Fast and Accurate Entity Linking via Graph Embedding

Alberto Parravicini
Politecnico di Milano
alberto.parravicini@polimi.it

Rhicheek Patra
Oracle Labs
rhicheek.patra@oracle.com

Davide B. Bartolini
Oracle Labs
davide.bartolini@oracle.com

Marco D. Santambrogio
Politecnico di Milano
marco.santambrogio@polimi.it

ABSTRACT

Entity Linking, the task of mapping ambiguous Named Entities to unique identifiers in a knowledge base, is a cornerstone of multiple Information Retrieval and Text Analysis systems. So far, no single entity linking algorithm has been able to offer the accuracy and scalability required to deal with the ever-increasing amount of data in the web and become a de-facto standard.

In this paper, we propose a framework for entity linking that leverages graph embeddings to perform collective disambiguation. This framework is modular as it supports pluggable algorithms for embedding generation and candidate ranking. With our framework, we implement and evaluate a reference pipeline that uses DBpedia as knowledge base and leverages specific algorithms for fast candidate search and high-performance state-space search optimization. Compared to existing solutions, our approach offers state-of-the-art accuracy on a variety of datasets without any supervised training and provides real-time execution even when processing documents with dozens of Named Entities. Lastly, the flexibility of our framework allows adapting to a multitude of scenarios by balancing accuracy and execution time.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Information systems** → **Information extraction**; • **Computing methodologies** → **Information extraction**; *Unsupervised learning*.

KEYWORDS

entity linking, graph embeddings, representation learning, text disambiguation

ACM Reference Format:
Alberto Parravicini, Rhicheek Patra, Davide B. Bartolini, and Marco D. Santambrogio. 2019. Fast and Accurate Entity Linking via Graph Embedding. In *2nd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (GRADES-NDA'19), June 30-July 5 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3327964.3328499

GRADES-NDA'19, June 30-July 5 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6789-9/19/06…$15.00
https://doi.org/10.1145/3327964.3328499

Figure 1: An example of Entity Linking where each mention in the original sentence is connected to a URI.

1 INTRODUCTION

Entity Linking (EL), sometimes called Named Entity Disambiguation (NED), is the task of mapping words of interest (usually named entities, e.g., names of persons, locations, companies, etc...) from an input text to corresponding unique entities - called Uniform Resource Identifiers (URIs) - in a target knowledge base. Being able to uniquely identify entities contained in a text is critical in many fields of application, such as text analysis, recommender systems, semantic search and chatbots. All of these fields benefit from high-level representations of the text, in which concepts relevant to the application are separated from noisy text.

1.1 Motivations

Entity linking is a deceptively simple task but over the years none of the multiple approaches, proposed in the literature, managed to become a de-facto standard [9, 11, 27, 29]. Indeed, there are several subtle challenges in the entity linking task (see fig. 1 for an example). For example, the words to be linked, known as *mentions* or *surface forms*, are often ambiguous if considered by themselves. For example, *"Zeppelin"* might refer to an airship, to the famous band *Led Zeppelin*, or to the *Apache Zeppelin* software. In order to correctly link such ambiguous mentions, the key is to consider the context. By leveraging the relationships in a Knowledge Base (KB), one can derive that *"Led Zeppelin"* relates to *hard rock* and the 70s, while the *Zeppelin* airship or the *Apache Zeppelin* would have significantly different relationships and thus appear in different contexts.

Another challenge for a practical entity linking system is to provide the results with a low latency, often in real-time. This requirement is very challenging when using large knowledge-bases, even when processing documents of moderate size. For example, using Wikipedia as the KB means analyzing nearly 9 million entities and around 165 million relationships. Next, a news article might contain dozens of mentions that need to be collectively disambiguated, leading to an enormous search space for the correct URIs.

Existing EL algorithms are optimized for single tasks or domains, such as web pages [9] or social media posts [3], and are unable to offer accuracy, scalability and low latency under highly variable conditions encountered in real-life applications.

1.2 Contributions

In this paper, we propose a novel EL algorithmic framework that combines neural graph embeddings with state-space search optimization. Our framework is composed of different building blocks that can be configured and tuned. We leverage this framework to design and evaluate an EL pipeline that uses DBpedia as the KB and can handle large documents.

Our approach, to the best of our knowledge, is the first to employ a graph representation based on neural embeddings to perform entity linking. Representing graphs with embeddings allows encoding in a low-dimensional space the complex topological relationships that exist among entities in the graph. Moreover, it is possible to encode in a unified way heterogeneous properties, such as textual labels, statistical measures, and other handcrafted features. We propose a novel method to exploit vertex embeddings to measure the degree of coherence, and thus the quality of an EL candidate.

Since an exhaustive evaluation of all possible combinations of the candidate URIs is computationally infeasible, we devise a novel state-space search optimization method to efficiently obtain high-quality candidate solutions, which converges with a high probability to the optimal possible solution or very close to it. Compared to traditional algorithms such as Personalized PageRank (PPR), this approach allows scaling our entity linking algorithm to large documents (with dozens of mentions) while exploiting the larger context they offer and providing high-quality linked entities to the end user.

