WebTLab: A cooccurrence-based approach to KBP 2010 Entity-Linking task

Norberto Fernández, Jesus A. Fisteus, Luis Sánchez, Eduardo Martín
{berto,jaf,luiss,emartin}@it.uc3m.es
Web Technologies Laboratory (WebTLab)
Department of Telematic Engineering
Universidad Carlos III de Madrid
Universidad 30, Leganés, Madrid, Spain

Abstract

In this paper we describe the approach followed and results obtained in the KBP-Entity Linking task of TAC 2010 by the Web Technologies Laboratory (WebTLab). The system implemented for the task is based on the intuition that instances in a knowledge base typically cooccur in documents with other related instances (for example, Obama and USA). Based on that intuition, the proposed system takes advantage of the information provided by the Wikipedia link structure to compute instance cooccurrence estimates, and uses them for disambiguation purposes, combinined with more classical techniques from the information retrieval community.

1 Introduction

This notebook paper describes the approach followed and results obtained in the KBP-Entity Linking task of TAC 2010 by the Web Technologies Laboratory[1] (WebTLab). The system implemented for the task is inspired by the following principle (that we call the *semantic coherence* principle): As instances in a knowledge base typically cooccur in documents with other related instances (for example, Obama and USA or Pau Gasol and Los Angeles Lakers) the occurrence of a certain instance in a document gives information about the occurrence of other instances. That is, in order to know which is the best instance in the KBP KB (Knowledge Base) for the entity in the query, we use as context the other instances that are mentioned by the document the query refers to. To achieve this goal, our system proceeds through the following stages:

1. It uses a combination of named entity recognizing tools to detect the entities (persons, locations, organizations) that appear in the document referred by the query.

2. The system looks for candidate instances for each named entity. An extended knowledge base was built with that purpose. It merges the instances in the KBP KB with information from their associated Wikipedia articles. 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[2]. For each named entity in the input document, the text of the named entity is used to query the index, obtaining the potential candidate instances in the extended KB.

3. A ranking process is carried out to decide which is the best candidate instance for each entity (including the one in the query). Due to the semantic coherence principle, the occurrence of a certain instance depends on the occurrence of other instances in the document. Therefore, all the entities have to be disambiguated at the same time. A PageRank-like algorithm (Fernández et al., 2007; Fernández et al., 2006) is used to achieve this goal. This algorithm uses as input the Lucene scores for the candidates (obtained in the previous stage) and information about instance cooccurrence. In particular, estimates about the cooccurrence of each pair A, B of instances in the extended

[1] http://webtlab.it.uc3m.es/index.html

[2] We used Lucene version 3.0.1 available at http://lucene.apache.org/ (27/Oct/2010)

KB are computed by counting the number of articles in Wikipedia than link both to A and B, as well as the direct links between A and B. The algorithm produces a score for each pair (entity, candidate). Taking into account these scores, a configured threshold, and the origin of the candidates (KBP KB, Wikipedia) the algorithm chooses a candidate or NIL.

Three different runs were submitted that differ in configuration parameters. Our best run achieved a 0.7698 accuracy over all queries, according to the official results.

The rest of this paper describes our system in depth: Section 2 describes the components of our system and the different processing stages and information sources. Section 3 shows the results obtained by the system in the competition. Finally, section 4 provides some concluding remarks and future lines of development of the system.

2 The WebTLab approach

The system that the WebTLab team submitted for evaluation to KBP 2010 consists of a set of components that are connected in a document processing pipeline. Within this pipeline, each component (except the first one) uses as input the results of the previous component plus additional information obtained from configuration, and/or from data sources generated by preprocessing the KBP knowledge base and the Wikipedia.

The architecture of the system showing all its components and information sources is depicted in figure 1. The next sections describe with more detail the functionality provided by each component and the information stored in each data source.

2.1 Document fetcher

The main goal of this component is to process the input XML file that contains the evaluation queries, and trigger the processing of those queries one by one. For each query, the contents of the context file are fetched from the local filesystem and are provided to the next component in the pipeline (*docXML* in figure 1), together with the query information: query identifier, entity text, and document identifier (*queryId*, *entText* and *docId* respectively in figure 1).

