International Journal of Innovative
Computing, Information and Control
Volume **8**, Number **11**, November **2012**

ICIC International ©2012 ISSN 1349-4198
pp. 7519–7541

WIKIIDRANK: AN UNSUPERVISED APPROACH FOR ENTITY LINKING BASED ON INSTANCE CO-OCCURRENCE

Norberto Fernández, Jesus A. Fisteus, Luis Sánchez
and Damaris Fuentes-Lorenzo

Web Technologies Laboratory
Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid
Avda. Universidad 30, Leganés, Madrid, Spain
{ berto; jaf; luiss; dfuentes }@it.uc3m.es

Received July 2011; revised January 2012

Abstract. *In recent times there has been an interest in developing automatic techniques to create and/or populate knowledge bases (KB) and ontologies. One initiative along this line is the Knowledge Base Population (KBP) track of the Text Analysis Conference (TAC), which offers a framework for the evaluation of automatic systems designed to populate ontologies using information found in unstructured text. In KBP, the knowledge base population process has been divided into two complementary tasks: entity linking, whose goal is to detect mentions in text to instances in a reference KB, and slot filling, which extracts from text facts about the previously detected instances and adds these facts to the KB. In this paper, an automatic, unsupervised algorithm for entity linking is presented. The main novelty of the proposed approach, named WikiIdRank, comes from applying a PageRank-like algorithm to a network of instance co-occurrences to address the task. The algorithm was implemented and evaluated in the KBP 2010 with positive results. Furthermore, an analysis of the impact of instance co-occurrence information in the entity linking process was carried out, and indicates a gain of accuracy around 16% compared with a baseline approach relying on information retrieval techniques.*
Keywords: Knowledge discovery, Ontology population, Entity linking, Named entity disambiguation, Instance co-occurrence

1. **Introduction.** Nowadays, digital contents are produced at an extraordinary pace. For instance, a recent report by IDC[1] estimates that the world will create a total number of 1.8 zettabytes of digital contents in 2011. It also indicates that the world's information is doubling every two years.

The large and quickly increasing volumes of digital contents that are available either in public (like the web) or private repositories (like corporate intranets), make it difficult to find valuable information resources and to analyze data to extract meaningful conclusions. This leads to a problem of information overload or *infobesity* [1] that affects both individuals and organizations.

In order to cope with this problem, initiatives like the Semantic Web [2, 3, 4] or the Linked Data [5][2] propose the creation of knowledge bases or ontologies [6] by extracting information from digital contents and representing it in a formal, well-structured manner. These knowledge bases or ontologies can be later used by software agents to carry out complex information processing tasks. For instance, they have been successfully applied

[1]http://spain.emc.com/leadership/programs/digital-universe.htm (30/Nov/2011)
[2]http://linkeddata.org/ (30/Nov/2011)

to tasks like information retrieval [7, 8], web page clustering or information integration [9] and in domains like medicine [10], news distribution [11], genetics [12].

However, the process of structuring the information in digital contents into knowledge bases or ontologies is not free from difficulties. These difficulties are a consequence of the *knowledge acquisition bottleneck* [13], a well known problem in the knowledge management area. Some of the causes of this problem are (1) the large scale of the contents to be processed; (2) the unstructured nature of a significant part of the contents (HTML, plain text), which makes the process of conversion into well-structured representations more difficult and, (3) the evolution of information over time, which imposes the need for maintenance in the knowledge bases and ontologies. Due to these difficulties, it is not surprising the interest in developing techniques to automate, at least partially, the process of building, populating and maintaining knowledge bases and ontologies.

One of the initiatives along this line is the Knowledge Base Population (KBP) track[3] of the Text Analysis Conference (TAC)[4]. This forum aims at offering a research environment for the competitive evaluation of automatic systems designed to populate ontologies using information found in unstructured text.

In the context of the KBP track, all participants are provided with a corpus of text documents and a reference knowledge base to be populated. The population process is divided into two different tasks: *entity linking* and *slot filling*. KBP participants may address both tasks or only the entity linking one. Basically, the main goal of a slot filling system is to populate the reference knowledge base with facts extracted from the different documents in the corpus. However, those facts should be linked to a concrete entry in the knowledge base (a.k.a. *instance*)[5], and thus a previous process of detecting mentions to instances in the text of documents is needed. For example, before a slot filling system can extract from a document a fact about the date of birth of a concrete instance of a person (like *Angela Merkel*), the mention to this concrete instance needs to be recognized in the document. Nevertheless, this recognition process is not always easy, as instances may be sometimes referred to with ambiguous names (like *Merkel*, which may represent *Angela Merkel* or *Alexander Merkel*), alternative names (like *Angela Kasner*), or even misspelled names (like *Angela Mierkel*). Successfully addressing these difficulties is the goal of the entity linking systems, which, given a document in the corpus and a piece of text mentioned in that document (a.k.a. *entity*), should find out in the reference knowledge base the instance, if it exists, that best represents the meaning of the entity in the context of the document. Thus, using accurate entity linking systems is crucial for the success of slot filling systems in their task of populating the reference knowledge base with reliable facts, because if the instances detected by the entity linking system were wrong, the facts extracted at a later step by the slot filling system would also be wrong.

In this paper an automatic, unsupervised approach to the KBP entity linking task is presented. The proposed algorithm, named WikiIdRank, is based on previous work of the authors in the context of semantic annotation of news items: the IdentityRank algorithm [14]. In particular, both IdentityRank and WikiIdRank are inspired in the *semantic coherence* principle as stated in [14]. According to this principle, instances are frequently mentioned in documents together with (i.e., *co-occur* with) other related instances (for example, the instances for *Obama* and *USA* or the ones for *Pau Gasol* and

[3]http://nlp.cs.qc.cuny.edu/kbp/2011/ (30/Nov/2011)

[4]http://www.nist.gov/tac/2011/ (30/Nov/2011)

[5]In this paper a common nomenclature within the Semantic Web community is adopted. However, it must be clarified that other communities, like the one researching in natural language processing, usually adopt a different terminology, using the term *surface form* to refer to what is called *entity* in the context of this paper and the term *entity* to refer to an *instance* in a knowledge base.

Los Angeles Lakers). Thus, the mention (*occurrence*) of a certain instance in a certain context (like a document) gives information about the occurrence of other instances in the same context. However, whereas IdentityRank was semi-supervised, and specifically designed to process news items' streams, WikiIdRank is an unsupervised algorithm and may be applied to different contents. In fact, the KBP corpus includes news items, but also web resources like blog entries.

The algorithm was implemented and evaluated in the context of KBP 2010, achieving positive results. Furthermore, an analysis of the impact of instance co-occurrence information in the entity linking process was carried out, and indicates a gain of accuracy around 16% compared with a baseline approach relying on information retrieval techniques.

The rest of this paper is organized as follows. Section 2 provides an overview of the entity linking task of KBP. Section 3 describes the proposed algorithm. In Section 4, the results obtained by empirical evaluation of WikiIdRank in the KBP environment are shown and discussed. Section 5 compares the approach with related work in the state of the art. Finally, Section 6 provides some concluding remarks and future lines of development of the system.

2. The Entity Linking Task.

Though the original description of the KBP 2010 entity linking track can be found on the web at the time of writing[6], for the sake of completeness, the most relevant aspects are covered in this section.

The main goal of a system participating in the entity linking task may be summarized as follows: given a *query* consisting of (1) a piece of text (entity) and (2) a document from a corpus where the text is mentioned, the system should find out the instance (if it exists) in a reference knowledge base that best represents the meaning of the entity in the context of the document. In case no appropriate instance were found in the knowledge base for the entity, the system should return *NIL* as answer for the query. The answers given by the automatic system to a set of queries provided by the organizing committee are compared to a golden standard of answers, manually built by a group of experts. The accuracy (percentage of correct answers) of the system is thus computed and used as metrics to compare different approaches.

A few phenomenon make this process of linking entities to knowledge base instances difficult. On the one hand, entities may be ambiguous, that is, the same piece of text may refer to different instances (for example, *George Washington* may refer to a jazz musician, a US president or a university). On the other hand, the same instance may be represented by several different entities: nicknames, aliases, acronyms, abbreviations, transliterations, etc. (for example, *Angela Merkel* and *Angela Kasner* may refer to the same person, *AZ* and *Arizona* refer to the same place, etc.). Successfully facing these phenomena is the target of entity linking systems.