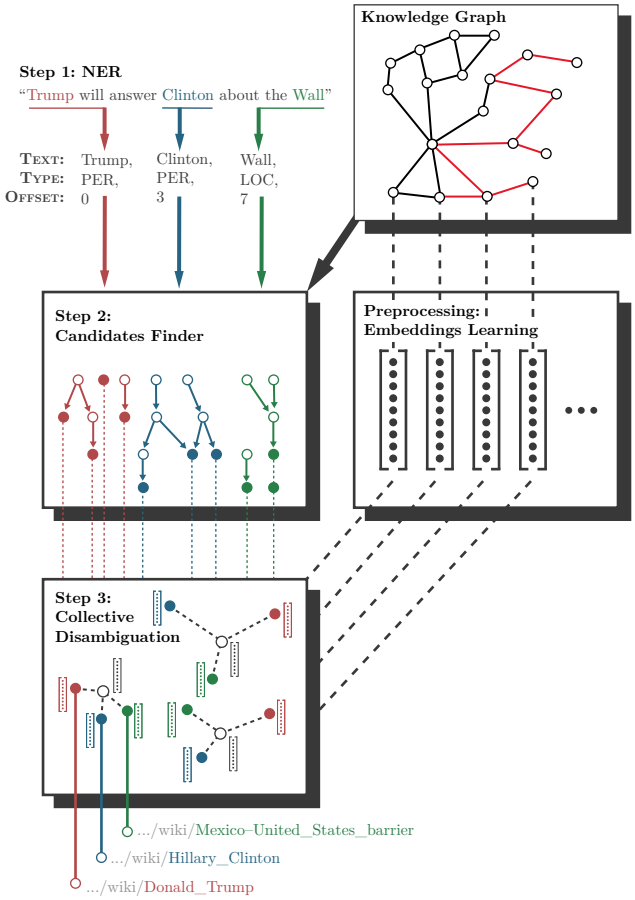

Figure 2: The building blocks of our EL pipeline
In the Knowledge Graph (KG), *disambiguation* and *redirection* links are highlighted in red. They are treated as directional links by the *candidate finder*.

2 RELATED WORK

Entity Linking (EL) has been a hot topic in industry and academia for the last decade. However, as of today a large number of challenges are still open [13], and many Entity Linking systems, with widely different strengths and weaknesses, have been proposed.

Modern Entity Linking systems can be divided into two categories: *text-based* approaches that make use of textual features extracted from large text corpora (e.g. word co-occurrence probabilities), and *graph-based* approaches that exploit the structure of Knowledge Graphs (KGs) to represent the relation of entities.

Among the most relevant text-based Entity Linking algorithms, Rao et al. [23] propose a two-step algorithm to link Named Entitys (NEs) to entities in a target Knowledge Base (KB). First, a set of candidate entities is chosen using string matching, acronyms, and known aliases. The best link among the candidates is chosen with a ranking Support Vector Machine (SVM) that uses linguistic features. The ideas presented in this paper, such as the initial candidate finder and the ranking SVM have seen usage in many other algorithms.

Modern Entity Linking systems do not limit their analysis to textual features generated from input documents or text corpora, but employ large KGs created from KBs such as Wikipedia. These systems extract complex features which take advantage of the KG topology, or leverage multi-step connections between entities, which would be hidden by simple text analysis. Moreover, it is inherently difficult to create Entity Linking systems based on Natural Language Processing (NLP) as it requires large text corpora or hand-crafted grammar rules that can hardly capture the complexity of language.

Han et al. are credited for the first Entity Linking system to employ a KG representation [9]. They propose the creation of a disambiguation graph (a subgraph of the KB which contains candidate entities). This graph is employed for a purely collective ranking procedure that finds the best candidate link for each mention.

Another famous EL approach is AIDA [11], which uses a greedy algorithm that identifies coherent mentions on a dense subgraph by considering context similarities and vertex importance features.

Alhelbawy et al. [1] are the first to employ PageRank (PR) to perform collective Entity Linking on a disambiguation graph.

Personalized PageRank (PPR) has also been employed in Entity Linking by Pershina et al. [21]. This approach computes PPR scores starting from each candidate vertex to find for each mention the candidate that is most coherent to the other mentions' candidates. This algorithm provides good results, but running PPR from each candidate vertex is excessively time-consuming. We experimented

with a similar strategy and decided to move to the approximated computation presented in this paper to provide faster computation.

Representing Wikipedia entities with embeddings has become rather common in recent EL algorithms: the approach by Zwicklbauer et al. performs a multi-step disambiguation procedure that leverages PR on a disambiguation graph enriched with textual embeddings created from Wikipedia pages. This fairly complex algorithm makes use of hand-made heuristics, but it improves results on a number of a datasets thanks to the use of textual embeddings.

Among industrial solutions, IBM [2] proposed in 2016 a framework for entity linking that employs a declarative language and logical constraints. Google is also active in the field of entity linking, as in 2013 they released a large dataset extracted from Wikipedia which can be used for testing or model training[1]. It is known from [12] that they employ approximate Bayesian inference through Gibbs Sampling, and they also incorporate statistical information extracted from the Wikipedia graph.

3 PROBLEM STATEMENT

Entity Linking (EL) is the task of finding the correct relationship between a textual surface form, called *mention*, and the entities of a Knowledge Base (KB). Usually, an input document contains more than a mention (for example, multiple names of cities, persons, companies,...). In this case, it is common to link all the mentions at once. Indeed, mentions appearing in the same sentence or paragraph are often related, and a good EL algorithm can disambiguate these mentions by mapping this relatedness to the rich semantic connections of the target KB.

More formally, given a list of textual mentions $(M_0, M_1, \ldots, M_N)$, each mention M_i has a list of candidates $(C_{i1}, C_{i2}, \ldots, C_{iK})$, which represent possible target entities in the KB. K might be different for different mentions. A candidate solution is a tuple $(C_{1i}, C_{2j}, \ldots, C_{Nq})$, with $C_{1i}, C_{2j}, \ldots$ being candidate vertices taken from the candidate set of each mention. Candidates are obtained by computing the string similarity of mentions with entities of the KB, which have been indexed for fast retrieval. As mentions can be ambiguous, we leverage Wikipedia *disambiguation* and *redirection* links to enrich the candidate set of each mention. A good candidate tuple will have candidates that are reasonably similar to the surface forms, but it will also contain related concepts. This intuition stems from the fact that sentences usually contain related and coherent entities, often belonging to a specific topic.