2.2 Text preprocessor

This component was implemented in order to prepare the XML text of the original document for later processing stages. The XML format in the original corpus includes some elements (for instance, *TRAILER*) that are optional, and thus, are not found in all the documents. The text preprocessor transforms the input format into the TREC XML format, so that later stages can expect a regular document structure. An example of the resultant XML structure is shown below:

```
<DOC>
  <DOCNO>The docId goes here</DOCNO>
  <TEXT>
     The text contents of all original
     tags are included here
  </TEXT>
</DOC>
```

The output of this component includes the query information as provided by the previous component plus a new element *docText* that contains the TREC XML text generated by the preprocessor.

2.3 Entity finder

Taking as input the contents of the *TEXT* element as provided by the text preprocessor, the entity finder component uses natural language processing tools to find occurrences of named entities (persons, locations, organizations). The implemented system uses a combination of three free, open source tools to carry out this task: University of Sheffield's GATE (Cunningham et al., 2002), Stanford's Named Entity Recognizer (NER) (Finkel et al., 2005) and University of Illinois at Urbana-Champaign's LbjNerTagger (Ratinov and Roth, 2009). 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, the component mixes the results, selecting the entities detected by at least two recognizers, to generate the definitive output entity list (*entities* in figure 1).

2.4 Entity filter

While most of the documents included in the KBP collection are relatively small (few kilobytes) there are some outliers that are specially

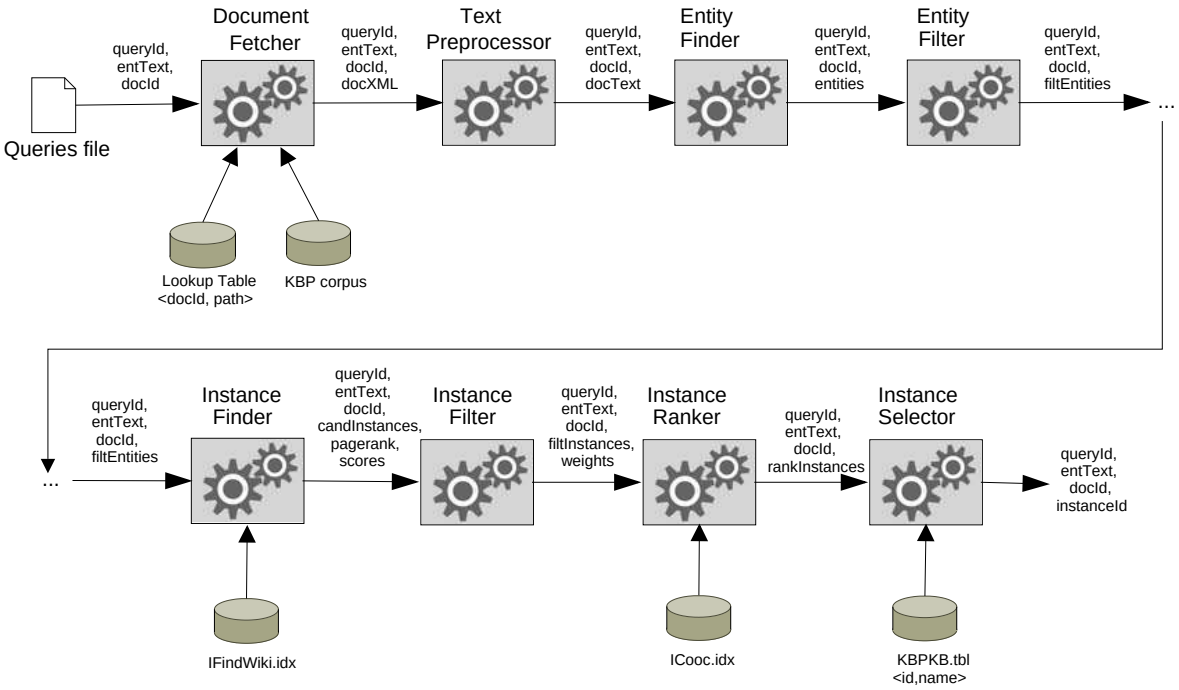

Figure 1: Document processing pipeline architecture.

big[3]. When analyzing these documents with the entity finder, many entities are detected[4] which may result in an overload for the next components of the pipeline. To avoid such overload, the entity filter 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 the entities with respect to the first occurrence in the document of the target entity (the text of the query), keeping those entities that appear nearer to it.