In order to address the entity linking task a system needs some information sources: the reference knowledge base, the corpus of documents, the evaluation dataset (queries) to be answered, etc. In particular, the following sources are provided by the KBP organizing committee to participating teams:

- A **knowledge base**, generated by processing the info-boxes of the articles in an English Wikipedia dump of 2008. The knowledge base provides some information about each of its 818,741 entries, including: a label or name, a type (PERson, ORGanization, Geo-Political Entity or UnKNown), a unique identifier, a set of properties or *facts* and a text snippet obtained from the original Wikipedia page. Listing 2

[6]http://nlp.cs.qc.cuny.edu/kbp/2010/KBP2010_TaskDefinition.pdf (11/Jul/2011)

shows an example of knowledge base entry, representing the instance for the city of *Kunduz*.

```
<entry wiki_title="Kunduz" type="GPE" id="E0675891" name="Kunduz">
  <facts class="Infobox␣City␣in␣Afghanistan">
    <fact name="official_name">Kunduz</fact>
    <fact name="province_name">Kunduz</fact>
    <fact name="latd">36.73</fact>
    <fact name="longd">68.86</fact>
    <fact name="population_total">264100</fact>
    <fact name="population_as_of">2006</fact>
    <fact name="population_rank">5th</fact>
    <fact name="elevation_m">397</fact>
    <fact name="Main_Language"><link>Pashto</link></fact>
  </facts>
  <wiki_text>
    <![CDATA[Kunduz
      Kunduz also known as Kunduz, Qonduz, Qonduz, Konduz, Konduz,
      Kondoz, or Qhunduz is a city in northern Afghanistan, the
      capital of Kunduz Province [...]
    ]]>
  </wiki_text>
</entry>
```

Listing 1. An example of entry from the KBP reference knowledge base

- A **corpus** of documents in English represented in XML format. The corpus includes a collection of approximately 1 million news articles and around 300,000 documents obtained from web data, most of them selected from blog content.
- A set of **queries** (2250 to be precise) to be answered, represented in XML format as shown in Listing 2. Each query consists of three information pieces: a unique query identifier, a piece of text, representing the entity to be linked to the knowledge base, and a reference to a document in the corpus where the entity is mentioned. This document provides context information for the linking process.

```
<query id="EL013663">
  <name>Kunduz</name>
  <docid>XIN_ENG_20070703.0409.LDC2009T13</docid>
</query>
```

Listing 2. An example of query

- Additionally, a set of more than 5000 **human assessments** was also provided. Each of these assessments consists on a query that has been previously answered by human experts, mapping its entity to an instance in the knowledge base or to *NIL*. They allow teams to test and tune their systems and may also be used as training information in the case of supervised systems.

The entity linking task in KBP 2010 included two different sub-tasks: in one of them (*general* sub-task) the text from the Wikipedia pages associated with the knowledge base entries may be used as support information in the linking process, whereas in the other (*no-wikitext* sub-task), this text must not be used. However, it is possible in both cases to compile lists of name variations or alternative labels for instances, based on information

in Wikipedia like hyperlinks and redirections. It is also possible to create a KB of world knowledge using Wikipedia or a Wikipedia derived-resource (like DBpedia [15]). As will be shown in next section, WikiIdRank exploits these possibilities to extend the reference knowledge base with additional information.

3. **WikiIdRank.** In this section, the inner details of the WikiIdRank algorithm are described. Basically, given a query (with its entity and context document) the algorithm proceeds through the following four sequential stages:

1. It uses a combination of state-of-the-art named entity recognition tools to *find the entities* (persons, locations, organizations) that appear in the context document. These are merged with the entity in the query, to obtain the final set of entities that will be further processed by the algorithm.
2. It *looks for candidate instances* for each of the entities detected in the previous stage. A *candidate instance* for an entity is any entry in the knowledge base that is considered to represent a potential meaning of the entity. For example, the instance described in Listing 1 can be considered as a candidate for the entity *Kunduz*. In order to carry out this process, an extended knowledge base was built, that merges the instances in the official KBP KB with additional information and new entries obtained from a Wikipedia dump dated in year 2010. For each instance in the extended KB, a set of labels, obtained from Wikipedia redirections, anchors and disambiguation pages, is also stored. All the extended KB entries are indexed by Lucene[7], an information retrieval system. For each entity in the input, the text of the entity is used to query the Lucene index, obtaining the potential candidate instances in the extended KB.
3. The different *candidate instances for the same entity are ranked* to decide which is the best one. This process is carried out for all the entities, including the one in the query. In order to compute this ranking, WikiIdRank puts into action the semantic coherence principle, using information about instance co-occurrence estimated from the hyperlink structure of Wikipedia. For each entity, the output of the ranking process is a set of pairs {*candidate instance, score*}.
4. Taking into account the scores provided by the previous step, a configured threshold and the candidates' origin in the extended KB (reference KBP KB or Wikipedia) the algorithm *selects a candidate or NIL* as answer for the entity in the initial KBP query.

These four different stages are implemented in four different components: *Entity Finder*, *Instance Finder*, *Instance Ranker* and *Instance Selector* respectively. Figure 1 shows the architecture of the system, including all its components and the information sources they use. The next sections describe with more detail the functionality provided by each component and the information stored in each source.

3.1. **Entity finder.** The entity finder component uses state-of-the-art natural language processing (NLP) tools to find occurrences of named entities (persons, locations, organizations) in the query context document. The implemented system uses a combination of three free, open source tools to carry out this task: University of Sheffield's GATE [16], Stanford's Named Entity Recognizer (NER) [17] and University of Illinois at Urbana-Champaign's LbjNerTagger [18]. The entity finder processes the input text with all these named entity recognizers, and obtains the list of the entities detected by each one. Then,

[7]http://lucene.apache.org/ (11/Jul/2011)

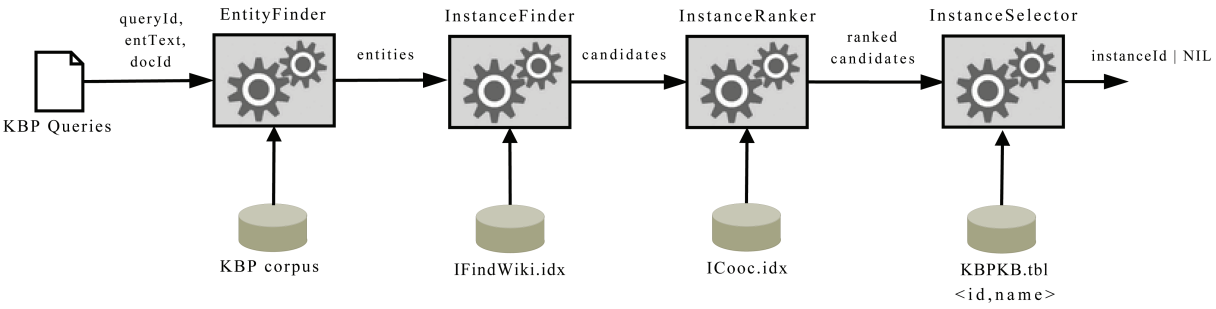

FIGURE 1. Document processing pipeline architecture

the component merges their results, selecting the entities detected by at least two recognizers. The entity in the initial query is also explicitly added if it is not detected by the NLP tools.

While most of the documents included in the KBP corpus are relatively small (few kilobytes) there are some outliers that are specially big. When analyzing these documents with the entity finder, many entities are detected[8], which may result in an overload for the next components of the WikiIdRank processing pipeline. To avoid such potential overload, the entity finder component filters out some entities so that at most 100 are handed to the next components. In order to select the entities to be removed, the filter takes into account the relative position of these entities with respect to the first occurrence in the document of the query text, keeping only those that appear nearer to it. The result of this filtering process is an entity set (*entities* in Figure 1), which is provided as input to the next component in the processing pipeline. Note that this set cannot be empty, because at least the entity in the query should be included in the entity finder results.

3.2. **Instance finder.** The main goal of this component is to obtain a set of candidate instances from the knowledge base for each entity detected by the entity finder. Basically, this process is carried out by matching the text of the entity with a set of names or labels associated to the instances in the knowledge base.

This component should be designed to minimize the possibility of leaving the instance that correctly represents the meaning of a certain entity out of the set of candidates for that entity. Due to this, it is important to include as many instances as possible in the knowledge base and to provide as many alternative names and labels as possible for each of these instances.

In accordance with this requirement, an extended knowledge base was built by merging the official KBP KB with information obtained from a Wikipedia dump downloaded from the English Wikipedia website[9] (the one dated 01/30/2010, to be more precise). In the extended KB, each instance represents a Wikipedia page in the dump (ignoring disambiguation pages and pages with special namespaces[10], and handling redirections as a sole entry). This results in a total of 9.604.232 entries, which cover all the ones in the original KBP KB, plus many new additions.

The extended knowledge base was later indexed in Lucene (*IFindWiki.idx* in Figure 1) to allow searching for instances. The fields stored in the index for each entry can be classified into two groups: Those that contain labels for the instance, and those that contain additional information about the instance (like its identifier). The fields in the first group are the ones used when looking for the candidates of a certain entity, by

[8]In one case, more than 2500 named entities were found on a single document.
[9]http://en.wikipedia.org/wiki/Wikipedia:Database_download (11/Jul/2011)
[10]http://en.wikipedia.org/wiki/Wikipedia:Namespace (11/Jul/2011)

matching the labels in the index with the text of the entity. This matching process may be an exact matching (that is, the text of the entity should be the same as the one in the label) or a free-text matching (where the words in the entity text are searched within the text of the labels). Due to this, the label fields are further divided into two groups: those that store the labels as keywords, used for exact matching, and those that store the same labels, but parsed to be used in free-text searching[11].