A candidate is evaluated with a function that considers [29]:

- A *"local"* score η that considers the similarity of the mentions with their candidates, and also the overall importance of the vertex in the graph (intuitively, if two candidates have the same similarity the most important one should be chosen).
- A *"global"* score γ that measures how related the candidates in a tuple are to each other. Measuring the relatedness of the candidates in the tuple is called *Collective Disambiguation*.

The goal is to find a candidate tuple $T = \{t_1, t_2, \ldots, t_N\}$ that maximizes both scores, among all the possible tuples. Local and global scores could be given different weights. The best tuple T^* is obtained as

$$T^* = \underset{T}{\operatorname{argmax}} \sum_{t_i \in T} \eta(t_i) + \gamma(T) \quad (1)$$

The evaluation of this function over all possible tuples T has exponential complexity, as there exist approximately $O(K^N)$ different tuples. Instead, we approximate the *global* component γ by exploiting the information contained in vertex embeddings, which are created from the KB to capture semantic relations between entities. Given a tuple, we can measure the average distance of the embeddings from the mean tuple embedding: a set of coherent candidates will have low average distance.

Our *local* function η scores candidate entities using string similarity with the mention they relate to and the *PageRank* score of the candidate entity.

3.1 Knowledge Graph Creation

Our EL algorithm links mentions to entities in Wikipedia, using a Knowledge Graph (KG) created from DBpedia[2]. DBpedia represents entities with a schema, and divides them in different types. For example, *Paris*[3] is an entity of type `PopulatedPlace`, and has properties such as `area`, `country` and `mayor`.

From a Knowledge Base represented with Resource Description Framework (RDF) such as DBpedia, it comes natural to create a Knowledge Graph, so that relations between entities can be more easily analyzed and leveraged for EL. Even though DBpedia links are directed, we consider our graph as undirected, as the information contained in DBpedia links is inherently bidirectional (with the exception of *redirection* and *disambiguation* links). We employ most of the files that compose the English DBpedia, and create a graph with around 12 million vertices and 170 million (bidirectional) edges.

DBpedia links that perform redirections (e.g. *NY*[4]) or are outlinks of disambiguation pages (e.g. *Paris*[5]) define a Directed acyclic graph (DAG) that is used to enrich the candidate set of each mention, by adding redirection and disambiguation targets to the set.

3.2 Vertex Embeddings Learning

A vertex embedding is an n-dimensional vector associated with the vertex of a graph, created to encode in a compact way the information of the vertex. This information could be related to the topology of the graph (e.g. information about the connectivity of a vertex or about its neighborhood) or encode additional features of the vertex (e.g. the category of a Wikipedia page). The key idea behind embeddings algorithms is that vertices that are similar or related in some way should also have similar embeddings.

Embeddings are not specific to graphs, but can also be used to represent images or text, often using very similar techniques. Indeed, some of the most famous vertex embeddings algorithms, such as DeepWalk[20], and *node2vec*[7] leverage ideas taken from the field of word embeddings. Both these algorithms, in fact, apply *word2vec*[17] to random walks, random sequences of adjacent vertices. In *word2vec*, using the *skip-gram* model, the embedding of a word w_i is learnt to maximize the probability of surrounding

[1] https://ai.googleblog.com/2013/03/learning-from-big-data-40-million.html

[2] https://wiki.dbpedia.org/

[3] http://dbpedia.org/page/Paris

[4] http://dbpedia.org/page/NY

[5] http://dbpedia.org/page/Paris_(disambiguation)

words $w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}$ that are often occuring near w_i (for a context window of size 5).

In our algorithm, we make use of a custom implementation of DeepWalk applied to our KG. As the random walks are vertex sequences, we can transform them into embeddings using *word2vec*.

Our implementation of DeepWalk doesn't use *hierarchical sampling* to approximate the softmax probabilities, like in the original paper. Instead, we employ *negative sampling*, as in *word2vec* and Node2Vec [7]. Negative sampling approximates the probabilities by sampling a small number of vertices (e.g. 5%) and is more efficient and flexible from a computational standpoint as it doesn't require an iteration over all the vertices to update a single embedding.

Recent vertex embeddings algorithms, such as GCN [14] or GraphSage [8] allows encoding vertex properties directly in the vertex embeddings. With these techniques, it's possible to leverage additional vertex information, such as the vertex labels, or even the embeddings generated from the text or images in each page, and obtain even more powerful representations. In this work, we limit ourselves to simpler embeddings to prove that our algorithm can be highly effective even with a limited amount of information, and leave the analysis of additional embeddings techniques as future work.

4 IMPLEMENTATION

In our framework, like other approaches in the field, Entity Linking (EL) is performed in two main steps. The first, the *candidates finder*, obtains, for each textual mention from an input document, a small number of candidate entities from the graph. The second, *collective disambiguation*, ranks the candidate entities to obtains the candidate tuple that best links each mention to the Knowledge Base (KB).

4.1 Candidate Finder

The goal of the *Candidate Finder* is to reduce the dimensionality of the problem by obtaining, for each mention found using Named Entity Resolution (NER), a small set of candidate entities. These entities are represented by vertices of the Knowledge Graph (KG), and the correct entity should be present among them. Candidate entities are chosen among the full set of vertices in the KG.

Similarly to most EL algorithms, we assume that Named Entities are already provided as input, and finding the candidates for each Named Entity (or *mention*) is the first step of our algorithm.

Each candidate vertex is assigned a score in $[0, 1]$ that represents the similarity of its identifier (i.e. the DBpedia entity name) to the surface form of the textual mention. To avoid a linear scan of all the vertices in the KG, which has complexity $|V|$ and is unsuitable for real-time usage, we use an index-based string search system. Experimenting with different string similarity metrics didn't lead to significant accuracy differences, so we opted for a straightforward combination of 2-grams and 3-grams similarity, which is extremely fast to compute thanks to our indexing approach.