2.5 Instance finder

The main goal of this component is to obtain a set of candidate instances from the extended knowledge base for each entity obtained at the previous stage. As said in the introductory section, our approach uses the official KBP knowledge base, but also takes advantage of information from Wikipedia.

Using a dump downloaded from the official English Wikipedia website[5] (the one dated 01/30/2010 to be more precise), a Lucene index was built (*IFindWiki.idx* in figure 1). Each entry in the index represents a Wikipedia article (ignoring disambiguation pages and pages with special namespaces –User, Talk, File, etc–, with redirections resolved). The following fields are stored for each entry:

- Fields for literal queries:
  - **source** The name of the Wikipedia page.
  - **anchors**, **redirects** and **disambiguation** These fields contain, respectively, the concatenated, tab separated, 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. These fields are intended to be used for literal queries. That is, Lucene matches values in these fields only if there is

[3]For example, the size of document LTW_ENG_20070309.0062.LDC2009T13.sgm is 968K

[4]The document LTW_ENG_20070309.0062.LDC2009T13.sgm contains more than 2500 named entities according to our experiments

[5]http://en.wikipedia.org/wiki/Wikipedia:Database_download (27/Oct/2010)

a perfect match between the text in the query and one of the entries in the source, anchors, redirects or disambiguations fields.

- Fields for non-literal 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 for other purposes:
  - **identifiers** The associated identifier in the official 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 official KB entries, the names of the pages are compared with the names of the 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 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 identitifers of both entries in the KBP KB are stored in this field, separated by whitespace.
  - **pagerank** Given the graph structure of the Wikipedia dump, the PageRank value of each page was computed and stored in the index for later use.

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[6], several ways of using the text of the entity are possible.

When designing the specific queries for Lucene, our goal was to maximize precision. Due to this, we started by looking for exact matches of the entity text in the index (the non-parsed fields). Obviously, maximizing precision comes at the cost of reducing recall. Therefore, no exact matches were found for some entities in the first prototype of the system. To address this limitation, the final system relied on a three-stage query mechanism. Each stage uses a query with more relaxed conditions than the previous one, and is executed only when the previous stage does not produce any results. The queries run in the three stages are shown in table 1.

In all the queries the Lucene boosts coefficients were configured so that the *source* field had a x10 boost, the *redirects*, *disambiguation* and *anchors* fields had a x5 boost and the *parsed* fields have the default x1 boosting.

The number of results returned by the queries was limited to a maximum (200 in our experiments). When more results were obtained, those with the lowest Lucene scores were dropped.

The output of this component is the list of candidates for each input named entity. For each candidate, the score given by Lucene to it and the value of its *pagerank* field are also provided to the next component. This output is represented

[6] More information on Lucene query capabilities and operators may be found in http://lucene.apache.org/java/3_0_1/queryparsersyntax.html (27/Oct/2010)

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 (~ 2)

Table 1: Queries run at each stage of the instance finder component

with the elements *candInstances*, *pagerank* and *scores* in figure 1.

2.6 Instance filter

This component limits the maximum number of candidate instances to be handed to the following components of the processing pipeline, to avoid potential overloads. To select the candidates that will survive the filtering process, the algorithm uses the following criteria:

- The target entity (the text of the query) should keep more candidates than the rest of the entities (context entities), to minimize the probability of filtering out the correct candidate from the list.

- The number of candidates for the target entity should depend on the mechanism used to obtain them. As seen in section 2.5, candidate instances may come from different query stages. As the first stages are based on more restrictive queries, it is expected that they also produce more reliable results. Thus, depending on the specific stage the candidates for the target entity come from, 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 field in the *IFindWiki.idx* index are taken into account (the instance finder component ranks them using only the Lucene information).