More in detail, the following fields are available for each entry in the Lucene index:

- Non-parsed fields, used for exact matching queries:
  - **source** The name of the Wikipedia page.
  - **anchors**, **redirects** and **disambiguation** These fields contain, respectively, the text of the link anchors, names of redirect pages, and names of disambiguation pages that point to the Wikipedia page of the index entry. All these strings can be considered as potential alternate labels for the topic represented by the entry.
- Parsed fields for non-exact matching queries:
  - **source-parsed**, **anchors-parsed**, **redirects-parsed** and **disambiguation-parsed** These fields contain the same information as the non-parsed ones described above. However, on the contrary to them, these fields are processed word by word. Therefore, it is not necessary to exactly match the text of the query: the relevance of an entry depends on the occurrences of the words in the query inside these fields.
- Fields containing additional information:
  - **identifiers** The associated identifier in the official KBP KB or an automatically generated one if the Wikipedia page is not included in the official KB. In order to define the mapping between the Wikipedia pages and the KBP KB entries, the names of the pages are compared with the names of the official KB entries. If an exact match is found, a mapping is defined. One problem that we found with this approach was due to redirections: in our system we treat pages with redirections as alternative representations of the same entry. As the Wikipedia version that we used (dated 01/30/2010) is different to the one the official KB was built from (dated October 2008), it happens that some redirection pages have changed, and Wikipedia pages that originally were different (and thus have different entries in the official KBP KB) are now redirected (and thus have a single entry in our index). The consequence is that a single entry in our index may have two or more different official KB identifiers. An example of this situation is the *Angikuni Lake*, which appears in the official KB as two separate entries *Angikuni Lake* and *Lake Angikune*, while in the version of Wikipedia we used they represent the same entry (*Lake Angikune* is just a redirection to *Angikuni Lake*). In these cases, the identifiers of both entries in the KBP KB are stored in this field.
  - **pagerank** Given the graph structure of the 2010 Wikipedia dump, the PageRank [19] value of each page was computed and stored in the index for later usage.

In order to find the most adequate candidate instances for each entity in the input, the instance finder component queries the Lucene index by using the entity text as query. As Lucene provides many query capabilities[12], several ways of using the text of the entity are possible.

[11]From an implementation point of view, the main difference between these two types of fields is the particular Lucene Analyzer used to process the text in the field: in one case it would be the KeywordAnalyzer and in the other the StandardAnalyzer. See Lucene documentation for details: http://lucene.apache.org/java/docs/index.html (11/Jul/2011)

[12]More information on Lucene query capabilities and operators may be found in http://lucene.apache.org/java/3_0_1/queryparsersyntax.html (11/Jul/2011)

In practice, the WikiIdRank approach relies on a three-stage query mechanism. The first stage looks into the non-parsed index fields with a query among quotes, looking for exact matches in these fields. When exact matches are not found, the second stage is run, which carries out a fuzzy search (looking for similar text using the Levenshtein metrics [20]) on the non-parsed fields as well as a quoted query in the parsed fields. In case this second stage does not find any candidate, the third stage is carried out. It looks into the parsed fields with and unquoted query and using the Lucene proximity operator, that matches the labels only in the case they contain the words in the query within a specific distance away.

As can be seen, each stage uses a query with more relaxed conditions than the previous one (and, thus, with lower precision but higher recall) and is executed only when the previous stage does not produce any results. Table 1 summarizes the aforementioned query stages.

TABLE 1. Queries run at each stage of the instance finder component

Stage	**Fields queried**	**Query conditions**
1	source, anchors, redirects, disambiguation	Quoted query
2	source, anchors, redirects, disambiguation	Lucene's FuzzySearch (similarity threshold 0.8)
	source-parsed, anchors-parsed, redirects-parsed, disambiguation-parsed	Quoted query
3	source-parsed, anchors-parsed, redirects-parsed, disambiguation-parsed	Unquoted query and Lucene proximity operator (proximal ≤ 2 words away)

This three-stage query process to the extended knowledge base index is repeated for each entity provided by the entity finder. Its results consist of a set of pairs {*candidate instance, Lucene score*} for each entity. Note that, for some entities, the set may be empty, if no candidate instance is found when searching the index.

As in the case of the entity finder (see Section 3.1), once these results have been obtained, they are filtered. The goal of this filter is to avoid potential overloads by limiting the maximum number of candidate instances per entity to be handed to the following component of the processing pipeline.

To select the candidates that survive the filtering process, the algorithm uses the following criteria:

- The target entity (the one in the KBP query) should keep more candidates than the rest of the entities (context entities), to minimize the possibility of filtering out the correct instance to be answered.
- The number of candidates for the target entity should depend on the concrete query stage where they were found. The rationale behind this criterion is that first stages are based on more restrictive queries and, due to this, it is expected that they produce more reliable results. Thus, depending on the specific stage, different thresholds for the maximum number of candidates are set (lesser when results are poorer, to avoid introducing too much noise in the system).
- When selecting the final candidates, both the scores provided by Lucene and the PageRank values stored in the Lucene index are taken into account.

In accordance with these requirements, the final candidate filtering process consists of the following three stages:

1. A weight is computed for each candidate, by linearly combining its Lucene score and the PageRank value stored in the Lucene index. That is, if M is the set of entities provided by the entity finder and $C(m)$ represents the set of candidate instances for a certain entity $m \in M$, the weight W_i of a candidate $I_i \in C(m)$ is

$$W_i = \alpha_L S_i + \alpha_{PR} PR_i \tag{1}$$

   where
   - S_i is the Lucene score for I_i, normalized so that $S_i \in [0, 1]$ and $\sum_{i=1}^{|C(m)|} S_i = 1$.
   - PR_i is the PageRank value for I_i, normalized so that $PR_i \in [0, 1]$ and $\sum_{i=1}^{|C(m)|} P R_i = 1$.
   - α_L and α_{PR} are configuration parameters, chosen so that $\alpha_L + \alpha_{PR} = 1$, $\alpha_L \in [0, 1]$ and $\alpha_{PR} \in [0, 1]$.
2. All the candidates for the same entity m (all $I_i \in C(m)$) are ranked using the previously computed weights W_i.
3. Candidates are filtered, keeping only the top-K of the ranking. The value of K depends on the entity and the query stage where its candidates were found. The target entity keeps 200 candidates in case they have been obtained in the first query stage, or only 30 if they have been obtained in the second or third stages. The context entities retain only a maximum of 15 candidates, independently of the query stage where they were found. Though a more restrictive approach taking the query stage into account is also possible for these entities (for instance, filtering out those whose candidates are not found in the first query stage), it was not implemented to avoid limiting too much the context information.

The final results of the instance finder component include, for each entity $m \in M$, a (potentially empty) list of pairs {*candidate instance, weight*} (that is, $\{I_i, W_i\}$, $\forall I_i \in C(m)$). The lists for all the entities (element *candidates* in Figure 1) are provided to the next component in the pipeline: the *Instance Ranker*.

3.3. **Instance ranker.** The main goal of the instance ranker component is to rank the set of candidate instances provided by the instance finder for each entity. This rank is later used to choose a single instance (or *NIL*) for the target entity.

It has to be noted that, as indicated in the previous section, the candidate instance list for a certain entity may be empty. If this is the case for the target entity, the algorithm will assign *NIL* as answer of the KBP query, without running the instance ranker. However, in the rest of this section it will be assumed that this is not the case.

The algorithm implemented in the instance ranker is based on the IdentityRank algorithm described in [14]. Basically, the principle that inspires the algorithm is the *semantic coherence* principle: as instances typically co-occur in documents with other related instances (for example, *Obama* and *USA* or *Pau Gasol* and *Los Angeles Lakers*), the occurrence of a certain instance gives information about the occurrence of other instances. There is, however, a difficulty with this approach: in order to use instance co-occurrence information for disambiguation, the actual instances that appear in the query context document need to be known. However, the instance ranker only knows the entities that are mentioned in the document and, for each of them, a list of candidate instances that can potentially represent the meaning of the entity. Moreover, due to the semantic coherence principle stated above, the occurrence of a certain instance depends on the occurrence of other instances in the context document. There is, therefore, a recursive problem: the decision about the linkage entity/instance (or entity/*NIL*) for a certain entity depends on the decisions taken for other entities and vice versa. This recursive problem may be

addressed by defining the linkages not only for the target entity but for all the entities at the same time.

Fortunately, as indicated in [14], the difficulty described above can be addressed by a PageRank-like algorithm. In PageRank, which is targeted at ranking web pages, *a page has high rank if the sum of the ranks of the pages that link to it is high* [19]. In the WikiIdRank case, the goal is to rank the possible identities (candidate instances) associated with a certain entity in the document. So, paraphrasing the sentence above, it can be said that *an instance has a high rank if the sum of the ranks in the document of the instances that typically co-occur with it is high.*

As reported in [14], this principle may be represented mathematically by a recursive matrix equation. Let C represent the set of candidate instances for all the entities in the document:

$$C = \bigcup_{\forall m \in M} C(m) \tag{2}$$

It might happen that the same instance is candidate for several entities, that is, it might appear in different $C(m)$. However, each instance is only included once in C.

