# An Integrated Approach for Large-scale Relation Extraction from the Web

Naimdjon Takhirov[1], Fabien Duchateau[2], Trond Aalberg[1], Ingeborg Sølvberg[1]

[1] Norwegian University of Science and Technology, NO-7491 Trondheim, Norway
{takhirov,trondaal,ingeborg}@idi.ntnu.no
[2] Université Lyon 1, LIRIS, UMR5205, Lyon, France
fduchate@liris.cnrs.fr

**Abstract.** Deriving knowledge from information stored in unstructured documents is a major challenge. More specifically, binary relationships representing facts between entities can be extracted to populate semantic triple stores or large knowledge bases. The main constraint of all knowledge extraction approaches is to find a trade-off between quality and scalability. Thus, we propose in this paper SPIDER, a novel integrated system for extracting binary relationships at large scale. Through series of experiments, we show the benefit of our approach, which in general, outperforms existing systems both in terms of quality (precision and the number of discovered facts) and scalability.

**Keywords:** Relation Extraction, Knowledge Bases, Web Mining

## 1 Introduction

Information available on the Web has the potential to be a great source of structured knowledge. However this potential is far from being realized. The main benefit of obtaining exploitable facts such as relationships between entities from natural language texts is that machines can automatically interpret them. The automatic processing enables advanced applications such as semantic search, question answering and various other applications and services. The **Linked Open Data** (LOD) aims at making the vision of creating a large structured database a reality. In this domain, the building of semantic knowledge bases such as DBpedia, MusicBrainz or Geonames is (semi-)automatically performed by adding new facts which are usually represented by triples. However, most of these triples express a simple relationship between an entity and one of its properties, such as the *birthplace of a person* or the *author of a book*. By mining structured and unstructured documents from the Web, one can provide more complex relationships such as *parodies*. A different vision known as the **web of concepts** shares similar objectives with LOD [12]. As a consequence, knowledge harvesting [9, 10, 14] and more generally open-domain information extraction [5] are emerging fields with the goal of acquiring knowledge from textual content.

In this paper, we propose a relation extraction approach named **SPIDER**[3]. It aims at addressing the previously mentioned issues by integrating the most relevant techniques to generate trustworthy patterns and relationships. SPIDER is based on a perpetual process of generating patterns and examples either in supervised or unsupervised mode. The main intuition is that generic patterns which are derived from similar sentences and which are discovered in trustworthy documents are useful for detecting relationships in other documents. In summary, our contributions in this paper are:

[3] **S**emantic and **P**rovenance-based **I**ntegration for **D**etecting and **E**xtracting **R**elations

– Extracting relationships from a Web-scale source is a major bottleneck. To the best of our knowledge, SPIDER is the first approach which does not require several days to perform a single iteration.
– Contrary to most knowledge extraction tools, we tackle the problem of uniquely identifying entities to extend their list of spelling forms and to facilitate the matching to LOD.
– SPIDER includes a flexible pattern definition scheme. This scheme is used to merge similar patterns for efficiency purposes. In addition, we introduce the notion of confidence score that controls the ranking of patterns. The confidence score evolves over time as the system runs continuously.
– Experiments confirm the benefits of our approach w.r.t. similar tools (*ReadTheWeb*, *Prospera*) in terms of quality and performance.

## 2 Overview

### 2.1 Problem Definition

The overall goal of SPIDER is to continuously generate relationships and patterns. Let us first define a **relationship**. It is a triplet <$e_1$, $\tau$, $e_2$> where $e_1$ and $e_2$ represent **entities** and $\tau$ stands for a **type of relationship**. An entity might be represented with different labels in natural language text, and we note $l_e \in \mathcal{L}$ one of the mentions for the entity $e$. An example of a relationship is *createdBy* between the work entity *The Lord of the Rings* and the person entity *J.R.R. Tolkien*. Note that both the entities and the type of relationship are uniquely identified using an URI. An **example** denotes a **pair of entities** ($e_1$, $e_2$) which satisfies a type of relationship.