Often, entities might appear with plenty of different surface forms: for example, *"New York"* could also be mentioned as *"NY"* or *"Big Apple"*. By building a graph with DBpedia redirection links, we can match a surface form such as *"NY"* to a vertex whose name is *"NY"*, and then follow the redirection link from vertex *"NY"* to vertex *"New_York"*. Another ambiguous scenario is related to very

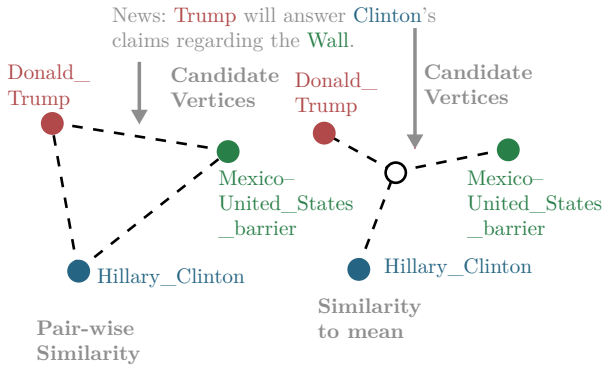

Figure 3: Different ways to compute the *global score* γ using vertex embeddings

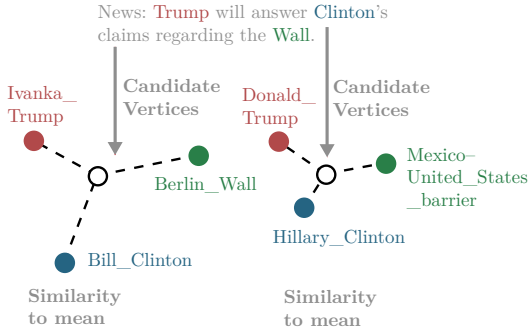

Figure 4: Vertices which are more strongly related have more similar embeddings, and give better EL

generic surface forms, which might be mapped to many different vertices. For example, *"Paris"* could be a French city, but also the name of many different movies. To deal with this scenario, we employ disambiguation pages and links. If a surface form is matched with a disambiguation page, we add all the destinations of the disambiguation page to the list of candidates.

In both cases, the score of the newly added candidates is given by the equation eq. (2) (where $\mathcal{S}$ is a string similarity function and we use the *"New York"* example for clarity).

$$score(\text{"}New\_York\text{"}) = max(\mathcal{S}(\text{"}NY\text{"}, \text{"}NY\text{"}), \mathcal{S}(\text{"}NY\text{"}, \text{"}New\_York\text{"})) \quad (2)$$

Redirection and disambiguation links might follow each other, but the subgraph induced by taking all the edges of this type is a Directed acyclic graph (DAG). Thus, we perform the score propagation step as a Breadth-First Visit (BFV) starting from the initial sets of candidates. From an implementation standpoint, we first retrieve the K candidates with the highest score, then perform score propagation. As such, the final sets might have a size smaller or larger than K.

4.2 Collective Disambiguation

At this stage of the algorithm, each mention has a set of candidate vertices, each with its own similarity score (computed by the *Candidate Finder*). The goal is to pick the best candidate vertex for each mention, and provide the final document entity linking.

Given a list of mentions ($M_0, M_1, \ldots, M_N$), each mention M_i has a list of candidates ($C_{i1}, C_{i2}, \ldots, C_{iK}$). A candidate solution is a tuple ($C_{1i}, C_{2j}, \ldots, C_{Nq}$), with $C_{1i}, C_{2j}, \ldots$ being candidate vertices from the candidate set of each mention. For simplicity and ease of explanation, we also refer to the elements of T as $\{t_1, t_2, \ldots, t_N\}$ to denote to the i-th tuple element. As explained by eq. (1), the goal of the algorithm is to find a tuple T^* where each candidate vertex is reasonably similar to the mention it refers to (according to the string similarity metric), but also, more importantly, where candidate vertices for different mentions are strongly related.

Optimizing the relation of different candidates, i.e. the *global score* γ in eq. (1), can be done in different ways. For example, we can define a pairwise vertex similarity metric γ_{ij} which is applied to each pair of candidates t_i, t_j in T, and obtain the *global score gamma* as the sum of γ_{ij} over all pairs of candidates. Then, we compute γ over all possible tuples T and keep the one with maximum value (of both γ and η, the *local score*) as final prediction T^*.

In our experiments, we initially used Personalized PageRank (PPR) as pairwise similarity function γ_{ij}. PPR gives the importance of each vertex in a graph with respect to a designated personalization vertex. Given a personalization vertex, vertices with high PPR score will be more relevant to it. We can use this idea to compute the relatedness of a tuple of candidate vertices. For each candidate vertex, we compute PPR using that vertex as personalization vertex, and store the PPR scores of all the other candidate vertices. Then, we look at the tuple of candidate vertices with the highest sum of pairwise PPR scores. This approach has the drawback of being computationally intensive, as it requires to compute PPR for each of the candidate vertices. For N mentions and K candidates per mention, we need to compute approximately N^2K^2 PPR scores, as PPR is generally not symmetric. Even if we considered a subgraph, as proposed in AGDISTIS [27], it would be infeasible to compute PPR thousands of times while keeping real-time latencies. Also, storing all the pre-computed $|\mathcal{V}|^2$ PPR scores requires hundreds of terabytes of storage, making this approach equally impractical.

Instead of relying on PPR, we developed an approach based on vertex embeddings, moved by the idea that embeddings can encode the same information of PPR. If two vertices are similar, their embeddings should also be similar. To compute the distance of two embeddings, we use cosine similarity.

The global score γ of a tuple can be computed in 2 ways:

(1) As the sum of pairwise similarities of all the candidates' embeddings.
(2) As the sum of the similarities of the candidates' embeddings from the mean vector of the tuple's embeddings.