Let $N = |C|$ and I_i, I_j any two instances included in C. Taking this into account, as it is explained in [14], the aforementioned principle can be mathematically modeled by a set of equations:

$$R_i = (1-\alpha) \sum_{j=1}^{N} a_{ij} R_j + \alpha E_i \quad i = 1 \ldots N \tag{3}$$

These equations may be represented in recursive, matrix form as follows:

$$R = (1-\alpha) AR + \alpha E \tag{4}$$

where

- $R \in \mathbb{R}^N$ is a vector that represents the rank of the candidates in the context document. The element R_i represents the rank of a specific candidate instance I_i.
- $A \in \mathbb{M}_{NxN}$ is a matrix where $a_{ij} \in \mathbb{R}$ represents the strength of the relationship that I_j has with I_i, that is, the proportional part of the I_j rank which is given to I_i. Those coefficients are computed by using instance co-occurrence information, as will be described later.
- $E \in \mathbb{R}^N$ is a vector that corresponds to a source of rank. Each component E_i can be used to adjust the rank of a certain instance I_i (give an *a priori* weight to certain candidates).
- α is a configuration parameter so that $\alpha \in [0, 1]$.

As indicated in [19], the matrix Equation (4) can be solved using numerical methods, like the *power method*. A description of the power method can be found at [21].

In order to estimate the co-occurrences between instances, WikiIdRank uses the Wikipedia link structure. In particular, two candidate instances are considered to co-occur when Wikipedia pages that link to both of them exist. For instance, in the Wikipedia dump used by the system, the instance referring to *Italy* is considered to co-occur with the instance for *European Union* because the Wikipedia page for *Rome* points to both the pages of Italy and the European Union. Additionally, two instances are also considered to co-occur when a direct link exists between the Wikipedia pages of the candidate instances (at least in one of the directions). For instance according to this principle, *Barack Obama* and *Columbia University* are also considered to co-occur, because there is a direct link from the Wikipedia page for Obama to the page of the Columbia University.

The co-occurrence information that is needed by the instance ranker component is stored in a Lucene index, represented as *ICooc.idx* in Figure 1. The entries stored into

the index, each of them representing a Wikipedia page (ignoring disambiguation pages and pages with special namespaces – User, Talk, File, etc. – and resolving redirections) consists of the following fields:

- **source** The name of the Wikipedia page as a keyword, that is, not parsed.
- **destination** This field represents the concatenated text of the different names of the Wikipedia pages that are pointed by the links in the source page (outgoing links). Each of the names is also treated as a keyword.

Taking into account the two different contributions to the co-occurrence information described above, Equation (4) has been replaced by:

$$R = (k_L A_L + k_C A_C)R + k_E E \tag{5}$$

Again, let $N = |C|$, and $I_i, I_j \in C$:

- R has the same meaning as in Equation (4)
- $A_C \in \mathbb{M}_{NxN}$ is a matrix. The value of each element a_{ij}^C in A_C, is computed using the information in the *ICooc.idx* index as follows:

$$a_{ij}^C = \begin{cases} 0.0 & i = j \\ |Q(d = I_i, d = I_j)|/|Q(d = I_j)| & i \neq j \end{cases} \tag{6}$$

The term $|Q(d = I_i, d = I_j)|$ represents the total number of documents returned by Lucene by querying *ICooc.idx* with a query that looks in the *destination* field for the quoted name of the instance I_j and for the quoted name of I_i (for example, *destination:"Madrid" AND destination:"Spain"*). Thus, the number of Wikipedia articles that include links to both pages is obtained. The term $|Q(d = I_j)|$ represents the total number of results returned by Lucene when querying the index with a query that looks in the *destination* field with the quoted name of the instance I_j, that is, the number of Wikipedia articles that contain links to I_j. Thus, the a_{ij}^C values can be interpreted as an estimation of $P(I_i/I_j)$. The A_C matrix is later normalized by dividing it by its norm one, so that $||A_C||_1 = 1$. Note that the strength of relations between instances is not required to be symmetric. That is, it is not always the case that $a_{ij}^C = a_{ji}^C$ and that $P(I_i/I_j) = P(I_j/I_i)$.

- $A_L \in \mathbb{M}_{N\times N}$ is the matrix that carries data about direct links in a Wikipedia page to other Wikipedia page (in contrast to A_C that is built based on the co-occurrence of links to two Wikipedia pages in a third one). Intuitively, it is clear that the instances that are mentioned in a Wikipedia page are related to the instance represented by such Wikipedia page. Thus, if a certain Wikipedia article links many times to another one, and the instance represented by the former article is likely to occur in the context document of the KBP query (that is, it has a high ranking value), then it is also likely that the pointed article and its associated instance (which is also a candidate) is also present.

  Specifically, the values of the A_L matrix, a_{ij}^L, are computed using the information in the *ICooc.idx* index according to the following equation:

$$a_{ij}^L = \begin{cases} 0.0 & \nexists I_j \to I_i \text{ or } i = j \\ score(Q(s = I_j, d = I_i)) & \exists I_j \to I_i \end{cases} \tag{7}$$

  where the $\to$ symbol is used to represent a link between two instances (link between the Wikipedia pages of the instances in the direction indicated by the arrow), and $score(Q(s = I_j, d = I_i))$ represents the Lucene score obtained by querying *ICooc.idx* with a query that looks in the *source* field for the quoted name of I_j, and in the field *destination* with the quoted name of the instance I_i. The A_L matrix is later

normalized by dividing it by its norm one, so that $||A_L||_1 = 1$. Again, it is not always the case that $a^L_{ij} = a^L_{ji}$, and thus the A_L matrix is not symmetric.

- $E \in \mathbb{R}^N$ represents a source of rank, as in Equation (4). The value of the element E_i is the weight (W_i) for the instance I_i, that the instance finder component provides to the instance ranker. As the same instance I_i might be candidate for several different entities with several different weights, the maximum weight across all entities is assigned. E is normalized so that $||E||_1 = 1$.
- k_L, k_C and k_E are configuration parameters so that $k_L, k_C, k_E \in [0, 1]$ and $k_L + k_C + k_E = 1$.

Using the power method, the matrix Equation (5) is solved, obtaining the vector R. Once R is computed, the ranking of each candidate instance in the context of the document being analyzed is known: the weight of the instance I_i is simply the component i of the vector R. For each entity $m \in M$ in the document, the algorithm returns a list with all the pairs {*candidate instance, weight*} for the entity (that is, $\{I_i, R_i\}$, $\forall I_i \in C(m) \subseteq C$), ordered by weight. The list obtained for the entity in the KBP query (element *ranked candidates* in Figure 1) is provided to the next component in the pipeline: the *Instance Selector*.

3.4. **Instance selector.** The main goal of the KBP entity linking system is to define a single assignment entity-instance (or entity-*NIL*) for the entity in the KBP query. The final decision process that selects the best answer (top ranked candidate instance as provided by the instance ranker or *NIL*) is implemented by the instance selector component. In particular, the following activities are carried out:

1. Selecting the candidate instance to be assigned to the target entity. When several candidates are available, the weights for the two candidates with highest rank are compared by computing the coefficient:

$$Plausibility = \frac{TopInstanceWeight}{SecondInstanceWeight} \tag{8}$$

   In case the weight of the second instance is similar to the one of the first instance, that is, when the coefficient is below a configurable threshold, NIL is returned instead of the top ranked instance. The rationale behind the thresholding mechanism is that when two candidates get a similar weight from the ranker, there is a less clear distinction between them, that is, the context may be proper for the occurrence of any of them and, thus, there is less confidence about the result. This mechanism is also useful to break ties between candidates, and to avoid giving too much credibility to small differences between instance weights due to numerical imprecision. Empirical tests on the human assessments provided by the KBP organization showed that, using the threshold, a slight gain in accuracy (around 3% or 4%) is obtained. A thresholding mechanism for instance/*NIL* selection is also used in other approaches like [22].

   In practice, two different threshold values are defined, selecting between them depending on how the instance finder has found the candidates. In case the instance candidates for the target entity have been obtained by the instance finder in the first query stage, the threshold (named σ_L) is lower (thus the L) than in case the candidates have been obtained in second or third query stages (threshold named σ_H, H for high), as these results are considered less reliable.
2. Addressing the problem of multiple identifier values that was referred in Section 3.2 when describing the *identifiers* index field. In case several official KB identifiers are available for the same candidate, the name in the official KB associated to each

identifier is looked up in a specific table (*KBPKB.tbl* in Figure 1) which contains the *id* and *name* of all the entries in the official KBP KB. The Levenshtein distance [20] is then used to compare the different names with the text of the target entity in the KBP query. The official KB entry whose name is the most similar to the text of the query is finally assigned.

3. As the system is using an extended KB, which contains more instances than the official KBP KB, in case the instance selected as the best candidate is not included in the official KB, *NIL* is answered.

4. **Evaluation.** In order to validate the approach described in Section 3, it was implemented and evaluated in the KBP 2010 scenario. In this section the results obtained by the system in the entity-linking task are shown (Section 4.1). In addition, tests were carried out with the core functionality of the instance ranker disabled, i.e., with no instance co-occurrence information, in order to measure its actual contribution in the accuracy of the system. The results of this second experiment are reported in Section 4.2. Finally, Section 4.3 shows a performance analysis of the algorithm, designed to measure the response time of WikiIdRank.

4.1. **Entity linking task results.** According to the KBP rules, each team taking part in the competition might submit at most three different runs of their systems for evaluation. In the case of WikiIdRank, three runs were submitted. The differences between these runs were not in the algorithm, which was in all the cases the one described in Section 3, but in the configuration parameters. The configurations for each submitted run are shown in Table 2.