The patterns are extracted from **a collection of documents** $\mathcal{D} = \{d_1, d_2, ..., d_n\}$. Although not limited to, these documents are webpages in our context. Each document is composed of **sentences**, which may contain mentions of the two entities. In that case, the sentence is extracted as a **candidate pattern**. We note $\mathcal{CP}$ the set of candidate patterns given a collection of documents and a set of initial examples. Each candidate pattern is defined as a tuple $cp = \{t_b, e_1, t_m, e_2, t_a\}$ with $t_b$, $t_m$ and $t_a$ respectively standing for the *text before*, the *text in the middle* and the *text after* the entities. A sentence *"Bored of the Rings is a parody of Lord of the Rings"* is transformed to a corresponding candidate pattern *{"", $e_1$, "is a parody of", $e_2$, ""}*.

From the set of candidate patterns $\mathcal{CP}$, we derive a set $\mathcal{P}$ of **generic patterns** by applying a strategy $s$. A **strategy** is defined as a sequence of **operations** $s =< o_1, o_2, ..., o_k >$, each operation aiming at generalizing the candidate patterns. Namely, this generalization implies the detection of frequent terms and the POS-tagging of the other terms of the candidate patterns. Thus, a strategy is a function such that $s(\mathcal{CP}) \rightarrow \mathcal{P}$. All generated patterns are associated to a specific type of relationship $\tau$. For instance, a generic pattern for the *parody* type is illustrated with *"{$e_1$} is/VBZ a/DT {parody, illusion, spoof} of/IN {$e_2$}"* [4].

Finally, a pattern and an example both have a **confidence score** noted $conf_p$ and $conf_e$ respectively. This score is based on the support, the provenance, the number of occurrences, the number of strategies and the iteration. A **pattern similarity** metric indicates the proximity between two (candidate) patterns. The notion of confidence score, as well as the one for operation, strategy, pattern similarity and (candidate) pattern, are further detailed in the next sections.

[4] VBZ=Verb, 3rd person sing. present, DT=Determiner, IN=Preposition or subordinating conjunction

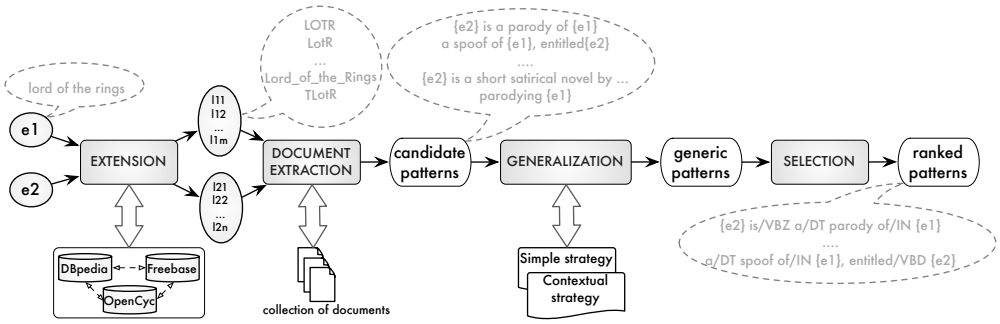



**Fig. 1.** The Pattern Generation Process

### 2.2 Workflow

Given two labels, SPIDER generates patterns and derives to a relationship. This pattern generation capability is guaranteed by the following two processes:

- **Pattern Generation** is in charge of detecting candidate patterns by using examples or provided entities and of generalizing these candidate patterns to obtain patterns for a given type of relationship (see Figure 1).
- **Example Generation** exploits the previously generated patterns in order to discover new examples which satisfy the type of relationship.

The **knowledge base** stores all generated examples and patterns. These examples can be used to maintain the system continuously running, but they can be exploited from a user perspective too.

## 3 Pattern Generation

The pattern generation process either requires a few examples for a given type of relationship so that patterns for this type of relationship can be automatically generated (supervised), or it directly tries to guess the type of relationship for two given labels (unsupervised). The process is similar in both modes and it is composed of three main steps: extension of entities, extraction of candidate patterns from the collection of documents and their refinement into patterns.

