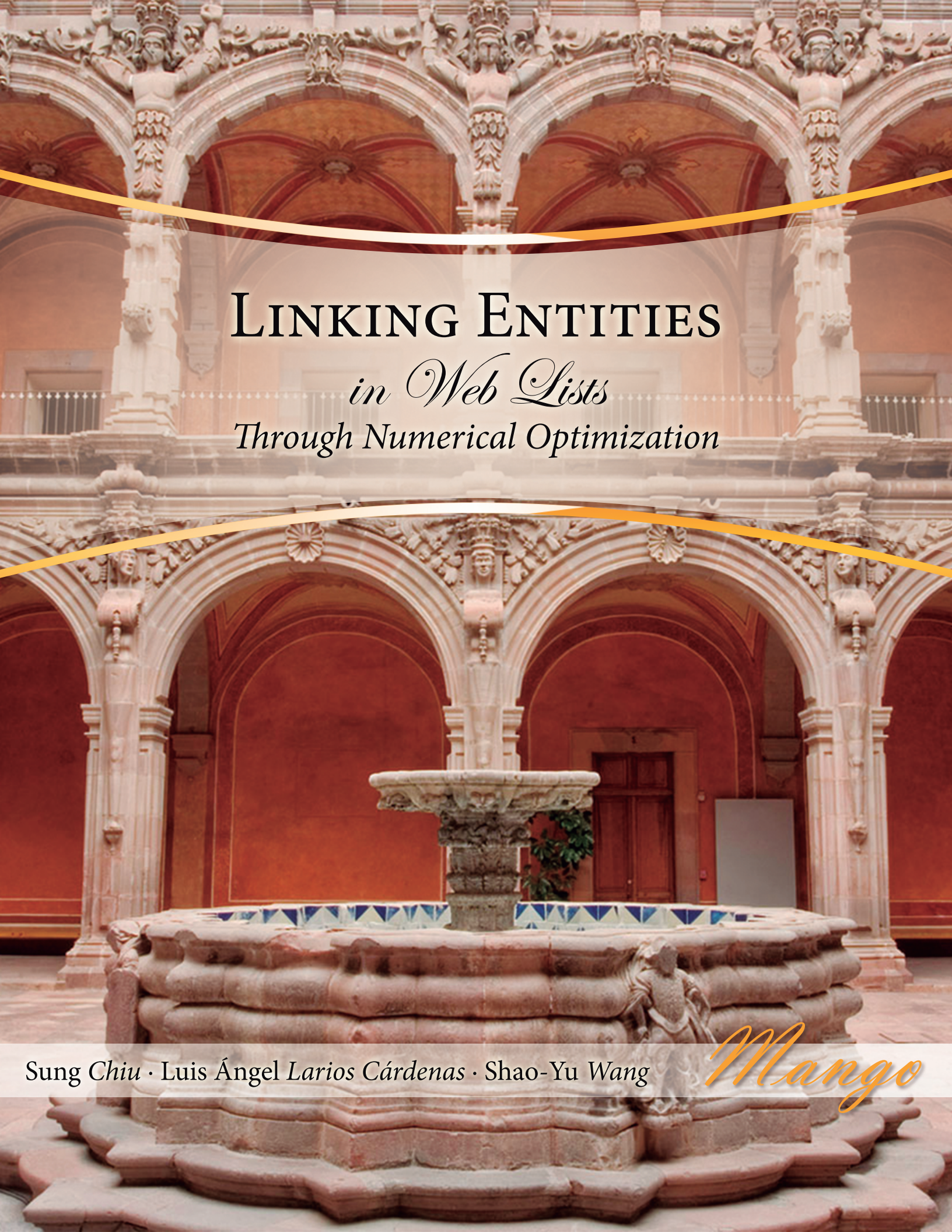


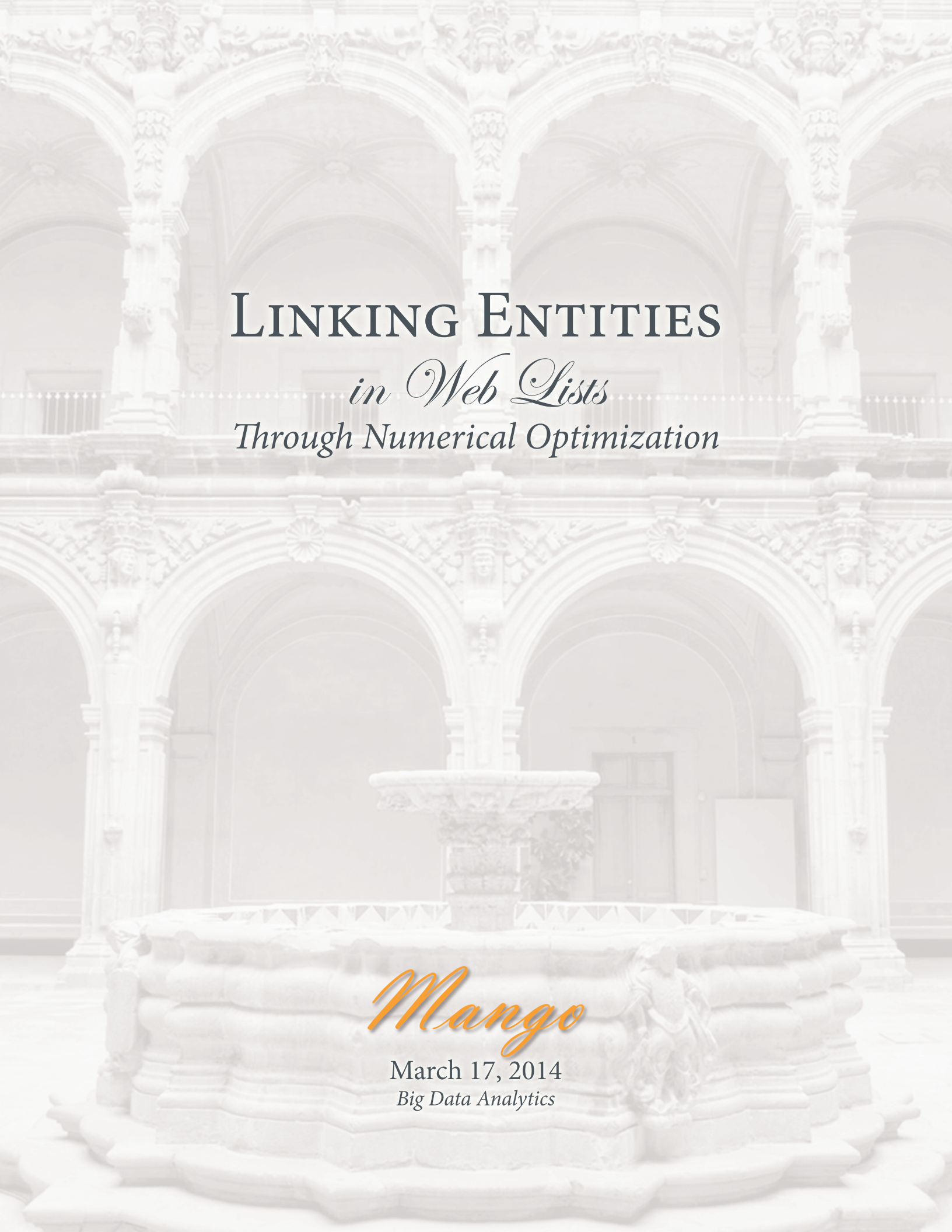
# Linking Entities

## *in Web Lists*

## *Through Numerical Optimization*

Sung *Chiu* · Luis Ángel *Larios Cárdenas* · Shao-Yu *Wang*

Mango



# Linking Entities

## *in Web Lists*

## *Through Numerical Optimization*

*Mango*

March 17, 2014

*Big Data Analytics*

COM SCI 249 Big Data Analytics

*Project Report*

# LINKING ENTITIES *in Web Lists* *Through Numerical Optimization*

***Team Mango**: Sung Chiu, Luis Ángel Larios Cárdenas, and Shao-Yu Wang*

## 1. INTRODUCTION

Linking web entities is the task of finding a mapping between an *entity mention* (after a proper named entity recognition process) and its counterpart in the real world. Linking entities has attracted lots of attention from web researchers since it is a potential approach to advancing the functionality of search engines to a more expressive semantic level, rather than just considering keywords inside queries as isolated units. In order to achieve this goal, some information-extracting machinery has to be able to recognize the coherence between named entities in a request, their similarity, and relatedness.