TABLE 2. Configuration of the WikiIdRank system for the three submitted runs

Run	**Instance Finder**		**Instance Ranker**			**Instance Selector**	
	α_L	α_{PR}	k_L	k_C	k_E	σ_L	σ_H
Run1	0.8	0.2	0.55	0.25	0.2	1.05	1.5
Run2	0.8	0.2	0.55	0.25	0.2	1.2	2.0
Run3	0.8	0.2	0.4	0.4	0.2	1.2	2.0

In order to establish appropriate values for the different configuration parameters, (α_L, α_{PR}, k_L, k_C, k_E, σ_L, σ_H) the system was run and tuned using the information given by the human assessments provided by the KBP organizers. Though there was no guarantee that these values would be also optimum for other sets of queries, after having analyzed the results obtained with the 2010 query set, we found those values to perform well also for that query set.

Results achieved by these runs, as reported by the organizing committee, are summarized in Table 3. The first column, *Run*, identifies the system run, whose configuration is described in Table 2. The column *All* indicates the accuracy (between 0 and 1) of a certain run in all the 2250 evaluation queries. The accuracy for the 1020 evaluation queries whose manual answer is an instance, is reported in the column *Non-NIL*. Finally, the last column, *NIL*, indicates the accuracy of the system for those evaluation queries (1230) where the correct manual answer was *NIL*.

4.2. **Analyzing the contribution of instance co-occurrence information.** The main novelty of the system presented in this paper with respect to other approaches to the entity linking problem is the use of instance co-occurrence information for disambiguation purposes. The objective of this section is to analyze its actual influence in the

TABLE 3. Results obtained by the different runs (accuracies between 0 and 1)

Run	All	Non-NIL	NIL
Run1	**0.7698**	0.6647	0.8569
Run2	0.7636	0.6098	0.8911
Run3	0.7596	0.6049	0.8878

system accuracy. In order to do that, the results achieved by the best run submitted to KBP were compared with two baseline approaches that only take into account the ranking as provided by the instance finder component. This ranking is based on information retrieval techniques (Lucene score, PageRank). Thus, it does not take advantage of the instance co-occurrence information.

Apart from that, the algorithm was also run with two additional configurations to analyze the impact in the final results of the different instance co-occurrence information sources (matrices A_L and A_C, see Section 3.3). The configurations compared were (see Table 4):

- **Lucene**: core algorithm of the instance ranker disabled (i.e., returning candidates in the same order as the instance finder module), with results ranked with 100% the score given by Lucene (without any PageRank contribution).
- **Lucene-pagerank**: core algorithm of the instance ranker disabled, with results ranked with 80% the score given by Lucene and 20% the value of PageRank computed for each candidate (i.e., same configuration for instance finder as in Run1, but without instance co-occurrence contribution).
- A_C **only**: all the components enabled, but the instance ranker component uses only the information about instance co-occurrence provided by the A_C matrix.
- A_L **only**: all the components enabled, but the instance ranker component uses only the information about instance co-occurrence provided by the A_L matrix.
- **Best run**: all the components enabled, with the configuration of the best run (Run1) as submitted to KBP 2010.

TABLE 4. Configuration parameters for the five runs used to analyze the impact of instance co-occurrence information in the final results of the system. The "Lucene" and "Lucene-pagerank" configurations have the core algorithm of the instance ranker disabled, i.e., $k_L = k_C = 0$ and act as a reference baseline.

	Run	Instance Finder		Instance Ranker			Instance Selector	
		α_L	α_{PR}	k_L	k_C	k_E	σ_L	σ_H
Baseline runs	Lucene	1.0	0.0	0.0	0.0	1.0	1.05	1.5
	Lucene-pagerank	0.8	0.2	0.0	0.0	1.0	1.05	1.5
Runs using co-occurrence information	A_C only	0.8	0.2	0.0	1.0	0.0	1.05	1.5
	A_L only	0.8	0.2	1.0	0.0	0.0	1.05	1.5
	Best run	0.8	0.2	0.55	0.25	0.2	1.05	1.5

Table 5 summarizes the results for those five configurations, using the same nomenclature as Table 3. According to the results shown in this table, the core algorithm of the instance ranker has a significant positive effect in the results of the system for the *non-NIL* queries. Overall accuracy increases from 0.6667 to 0.7698, a gain of 15.46%, when the *Best run* is compared to the best of the baseline runs (*Lucene-pagerank*). The accuracy

for *NIL* answers is better when running only Lucene. However, this is just an effect of the system returning more *NIL* answers, which reduces the success rate for *non-NIL* queries.

TABLE 5. Overall results of the system with five different configurations

	Run	All	Non-NIL	NIL
Baseline runs	Lucene	0.6387	0.2912	0.9268
	Lucene-pagerank	0.6667	0.3725	0.9106
Runs using co-occurrence information	A_C only	0.7458	0.6294	0.8423
	A_L only	0.7556	0.6284	0.8610
	Best run	**0.7698**	0.6647	0.8569

Table 5 also shows that, even considered only by themselves, both instance co-occurrence contributions outperform the *Lucene-pagerank* baseline, which stresses the previous conclusion. It can also be seen that there are minor differences between the A_C *only* and A_L *only* cases, mainly due to the later performing slightly better in the *NIL* queries, though there are no significant differences between them in the *non-NIL* case. Finally, it has to be noted that the *Best run*, that takes into account all the contributions, provides better results than all the individual components, which confirms that the combination of the different features is beneficial.

To further validate the previous conclusions, the accuracy-at-k of the five different runs was also computed. The accuracy-at-k of a run is defined, for a given integer k, as the fraction of queries with a *non-NIL* manual answer in which that answer is in the first k results given by the ranker. For example, accuracy-at-2 represents the ratio of queries with a *non-NIL* correct answer for which the answer is the first or second result produced by the instance ranking process. Figure 2 compares the value of accuracy-at-k for the five runs. Figure 3 shows the same results as Figure 2, but with a zoom along the X and Y axes to provide a clearer view of the differences between the five configurations tested.

The accuracy-at-k values shown in Figures 2 and 3 confirm also the significant effect of the instance co-occurrence information: it increases the probability of the correct answer being at the top of the ranked list of candidates.

Figure 2 shows also that the five configurations converge to an accuracy-at-k of 0.8569 when $k = 200$. This result indicates that in 14.31% of the queries with *non-NIL* manual answer (that is, in 146 out of 1020 queries) the instance finder did not return the correct answer within the list of candidates. Thus, the *recall* of the instance finder, defined as the percentage of queries with *non-NIL* manual answer in which the only relevant result (the correct answer to the query) appears in the list of candidates, is 85.69%.

Note that, when the instance finder is not able to provide the correct answer among the candidate instances, the instance ranker never returns that correct answer. Thus, the recall of the instance finder has a very significant impact in the final quality of the system. Due to this, improving this recall may constitute an interesting approach to increase the system accuracy. In this sense, a potential improvement that needs to be explored in future versions of WikiIdRank is using a query expansion mechanism, as is suggested by some related works [23]. The idea is to extend the query used by the instance finder component with additional context information obtained from the document, with the aim of increasing the number of queries where the correct answer is included among the candidate instances.

4.3. **Performance analysis.** Apart from the accuracy of the system, another aspect that needs consideration when evaluating the quality of an entity linking system is its performance. In particular, a relevant parameter to be measured is the response time of