### 3.1 Extending Entities

In a document, entities are not uniquely identified by a label but they have alternative labels or spelling forms. Therefore, extending these entities with their alternative labels is a crucial step and it requires the correct identification of the entity. For instance, the entity *"Lord of the Rings"* can be labeled *"LOTR"* or *"The Lord of the Rings"*. To avoid missing potentially interesting relationships, we search for these alternative forms of spelling in the documents. Given an entity $e$ represented by a label $l$, the goal is to discover its set of alternative labels $\mathcal{L}_e = \{l, l^1, l^2, ..., l^n\}$. The idea is to match the entity against LOD semantic knowledge bases to obtain this list of alternative labels. Namely, we build various queries by decomposing the

initial label and we query in the aliases attributes of knowledge bases (i.e., *common.topic.alias* for Freebase, *wikiPageRedirects* for DBpedia, etc.). In most cases, several candidate entities are returned and the tool tries to automatically select the correct one.

The process of automatically selecting the correct entity is achieved as follows. First, an `AND` query is constructed with the two labels. Clusters of documents are built representing documents belonging to a set of specific type of entities. The $n$ number of words around labels are extracted and stemming performed on words. Our assumption is based on the fact that documents mentioning the same entities tend to have similar words. Therefore, a graph of semantically related words is built. The most important documents in the cluster are then compared against the abstract of the automatically selected entities. Next, we extract frequent terms from the most important documents in the result set and use these frequent words as extensions. Note that if disambiguation is not possible, we discard the example and we do not use it for subsequent pattern generation. The result of the extension process is a list of alternative labels as illustrated in Figure 1.

The main issue in this step deals with the absence of the two labels in any knowledge base, which means that the entities cannot be extended. The number of retrieved documents in that case could not be sufficient to extract good candidate patterns. The first solution consists of analyzing these retrieved documents to detect potential alternative spellings by applying metrics such as tf-idf and Named Entity Recognition techniques. Another possibility is to relax the similarity constraint when searching a label in a knowledge base. In other words, a strict equality measure would not be applied between the label and the candidate spelling forms from a knowledge base. Rather n-grams or Levenshtein similarity metrics with a high threshold would be a better choice.

### 3.2 Extraction of Candidate Patterns

As depicted in Figure 1, the outcome of document extraction process is candidate patterns. Given the lists of extended labels for both entities, our tool associates all alternative labels of the first entity to all labels of the second entity (Cartesian product) to build different queries. The documents resulting from these queries are ranked according to their relevance score. The candidate patterns are extracted by parsing these documents and locating the sentences with co-occurrence of both entities (defined by a maximum number of words between them, currently 15 words). Note that we include in the candidate patterns the text before and after the entities to obtain full sentences. The final step aims at refining the candidate patterns to obtain patterns.

### 3.3 Selection of Patterns

The last issue deals with the selection or ranking of the generic patterns. Thus, a **confidence score** noted $conf_p$ is computed for each pattern $p$ with Formula 1. Our intuition is to exploit all information which allowed the discovery of the patterns and to compare a pattern with the ones of the same type of relationship.

$$conf(p) = \left( \frac{\alpha sup_p + \beta occ_p + \gamma prov_p}{\alpha + \beta + \gamma} \right) \quad (1)$$

The **support** $sup_p$ is defined as the ratio between the number of examples $ex_p$ that this pattern is able to discover and the total number of examples $ex_\tau$ discovered by all patterns of the same type of relationship $\tau$. Note that the support cannot be computed at the first iteration.

Similarly, the **occurrency** $occ_p$ stands for the number of candidate patterns which led to the generation of the pattern $p$. It is normalized by the total number of candidate patterns used to generalize all patterns of the same type of relationship $\tau$.