In our experiments, we focus on the second implementation, as it has evaluation complexity $O(N)$ instead of $O(N^2)$. From a theoretical standpoint, maximizing the similarity w.r.t. the mean tuple embedding can be seen as a variance minimization of the tuple's embeddings, if Euclidean distance is used to compare embeddings. This interpretation intuitively shows why computing γ in such a way leads to predicting strongly related vertices.

The equation to compute γ with the second technique is given in eq. (4). Function $\mathbf{e}(t_i)$ retrieves the embedding of a vertex t_i, while $\bar{\mathbf{e}}(T)$ is the mean embedding of a tuple T (eq. (3)).

$$\bar{\mathbf{e}}(T) = \frac{1}{|T|} \sum_{t_i \in T} \mathbf{e}(t_i) \tag{3}$$

$$\gamma(T) = \sum_{t_i \in T} \frac{\langle \mathbf{e}(t_i), \bar{\mathbf{e}}(T) \rangle}{\|\mathbf{e}(t_i)\|_2 \|\bar{\mathbf{e}}(T)\|_2} \tag{4}$$

The *local score* η of a tuple is computed as the sum of the independent local scores of each candidate vertex. The local score of a candidate is the weighted sum of its candidate similarity and of its PageRank score (pre-computed in the preprocessing of the KG).

As stated before, the goal of the algorithm is to find a tuple of candidates that maximize a (weighted) sum of global and local scores. Evaluating all combinations of candidates is not feasible for documents with more than a handful of mentions (e.g. 10 mentions with 10 candidates each would result in 10^10 different tuples). Instead, we devised a heuristic optimization algorithm based on state-space search exploration that is able to converge to good solutions (without guarantee of convergence to the global optimal) in a very small amount of iterations. The basic idea of the optimization algorithm is to iteratively improve an existing candidate tuple. Starting from an initial tuple, at each step the algorithm will create a new tuple which has a better score. The algorithm terminates after a given amount of steps, or when an early-stop termination criterion is met (e.g. the score didn't improve for the last n iterations). The initial tuple is the one with maximum *local score*, found in $O(KN)$ time.

In each optimization step, the algorithm first creates a specified number of new tuples by modifying a random number of candidates. The best of these new tuples is then picked. This new tuple is then optimized with a greedy procedure. For each mention (in random order), we pick the candidate that maximizes the overall tuple score (i.e. *local* and *global*, while other candidates are kept (temporarily) fixed. Pseudo-code for the collective disambiguation algorithm is provided in algorithm 1.

5 EXPERIMENTAL EVALUATION

In this section, we describe the creation of the Knowledge Graph (KG) from DBpedia and the embeddings generation (section 5.1), the runtime hyper-parameters of the Entity Linking (EL) algorithm (section 5.2) and the evaluation metrics we adopted (section 5.3). We compare our EL algorithm against multiple state-of-the-art EL systems, on a variety of data sets that allow us to measure the effectiveness of our approach under many different scenarios. We also provide an analysis of the execution time of our algorithm, and show how accuracy and document analysis latency can be adapted to the user's needs with simple configuration changes.

5.1 Preprocessing

Our KG is created starting from DBpedia 2016 – 10. Before creating the KG, we clean the entity names by removing characters such as asterisks, quotes and spaces, and by removing *disambiguation* suffixes. To create a KG as large as possible, we treat literal values as

Function `optimizer(`*candidates*`)`**:**

// The initial state is the tuple with maximum η

T = `find_initial_state(`*candidates*`)`

s = `compute_score(`T`)`

best_tuple = T

best_score = s

while *stop condition not met* **do**

// Create *num_children* new tuples by modifying random elements of T. Keep the new tuple with highest score

T = `new_tuples(`*T, num_children*`)`

T = `optimize_tuple(`T`)`

s = `compute_score(`T`)`

if *curr_score* $\geq$ *best_score* **then**

best_tuple = T

best_score = s

end

end

return best_tuple, best_score

Function `compute_score(`T`)`**:**

// Local score is a weighted sum of string similarity and PageRank, $\gamma(T)$ is computed as in eq. (4)

// α and β are hyper-parameters

$\eta_1 = \sum_{t_i \in T} string\_sim(t_i)^\alpha$

$\eta_2 = \sum_{t_i \in T} PR(t_i)^\beta$

return $w_1 \cdot \eta_1 + w_2 \cdot \eta_2 + (1 - w_1 - w_2) \cdot \gamma(T)$

Function `optimize_tuple(`T`)`**:**

for $i \in \{1, \ldots, T.size\}$ **do**

T = `optimize_mention` (T, i)

end

return T

Function `optimize_mention(`T, i`)`**:**

// Given a tuple T and a position i, find for the mention in that position the candidate that maximizes the tuple score

return T

Algorithm 1: The collective disambiguation algorithm. Its input is, for each mention to be linked, a set of candidate vertices, as found by the *candidates finder*. The algorithm stops after a fixed number of iterations, or if the score hasn't improved for n steps

entity names whenever possible (i.e. if an entity with the same value as the literal exists). Overall, the KG has 12 million entities/vertices and 170 million links/edges.

Before computing PageRank and creating the embeddings of the vertices, we undirect the graph and make every link bidirectional. PageRank is computed with standard parameters (e.g. *damping* factor of 0.85), and runs for about 800 iterations before convergence.

Vertex embeddings are created with DeepWalk, using an embedding size of 160, a *random walk length* of 8, 12 random walks for each vertex, and 6 epochs. The other parameters are left to the default values proposed in the original paper [20]. Preprocessing time depends on the parameters used, but takes on average between 4 to 8 hours.

Experiments were mostly run on a single server with an Intel® Xeon® CPU E5-2680 v2 with 20 cores, 40 threads at 2.80GHz and 378GB of RAM.