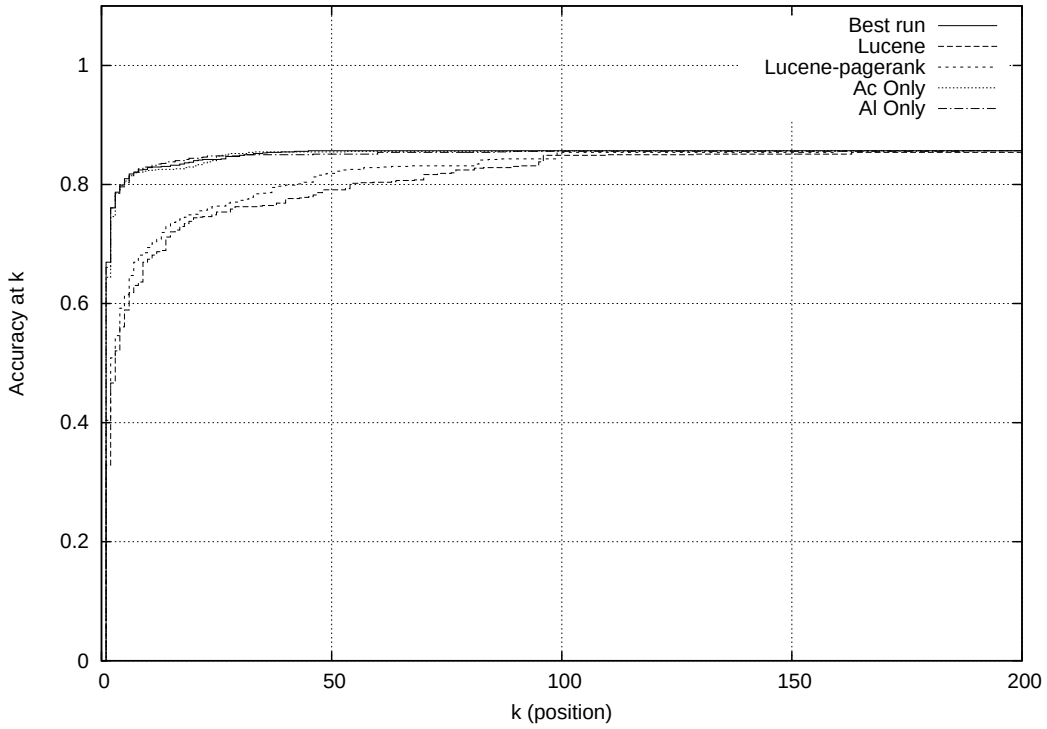

FIGURE 2. Accuracy-at-k for the system with five different configurations

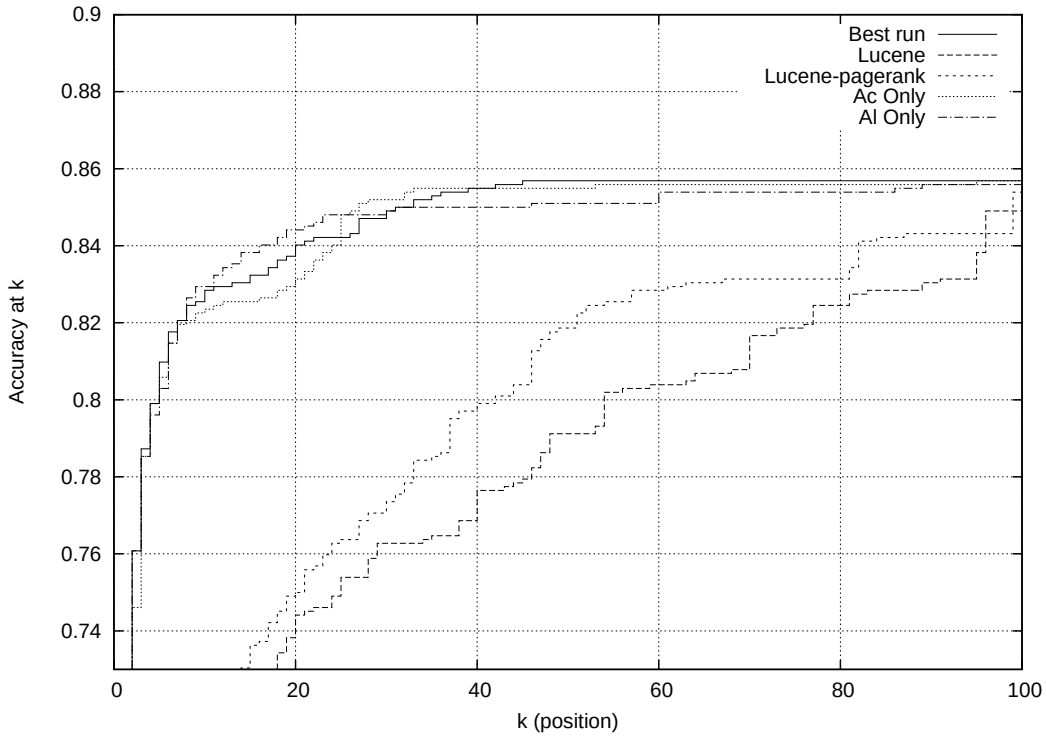

FIGURE 3. Accuracy-at-k for the system with five different configurations (zoomed in)

the algorithm, that is, the average time that the system needs to process a document. This parameter is very relevant, because in case an entity linking system were very accurate,

but its response time were too long (for instance, in case it took hours to process a single document) the practical usefulness of the system would be limited.

A performance analysis of WikiIdRank is shown in this section. This analysis includes not only an evaluation of the total response time of the algorithm, but also information about how the different processing stages of the algorithm (EntityFinder, InstanceFinder, InstanceRanker) affect the global system performance.

In order to carry out this performance analysis, the WikiIdRank implementation was run on the KBP 2010 corpus in a Debian Linux 2.6.32 machine with 8 GB RAM, 1 TB SATA-II hard disk and an Intel(R) Core(TM) i7 860 at 2.80GHz processor. The total response time of the algorithm, as well as the response time of the main processing stages, was measured for each document in the corpus.

TABLE 6. Median, mean and maximum response time for each processing stage and the complete system (in seconds)

Time	**Median (sec.)**	**Mean (sec.)**	**Max (sec.)**
t_{EF}	0.48	0.68	14.32
t_{IF}	18.84	30.23	304.08
t_{IR}	2.70	8.16	287.63
t_{TOT}	25.85	39.07	426.80

Table 6 provides a statistical summary of the measured response times. In particular, table columns show the median, mean and maximum response time (in seconds) measured for each processing stage (EntityFinder, t_{EF}, InstanceFinder, t_{IF}, and InstanceRanker, t_{IR}) as well as the total response time of the system (t_{TOT}).

As can be seen in Table 6, the InstanceFinder and InstanceRanker stages are the two most computationally expensive. In order to explain this result, it has to be taken into account that both the InstanceFinder and InstanceRanker components carry out multiple queries to Lucene indexes. As these indexes are stored in hard disk, this process requires extensive disk access, which has a negative impact in the response time. In particular, as is also shown in Table 6, this impact specially affects the InstanceFinder, due to the fact that the three-stage query process used by this component is more complex than the one used by the InstanceRanker. Thus, in order to reduce response time, loading the Lucene indexes into RAM memory or storing them into an SSD (Solid State Drive) disk could be potential ways to be considered in the future.

Looking at Table 6 it can also be seen that the maximum total processing time of a single document in the KBP 2010 corpus was around 427 seconds, which, though not unreasonable, would prevent the usage of the system in restrictive real time environments. However, the table also shows that the median and mean total response time were much shorter (around 26 and 39 seconds, respectively), which indicates that the relatively long processing time may be due to outliers.

In order to give more precise information, the histograms of t_{EF}, t_{IF}, t_{IR} and t_{TOT} are shown in Figure 4. As can be seen, they follow the pattern of a long tail distribution, which explains the difference between the mean/median times and the maximum times described in Table 6. Note that, according to the information displayed in Figure 4, the total processing time, t_{TOT}, for most of the processed documents lies below the 90 second threshold. In particular, these documents constitute the 91.29% (2054 out of 2250) of the total number of documents, which can be considered a positive result: though some documents took several minutes to be processed by the WikiIdRank algorithm, the vast majority required less than one minute and a half.

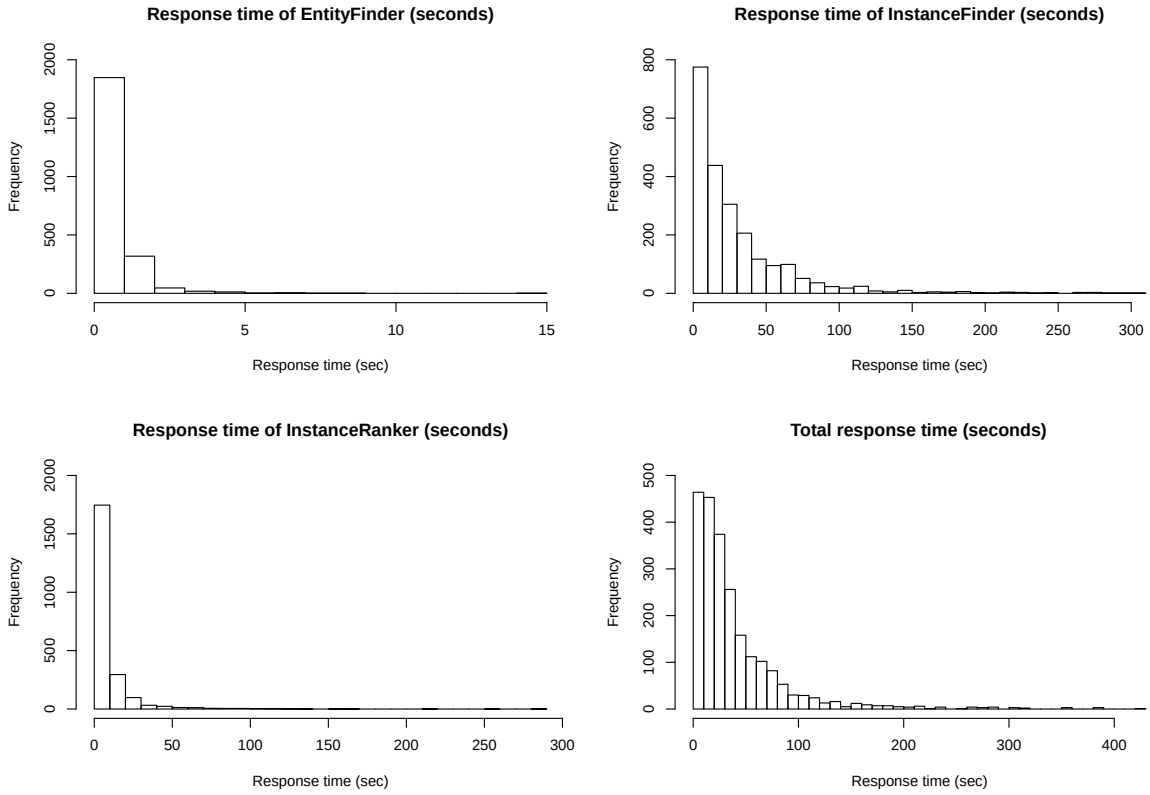

FIGURE 4. Histograms showing the response time of the EntityFinder (t_{EF}, top left), InstanceFinder (t_{IF}, top right), InstanceRanker (t_{IR}, bottom left) and total of the full system (t_{TOT}, bottom right)

5. **Related Work.** Due to its prominent role in the process of automatic knowledge base population, the entity linking problem has been addressed by several works in recent times. In this section, a comparative analysis of these approaches in relation to the work described in this paper is presented.