The algorithm implemented in the instance filter computes first new weights for all the candidates of each entity, by linerly combining the Lucene scores and PageRank values with coefficients α_L and α_{PR} respectively. Candidates are then re-ranked using those weights and filtered: the entities other than the target one retain at most 15 candidates, whereas 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 identifiers of the candidates that survive the filtering process (element *filtInstances* in figure 1) and their new weights (*weights* in figure 1) are provided to the next component in the pipeline.

2.7 Instance ranker

The result of the execution of the instance filter component is a (maybe empty) list of candidate instances for each named entity, with a weight associated to each candidate. As the final goal of the KBP entity linking system is to define a single assignment entity-instance (or entity-NIL), a decision process that selects the best candidate for each entity is needed.

The main goal of the instance ranker is to rank the set of candidates for each entity returned by the entity filter. This rank is later used by the next component in the processing pipeline (the instance selector, see section 2.8) to choose a single instance (or NIL) for the target entity.

The algorithm implemented in the instance ranker is an adaptation of the IdRank algorithm described in (Fernández et al., 2007) and (Fernández et al., 2006). Basically, the principle that inspires the algorithm is the *semantic*

coherence principle: as instances typically cooccur 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 cooccurrence information for disambiguation, the actual instances that appear in the document need to be known. However, only the entities, not the instances, are known. Moreover, due to the semantic coherence principle stated above, the occurrence of a certain instance depends on the occurrence of other instances in the document. There is, therefore, a recursive problem. It has to be addressed by defining the assignments entity/instance not only for the target entity but for all the entities. That is, all the entities need to be disambiguated at the same time.

Fortunately, as shown in (Fernández et al., 2007) and (Fernández et al., 2006), 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* (Page et al., 1998). In our case, we are interested in ranking 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 cooccur with it is high*.

As it is explained in (Fernández et al., 2007), the principle stated above can be mathematically represented by the following matricial equation:

$$R = (1 - \alpha)AR + \alpha E \quad (1)$$

Where, if N represents the total number of non-duplicate candidate instances for all the entities in the document:

- A is a matrix $A \in \mathbb{M}_{NxN}$ 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 ranking which is given to I_i. Those coefficients are computed by using instance cooccurrence information, as will be described later.

- R is a vector $R \in \mathbb{R}^N$ that represents the ranking of the candidates in the context document. The element R_i represents the ranking of a specific candidate instance I_i.

- E is a vector $E \in \mathbb{R}^N$ that corresponds to a source of rank. As it is shown in (Fernández et al., 2007), 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 $\alpha \in [0, 1]$.

This matricial equation can be solved with different numerical methods, like the *power method*, as proposed in (Page et al., 1998). A description of the power method can be found at (Mathews and Fink, 2004).

In our approach, two candidate instances are considered to cooccur when Wikipedia pages that link to them exist. For instance, the instance referring to *Italy* is considered to cooccur with the instance for *European Union* because the Wikipedia page for *Rome* points to both the pages of Italy and the European Union. Additionaly, two instances are also considered to cooccur 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 cooccur, because there is a direct link from the Wikipedia page for Obama to the page of the University.

The cooccurrence 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.

- **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). In order to separate the different names the tabulator character is used.

Taking into account the two different contributions to the cooccurrence information described above, the equation (1) has been adapted to:

$$R = (k_L A_L + k_C A_C)R + k_E E \qquad (2)$$

Again, if N represents the total number of non-duplicate candidate instances for all the entities in the document:

- A_C is a matrix $A_C \in \mathbb{M}_{NxN}$. The values of A_C, a_{ij}^C are computed using the information in the *ICooc.idx* index as follows:

$$a_{ij}^C = \begin{cases} 0.0 & i = j \\ |Q(d = I_i, d = I_j)|/|Q(d = I_j)| & i \neq j \end{cases} \qquad (3)$$

  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. Thus, the number of index entries 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 documents 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, as in real life, 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 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 coocurrence 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. However, how to compute context similarities from this intuition is not clear. We present here the initial approach we have followed for KBP 2010, although other possibilities could be explored in the future.