5.2 Runtime Hyper-Parameters

Our EL algorithm is fully unsupervised, and doesn't require any training [10] or prior probabilities estimations [29] to be effective. It supports, however, a number of hyper-parameters that can be tuned to obtain a flexible trade-off between inference time and accuracy, according to the users' needs. In the list below, w_1, w_2, α and β influence the scoring function, and mostly serve as scaling factors to make the *local* and *global* scores comparable.

We report the values used in the experiments, although our algorithm is very robust to small changes of parameters, and modifications don't usually affect accuracy in a noticeable way.

- K: the number of candidates per mention obtained by the *candidate finder*. With just 100 candidates, the correct entity link is among these candidates more than 95% of the times. Higher values of K don't improve this percentage in a significant way. Lower values of K (e.g. 10) can greatly decrease the execution time of the algorithm.
- w_1, w_2: weights attributed to the *local* score components. They influence the string similarity and the PageRank score of a candidate entity, respectively. Reasonable values are $w_1 \in [0.4, 0.5]$ and $w_2 \in [0.25, 0.3]$. $1 - w_1 - w_2$ is the weight given to γ, the global score.
- α, β: exponents to which the string similarity and the PageRank score of a candidate entity are raised. They work as normalization/scaling factors. Reasonable values are $\alpha \in [0.8, 1]$ and $\beta = 0.1$.
- `num_steps`: number of steps done by the optimization algorithm. Usually even 10 steps are enough to provide good results, but occasionally larger values (e.g. 40) can be useful. Low values can boost significantly the inference time, with minimum loss of accuracy. By default, it is used an *early stop* value of ½`num_steps`.

5.3 Evaluation Metrics and Data Sets

The literature makes use of multiple metrics to evaluate the performance of different EL systems. The lack of a common metric that is universally shared by different authors makes sometimes hard to compare different EL techniques. In our work, we focus on *micro-averaged* Bag-of-Concepts (BoC) *precision* π, *recall* ρ and *F1*. We also provide values of *micro-averaged* and *macro-averaged accuracy a*. Results obtained by our algorithm are compared to other EL systems using the same metrics chosen by the respective authors. No other author reported metrics related to execution time.

Accuracy a is simply the percentage of correctly predicted entity links. *Macro-averaged accuracy* is the average of the accuracy values measured on each document, while *micro-averaged accuracy* is measured on the entire data set at once. As documents in a data set could have a significantly different number of mentions, micro-averaging assigns higher impact to documents with more mentions, and gives a better picture of the real performance of the algorithm.

Table 1: Summary of the characteristics of different data sets used in the evaluation

Data Set	Type	#docs	#mentions	Mentions per doc.
ACE2004	news	57	252	4.44
AQUAINT	news	50	727	14.54
MSNBC	news	20	658	32.90
N3-Reuters	news	128	650	5.08
N3-RSS-500	Rss-feeds	500	524	1.05

Given a set of document D and a document $d_i \in D$, for which there is a prediction entity set P_i and a golden entity set G_i, one can define micro-averaged BoC as

$$\pi = \frac{|\bigcup_{d_i \in D} G_i \cap P_i|}{|\bigcup_{d_i \in D} P_i|} \quad \rho = \frac{|\bigcup_{d_i \in D} G_i \cap P_i|}{|\bigcup_{d_i \in D} G_i|} \quad F1 = 2\frac{\pi \cdot \rho}{\pi + \rho} \tag{5}$$

Even though BoC works on sets of elements and it doesn't check that prediction-golden pairs are correct, it gives numerical values very similar to *accuracy* and is often used in the literature as the performance metric of choice [29].

In our evaluation, we focus on 5 data sets that are commonly used as a benchmark of EL algorithms, thanks to the significantly different types of documents they contain and because they are among the largest available in this field [29]. The following list details the data sets used in our experiments.

- **ACE2004**: this small data set was assembled by Ratinov et al. [24] and contains 57 articles, with 253 mentions.
- **AQUAINT**: this data set contains 50 documents and 727 mentions (14.54 mentions per document), extracted from news belonging to the Xinhua News Service, the New York Times, and the Associated Press. It was created by Milne and Witten [18] and commonly used in the literature (among other, by [15] and [29]).
- **MSNBC**: the corpus was created by Cucerzan et al. [4] and contains 20 documents with 658 mentions (32.90 mentions per document), extracted from news articles. It contains the largest documents of all our data sets, and it is a good benchmark to measure the execution time and flexibility of our algorithm.
- **N3-Reuters**: created by Röder et al. [25], this corpus contains 128 economic news articles with 650 mentions.
- **N3-RSS-500**: this data set contains 500 documents extracted from RSS feeds, with topics related to politics and economy. It was created by Gerber et al. [6].

Note that there exists a 1-to-1 mapping between Wikipedia and DBpedia URIs, so using DBpedia for our KG allows us to easily retrieve the corresponding Wikipedia URIs.

5.4 Accuracy Analysis

Our algorithm is tested against the following EL systems: AIDA [11], Wikifier [24], DoSeR [29], WAT [22], Babelfy [19] and Spotlight [16]. Results for these algorithms are taken from Zwicklbauer et al. [29], who in turn computed them through GERBIL (General Entity Annotator Benchmark), proposed by Uzbeck et al. [28]. GERBIL is a web platform that enables the comparison of different EL systems over a variety of data sets, and ensures that results are comparable. Our experiments were run with the same scenarios provided by GERBIL. Mentions whose corresponding entities are not present in the KG are not considered in the evaluation of F1, precision, recall and accuracy. Results are obtained using the best parameters and configurations provided by the authors.