This analysis is structured into two subsections. Firstly, Section 5.1 reports on the qualitative aspects of the comparison, centered on the techniques used to address the entity linking task. Secondly, Section 5.2 addresses the comparative analysis from the quantitative point of view, centered on the accuracy of the system.

5.1. **Qualitative comparison to related work.** From a qualitative point of view, the different works that address the entity linking task may be classified into several groups taking into account the techniques used to approach the problem:

- A first group of works, including for instance the ones in [24, 25, 26], rely on building a vector space model to address the task. In this approach, the reference KB entries (or their corresponding Wikipedia articles) and the query context document are modeled as vectors. These vectors are later compared by computing the scalar product or other vector similarity metrics to decide which KB entry is the best answer for the query (or *NIL*). However, the works in this family differ in the features that are considered when modelling the documents and KB entries as vectors.
- A second group is composed of works relying on information retrieval techniques, like for instance [27, 28, 29, 30, 31]. In this case the reference KB (possibly extended with information from Wikipedia) is indexed in an information retrieval system (like

Lucene or Indri[13]). This index is later searched with a query built from the target entity text and/or text from the context document. The results obtained are later used to decide which instance (if any) is the best for the target entity.

- Machine learning techniques are also commonly used to address the entity linking task. Some works use Support Vector Machines (SVMs) [32, 33, 34]. Some are based on agglomerative clustering techniques [35]. Others rely on regression-based classifiers [36].
- Using the information provided by the graph of Wikipedia links is also explored in some works, like [37] or [38]. In [37] the *GRAPH* algorithm is described. Basically, *GRAPH* is similar to WikiIdRank in the sense that it builds a graph where nodes represent the candidate instances (including *NIL* among them) for the target entity as well as candidate instances for up to other 5 named entities in the context document. The algorithm places edges connecting the nodes according to connectivity information obtained from the KB (for example, two nodes are connected when they are one or two clicks away in Wikipedia). Nodes are assigned initial weights and an error back-propagation algorithm is used to assign weights also to edges. Finally, once the graph is built, a random walk is carried out on this graph, to obtain the final node weights. The node with the biggest weight is selected. In [38] the authors describe a *graph based reweighing* technique. For each named entity considered in the context document, its candidates are found using information retrieval techniques. A set containing the in-links to the top ranked candidate instance of each entity is built. Then a score is computed for each candidate of each entity using the intersection of the in-links to the candidate with the in-links in the previously built set. This score is used in the candidate ranking process together with other features.
- Additional related works are based on the usage of heuristics or custom similarity metrics. Examples of this type include [22] which uses a heuristic based on detecting the named in the context document and counting how many of them are mentioned in the facts included in the KB entry of each candidate; [29], which relies on a similarity metrics based on comparing the name and type of each candidate instance with the entity query text and type; [31], which makes use of heuristics rules; [39] which uses a similarity metrics based on comparing the noun phrases found in the query context document with the ones found in the text of the KB entries of the different candidate instances; [40] that uses a technique based on look-up on a cascade of specifically built dictionaries or [23] that uses the normalized Google distance [41]. This last work is specially relevant in the context of this paper, because the authors propose the usage of *concepts*, instead of text terms, as context information for linking. These concepts are just other Wikipedia articles (that is, instances) referenced in the context document. The concepts are detected by selecting low ambiguity terms in the document that are easy to link to Wikipedia. Once the concepts are detected, the similarity between the candidate instances and each of these concepts is computed using the normalized Google distance, and the average similarity with all the concepts of a candidate is used as one of the features for candidate ranking.
- Obviously, it is possible to design hybrid systems that rely on different families of techniques to address the task. Some examples are: [26], based both on a vector space model and custom similarity metrics; [29], which combines information retrieval techniques with an heuristic similarity metrics; and [31], which uses both information retrieval techniques and a rule-based heuristic.

[13]http://www.lemurproject.org/indri/ (11/Jul/2011)

According to the previously described classification, WikiIdRank may be considered a hybrid system that combines information retrieval techniques (features included in the E vector in Equation (5)) with specific Wikipedia graph analysis techniques (instance co-occurrence information, matrices A_L and A_C in Equation (5)).

Though some of the aforementioned related works use features based on the link structure of Wikipedia (like [38]) or propose the usage of other instances detected in the context document as features (for example, [23]), the idea of applying a PageRank-like algorithm to a network of co-occurrences of candidate instances to rank these candidates is, to the knowledge of the authors, novel to WikiIdRank.

In this sense, the most similar related work is the *GRAPH* algorithm described in [37]. Both WikiIdRank and *GRAPH* rank the candidate instances for the different entities in the context document using information about the relations (links) between these candidates. Thus, while most of the related works are centered on disambiguating named entities in a one-at-a-time basis, WikiIdRank and *GRAPH* have the potential advantage of co-disambiguating the different named entities that appear in the query context document all at the same time in a single run. However, both approaches differ in how this is done (random walk in *GRAPH*, PageRank-like algorithm in WikiIdRank). They also differ in the features used: though the direct linkage between candidate instances is considered in both systems, the co-occurrence in other Wikipedia pages (matrix A_C in WikiIdRank) is not taken into account in *GRAPH*. These differences in the mechanism used to address the task imply also differences in the results obtained by both approaches, as will be shown in the next section.

5.2. **Quantitative comparison with related work.** From the quantitative point of view, a first aspect that needs consideration is that some of the approaches described in Section 5.1 ([24, 25, 32, 35]) have been developed outside of the KBP context. Due to this, they use a different evaluation corpus to the one provided in TAC KBP. Furthermore, there are also differences in the evaluation metrics (F-measure is used instead of accuracy in [35]) and in whether the *NIL* case is taken into account or not (for instance, [24, 25, 35] do not consider that case). Due to this, the empirical results reported in these works are not directly comparable with the ones in this paper.

Taking this into account, the quantitative analysis of the related work will be centered in KBP 2010 approaches, which use the same corpus and evaluation metrics. In particular, due to the fact that WikiIdRank does not use the text of Wikipedia pages to carry out the linking process, the comparison will focus on the systems taking part in the *no-wikitext* sub-task (see Section 2), which imposes this restriction, thus making the comparison fair.

Table 7 summarizes the results achieved by the *no-wikitext* systems, comparing them to WikiIdRank. It has to be noted that, though the system in [40] did not take part in the *no-wikitext* sub-task, its authors indicate that it is compatible with that sub-task, and it has been, thus, included. The first column in the table is a reference to the system description. The second column provides the accuracy as reported by the KBP organizers. As can be seen, the WikiIdRank approach outperforms all but two of the systems, including the one described in [37]. However, taking into account that the system in [33] is a supervised one, the WikiIdRank approach is the second among the unsupervised.

Finally, it has to be pointed out that the work described in this paper extends the one initially reported to KBP 2010 [42], including a more detailed evaluation and comparison with related approaches, and continues former work of the authors described in [14, 43]. Compared with these two precedents, the algorithm presented in this paper is unsupervised, domain-independent and has been evaluated in a large corpus on the context of an international forum as the TAC KBP.

TABLE 7. Quantitative comparison with related approaches in the *no-wikitext* sub-task

Reference	Accuracy
[40]	0.80
[33]	0.7791
WikiIdRank	**0.7698**
[29]	0.7547
[36]	0.6978
[37]	0.6716
[28]	0.66
[22]	0.6458
[27]	0.4396

6. **Conclusions and Future Lines.** In this paper an automatic, unsupervised approach to the entity linking task of the Knowledge Base Population (KBP) track within the Text Analysis Conference (TAC) was presented. The proposed algorithm, named WikiIdRank, is inspired in the semantic coherence principle, which states that instances are frequently mentioned in documents together with (i.e., *co-occur* with) other semantically related instances. Following this intuition, the main novelty of WikiIdRank consists in applying a PageRank-like algorithm to a network of instance co-occurrences obtained from the link structure of Wikipedia to address the entity linking task.

The algorithm was implemented and evaluated in the context of KBP 2010, achieving positive results. In particular, when compared to the systems that took part in the *no-wikitext* sub-task of KBP 2010, WikiIdRank ranks as third (or as second, taking only unsupervised systems into account).

The *a posteriori* analysis of the system shows that the instance co-occurrence information has a significant positive impact on its accuracy. In particular, as has been shown in the evaluation section, the usage of instance co-occurrence information provides a gain of accuracy around 16% compared with a baseline approach relying on information retrieval techniques. Furthermore, a performance analysis of the algorithm was also carried out. This analysis indicates that the average processing time per document is in the order of seconds.