The task of semantic relationship discovery is perhaps the most demanding part of the whole endeavor. Consider, for instance, the updating process of a knowledge base. When new information or events occur in the real world, it is desirable to incorporate this knowledge into the corpus in an automatic and efficient manner; however, the particular entity for which new facts have been collected may have different names (nicknames or **surface names**[1]), and, yet, different entities can share the same name that has been mentioned in the piece of novel knowledge to be incorporated. This poses the problem of **disambiguation**, which consists of accurately identifying the correct world entity match for which we wish to update its information in the knowledge base. For example, the city of Los Angeles has two names through which people identify it, e.g. "Los Angeles" and "L.A.;" they both are surface names that refer to the same real world entity. Also, considering the sentence "Michael Jordan won his first championship with the Bulls in 1991," one can observe that the **entity mention** "Michael Jordan" may refer to either the basketball player or to Prof. Michael Jordan, who is a faculty member of UC Berkeley. This means that, for the same surface name, we may have several **candidate mappings** from which we should choose the right one. Basically, this is possible by leveraging information from the context of a sentence, e.g. after seeing the words "championship" and "(Chicago) Bulls," one can infer that the mapping to "Michael Jordan" is "Michael Jordan the basketball player." But, how can this be done automatically?

The pioneer effort for linking entities in free text with a knowledge base saw the light in 2006, by the hand of R. Bunescu and M. Pasca. They mainly focused on entity mentions detected within web documents. They employed several disambiguation resources that leveraged the rich structure of the global encyclopedia by excellence: Wikipedia. Furthermore, Bunescu and Pasca built a context-article cosine similarity model and an SVM based on a taxonomy kernel in order to identify the real entities under analysis. One year later, another researcher, Cucerzan, took over the disambiguation quest on web documents and introduced a data structure for holding and organizing Wikipedia vocabulary. Such a structure was divided into two parts: the first one contained surface forms, while the second one stored the associated entities along with contextual information. In Cucerzan's framework, disambiguation was performed by measuring similarity among named entities in the document and choosing the best match with respect to a set of candidates. To date, Cucerzan's idea is still used as foundation to develop more sophisticated disambiguation frameworks.

The emergence of huge, open knowledge bases, such as Wikipedia, YAGO, and WordNet, has catalyzed the construction of systems that are capable of disambiguating entities identified in web texts, microblogs, and web lists. These global encyclopedias contain constantly updated information for almost any entity of interest. They also provide semantic relationships (in a form of internal and inter-page links), and an expressive taxonomy described as a hierarchical set of categories which entities belong to. In this fashion, researchers are now leveraging available meta-information to define *semantic relatedness* and *semantic associativity* among entities for the disambiguation task.

[1] Throughout this paper the terms *surface name*, *surface form*, *list item*, *entity mention*, and *named entity* are used interchangeably.

On the other hand, free-form texts, microblogs, and even lists are three main ways to accommodate information inside webpages. Casting the task of linking web entities within any of them poses the same problem of disambiguation under different perspectives and limitations. For example, free-form texts have one unique context, from where one expects a global coherence among entity mentions that appear in it. In effect, if the text talks about a particular topic, then all entities must share certain degree of relatedness for they would not be there otherwise. Another scenario is that of microblogs (e.g. Twitter), which share a reduced, local coherence for any one entry. Actually, microblogs are of such a free-form nature that entities that appear in one entry may be completely non-related to the entities that appear in a different entry: in fact, most of the time they belong to different topics and are limited to a few dozens of words in body extent. In that case, the entity linking task becomes more involving since one has to determine/approximate the user interests so as to infer which entities are more likely to appear in any microblog entry, without putting aside the valuable information that comes from the existing relationship among entity mentions at their intra level. Finally, web lists are the most constrained forms to permit entity disambiguation because the "textual coherence" is practically non-existence (not even locally). This limitation calls for another approach to replace the lack of context. In this project, we have embraced the task of linking web entities in web lists, solving the problem of disambiguation through numerical optimization in order to get the highest possible qualified group of entity matches from a set of candidate mapping entities.

Tables (i.e. <table></table>) inside web pages are the preferred structure to accommodate information or tuples that, in one or another way, share some semantic similarity, rather than semantic relatedness (which is what we usually find in web texts). Perhaps, one of the columns in a table bears the titles of the all-times literary masterpieces or the names of the most renowned writers. For instance, around 75% of the tables on the web typically have a subject column that contains entities that describe what that table is about. Linking web entities, in this case, would be helpful at annotating the subject column to enrich the content of the web page by providing additional references from the actual list items to their respective entry in a global knowledge base, like Wikipedia.

Therefore, given a knowledge base about world's entities, (containing a descriptive taxonomy and relationships among them), and a corpus of web lists, we aim at linking the entity mentions (or list items) to their corresponding real world entities in an automatic way

## 2. Related Work

Many researchers have delved into the task of linking web entities, approaching the problem more or less in the same way. Essentially, this paradigm attempts to capture the surrounding context of an entity mention, so that the context can be further compared to its homologue from several possible correspondences in the knowledge base. The availability of such a context for the input entities is not always complete, and, in those cases, researchers need to look for the missing information somewhere else.

As we mentioned above, linking entities in the web has mainly focused in three areas, listed in descending order, according to their easy-to-access context information availability:

- Topic-based web texts.
- Microblog posts.
- Web lists.

The first large-scale named entity disambiguation system was proposed by Cucerzan, 2007. His framework employs a large amount of information automatically extracted from Wikipedia. Each entity is represented with a high-dimensional feature vector $\mathbf{g}(\gamma)$, which is used in an inner product expression $r(\gamma, \gamma') = \mathbf{g}(\gamma)^T \mathbf{g}(\gamma')$ to compute the similarity of the contexts of $\gamma$ and $\gamma'$. Letting $\Gamma_0$ be the set of entity disambiguation candidates for all spots on a page, one can compute the average vector $\mathbf{g}(\Gamma_0) = \sum_{\gamma \in \Gamma_0} \mathbf{g}(\gamma)$. Thus, the score of candidate $\gamma$ for spot $s$ can be calculated considering two factors: a local context compatibility and $\mathbf{g}(\gamma)^T \mathbf{g}(\{\Gamma_0 \backslash \gamma\})$, that is reminiscent of leave-one-out cross validation. Cucerzan's system yields high accuracy, exceeding 91 and 88% in tests performed on news data and Wikipedia texts. In addition, because the processes use very little language-dependent resources, his framework can be easily adapted to languages other than English. However, a problem with his approach is that $\Gamma_0$ is contaminated with all possible disambiguation candidates for all spots, so the "check for agreement with the majority" may be misleading (Kulkarni, S., 2009). Moreover, his approach requires a local context that is found only in the neighborhood of entity mentions within web text; in our case, web lists, lack precisely of this context, making Cucerzan's algorithm unsuitable for the purpose of our work.