  The idea that we have explored in this paper is to assume that if a certain Wikipedia article links many times to another one, and the instance represented by the first article is likely to occur in the document being processed (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, A_L is a matrix $A_L \in \mathbb{M}_{NxN}$, whose values, a_{ij}^L, are computed using the information in the *ICooc.idx* index using the following formula:

$$a_{ij}^L = \begin{cases} 0.0 & \nexists I_j \rightarrow I_i \ \ or \ \ i = j \\ score(Q(s = I_j, d = I_i)) & \exists I_j \rightarrow I_i \end{cases} \qquad (4)$$

  Where the $\rightarrow$ 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 $a_{ij}^L = a_{ji}^L$, and thus the A_L matrix is not symmetric.

- R is a vector $R \in \mathbb{R}^N$ that represents the ranking of the candidates in the context of document. The element R_i represents the ranking of a specific candidate instance I_i.

- E is a vector $E \in \mathbb{R}^N$. The value of the element E_i is the weight for the instance I_i that the instance filter component provides to the instance ranker. As it might happen that the same instance I_i is 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 matricial equation (2) 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 in the document, the algorithm returns a vector with all the pairs (candidate instance, weight) for the entity (see *rankInstances* in figure 1), which is processed by the next component in the pipeline.

2.8 Instance selector

The goals of this last component of the processing pipeline are twofold:

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

$$Plausibility = \frac{TopInstanceWeight}{SecondInstanceWeight} \quad (5)$$

   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, because there is less confidence about the result.

   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. In order to establish appropriate values for σ_L and σ_H, the system was run with the set of queries of 2009. Based on the logs of the system, an auxiliary script analyzed many combinations of (σ_L, σ_H) to determine the combination of thresholds that would provide the best results ($\sigma_L = 1.2$ and $\sigma_H = 2.0$). There is no guaratee that these values will still be optimum for other sets of queries. However, after having analyzed the results obtained with the 2010 query set, we found those values to be optimum also for that query set.

2. Addressing the problem of multiple identifier values that was described in section 2.5. 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 an specific table (*KBPKB.tbl* in figure 1) which contains the *id* and *name* of all the entries in the official KBP KB. A text similarity metrics (the Levenshtein distance (Levenshtein, 1966)) is then used to compare the different names with the text of the target entity. The official KB entry whose name is the most similar to the text of the query is finally assigned.

3 Evaluation

This section shows the results obtained by our system in the 2010 entity-linking task. Three different runs were submitted. The differences between these runs were not in the algorithm, which was in all the cases the one described in previous sections, but in the configuration parameters. The configuration of each run is shown in table 2.

Results achieved by these runs, as reported by the organizing committee, are summarized in the following tables: table 3 shows the global results obtained by the different runs; table 4 shows the results for each entity type.

As it can be seen by looking at the *All* columns in table 4, the algorithm provides better results for persons than for organizations and geopolitical entities. One of the reasons that might explain this fact is that the algorithm is based on instance cooccurrence information and, whereas the context for persons is usually well-defined, for organizations and GPEs it can include many

Run	Instance finder		Instance ranker			Instance selector	
	α_L	α_{PR}	k_L	k_C	k_E	σ_L	σ_H
WebTLab1	0.8	0.2	0.55	0.25	0.2	1.2	2.0
WebTLab2	0.8	0.2	0.55	0.25	0.2	1.05	1.5
WebTLab3	0.8	0.2	0.4	0.4	0.2	1.2	2.0

Table 2: Configuration of the WebTLab annotation system for the three submitted runs

Run	2250 queries	1020 non-NIL	1230 NIL
WebTLab1	**0.7698**	0.6647	0.8569
WebTLab2	0.7636	0.6098	0.8911
WebTLab3	0.7596	0.6049	0.8878

Table 3: Global results obtained by the different runs (KBP 2010)

