiguation task in an intuitive manner. Moro et al. [18] decided to build their graph using random walks,

[18] Full KORE-50 results: http://gerbil.aksw.org/gerbil/experiment?id=201801280015
[19] Full CoNLL results: http://gerbil.aksw.org/gerbil/experiment?id=201801280016
[20] https://nlp.stanford.edu/software/CRF-NER.shtml

while Piccinno et al. [19] suggested a plethora of different algorithms for the solution of their graph. Ganea et al. [5] introduced a probabilistic graphical model for collective disambiguation.

7 CONCLUSIONS

In this paper we introduced GEEK, a framework that tackles the difficult task of NERD by combining well-established measures, such as commonness of entities and coherence, with new potent tools, such as the Google Knowledge Graph Search API. We proposed a graph representation of texts, which allowed us to model disambiguation as a concise graph-based problem and solve it using a heuristic method that finds the best clique in a stepwise manner.

The main strength of GEEK is that it is built on top of publicly available data sources and has a straightforward graph algorithm at its core. Moreover, the experimental evaluation on GERBIL showed that GEEK produces competitive scores in comparison with other state-of-the-art systems, which suggests that one does not always need specialized knowledge bases or sophisticated disambiguation algorithms to achieve a high quality disambiguation result.

ACKNOWLEDGEMENTS

We acknowledge support of this work by the project "APOLLONIS" (MIS 5002738) which is implemented under the Action "Reinforcement of the Research and Innovation Infrastructure", funded by the Operational Programme "Competitiveness, Entrepreneurship and Innovation" (NSRF 2014-2020) and co-financed by Greece and the European Union (European Regional Development Fund).

REFERENCES

[1] Razvan C Bunescu and Marius Pasca. 2006. Using Encyclopedic Knowledge for Named entity Disambiguation.. In *Eacl*, Vol. 6. 9–16.

[2] R. L. Cilibrasi and P. M. B. Vitanyi. 2007. The Google Similarity Distance. *IEEE Transactions on Knowledge and Data Engineering* 19, 3 (March 2007), 370–383.

[3] Silviu Cucerzan. 2007. Large-Scale Named Entity Disambiguation Based on Wikipedia Data. In *Proceedings of EMNLP-CoNLL 2007*. 708âĂŞ716.

[4] Paolo Ferragina and Ugo Scaiella. 2010. TAGME: On-the-fly Annotation of Short Text Fragments (by Wikipedia Entities). In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM '10)*. ACM, New York, NY, USA, 1625–1628.

[5] Octavian-Eugen Ganea, Marina Ganea, Aurelien Lucchi, Carsten Eickhoff, and Thomas Hofmann. 2016. Probabilistic Bag-Of-Hyperlinks Model for Entity Linking. In *Proceedings of the 25th International Conference on World Wide Web (WWW '16)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 927–938.

[6] Stephen Guo, Ming-Wei Chang, and Emre Kiciman. 2013. To Link or Not to Link? A Study on End-to-End Tweet Entity Linking.. In *HLT-NAACL*. 1020–1030.

[7] Ben Hachey, Will Radford, Joel Nothman, Matthew Honnibal, and James R. Curran. 2013. Evaluating Entity Linking with Wikipedia. *Artificial Intelligence* 194 (2013), 130 – 150.

[8] Xianpei Han and Le Sun. 2011. A Generative Entity-mention Model for Linking Entities with Knowledge Base. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1 (HLT '11)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 945–954.

[9] Xianpei Han and Le Sun. 2012. An Entity-topic Model for Entity Linking. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL '12)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 105–115.

[10] Xianpei Han, Le Sun, and Jun Zhao. 2011. Collective Entity Linking in Web Text: A Graph-based Method. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '11)*. ACM, New York, NY, USA, 765–774.

[11] Johannes Hoffart. 2015. *Discovering and disambiguating named entities in text*. Ph.D. Dissertation. UniversitÃđt des Saarlandes, Postfach 151141, 66041 SaarbrÃijcken.