Table 2: Results of micro and macro-averaged accuracy *a*, and micro-averaged precision π, recall ρ and F1 score obtained by our algorithms on different data sets

Data Set	Macro acc.	Micro acc.	π	ρ	F1
ACE2004	0.85	0.85	0.83	0.83	0.83
AQUAINT	0.84	0.86	0.86	0.86	0.86
MSNBC	0.90	0.89	0.90	0.94	0.92
N3-Reuters	0.76	0.76	0.78	0.87	0.82
N3-RSS-500	0.68	0.72	0.71	0.73	0.72

Table 3: Results of BoC micro-averaged F1 score of different state-of-the-art EL algorithms on multiple data sets

Data Set	Ours	DoSeR	WK	AIDA	WAT	BB	SL
ACE2004	0.84	**0.90**	0.83	0.81	0.80	0.56	0.71
AQUAINT	**0.86**	0.84	**0.86**	0.53	0.77	0.65	0.71
MSNBC	**0.92**	0.91	0.85	0.78	0.78	0.60	0.51
N3-Reuters	0.82	**0.85**	0.70	0.60	0.64	0.53	0.58
N3-RSS-500	0.72	**0.75**	0.73	0.71	0.68	0.63	0.62

WK is Wikifier, **BB** is Babelfy, **SL** is Spotlight

To the best of our knowledge, DoSeR and Wikifier are the best performing EL algorithms when tested against our evaluation data sets. Some of the algorithms in our analysis provide links to knowledge bases different from Wikipedia or DBpedia (for example, AIDA uses YAGO2 alongside DBpedia), but in all cases there exists 1-to-1 mapping between the entities of different Knowledge Base (KB), which enables the comparison of these systems.

Table 2 shows different metrics of performance of our EL algorithm across different data sets. It can be seen that the lowest accuracy is obtained on the **N3-RSS-500** data set: this data set has, on average, just 1 mention per document. As such, we cannot rely on the *collective disambiguation* step to provide good results. Instead, the final predictions will be mostly based on the *candidate finder* and the *PageRank* score of each candidate. On the other hand, we see that results on the other data sets are closer to each other, even though the average number of mentions in their documents is highly different. This result suggests that our algorithm doesn't require a large context to provide good predictions, but even 4 or 5 mentions are often enough. Also, our EL system seems to favor high *recall* ρ over high *precision* π (this is especially evident with

the **N3-Reuters** data set). This implies that, broadly speaking, most of the entities that should be retrieved are actually found by our algorithm. In our experiments, we managed to increase *precision* by giving higher weight to the *global score* γ. The intuition is that there might be mentions which should be linked to the same entity. Choosing the same entity for multiple mentions will give a higher *global score*, as the mean embedding will tend to the embedding of the repeated entity. Thus, giving higher weight to γ will improve *precision*. Still, the settings used in our experiment gave a better balance of π and ρ, and an overall higher F1 score.

Table 3 shows how our algorithm fares against other state-of-the-art EL algorithms. We can see how only *DoSeR*, and (limited to the AQUAINT data set) *Wikifier* provide strong competition to our algorithm. Still, *Wikifier* is a supervised algorithm as it uses a ranking Support Vector Machine (SVM), while *DoSeR* requires the estimation of prior probabilities given *mention-target entity* pairs. Compared to them, our algorithm doesn't require any training, and is significantly simpler. Moreover, our algorithm can easily be applied to other domains other than Wikipedia, as it doesn't require any text analysis or information specific to Wikipedia (e.g. the *mention-target entity* pairs) and provides much higher modularity, as most of its components (the *embedding* algorithm, the *string matching* algorithm, and the *optimization* algorithm) can be updated or replaced according to the user's need, without having to retrain the EL algorithm.

5.5 Execution Time Analysis

The focus of our EL algorithm is not just on the accuracy of its predictions, but also on providing good results in a very short, and possibly real-time, time-frame (here, we consider the algorithm to be real-time if the end-to-end analysis takes less than one second). Moreover, users should be able to tune the algorithm according to their needs, and to whether they require higher accuracy, higher throughput or lower latency.

Figure 5 provides a breakdown of the execution time of our EL algorithm, to better understand which parts of the algorithm take the most time. We show how the execution time can be influenced using different settings, and how a minimal drop in accuracy can translate to a significant reduction in the document analysis latency. We measure times on the **MSNBC** data set as it has the highest *mention per document* value among our data sets (32), and provides a reasonable upper bound on the execution time of our algorithm. Most of the execution time of the algorithm is spent either in the *string-matching* step of the *candidate finder*, or in the *collective disambiguation* step. The overheads (everything which doesn't belong to a specific step) are negligible, as it can be measured by subtracting the **CF** and **Dis.** columns from the total execution time. In the **low-latency** configuration we lower the number of candidates per mention, and the number of optimization steps: the execution time improves by almost a factor of 6x compared to the **default** configuration, with a loss of F1 score of only 2% absolute points.

Our EL algorithm is fully multi-threaded, and both the *candidate finder* and the *disambiguation step* are implemented in a parallel fashion. The *candidate finder* can process multiple mentions at the same time, and the underlying string matching algorithm used to find the candidates of each mention is also highly parallelized. The

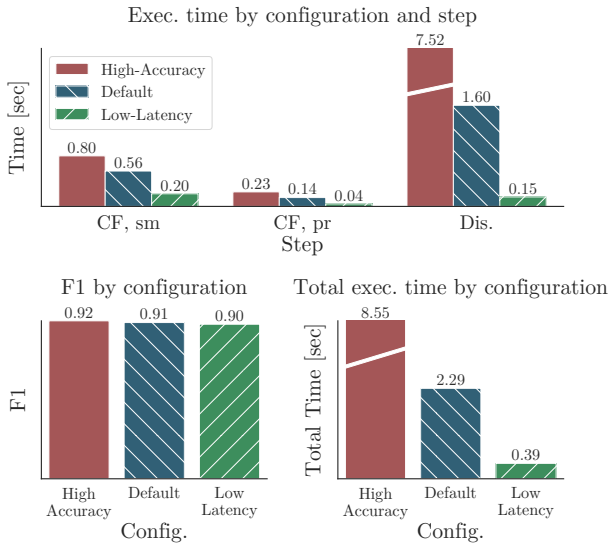

Figure 5: Execution time with different configurations

CF, sm is the string-matching part of the *candidates finder*, while **CF, pr** is the score propagation using redirection and disambiguation links. **Dis.** is the *collective disambiguation* step.

disambiguation algorithm analyzes multiple candidate tuples at once, and the computation of the *global score* γ of each tuple is also multi-threaded, being mostly vector arithmetic.