Despite the recognized positive impact of instance co-occurrence information in the entity linking process, the WikiIdRank system may be subject to further improvements. For instance, as shown in the evaluation section, the instance finder component has a relatively low recall, and is by itself responsible of the errors in 14.31% of the queries that have a *non-NIL* correct answer. This is an aspect that clearly needs consideration. Furthermore, another aspect that needs to be explored in the future is including in the context information not only candidate instances associated to named entities, but also potential links to Wikipedia pages about common topics (like *programming language* and *president*). The intuition is that adding these links may enrich the available context information and provide better results at the ranking stage, an aspect that obviously needs empirical testing. Combining the instance co-occurrence information with additional features used by related works, to test whether the combination is positive or not, may be another line worth exploring. Finally, adapting the algorithm to run on multilingual scenarios is also considered as a potential future work. This adaptation would benefit from the existence of versions of Wikipedia in different languages[14], as well as natural language processing tools for languages other than English.

[14]http://meta.wikimedia.org/wiki/List_of_Wikipedias (11/Jul/2011)

Acknowledgment. This work has been partially funded by the Spanish Ministry of Science and Innovation under contract TIN2008-05163, (*LEARN3: Hacia el aprendizaje en la 3ª fase*), and by the Community of Madrid under contract S2009/TIC-1650 (*eMadrid, Investigación y Desarrollo de Tecnologías para E-Learning en la Comunidad de Madrid*).

REFERENCES

[1] D. Bawden and L. Robinson, The dark side of information: Overload, anxiety and other paradoxes and pathologies, *Journal of Information Science*, vol.35, no.2, pp.180-191, 2009.

[2] T. Berners-Lee, J. Hendler and O. Lassila, The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities, *Scientific American*, vol.284, no.5, pp.34-43, 2001.

[3] L. Sánchez and N. Fernández, The semantic web: Fundamentals and a brief state-of-the-art, *The European Journal for the Informatics Professional*, vol.6, no.6, pp.5-11, 2005.

[4] N. Shadbolt, T. Berners-Lee and W. Hall, The semantic web revisited, *IEEE Intelligent Systems*, vol.21, no.3, pp.96-101, 2006.

[5] C. Bizer, T. Heath and T. Berners-Lee, Linked data – The story so far, *Int. J. Semantic Web Inf. Syst.*, vol.5, no.3, pp.122, 2009.

[6] A. Gómez-Pérez, M. Fernández-López and O. Corcho, *Ontological Engineering with Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web*, Springer, 2004.

[7] D. Vallet, M. Fernández and P. Castells, An ontology-based information retrieval model, *The Semantic Web: Research and Applications, Lecture Notes in Computer Science*, vol.3729, pp.455-470, 2005.

[8] M. Fernández, I. Cantador, V. López, D. Vallet, P. Castells and E. Motta, Semantically enhanced information retrieval: An ontology-based approach, *Web Semantics: Science, Services and Agents on the World Wide Web*, vol.9, no.4, pp.434-452, 2011.

[9] V. Alexiev, M. Breu, J. de Bruijn, D. Fensel, R. Lara and H. Lausen, *Information Integration with Ontologies: Experiences from an Industrial Showcase*, Wiley, 2005.

[10] D. M. Pisanelli, *Ontologies in Medicine*, IOS Press, 2004.

[11] N. Fernández, D. Fuentes, L. Sánchez and J. A. Fisteus, The NEWS ontology: Design and applications, *Expert Systems with Applications*, vol.37, no.12, pp.8694-8704, 2010.

[12] T. Z. Berardini, V. K. Khodiyar, R. C. Lovering and P. Talmud, The gene ontology in 2010: Extensions and refinements *Nucleic Acids Research*, vol.38, pp.331-335, 2010.

[13] B. G. Buchanan and D. C. Wilkins, *Readings in Knowledge Acquisition and Learning: Automating the Construction and Improvement of Expert Systems*, Morgan Kaufmann, 1993.

[14] N. Fernández-García, J. Blázquez-del-Toro, L. Sánchez-Fernández and A. Bernardi, IdentityRank: Named entity disambiguation in the context of the NEWS project, *Proc. of the 4th European Semantic Web Conference, Lecture Notes in Computer Science*, vol.4519, pp.640-654, 2007.

[15] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak and S. Hellmann, DBpedia – A crystallization point for the web of data, *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, vol.7, no.3, pp.154-165, 2009.

[16] H. Cunningham, D. Maynard, K. Bontcheva and V. Tablan, GATE: A framework and graphical development environment for robust NLP tools and applications, *Proc. of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.

[17] J. R. Finkel, T. Grenager and C. Manning, Incorporating non-local information into information extraction systems by gibbs sampling, *Proc. of the 43rd Annual Meeting of the Association for Computational Linguistics*, pp.363-370, 2005.

[18] L. Ratinov and D. Roth, Design challenges and misconceptions in named entity recognition, *Proc. of the 13th Conference on Computational Natural Language Learning*, 2009.

[19] L. Page, S. Brin, R. Motwani and T. Winograd, The PageRank citation ranking: Bringing order to the web, *Technical Report*, Stanford University, 1998.

[20] V. I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, *Soviet Physics Doklady*, vol.10, no.8, pp.707-710, 1966.

[21] J. H. Mathews and K. D. Fink, *Numerical Methods Using MATLAB*, 4th Edition, 2004.

[22] Y. Song, Z. He and H. Wang, ICL KBP approaches to knowledge base population at TAC 2010, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, NIST, Maryland, USA, 2010.

[23] J. Lehmann, S. Monahan, L. Nezda, A. Jung and Y. Shi, LCC approaches to knowledge base population at TAC 2010, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, NIST, Maryland, USA, 2010.
[24] S. Cucerzan, Large-scale named entity disambiguation based on Wikipedia data, *Proc. of EMNLP-CoNLL*, pp.708-716, 2007.
[25] H. T. Nguyen and T. H. Cao, Exploring Wikipedia and text features for named entity disambiguation, *Lecture Notes in Computer Science*, vol.5991, pp.11-20, 2010.
[26] Z. Chen, S. Tamang, A. Lee, X. Li, W. Lin, M. Snover, J. Artiles, M. Passantino and H. Ji, CUNY-BLENDER TAC-KBP2010 entity linking and slot filling system description, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[27] D. Nemeskey, G. Recski, A. Zséder and A. Kornai, BUDAPESTACAD at TAC 2010, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[28] V. Varma, P. Bysani, K. Reddy, V. B. Reddy, S. Kovelamudi, S. R. Vaddepally, R. Nanduri, N. K. Kumar, S. Gsk and P. Pingali, IIIT hyderabad in guided summarization and knowledge base population, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[29] S. Gottipati and J. J. Jiang, SMU-SIS at TAC 2010 – KBP track entity linking, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[30] J. Yu, O. Mujgond and R. Gaizauskas, The university of sheffield system at TAC KBP 2010, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[31] S. Gao, Y. Cai, S. Li, Z. Zhang, J. Guan, Y. Li, H. Zhang, W. Xu and J. Guo, PRIS at TAC2010 KBP track, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[32] R. Bunescu and M. Pasca, Using encyclopedic knowledge for named entity disambiguation, *Proc. of EACL 2006, the 11st Conference of the European Chapter of the Association for Computational Linguistics*, 2006.
[33] P. McNamee, HLTCOE efforts in entity linking at TAC KBP 2010, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[34] W. Zhang, Y. C. Sim, J. Su and C. L. Tan, NUS-I2R: Learning a combined system for entity linking, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[35] X. Han and J. Zhao, Named entity disambiguation by leveraging Wikipedia semantic knowledge, *Proc. of the 18th ACM Conference on Information and Knowledge Management*, New York, NY, USA, pp.215-224, 2009.
[36] C. Pablo-Sánchez, J. Perea and P. Martínez, Combining similarities with regression based classifiers for entity linking at TAC 2010, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[37] A. Goldschen, B. Kapp, T. Meyer, B. Onyshkevych and P. Schone, TCAR at TAC-KBP-2010, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[38] W. Radford, B. Hachey, J. Nothma, M. Honnibal and J. R. Curran, Document-level entity linking: CMCRC at TAC 2010, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[39] A. Thomas, A. Rawal, M. K. Kowar, S. Sharma, S. Pitale and N. Kharya, Bhilai Institute of Technology Durg at TAC 2010: Knowledge base population task challenge, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[40] A. X. Chang, V. I. Spitkovsky, E. Yeh, E. Agirre and C. D. Manning, Stanford-UBC entity linking at TAC-KBP, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[41] R. L. Cilibrasi and P. M. B. Vitanyi, The Google similarity distance, *IEEE Trans. on Knowl. and Data Eng.*, vol.19, no.3, pp.370-383, 2007.
[42] N. Fernández, J. A. Fisteus, L. Sánchez and E. Martín, WebTLab: A cooccurrence-based approach to KBP 2010 entity-linking task, *Proc. of the 3rd Text Analysis Conference*, National Institute of Standards and Technology, Maryland, USA, 2010.
[43] N. Fernández, J. M. Blázquez, L. Sánchez and V. Luque, Semantic annotation of web resources using IdentityRank and Wikipedia, *Advances in Soft Computing*, vol.43, pp.100-105, 2007.