One more recent framework by Shen, W., et al, 2012, called LINDEN, leaves behind the simplistic model of the *bag of words* used in Cucerzan's algorithm

and incorporates new features to construct a semantic network for the annotation problem of web texts. These new features exploit Wikipedia's structure and the taxonomy of the knowledge base YAGO. Through this network they obtain measures for semantic associativity and for semantic similarity. Additionally, they compute a global coherence among the entity mentions and a *prior probability* (explained later in this paper) for all candidates. Moreover, they learn a threshold that allows their framework to predict unlinkable mentions in the analyzed text. As a result, LINDEN is a robust framework whose accuracy surpassed 94% in all of the test cases where there was minimal hand-tuning/intervention to make documents appropriate to their system. However, the authors point out that, in particular, the *semantic associativity* plays an important role in their framework. This measure happens to rely on the extraction of Wikipedia concepts inside the text. Therefore, if the document contains itself very few Wikiconcepts (or internal links to Wikipedia articles), the accuracy may drop significantly to almost entirely depend on the candidates' prior probabilities. For this reason, LINDEN is suitable for neither microblogs, nor web lists, since the lack of enough contextual information hinders the performance of the whole algorithm.

With respect to microblogs, Shen, W., et al, 2013, have showed a modest success solving the linking task in Twitter entries. Their goal is to map every named entity in a user's tweets to their real world counterpart in the knowledge base. Their approach is similar to our work, but, according to their observation, every user has preferences or interests that aid to counteract the noisy, short, and not-so-consistent semantic context within tweets. They use this "interest" assumption to disambiguate name entities by building a graph whose edges express a connection between entity mentions and their possible candidate mappings. Given the user interest, the linking quality between, say, Michael Jordan and Chicago Bulls, is enhanced inside the graph via their *User Interest Propagation Algorithm*, which takes into account user preferences for other subjects like, in this case, the named entity NBA – an indicative of a sports aficionado tweeter. Interestingly, though, KAURI (their system) achieves relatively high accuracy (around 85%) after lots of hand-depuration to remove undesired, extremely noisy tweets from their ground truth data set. Anyways, since we are dealing with web lists here, we do not have something conceptually similar to user preference or interest. Nonetheless, we do utilize some of their metrics such as the *prior probability* to find the best group of candidate mapping entities.

Then, in the field of annotating web lists, we find the approach by Limaye, G., et al, 2010, being somehow similar to our work. However, instead of linking each named entity in the web list, their task is to find the column type and relationship between columns in a table since a lot of tables may not contain such information. In order to find the column type and relations, they use a catalog collected mainly from YAGO. Their algorithm contains 5 functions: one for cell text and name entities, another for column header and column type, a third one for column type and entities, a fourth one for relations and column type pairs, and the last one for relations and entity pairs. Their goal is to maximize the product of those 5 equations. Nonetheless, their work is only limited to catalogs from YAGO, excluding any rich context information from Wikipedia to perform entity annotation. Moreover, if a table has too many rows, finding the optimal value of those 5 equations by computing pairs of entities is not efficient.

Lastly, the work on which we base our implementation is LIEGE (Shen, W., et al, 2012). LIEGE is the most up-to-date data mining approach to annotate list items, considering the lack of context information around the entities that one wants to identify. It leverages the rich semantic structure of Wikipedia and YAGO to compute popularity and similarity measures for all possible candidate mapping sets. However, being the underlying knowledge base so large, LIEGE performs a lot of preprocessing to generate its dictionary and to compute n-gram vectors for all real-world entities. We believe that one can save preprocessing time and space, in particular for the computation of n-gram vectors, if we only scan the Wikipedia pages of the candidate mappings of the surface name being queried. Moreover, they use an *Iterative Substitution Algorithm* that, though fast, converges to a local maximum that is roughly 90% successful at the annotation task. We can do better if we give a chance to not-so-popular candidate mappings and restart the greedy improvement process without excluding those candidate entities. This can be done via numerical methods such as *Simulated Annealing*, which is one contribution of the current work we present.

In our implementation, we propose the use of *Simulated Annealing* for maximizing a *Linking Quality Metric*. This technique is suitable for optimization problems of large scale, especially ones where a desired global extremum is hidden among many poorer, local extrema (Press, W., et al, 2007). *Simulated Annealing* has been successfully employed for solving the famous *Traveling Salesman Problem*, to find the best arrangement of several hundred thousand circuit elements on a tiny silicon substrate, and even for learning low-level behaviors in artificial animals. Among these applications we can find examples of combinatorial

maximization, where the number of elements in the configuration space is factorially large, so that they cannot be explored exhaustively. *Simulated Annealing* is based on the so called Boltzmann probability distribution, which is used to determine when a normal maximization process should leave the greedy up-hill exploration to undergo another "not as good" path. By opting for "changes of direction" during optimization we may wind up finding a global maximum after resuming to the usual up-hill procedure. Thus, this method is suitable for optimizing a *Linking Quality Metric* because it permits a random exploration of possible sets of candidate mappings that LIEGE's Iterative Substitution Algorithm simply ignores. However, this technique heavily relies on an annealing schedule parameter. This parameter is fundamental to define how often we should accept a "not as good" configuration state, and it usually requires physical insight or trial-and-error experiments in order to set it to a working value.

## 3. METHODOLOGY

We have chosen to depart from the guidelines established in LIEGE (Shen, W., et al, 2012). LIEGE is an annotation framework that tries to identify the real world's matching entities to the surface names in a web list, under the assumption that those entities share up to a certain degree the same conceptual type.

The general process of our system is the following:

- **Build a dictionary**, which contains entries for all possible surface names, aliases, and nicknames that refer to a given real world, knowledge base entity.
- For a web list, **enumerate all candidate mapping entities** (possible real world entities) that have as a surface name the entity mention that appears in the list.
- **Compute the *Linking Quality Metric*** for a choice of group of candidate mapping entities. The *Linking Quality Metric* makes use of the *prior probability* (popularity) of a candidate mapping entity and the *coherence* of such a candidate with respect to the rest of mapping entities that have been selected for the other surface names.
- Use the ***Iterative Substitution Algorithm*** and ***Simulated Annealing*** to find a nearly optimum choice of candidate mappings.