$$supp_p = \frac{ex_p}{ex_\tau} \qquad\qquad occ_p = \frac{occ_p}{occ_\tau}$$

The **provenance** $prov_p$ refers to the relevance of the documents from which the candidate patterns which generalize a given pattern have been extracted. The relevance is evaluated given three metrics: the relevance score namely applies tf-idf on the content of the document and its values are bounded by a maximal value which depends on the query. PageRank[5] is widely known due to the Google search engine. The PageRank scores of our collection are in the range $[0.15, 10]$. Finally, SpamScore indicates the probability that a document is a spam or not [8]. The idea is to average the scores returned by these three metrics for all documents from which patterns have been derived. We note $\mathcal{D}_p$ this set of documents for the pattern $p$, and $d_p^i$ a document of this set. The following formula computes a score in the range $[0, 1]$ to evaluate the average relevance of this set of documents, and thus the provenance of the pattern:

$$prov_p = \frac{\sum^{d_p^i} \frac{1}{3} \left( \frac{relevance(d_p^i)}{max(relevance(\mathcal{D}_p))} + \frac{spamscore(d_p^i)}{100} + \frac{prank(d_p^i)}{10} \right)}{|\mathcal{D}_p|}$$

## 4 Relationship and Example Discovery

In the previous step, we have generated patterns for a given type of relationship. SPIDER aims at discovering relationships between entities and discovering new examples.

### 4.1 Challenges for Exploiting Patterns

Using the generated patterns to discover knowledge from plain texts involves the addressing of the following two challenges: pattern similarity and NER.

**Pattern similarity**. When analyzing sentences in a document, SPIDER needs to evaluate the similarity between a sentence and a pattern. Thus, we have designed a pattern similarity metric. The intuition which underlies our metric is twofold: (i) the presence of frequent terms in the sentence is crucial while there is more flexibility for less important terms and (ii) the position of the words should be taken into account. First, the sentence is transformed (cleaning, POS-tagging and replacing the mentions of the entities) so that both the sentence $s$ and the pattern $p$ are composed of POS-tagged terms. The pattern may also include a list of frequent terms, which is noted $\mathcal{FT}_p$. The idea is to compute $\Delta$, the minimal total distance to transform the sentence into the pattern. Thus, our metric is an adaptation of the Levenshtein distance [7]. However, we do not compute a number of operations (delete, transform or add a term) between two words or characters, but rather we evaluate the semantic distance between two words. Given the $i^{th}$ word $w_p^i$ associated to the tag $t_p^i$ in the pattern $p$ and the word $w_s^j$ with the tag $t_s^j$ in the sentence $s$, we compute their semantic distance $semdist(w_p^i, w_s^j)$ using the following formula:

$$semdist(w_p^i, w_s^j) = \begin{cases} 0.0 & \text{if } w_s^j \in \mathcal{FT}_p \\ resnik(w_p^i, w_s^j) & \text{if } w_s^j \notin \mathcal{FT}_p \text{ and } t_p^i = t_s^j \\ 1.0 & \text{otherwise} \end{cases} \quad (2)$$

[5] `http://j.mp/Clueweb09-Pagerank`

Namely, the distance is equal to 0 if the word in the sentence is a frequent term. POS-tagged terms whose tags are identical (e.g., two verbs) have a similarity obtained by applying the Resnik distance in Wordnet [13]. Else, words with different tags have the maximal distance. The minimal total distance $\Delta(p, s)$ between a pattern $p$ and a sentence $s$ is computed by the matrix algorithm of the Levenshtein distance[6] using the semantic distance for all pairs of words. The distance is normalized in the range $[0, 1]$ with Formula 3 which assesses the similarity $pattsim$.

$$pattsim(p, s) = \frac{1}{1 + \Delta(p, s)} \tag{3}$$

Our metric is flexible because extra or missing words in a sentence do not significantly affect the similarity value. Similarly, less important words are mainly compared on their nature (POS-tag). Finally, we can select the sentences which are modeled by a pattern according to a threshold.

**NER**. When a sentence in a document corresponds to a pattern, our approach needs to identify and extract entities contained in the sentence. Thus, the NER issue is crucial as it determines (part of) the output. Indeed, it is necessary to correctly identify the (labels of) entities in the sentence based on a pattern. To solve this issue, we rely on the formalism of our patterns: since they have been POS-tagged, the tags serve as a delimiter and may constraint the candidate entities.