[12] Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. 2011. Robust Disambiguation of Named Entities in Text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '11)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 782–792.

[13] Sayali Kulkarni, Amit Singh, Ganesh Ramakrishnan, and Soumen Chakrabarti. 2009. Collective Annotation of Wikipedia Entities in Web Text. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '09)*. ACM, New York, NY, USA, 457–466.

[14] Xiaohua Liu, Yitong Li, Haocheng Wu, Ming Zhou, Furu Wei, and Yi Lu. 2013. Entity Linking for Tweets.. In *ACL (1)*. 1304–1311.

[15] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. 55–60.

[16] David Milne and Ian H. Witten. 2008. Learning to Link with Wikipedia. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM '08)*. ACM, New York, NY, USA, 509–518.

[17] Sean Monahan, John Lehmann, Timothy Nyberg, Jesse Plymale, and Arnold Jung. 2011. Cross-Lingual Cross-Document Coreference with Entity Linking.. In *TAC*.

[18] Andrea Moro, Alessandro Raganato, and Roberto Navigli. 2014. Entity Linking meets Word Sense Disambiguation: A Unified Approach. *Transactions of the Association for Computational Linguistics* 2 (2014).

[19] Francesco Piccinno and Paolo Ferragina. 2014. From TagME to WAT: A New Entity Annotator. In *Proceedings of the First International Workshop on Entity Recognition & Disambiguation (ERD '14)*. 55–62.

[20] Ken Q. Pu, Oktie Hassanzadeh, Richard Drake, and Renée J. Miller. 2010. Online Annotation of Text Streams with Structured Entities. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM '10)*. ACM, New York, NY, USA, 29–38.

[21] Lev Ratinov, Dan Roth, Doug Downey, and Mike Anderson. 2011. Local and Global Algorithms for Disambiguation to Wikipedia. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1 (HLT '11)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 1375–1384.

[22] W. Shen, J. Wang, and J. Han. 2015. Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *IEEE Transactions on Knowledge and Data Engineering* 27, 2 (Feb 2015), 443–460.

[23] Wei Shen, Jianyong Wang, Ping Luo, and Min Wang. 2012. LIEGE:: Link Entities in Web Lists with Knowledge Base. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '12)*. ACM, New York, NY, USA, 1424–1432.

[24] Wei Shen, Jianyong Wang, Ping Luo, and Min Wang. 2012. LINDEN: Linking Named Entities with Knowledge Base via Semantic Knowledge. In *Proceedings of the 21st International Conference on World Wide Web (WWW '12)*. ACM, New York, NY, USA, 449–458.

[25] Wei Shen, Jianyong Wang, Ping Luo, and Min Wang. 2013. Linking Named Entities in Tweets with Knowledge Base via User Interest Modeling. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. ACM, New York, NY, USA, 68–76.

[26] Avirup Sil and Alexander Yates. 2013. Re-ranking for joint named-entity recognition and linking. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management (CIKM '13)*. ACM, New York, NY, USA, 2369–2374.

[27] Tadej Štajner and Dunja Mladenić. 2009. *Entity Resolution in Texts Using Statistical Learning and Ontologies*. Springer Berlin Heidelberg, Berlin, Heidelberg, 91–104.

[28] Ricardo Usbeck, Michael Röder, Axel-Cyrille Ngonga Ngomo, Ciro Baron, Andreas Both, Martin Brümmer, Diego Ceccarelli, Marco Cornolti, Didier Cherix, Bernd Eickmann, Paolo Ferragina, Christiane Lemke, Andrea Moro, Roberto Navigli, Francesco Piccinno, Giuseppe Rizzo, Harald Sack, René Speck, Raphaël Troncy, Jörg Waitelonis, and Lars Wesemann. 2015. GERBIL: General Entity Annotator Benchmarking Framework. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 1133–1143.