In the last step, we propose *Simulated Annealing* in addition to LIEGE's approach. In LIEGE, the *Iterative Substitution Algorithm* is used because it is simple and fast enough at finding a good group of candidates for matching the list items. However, the algorithm is a greedy hill-climbing procedure that stops searching when there is no more improvement. We have observed that finding the optimum group of candidate mappings is of combinatorial nature, and it is, therefore, suitable for a numerical method. With *Simulated Annealing* it is possible to skip local maxima by "going down the hill" with certain probability. Thus, *Simulated Annealing* explores a broader scope of the Linking Quality Metric's domain to finally wind up choosing the best possible set of candidate mapping entities.

| *surface form* | *real entity* | *count* |
|---|---|---|
| Microsoft Corporation | Microsoft | 16 |
| Michael Jordan | Michael Jordan | 65 |
| | Michael I. Jordan | 10 |
| | Michael Jordan (mycologist) | 7 |
| | Michael Jordan (footballer) | 3 |
| New York | New York City | 121 |
| | New York (magazine) | 12 |
| | New York (film) | 7 |
| | "New York" (Eskimo Joe song) | 5 |

*Figure 1, an excerpt of the dictionary.*

### 3.1. Building the Dictionary

The dictionary is a set of tuples $\langle surface\ form, real\ entity, count \rangle$ and stores all possible knowledge base entities that can be recognized with a given surface form or name. Additionally, it keeps track of how many times a real entity is referred to by using any surface form inside the knowledge base. An example of dictionary is shown in figure 1.

We build the dictionary by taking advantage of the rich structure and information stored in Wikipedia. Being the most widespread and complete, global encyclopedia, Wikipedia maintains a record of world entities, and it is collectively updated as soon as a new event takes place. With this in mind, we use the following Wikipedia constructs to create the dictionary:

- **Entity pages**: We adopt Wikipedia entity pages as being the world entities we seek to link to named entities in lists. Each Wikipedia entity page is unique, making it suitable for our needs. Therefore, we add the title of such a page to both $surface\ form$ and $real\ entity$.
- **Redirect pages**: Some world entities can be recognized by different aliases or nicknames. In Wikipedia, all of these nicknames are registered in redirect pages that automatically refer readers to the real world entities associated with them. For each redirect page we add its title to $surface\ form$ and its redirection page title to $real\ entity$.
- **Disambiguation pages**: A disambiguation page contains links to all possible real world entities that are referred to by a single surface

| *Web List* | *Candidate Mapping Entities* |
|---|---|
| A Tale of Two Cities | *A Tale of Two Cities*, *A Tale of Two Cities (musical)*, *A Tale of Two Cities (1935 film)*, *A Tale of Two Cities (lost)*, *A Tale of Two Cities (1911 film)*, *A Tale of Two Cities (1958 film)* |
| The Da Vinci Code | *The Da Vinci Code*, *The Da Vinci Code (film)* |
| The Godfather | *The Godfather*, *The Godfather (novel)*, *Charles Wright (wrestler)*, *The Godfather: The Game* |
| Gone with the Wind | *Gone with the Wind (film)*, *Gone with the Wind*, *Gone with the Wind (song)*, *Gone with the Wind (musical)* |
| Fear of Flying | *Fear of Flying (The Simpsons)*, *Fear of Flying*, *Fear of Flying (novel)*, *Fear of Flying (album)* |
| The Catcher in the Rye | *The Catcher in the Rye* |

*Figure 2, a candidate mapping entity set obtained from a dictionary. Candidates are listed in descending order with respect to prior probability (popularity). The right mapping entity underlined.*

form. Thus, we add the title of a disambiguation page to $surface\ form$ and the list of links that appear in it to $real\ entity$.

- **Hyperlinks**: Inside the *Wikitext*[2] of any page there may be links to other Wikipedia pages but appearing with a possible text variation to that of the Wikipedia entity page. The anchor texts in those links are other ways to refer to the linked real world entities. Thus, we add anchors to $surface\ form$ and target page titles to $real\ entity$.

### 3.2. Enumerating Candidate Mapping Entities

With a dictionary at hand, one can obtain the collection of candidate mappings associated with all surface forms (entity mentions) that appear in the web list. Then, picking one candidate for each named entity allows us to build a group $M$ of candidate mappings on which we can perform the *Linking Quality Metric* (introduced in later sections) and determine if that choice of mappings is the best we can do. Figure 2 is an example of candidate mappings extracted from a dictionary for the task of linking entities in a list of best-seller books.

### 3.3. The Linking Quality Metric

In order to know if the choice $M$ of candidate mappings for the named entities is a good one, we have to compute the *Linking Quality Metric* over each selected candidate entity and later consider the sum of the individual qualities as the overall metric. The *Linking Quality Metric* is a convex, linear combination of three measures: *prior probability*, *hierarchical similarity*, and *distributional context similarity*. These three measures are explained in detail in the following content.

[2] Wikitext is the textual content of a Wikipedia page; it follows the structuring rules of so called Wiki Markup, where links appear as [[surface form | real entity]].

#### *Prior probability*

Among the set of candidate mappings for a given entity mention, candidates have different levels of popularity: some of them are the most preferred among users, while others are pretty obscure in their usage. We capture this measure of popularity by leveraging the *count* data we collect when building the dictionary of figure 1. Formally, the *prior probability* is defined as

$$P_{pr}(r_{i,j}) = \frac{count(r_{i,j})}{\sum_{u=1}^{|R_i|} count(r_{i,u})}$$

Where $r_{i,j}$ is the $j^{th}$ candidate mapping of the $i^{th}$ entity mention, and $|R_i|$ is the number of candidate mappings for the $i^{th}$ surface form in the web list.

#### *Coherence*

Intuitively, all real mapping entities in a web list should share some conceptual type. We can capture this concept in a *coherence measure*. Coherence is an indication of how much a candidate mapping is similar to the real world mapping entities identified for the rest of the items in the list. The formal expression for coherence is:

$$Coh(r_{i,j}) = \frac{1}{|L|-1} \sum_{u=1, u\neq i}^{|L|} Sim(r_{i,j}, m_u)$$

where $m_u$ is the mapping entity for the list item $l_u \in L$, and $Sim(e_1, e_2)$ is a function that measures semantic similarity between two real world entities. Specifically, one can carry out $Sim(e_1, e_2)$ through a linear combination of two more particular expressions: *type hierarchy based similarity* and *distributional context similarity*.

***Type hierarchy based similarity***

In this case, we are required to know what hierarchical categories or types a candidate mapping entity belongs to. In our case, we have extracted the taxonomy of YAGO to obtain a hierarchical representation of all known real world entities. Thus, two candidate entities are semantically similar if their corresponding types are in close places within the YAGO's taxonomy. To measure $Sim_{hr}(e_1, e_2)$ between both entities, we first need to determine how similar their sets of types $T(e_1)$ and $T(e_2)$ are. Also, for each $t_1 \in T(e_1)$ we define $\varepsilon(t_1)$ in the other set $T(e_2)$ as

$$\varepsilon(t_1) = \arg\max_{t_2 \in T(e_2)} Sim_{hr}(t_1, t_2)$$

because $T(e_1)$ and $T(e_2)$ may have different sizes. Also,

$$Sim_{hr}(t_1, t_2) = \frac{2 \times \log P(t_0)}{\log P(t_1) + \log P(t_2)}$$

which is the similarity between two types, where $t_0$ is the *Lowest Common Ancestor* for both $t_1$ and $t_2$ in the hierarchy, and $P(t_k)$ is the probability that a randomly selected entity $e_k$ belongs to the set of entities having type $t_k$ as ancestor. It is worth mention here that YAGO's taxonomy is not a tree *per se* but a directed acyclic graph (DAG). Therefore, we select the *Lowest Common Ancestor* or *LCA* by first intersecting both sets of types $T(e_1)$ and $T(e_2)$, from where we get a collection of common ancestors. Then, recursively, we pick the common ancestor whose distance to a "root" is the maximum (counting edges), and consider it as the *LCA* we use in place of $t_0$.