### 4.2 Discovering the Type of Relationship

Given two labels (representing an entity), the goal is to determine the possible type(s) of relationships between these two entities. The two entities are first extended to obtain their alternative labels (see Section 3.1). A set of documents is analyzed to extract all sentences which contain one label of each entity, and these sentences are then compared to all patterns stored in SPIDER's knowledge base (see Section 4.1). If a sentence is similar to a trustable pattern (with a sufficient confidence score), then the type of relationship corresponding to this pattern is proposed to the user. Note that if there is no trustable pattern in the knowledge base, the user has the possibility to provide training data to the system.

### 4.3 Discovering New Examples

The discovery of new examples is at the basis of the *never-ending* feature which enables the feeding of the knowledge base with additional training data for generating new patterns. SPIDER selects in the knowledge base the patterns of a given type of relationship. It retrieves a set of documents by querying each de-tagged pattern (or only their frequent terms). We compute the similarity between a pattern and the sentences of the documents which include a frequent term of this pattern. If the sentence is modeled by the pattern, then we apply the NER techniques for discovering the two entities. Note that each discovered example has a confidence score, which is computed with the same formula as in Section 3.3 for the confidence of a pattern, except that $sup_e$ replaces $sup_p$ and it indicates the number of patterns which discovered this example. Examples with a very low confidence are discarded while others are stored in the knowledge base.

[6] `http://j.mp/Levenschtein`

## 5 Scalability

In order to efficiently process documents, we distribute the jobs into several machines. MapReduce inspired techniques have been popular to tackle such tasks. Therefore, the collection is indexed with Hadoop enabling efficient indexing and searching. To compute statistics, SPIDER makes use of Pig[7] which is a high level platform for analyzing a large collection of data.

Additionally, we propose a document partition to incrementally provide results on a subset of the collection. The general idea is that highly ranked documents should be a better source for obtaining patterns than those with lower PageRank, SpamScore and relevance score values. The size of a partition depends on the quality of the obtained patterns and examples. SPIDER is able to automatically tune the ideal size of a partition. Initially, the documents are sorted by their PageRank, SpamScore and the relevance score as described above. The top $k$ documents are selected for analysis from the head of the ranked list of documents. For the $i$-th round, the cursor is moved to the range $[k, i \times k]$ and the documents in that range are picked out for analysis. Furthermore, when there are too few patterns discovered for two given labels out of these initial set of documents, the partition size is subsequently adjusted to a higher number. The combination of scores as well as the partitioning mechanism makes obtaining the URLs of the documents and their content for a given query fast, i.e., it only requires a few seconds. This efficiency is demonstrated in Section 7.2.

## 6 Related Work

Relationship extraction has been widely studied in the last decade. **Supervised systems** for relationship extraction are mainly based on kernel methods, but they suffer from the processing costs and the difficulty for annotating new training data. One of the earliest semi-supervised system, **DIPRE**, uses a few pairs of entities for a given type of relationship as initial seeds [3]. It searches the Web for these pairs to extract patterns representing a relationship, and use the patterns to discover new pairs of entities. These new entities are integrated in the loop to generate more patterns, and then find new pairs of entities. **Snowball** [1] enhances DIPRE in two different directions. First, a verification step is performed so that generated pairs of examples are checked with MITRE named entity tagger. Secondly, the patterns are more flexible because they are represented as vectors of weighted terms, thus enabling the clustering of similar patterns.

**Espresso** [11] is a weakly-supervised relation extraction system that also makes use of generic patterns. The system is efficient in terms of reliability and precision. However, experiments were performed on smaller datasets and it is not known how the system performs at Web-scale.

**TextRunner** brought further perspective in the field of Open Information Extraction, for which the types of relationships are not predefined [2]. A self-supervised learner is in charge of labeling the training data as positive or negative, and a classifier is then trained. Relations are extracted with the classifier while a redundancy-based probabilistic model assigns confidence scores to each new examples. The system was further developed into a **ReVerb** framework [6] which improves the precision-recall curve of the TextRunner.