Run	ORG			GPE			PER		
	All	non-NIL	NIL	All	non-NIL	NIL	All	non-NIL	NIL
WebTLab1	0.7613	0.6513	0.8363	0.6569	0.6441	0.6829	0.8908	0.7324	0.9535
WebTLab2	0.7707	0.6086	0.8812	0.6262	0.5706	0.7398	0.8935	0.7042	0.9684
WebTLab3	0.7680	0.6053	0.8789	0.6195	0.5666	0.7276	0.8908	0.6948	0.9684

Table 4: Results for each entity type obtained by the different runs (KBP 2010)

different entities for many different periods, and due to several different reasons.

According to the official results, shown in (Ji et al., 2010), the WebTLab approach to entity linking ranked 7th among the 16 teams that submitted results to the task. Given that the best team achieved a 0.8680 total accuracy, and that human evaluators obtained an agreement rate over the 90%, our system has a room for improvement for future editions. However, on the positive side, the approach that was followed by WebTLab was an unsupervised one, which may partially explain the rank of the system, and opens the door to improvements based on combining the system with supervised techniques.

4 Conclusions and future lines

The system presented in this paper combines traditional information retrieval methods with instance coocurrence information for disambiguation purposes.

Results show that the use of coocurrence information provides better results for persons than for organizations and, especially, geo-political entities.

Some potential future lines of development of the system described in the paper are:

- We want to improve the process of collection of candidate instances which, as we have explained, is by itself responsible in our system of the errors in 12,7% of the queries that have a *non-nil* correct answer.
- We plan to explore different alternatives to exploit direct links between Wikipedia articles, as we have discussed in section 2.7.
- In current system implementation we have relied on Wikipedia links and cooccurrences among the candidate instances for the input entities as main disambiguation information. An aspect we want to explore in the near future is try to extend the context information not only with instances associated to named entities, but also detecting in the input documents potential links to Wikipedia pages about common topics (like for instance *programming language*, *president*, etc). Our intuition is that adding these potential links to the disambiguation process may enrich the available information and provide better results, and aspect that obviously needs empirical testing.

- The knowledge base provided by TAC organizers includes for its entries and associated type (person, GPE, organization or unknown). The named entity recognizers that are used within the entity finder component, also provide a type for each entity. At the moment our system takes no advantage of this data, so a potential future line may address this limitation, specially taking into account that the type of entity to be disambiguated has an impact on the algorithm results.

- Finally, combining the unsupervised approach described in this paper with supervised machine learning techniques may be also a path worth exploring.

Acknowledgments

This work has been partially funded by the Spanish Government under contract ITACA (TSI2007-65393-c02-01).

References

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

N. Fernández, J. M. Blázquez, L. Sánchez, and V. Luque. 2006. Exploiting Wikipedia in integrating semantic annotation with information retrieval. In *Proceedings of the 4th International Atlantic Web Intelligence Conference, AWIC 2006, Beer-Sheva, Israel, June 2006*, volume 23 of *Studies in Computational Intelligence*, pages 61–70. Ed. Springer, June.

N. Fernández, J. M. Blázquez, L. Sánchez, and A. Bernardi. 2007. IdentityRank: Named entity disambiguation in the context of the news project. In *Proceedings of the 4th European Semantic Web Conference, ESWC, LNCS 4519*, pages 640–654.

J. R. Finkel, T. Grenager, and C. Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL), pp. 363-370*.

H. Ji, R. Grishman, and H. T. Dang. 2010. Overview of the TAC 2010 Knowledge Base Population Track. In *Proceedings of TAC/KBP 2010*.

A. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):787–793.

J.H. Mathews and K. D. Fink. 2004. *Numerical Methods using MATLAB*. 4th edition. ISBN: 978-0130652485.

L. Page, S. Brin, R. Motwani, and T. Winograd. 1998. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford University, January.

L. Ratinov and D. Roth. 2009. Design Challenges and Misconceptions in Named Entity Recognition. In *Proceedings of the 13th Conference on Computational Natural Language Learning (CoNLL)*.