Next, we can find the similarity from types $T(e_1)$ to $T(e_2)$ by using

$$Sim_{hr}\big(T(e_1) \to T(e_2)\big) = \frac{\sum_{t_1 \in T(e_1)} Sim_{hr}\big(t_1, \varepsilon(t_1)\big)}{|T(e_1)|}$$

And, finally,

$$Sim_{hr}(e_1, e_2) = \frac{Sim_{hr}\big(T(e_1) \to T(e_2)\big) + Sim_{hr}\big(T(e_2) \to T(e_1)\big)}{2}$$

is the required *type hierarchy based semantic similarity* measure that we need

***Distributional context similarity***

To define the context similarity of two entities we resort again to Wikipedia in order to retrieve their two corresponding Wikitexts. From these Wikitexts we collect monograms together with their frequencies to conform a vector $\mathbf{V}_e$ for entity $e$. Accordingly, we compute the *distributional context similarity* as the *cosine similarity* of both monograms vectors with the following formula:

$$Sim_{ds}(e_1, e_2) = \frac{\mathbf{V}_{e_1}}{\|\mathbf{V}_{e_1}\|} \cdot \frac{\mathbf{V}_{e_2}}{\|\mathbf{V}_{e_2}\|}$$

***Linking Quality Metric***

Now, given all measures we have listed, we can evaluate how good a set of candidate mappings $M$ is with respect to the similarity that each of them holds to one another. This linking goodness is expressed through the Linking Quality Metric, which is a convex, linear combination of the prior probability, the type hierarchy based similarity, and the distributional context similarity:

$$LQ(M) = \alpha * \sum_{s=1}^{|L|} P_{pr}(m_s) + \frac{\beta}{|L|-1} \sum_{s=1}^{|L|} \sum_{u=1,\, u \neq s}^{|L|} Sim_{hr}(m_s, m_u) + \frac{\gamma}{|L|-1} \sum_{s=1}^{|L|} \sum_{u=1,\, u \neq s}^{|L|} Sim_{ds}(m_s, m_u)$$

where $\alpha + \beta + \gamma = 1$ is a constraint on the weights to provide different levels of importance to each of the individual measurements. In our work, $\alpha$, $\beta$, and $\gamma$ are set 0.25, 0.45, and 0.3, respectively, in accordance to the experiments we conducted on a web list of 35 items.

## 3.4. Optimization via Simulated Annealing

The *Liking Quality Metric* is a function that retrieves the goodness of a selected group of candidate mappings for each surface name. Our purpose in this work is to find out which combination of candidate mapping entities yields the highest possible $LQ$. As a reference, Shen, W., et al, 2012 implemented the *Iterative Substitution Algorithm* to greedily improve the selection iteration by iteration. Though fast, their algorithm is fated to achieve $LQ$ local maxima. We avoid that by introducing *Simulated Annealing*. In our implementation, algorithm 1 describes this numerical method where all constants have been set to the same values used to solve the problem of the *Traveling Salesman* (Press, W., et al, 2007) since it shares the combinatorial nature with our case.

**SimulatedAnnealing**()

***Input***: Candidate mapping entity sets $R$ for all surface names in $L$
***Output***: Mapping entity list $M$

```
Set TEMP_FACTOR = 0.9                          //Annealing schedule
Set Temperature = 0.5                          //Annealing temperature
Set MaxChanges = 50 * |L|                      //Maximum number of changes at each temperature step
Set MaxSuccChanges = 5 * |L|                   //Limit for successful changes before continuing
Set TempSteps = 100                            //Maximum number of temperature adjustments

Set m_i = r_{i,0}                              //Initial mappings M by taking the first candidate for each l_i ∈ L
Set LQ_max = ComputeLQ( M )                    //Liking Quality Metric of initial M

For I = 0 to TempSteps
  Set nSucc = 0                                //Count number of successful changes

  For J = 0 to MaxChanges
    Set s = Rand( |L| )                        //A random surface name index
    If |R_s| = 1, continue loop                //Skip surface names with only one candidate

    Set c = Rand( |R_s| ), r_{s,c} ≠ m_s       //A random candidate for s^th surface name, not currently in M

    Set M_r = {M − m_s} ∪ r_{s,c}              //A tryout new set of mappings
    Set LQ = ComputeLQ( M_r )                  //Linking Quality of tryout mappings
    Set Diff = LQ − LQ_max

    If Diff > 0 or Rand( 1 ) < e^(Diff/Temperature)   //Metropolis decision algorithm
      Increment nSucc by one                   //A successful change
      Set LQ_max = LQ
      Set M = M_r                              //New set of mappings
    EndIf

    If nSucc = MaxSuccChanges                  //If it reached limit for successful changes
      Break loop

  EndFor

  Set Temperature = Temperature * TEMP_FACTOR  //Adjust temperature

  If nSucc = 0                                 //If no changes on this iteration
    Break loop

EndFor

EndFunction
```

*Algorithm 1, Simulated Annealing for finding the best set of mapping entities.*

## 4. EVALUATION

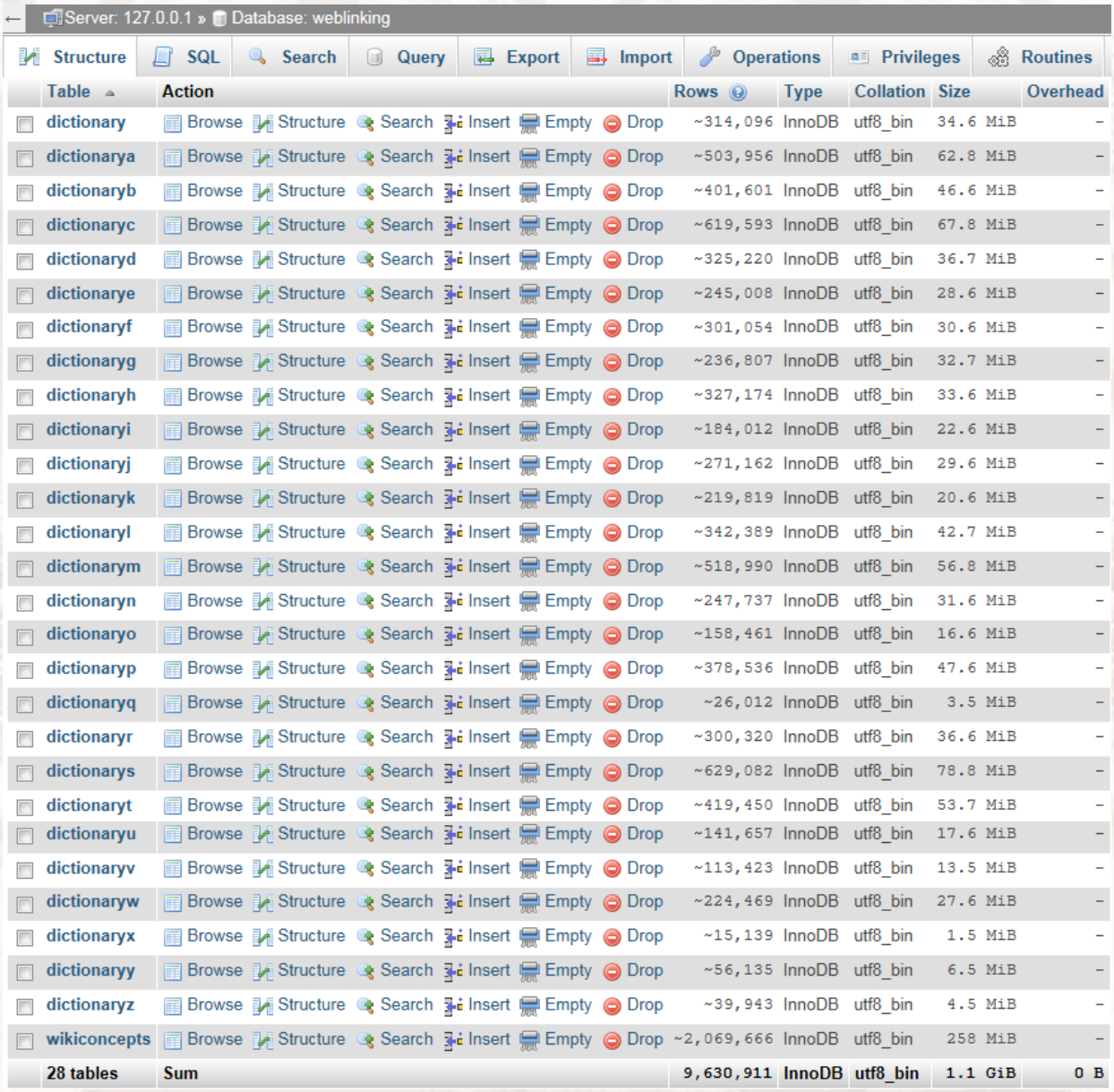
In order to adequately tackle the task of linking web entities we require a knowledge base. Since we are following the guidelines from the original implementation of LIEGE, there are two main sources of meta-information that we used to carry out the computation of the *Linking Quality Metric*:

- Wikipedia
- YAGO

Wikipedia is available for downloading for offline reading and research purposes. We obtained the Wikipedia dump from January 3, 2008, (MediaWiki, 2008), to perform the following activities:

- Build the dictionary.
- Compute the prior probability measure from the *count* information stored in the dictionary.
- Generate vectors of monograms to compute the *distributional context similarity* described in section 3.3.

On the same line as Wikipedia, YAGO has become an open web ontology that integrates taxonomic information from Wikipedia and WordNet in a more compact, yet expressive way. YAGO is available for free download, and we have used YAGO2s'




| Table | Action | Rows | Type | Collation | Size | Overhead |
|---|---|---|---|---|---|---|
| dictionary | Browse Structure Search Insert Empty Drop | ~314,096 | InnoDB | utf8_bin | 34.6 MiB | - |
| dictionarya | Browse Structure Search Insert Empty Drop | ~503,956 | InnoDB | utf8_bin | 62.8 MiB | - |
| dictionaryb | Browse Structure Search Insert Empty Drop | ~401,601 | InnoDB | utf8_bin | 46.6 MiB | - |
| dictionaryc | Browse Structure Search Insert Empty Drop | ~619,593 | InnoDB | utf8_bin | 67.8 MiB | - |
| dictionaryd | Browse Structure Search Insert Empty Drop | ~325,220 | InnoDB | utf8_bin | 36.7 MiB | - |
| dictionarye | Browse Structure Search Insert Empty Drop | ~245,008 | InnoDB | utf8_bin | 28.6 MiB | - |
| dictionaryf | Browse Structure Search Insert Empty Drop | ~301,054 | InnoDB | utf8_bin | 30.6 MiB | - |
| dictionaryg | Browse Structure Search Insert Empty Drop | ~236,807 | InnoDB | utf8_bin | 32.7 MiB | - |
| dictionaryh | Browse Structure Search Insert Empty Drop | ~327,174 | InnoDB | utf8_bin | 33.6 MiB | - |
| dictionaryi | Browse Structure Search Insert Empty Drop | ~184,012 | InnoDB | utf8_bin | 22.6 MiB | - |
| dictionaryj | Browse Structure Search Insert Empty Drop | ~271,162 | InnoDB | utf8_bin | 29.6 MiB | - |
| dictionaryk | Browse Structure Search Insert Empty Drop | ~219,819 | InnoDB | utf8_bin | 20.6 MiB | - |
| dictionaryl | Browse Structure Search Insert Empty Drop | ~342,389 | InnoDB | utf8_bin | 42.7 MiB | - |
| dictionarym | Browse Structure Search Insert Empty Drop | ~518,990 | InnoDB | utf8_bin | 56.8 MiB | - |
| dictionaryn | Browse Structure Search Insert Empty Drop | ~247,737 | InnoDB | utf8_bin | 31.6 MiB | - |
| dictionaryo | Browse Structure Search Insert Empty Drop | ~158,461 | InnoDB | utf8_bin | 16.6 MiB | - |
| dictionaryp | Browse Structure Search Insert Empty Drop | ~378,536 | InnoDB | utf8_bin | 47.6 MiB | - |
| dictionaryq | Browse Structure Search Insert Empty Drop | ~26,012 | InnoDB | utf8_bin | 3.5 MiB | - |
| dictionaryr | Browse Structure Search Insert Empty Drop | ~300,320 | InnoDB | utf8_bin | 36.6 MiB | - |
| dictionarys | Browse Structure Search Insert Empty Drop | ~629,082 | InnoDB | utf8_bin | 78.8 MiB | - |
| dictionaryt | Browse Structure Search Insert Empty Drop | ~419,450 | InnoDB | utf8_bin | 53.7 MiB | - |
| dictionaryu | Browse Structure Search Insert Empty Drop | ~141,657 | InnoDB | utf8_bin | 17.6 MiB | - |
| dictionaryv | Browse Structure Search Insert Empty Drop | ~113,423 | InnoDB | utf8_bin | 13.5 MiB | - |
| dictionaryw | Browse Structure Search Insert Empty Drop | ~224,469 | InnoDB | utf8_bin | 27.6 MiB | - |
| dictionaryx | Browse Structure Search Insert Empty Drop | ~15,139 | InnoDB | utf8_bin | 1.5 MiB | - |
| dictionaryy | Browse Structure Search Insert Empty Drop | ~56,135 | InnoDB | utf8_bin | 6.5 MiB | - |
| dictionaryz | Browse Structure Search Insert Empty Drop | ~39,943 | InnoDB | utf8_bin | 4.5 MiB | - |
| wikiconcepts | Browse Structure Search Insert Empty Drop | ~2,069,666 | InnoDB | utf8_bin | 258 MiB | - |
| **28 tables** | **Sum** | **9,630,911** | **InnoDB** | **utf8_bin** | **1.1 GiB** | **0 B** |

*Figure 3, the weblinking database viewed in phpMyAdmin web tool.*

Taxonomy, Types, and Transitive Themes as of February 14, 2014 (YAGO, 2014) to perform the computation of the *type hierarchy based similarity*. Referring back to section 3.3, the latter similarity measure is linearly combined with the *distributional context similarity* in order to obtain the notion of *coherence* for the *Linking Quality Metric*.

We have also used open source Python code (Jodaiber, 2011, and MediaLab, 2013) to parse the *.XML dump file from Wikipedia into JSON objects and other clean formatted files.

Our software implementation for linking web entities is written completely in Java from the reparsing process for dumping information into the dictionary, to the actual computation of the *Linking Quality* and the programming of *Simulated Annealing*. We have also integrated a graphical user interface, so that it is possible to type up to 8 surface names of similar entities for our software to figure out the real mappings from the set of generated candidates.