[7] `http://pig.apache.org`

**ReadTheWeb / NELL** [4] is another project that aims at continuously extracting categories (e.g., the type of an entity) and relationships from web documents and improving the extraction step by means of **learning** techniques. Four components including a classifier and two learners are in charge of deriving the facts with a confidence score. According to the content of the online knowledge base, more iterations provide high confidence scores (almost $100\%$) for irrelevant relationships. Contrary to NELL, we do not assume that one document returning a high confidence score for a given relationship is sufficient for approving this relationship. In addition, NELL is mainly dedicated to the discovery of categories ($95\%$ of the discovered facts) rather than relationships between entities.

A recent work reconciles three main issues in terms of precision, recall and performance [10]. Indeed, **Prospera** utilizes both pattern analysis with n-gram item sets to ensure a good recall and rule-based reasoning to guarantee an acceptable precision. The performance aspect is handled by partitioning and parallelizing the tasks in a distributed architecture. A restriction of this work deals with the pattern which only covers the middle text between the two entities. This limitation affects the recall, as shown with the example *"Lord Of The Rings, which Tolkien has written"*.

Contrary to the oldest systems which include hard representations of patterns, Prospera and SPIDER includes a more **flexible definition of the patterns**, so that similar patterns can be merged. In addition, the patterns are at the sentence level, which means that the texts before, after and between the entities are considered. Our **confidence score** for a pattern or an example takes into account crucial criteria such as provenance. In addition, the support and the occurrence scores are correlated within the same type of relationship, thus the confidence in a pattern or an example may decrease over time. To the best of our knowledge, none of these approaches deal with the issue of **identifying the alternative labels** of an entity. In the future, we plan to demonstrate the impact of these alternative labels on the recall. Finally, our approach is **scalable** with document partitioning based on smart sorting using the SpamScore, PageRank and relevance score of the documents.

## 7 Experimental Results

Our collection of documents is the English portion of the ClueWeb09 dataset (around $500$ million documents). For the components, we have used the contextual strategy, with the Maxent POS-tagger provided with StandfordNLP[8]. The NER component is based on the OpenNLP toolkit[9]. Evaluation is performed using well-known metrics such as precision (number of correct discovered results divided by the total number of discovered results). The recall can only be estimated since we cannot manually parse the $500$ million documents to check if some results have been forgotten. However, we show that the number of correct results increases over time.

### 7.1 Quality Results

In this section, we present our results in terms of quality of label extension, relationship discovery and comparison with state of the art baseline knowledge extraction tools. The evaluation of relationship discovery depends on the quality of generated patterns and hence we present this evaluation rather than the pattern generation itself.

[8] `http://nlp.stanford.edu/software/CRF-NER.shtml`
[9] `http://opennlp.apache.org/`

**Relationship Discovery**. We evaluate the quality obtained by SPIDER when running the first use case (Section 4.2). Given two labels (representing entities), we **search for the correct type of relationship** which links them. To fulfill this goal, we have manually designed a set of 200 relationships, available at this URL[10]. Note that the type of relationship associated to each example is the most expected one, but several types of relationship are possible for the same example. Table 1 provides a sample of examples (e.g., *Obama, Hawai*) and some candidate types of relationship discovered by SPIDER (e.g., *birthplace*). A **bolded** type of relationship indicates that it is correct for this example. A second remark about our set of relationships deals with the complexity of some relationships (e.g., <*cockatoo, tail, yellow*>). The last column shows the initial confidence score computed for the candidate relationship. The quality

| Example | Discovered type of relationship | Confidence score |
|---|---|---|
| *Obama, Hawai* | **birthplace** | 0.42 |
| | senator | 0.31 |
| | president-elect | 0.18 |
| *cockatoo, yellow* | amazon | 0.32 |
| | parrot | 0.31 |
| | **tail** | 0.16 |
| *eucalyptus, myrtaceae* | plant | 0.51 |
| | **family** | 0.43 |
| | specie | 0.27 |
| *Bartolomeo Cristofori, piano* | **inventor** | 0.60 |
| | instrument | 0.43 |
| | **maker** | 0.19 |