Users can also choose whether to process multiple documents at once (to maximize throughput), or to fully dedicate the machine resources to a single document (to minimize latency). To compute the execution time breakdown in a reliable way, we disabled the analysis of multiple documents in parallel. Our analysis doesn't consider the time required for the initial Named Entity Resolution (NER) step, as our focus is on the EL algorithm. Still, the availability of high-performance NER algorithms [5, 26] implies that real-time latencies are still achievable.

6 CONCLUSION AND FUTURE WORK

We presented a novel unsupervised Entity Linking (EL) algorithm and framework that leverages vertex neural embeddings to provide high accuracy and performance, along with an unmatched degree of modularity and flexibility. Our approach can rival and exceed multiple supervised state-of-the-art algorithms, and provide real-time analyses without significant sacrifices in terms of accuracy. The results presented in this work were achieved using *DeepWalk*, a fairly old and simple graph embedding algorithm. We plan to test more recent algorithms and make use of graph properties (such as entity labels, or textual embeddings created from Wikipedia pages) to further improve accuracy. Thanks to the structure of the framework, we don't expect any degradation in inference performance when using better embeddings.

REFERENCES

[1] Ayman Alhelbawy and Robert Gaizauskas. 2014. Collective named entity disambiguation using graph ranking and clique partitioning approaches. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. 1544–1555.

[2] Douglas Burdick, Ronald Fagin, Phokion G Kolaitis, Lucian Popa, and Wang-Chiew Tan. 2016. A declarative framework for linking entities. *ACM Transactions on Database Systems (TODS)* 41, 3 (2016), 17.

[3] Amparo E Cano, Giuseppe Rizzo, Andrea Varga, Matthew Rowe, Milan Stankovic, and Aba-Sah Dadzie. 2014. Making sense of microposts:(# microposts2014) named entity extraction & linking challenge. In *CEUR Workshop Proceedings*, Vol. 1141. 54–60.

[4] Silviu Cucerzan. 2007. Large-scale named entity disambiguation based on Wikipedia data. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

[5] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, 363–370.

[6] Daniel Gerber, Sebastian Hellmann, Lorenz Bühmann, Tommaso Soru, Ricardo Usbeck, and Axel-Cyrille Ngonga Ngomo. 2013. Real-time RDF extraction from unstructured data streams. In *International Semantic Web Conference*. Springer, 135–150.

[7] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[9] 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*. ACM, 765–774.

[10] Johannes Hoffart, Yasemin Altun, and Gerhard Weikum. 2014. Discovering emerging entities with ambiguous names. In *Proceedings of the 23rd international conference on World wide web*. ACM, 385–396.

[11] 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*. Association for Computational Linguistics, 782–792.

[12] Neil Houlsby and Massimiliano Ciaramita. 2014. A scalable gibbs sampler for probabilistic entity linking. In *European Conference on Information Retrieval*. Springer, 335–346.

[13] Heng Ji, Joel Nothman, Ben Hachey, and Radu Florian. 2015. Overview of TAC-KBP2015 Tri-lingual Entity Discovery and Linking.. In *TAC*.

[14] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[15] Xiao Ling, Sameer Singh, and Daniel S Weld. 2015. Design challenges for entity linking. *Transactions of the Association for Computational Linguistics* 3 (2015), 315–328.

[16] Pablo N Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. 2011. DBpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th international conference on semantic systems*. ACM, 1–8.

[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[18] David Milne and Ian H Witten. 2008. Learning to link with wikipedia. In *Proceedings of the 17th ACM conference on Information and knowledge management*. ACM, 509–518.

[19] 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), 231–244.

[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[21] Maria Pershina, Yifan He, and Ralph Grishman. 2015. Personalized page rank for named entity disambiguation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 238–243.

[22] 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*. ACM, 55–62.

[23] Delip Rao, Paul McNamee, and Mark Dredze. 2013. Entity linking: Finding extracted entities in a knowledge base. In *Multi-source, multilingual information extraction and summarization*. Springer, 93–115.

[24] 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*. Association for Computational Linguistics, 1375–1384.

[25] Michael Röder, Ricardo Usbeck, Sebastian Hellmann, Daniel Gerber, and Andreas Both. 2014. N^3-A Collection of Datasets for Named Entity Recognition and Disambiguation in the NLP Interchange Format.. In *LREC*. 3529–3533.

[26] Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. 2017. Fast and accurate entity recognition with iterated dilated convolutions. *arXiv preprint arXiv:1702.02098* (2017).

[27] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Michael Röder, Daniel Gerber, Sandro Athaide Coelho, Sören Auer, and Andreas Both. 2014. AGDISTIS-graph-based disambiguation of named entities using linked data. In *International semantic web conference*. Springer, 457–471.

[28] Ricardo Usbeck, Michael Röder, Axel-Cyrille Ngonga Ngomo, Ciro Baron, Andreas Both, Martin Brümmer, Diego Ceccarelli, Marco Cornolti, Didier Cherix, Bernd Eickmann, et al. 2015. GERBIL: general entity annotator benchmarking framework. In *Proceedings of the 24th international conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1133–1143.

[29] Stefan Zwicklbauer, Christin Seifert, and Michael Granitzer. 2016. Robust and collective entity disambiguation through semantic embeddings. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 425–434.