Furthermore, we have used MySQL to efficiently access the dictionary and the YAGO's taxonomy in a local machine. With this server we created 2 databases:

- **weblinking**, which contains a dictionary split into 27 tables (one per leading English alphabetic letter plus an additional table for surface names that begin with other non-alphabetical characters) and a table of *Wikiconcepts* (e.g. Wikipedia page titles). The latter table stores the paths to files that hold the

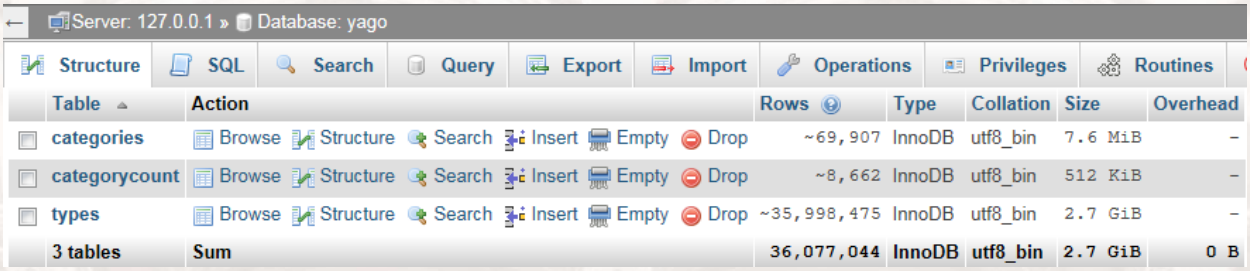



*Figure 4, the yago database viewed in phgMyAdmin web tool.*

Wikitext for each real world entity. This way, we avoid overwhelming our MySQL database with actual textual content. The use of indexes has helped us speed up access to dictionaries and to the *count* value to compute the *prior probabilities* as well.

- **yago**, which contains three tables: one for YAGO's taxonomy, holding who is parent of who within the hierarchy of Wordnet categories (we have removed Wikipedia categories); one for categories count, holding the different categories and the number of entities that belong to each type; and another one for types, having all categories (transitive relations inclusive) that each real world entity belongs to.

Figures 3 and 4 show screenshots of the *phpMyAdmin* tool listing the two aforementioned databases. The final sizes of both databases were estimated as 1.1 GB for *weblinking* and 2.7 GB for *yago*.

Unlike the work of Shen, W., et al., 2012, due to time constraints, we could not prepare training data sets to run an SVM in order to find out the real values for the weights $\alpha$, $\beta$, and $\gamma$. We instead used experimental constants that showed acceptable performance at annotating 35 literary masterpieces. Further details on this test are given in section 5.

To evaluate the performance of our LIEGE version, we have collected the surface names of 200 literary masterpieces (including novels, tales, and other books) to create our own web list. Furthermore, to check for accuracy, we have compared the number of correctly identified real world entities versus the expected number of annotated entities. We have run our system in a PC with Intel Core i5® and Windows 7 x64, using Eclipse, which was the platform we employed to develop the software.

## 5. Results

Since we could not prepare training data for an SVM, we approximated good values for the weight parameters in the *Linking Quality Function* by trial and error on a collection of 35 surface names. These named entities were extracted from the 200 literary masterpieces we collected. Figure 5 shows the "training" set we employed for this task, together with the expected mappings and the predicted real world entities. In general, only 1 out of the 35 samples was incorrectly annotated, which represents a (naïve) accuracy of 0.97.

In all of our experiments, we tried both the LIEGE's *Iterative Substitution Algorithm* and our *Simulated Annealing* approach. Particularly, in small sets like those shown in figures 2 and 5, both algorithms had the same performance and made the same errors, which pointed out that, despite all, the *Iterative Substitution Algorithm*, besides being faster than *Simulated Annealing*, is also a good, greedy optimizer. Moreover, the execution time for *Simulated Annealing* was always double or more than that of the *Iterative Substitution Algorithm*. We believe this "flaw" of the numerical method comes from its exploratory nature because instead of choosing only improving options, sometimes it goes down the hill in pursuit of a hidden global maximum.

During our experiments we also noticed that the computation of the *type hierarchy based similarity* is very expensive because for two candidate mappings to compare it has to retrieve their type sets, intersect them, build the taxonomic "sub-DAG" of such intersection, and find the *LCA*. For this reason, we cached this similarity values computed for pairs of entities into hash maps so that later comparisons of the same candidate mappings would query the hash map before going through the whole process to obtain the *type hierarchy based similarity*. In general, this technique saved us half the time for both implemented algorithms, but it did not boost *Simulated Annealing* beyond since this method kept visiting new pairs of entities more often than the *Iterative Substitution Algorithm*. These "random walks" paid the price of performance time because of the frequent computation of the similarity of new pairs of candidate mappings.

After setting the weight parameters to $\alpha = 0.25$, $\beta = 0.45$, and $\gamma = 0.3$, we run our system with both *Iterative Substitution Algorithm* and *Simulated Annealing* over 100 list items, again extracted from

| Surface Names | Real Mappings | Predicted Mapping Entities |
|---|---|---|
| Absalom | Absalom,_Absalom! | Absalom,_Absalom! |
| Adventures of Huckleberry Finn | Adventures_of_Huckleberry_Finn | Adventures_of_Huckleberry_Finn |
| Alice in Wonderland | Alice's_Adventures_in_Wonderland | Alice's_Adventures_in_Wonderland |
| Anna Karenina | Anna_Karenina | Anna_Karenina |
| Beloved | Beloved_(novel) | Beloved_(novel) |
| Birdsong | Birdsong_(novel) | Birdsong_(novel) |
| Don Quixote | Don_Quixote | Don_Quixote |
| Dracula | Dracula | Dracula |
| Gulliver's Travels | Gulliver's_Travels | Gulliver's_Travels |
| Little Women | Little_Women | Little_Women |
| Madame Bovary | Madame_Bovary | Madame_Bovary |
| Moby Dick | Moby-Dick | Moby-Dick |
| Nineteen Eighty-Four | Nineteen_Eighty-Four | Nineteen_Eighty-Four |
| Nostromo | Nostromo | Nostromo |
| One Hundred Years of Solitude | One_Hundred_Years_of_Solitude | One_Hundred_Years_of_Solitude |
| Pedro Paramo | Pedro_Páramo | Pedro_Páramo |
| Robinson Crusoe | Robinson_Crusoe | Robinson_Crusoe |
| The Adventures of Tom Sawyer | The_Adventures_of_Tom_Sawyer | The_Adventures_of_Tom_Sawyer |
| The Autumn of the Patriarch | The_Autumn_of_the_Patriarch | The_Autumn_of_the_Patriarch |
| The Call of the Wild | The_Call_of_the_Wild | The_Call_of_the_Wild |
| The Castle | The_Castle_(novel) | The_Castle_(novel) |
| The Count of Monte Christo | The_Count_of_Monte_Cristo | The_Count_of_Monte_Cristo |
| The Divine Comedy | Divine_Comedy | Divine_Comedy |
| The Hobbit | The_Hobbit | The_Hobbit |
| The Picture of Dorian Gray | The_Picture_of_Dorian_Gray | The_Picture_of_Dorian_Gray |
| The Portrait of a Lady | The_Portrait_of_a_Lady | The_Portrait_of_a_Lady |
| The Summons | The_Summons | The_Summons |
| The Sun also Rises | The_Sun_Also_Rises | The_Sun_Also_Rises |
| The Trial | The_Trial | The_Trial |
| The Turn of the Screw | The_Turn_of_the_Screw | The_Turn_of_the_Screw |
| To Kill a Mockingbird | To_Kill_a_Mockingbird | To_Kill_a_Mockingbird |
| Tropic of Cancer | Tropic_of_Cancer_(novel) | Tropic_of_Cancer_(novel) |
| Ulysses | Ulysses_(novel) | Ulysses_(novel) |
| Waiting for Godot | Waiting_for_Godot | Waiting_for_Godot |
| Winnie-the-Pooh | Winnie-the-Pooh_(book) | Winnie-the-Pooh |

*Figure 5, a small training set to determine the parameter weights in the Linking Quality Metric. Notice that candidate mapping entities are expressed in the form of Wikipedia entities, with underscores instead of spaces.*

the manually annotated set of 200 literary masterpieces. Results are depicted in figure 6.