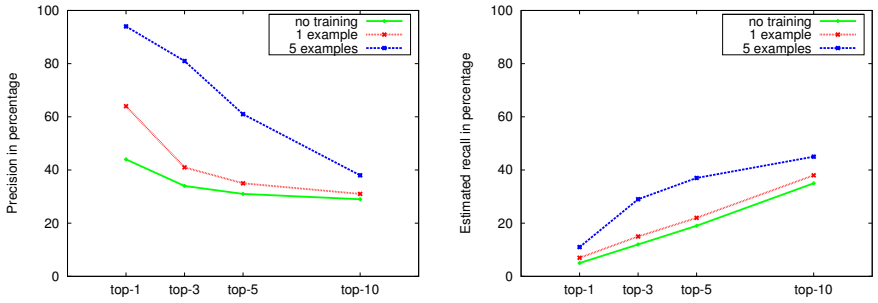
**Table 1.** Examples of Discovered Types of Relationship and Confidence Scores

is measured in terms of precision at different top-k. Indeed, SPIDER outputs a ranked list of relationship types according to their confidence scores. In addition, our approach is able to run with or without training data. Thus, we have tested the system when a few training data have been provided. Using 1 training example means that the system has randomly selected 1 correct example for bootstrapping the system. Experiments with the training data are based on cross-validation and 5 runs reduce the impact of randomness. The manual validation of the discovered relationships has been performed by 8 people. This manual validation includes around 3000 invalid relationships and 600 correct ones, and it facilitates the automatic computation of precision. In addition, we are able to estimate the recall, i.e. to evaluate the number of correct types discovered during a run w.r.t. all validated types. This is an estimation because there may exist more correct types of relationship than the ones which have been validated. Besides, a discovered type may have a different spelling from a validated type while both have the same meaning, thus decreasing the recall.

Figures 2(a) and 2(b) respectively depict the **average precision and the average recall** for the 200 relationships by top-k and by number of provided training data. We first notice that SPIDER achieves low quality without training data (precision from 40% at top-1 to 30% at top-10). The estimated recall values are also quite low at top-1 because there is an average of 3 correct types of relationships for each example. The top-3 results are interesting with 5 training data: the precision is acceptable (more than 80%) while the recall value (32%) indicates that one type of relationship out of three is correctly identified. Since our dataset contains

[10] `http://j.mp/apweb2013`

complex relationships, this configuration is promising for bootstrapping the system. Precision strongly decreases at top-5 and top-10, mainly because each example roughly includes 3 types. However, the top-5 and top-10 recall values indicate that we discover more correct examples. Finally, we notice that providing a few training data (5 examples) enables at least a 10% improvement both for precision and recall. This remark is important since our approach aims at running perpetually by reusing previously discovered examples and patterns.




(a) Precision (b) Estimated Recall

**Fig. 2.** Quality results according to top-k and Training Data

The quality results are subject to the complexity of the set of relationships, since we have selected some complex ones to discover, such as <*"cockatoo", "tail", "yellow"*>. Other problems of disambiguation occurred, for instance the example *"Chelsea", "London"* mainly returns types of relationships about accommodations because Chelsea is identified as a district of London and not as the football team.

**Baseline Comparison**. A final experiment aims at comparing our system with two other approaches, ReadTheWeb (NELL) [4] and Prospera [10], both described in Section 6. These two approaches have been chosen as baseline because the dataset along with the results are available online. An evaluation of these tools is described online[11]. Since the seed examples are available, we have used them as training data. Table 2 summarizes the **comparison between the three systems** in terms of estimated precision, as explained in the experiments reported in [4, 10]. Similarly to Prospera and ReadTheWeb, our precision is an estimation due to the amount of relationships to validate. Namely, 1000 random types have been validated for each relationship. The average precision of the three systems is the same (around 0.91). However, the total number of facts discovered by SPIDER (71,921) is 36 times higher than ReadTheWeb (2,112) and 1.3 times higher than Prospera (57,070), outperforming both baselines.