From figure 6 we can observe that the benefits of *Simulated Annealing* are not yet seen mainly because our test case is not large enough to make the *Iterative Substitution Algorithm* get stuck in a local maximum. Also, we notice that, with an increased number of candidate mappings, the annotation time grows exponentially (from 2,051 seconds for annotating 35 surface names, to 15,484.4 seconds for 100 named entities), which is a grave constraint for *Simulated Annealing*… not yet for the *Iterative Substitution Method*. For this reason, testing our system over the 200 hundred manually annotated books/novels would be prohibited within the time frame we were allotted.

## 6. Discussion

We have presented our implementation of LIEGE, a framework for annotating entities in web lists, based on the work of Shen, W., et al, 2012. We aggregated a numerical method, known as *Simulated Annealing*, to solve the combinatorial problem of choosing the best possible set of candidate mappings with respect to the *Linking Quality Metric*.

Although we expected to achieve better accuracy rate than the one Shen, W., et al, 2012 obtained, our results demonstrated that their approach with the *Iterative Substitution Algorithm*, though greedy, performs well at annotating web lists of small size. Due to time constraints, we were not able to run the linking task on larger sets with *Simulated Annealing* because our experiments indicated that such numerical method would take more than twice time to converge than the *Iterative Substitution Algorithm*. The prohibitive execution time comes from the expensive computation of *type hierarchy based similarity* measure. Had we removed it from the Linking Quality Metric, we would have obtained

| | Iterative Substitution Algorithm | Simulated Annealing |
|---|---|---|
| *Annotation time (seconds)* | 7,891.3 | 15,484.4 |
| *Correct mappings* | 93 | 93 |
| *Wrong mappings* | 7 | 7 |
| *Accuracy* | 0.93 | 0.93 |

*Figure 6, performance of the Iterative Substitution Algorithm versus Simulated Annealing.*

poorer mapping results since in web lists the hierarchical similarity plays an important role due to the lack of local context information that is present in web texts (Shen, W., et al, 2012 - LINDEN) or microblogs (Shen, W., et al, 2013).

Additionally, we decided to manually tune the weight parameters for the linear combination expression of the *Linking Quality*. The main restriction in our case was time allotted to complete our work, and we could not generate our sets for training a SVM. Regardless of such flaw, we believe our framework is, at least, as robust as the original LIEGE in accordance to the results we collected in section 5. The main contribution, though, in our implementation, is the introduction of *Simulated Annealing*, which theoretically should outperform the *Iterative Substitution Algorithm* when the input web list is sufficiently large and with many candidate mapping entities for each surface name.

We would also like to note some other problems we run into, which consumed some time mainly when building the dictionaries. Being Wikipedia too large and with multi-language-based text (despite the fact that we worked specifically with the English version of it), we came across several encoding problems because, obviously, ASCII was not sufficient, nor was Unicode. So, it took us quite a while to work around this issue by configuring everything (from developing platform to dictionary tables and test cases) into UTF-8. We therefore recommend all future work based on our idea to follow our coding (data transmissions) and decoding (texts and data reception) schemes to UTF-8. This would save lots of time at debugging, especially because the work horse of this framework relies on strings matching and comparison.

As of future work, we would like to, first, create real training data sets to use SVM and tune the weighting parameters of the *Linking Quality Metric*. It is also possible to take this idea of mapping entities in web lists to languages other than English. However, ones has to be able to find an ontology that categorizes all entities in that language… in other words, there should exist a YAGO equivalence for the targeted language. Nowadays, there are other web resources for various languages; for instance, we have a Wikipedia version in Spanish, whose equivalent YAGO ontology might be matched to the *Real Academia de la Lengua Española*, which is the maximal authority in Spanish grammar, with presence on the web nowadays.

To wrap up, linking entities in web lists is a time consuming task giving the current tools because, from our demonstrations, we are required to run the annotating framework on multicore platforms if we aim to achieve real time speeds. Yet, the principles have been established, and we would like to see these NLP-Big Data Analytic resources available for regular computing power somewhere in the near future.

## 7. REFERENCES


BUNESCU, R., AND PASCA, M. 2006. Using Encyclopedic Knowledge for Named Entity Disambiguation. *In Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06): 9-16.*

CUCERZAN, S. 2007. Large – Scale Named Entity Disambiguation Based on Wikipedia Data. *In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning: 708 – 716.*

JODAIBER, 2011. Annotated WikiExtractor. *Available from http://github.com/jodaiber/Annotated-WikiExtractor as for February 7, 2014.*

KULKARNI, S., SINGH, A., RAMAKISHNAN, G., AND CHAKRABARTI, S. 2009. Collective Annotation of Wikipedia Entities in Web Text. *In Proceedings of KDD '09: 457 – 465.*

LIMAYE, G., SARAWAGI, S., AND CHAKRABARTI, S. 2010. Annotating and Searching Web Tables Using Entities, Types, and Relationships. *In Proc. VLDB Endow., 3: 1338 – 1347.*

PRESS, W., TEUKOLSKY, S., VETTERLING, W., AND FLANNERY, B. 2007. Numerical Recipes: The Art of Scientific Computing. *Third Edition. Cambridge University Press. 549 – 552.*

SHEN, W., WANG J., LUO P., AND WANG, M. 2013. Linking Named Entities in Tweets with Knowledge Base Via User Interest Modeling. *In Proceedings of SIGKDD '13. 68 – 76.*

SHEN, W., WANG J., LUO P., AND WANG, M. 2012. LIEGE: Link Entities in Web Lists with Knowledge Base. *In Proceedings of SIGKDD '12, 1424 – 1432.*

SHEN, W., WANG, J., LUO, P., AND WANG, M. 2012. LINDEN: Linking Named Entities with Knowledge Base Via Semantic Knowledge. *In Proceedings of WWW '12, 449 – 458.*

MEDIALAB, 2013. WikiExtractor. *Available from http://medialab.di.unipi.it/wiki/Wikipedia_Extractor as of February 7, 2014.*

MEDIAWIKI. 2008. English Wikipedia Dump Archive from January 3, 2008. *Data set available at http://dumps.wikimedia.org/archive/enwiki/20080103/ as of February 7, 2014.*

YAGO, 2014. Taxonomy, Types, and Transitive Themes. *Data sets available at http://www.mpi-inf.mpg.de/yago-naga/yago/downloads.html as of February 14, 2014.*