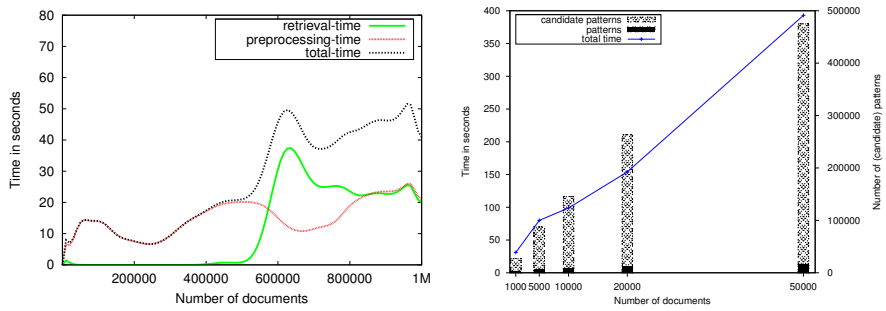
Prospera provides slightly better quality results than our approach on *AthletePlaysForTeam* relation. However, several factors have an influence on the precision results between Prospera, ReadTheWeb and SPIDER. First, Prospera is able to use seeds and counter seeds while we only rely on positive examples. On the other side, Prospera includes a rule-based reasoner combined with the YAGO ontology. Although SPIDER does not support this feature,

[11] `http://www.mpi-inf.mpg.de/yago-naga/prospera/`

the combination of POS-tagged patterns and NER techniques achieves outstanding precision values.

| Relation | RTW | Prospera | SPIDER |
|---|---|---|---|
| AthletePlaysForTeam | **1.00** (456) | 0.82 (14,685) | 0.80 (15,234) |
| TeamWonTrophy | 0.68(397) | 0.94 (98) | **0.96** (92) |
| CoachCoachesTeam | **1.00** (329) | 0.88 (1,013) | 0.90 (1,629) |
| AthleteWonTrophy | n/a | 0.92(10) | **0.94** (124) |
| AthletePlaysInLeague | n/a | 0.94 (3,920) | **0.95** (4,211) |
| CoachCoachesInLeague | n/a | **0.99** (676) | 0.89 (741) |
| TeamPlaysAgainstTeam | **0.99** (1,068) | 0.89 (15,170) | 0.93 (15,729) |
| TeamPlaysInLeague | n/a | 0.89 (1,920) | **0.95** (2,409) |
| TeamMate | n/a | **0.86** (19,578) | 0.84 (31,752) |

**Table 2.** Estimated Precision (with Number of Discovered Facts) values obtained by ReadTheWeb (RTW), Prospera and SPIDER.




(a) Retrieval and Preprocessing Performance (b) Query Performance

**Fig. 3.** Performance results

### 7.2 Performance

Since knowledge extraction systems deal with large collections of documents, they need to be scalable. Figure 3(a) depicts the performance of SPIDER for retrieving and preprocessing (i.e., clean up the header, remove html tags) the documents. The total time (sum of retrieval and preprocessing) is also indicated. Although there is no caching, the total time is not significant for collecting and preprocessing one million documents (around 40 seconds). Note that in real cases, a conjunctive query composed of two labels rarely returns more than $20,000$ documents. The peak for retrieval at $600,000$ documents is due to an overhead processing from the thread manager. Increasing the number of threads above 400 leads to higher thread switching latency while decreasing this number only reports the peak earlier during the process. This issue could be simply solved by dispatching this task on different servers.

ReadTheWeb and Prospera expose their knowledge base but not their tools. Thus, it is not possible to evaluate the three systems on the same hardware and the following comparison is based on the performance described in the original research papers. It mainly aims at showing

the significant improvement of SPIDER over ReadTheWeb and Prospera, for a better average quality. To produce the results shown in Table 2, Prospera needed more than 2 days using 10 servers with 48 GB RAM [10]. In a similar fashion, ReadTheWeb has generated an average of 3618 facts per day during the course of 67 days using the Yahoo $M45$ supercomputing cluster [4]. On the contrary, our approach performed the same experiment in a few hours. The generation of facts for each type of relationship took between 20 to 60 minutes with four servers equipped with 24 GB RAM. Although these values are mainly indicative, SPIDER is more efficient than the two other systems when dealing with dynamic and large-scale environments.

## 8 Conclusion

In this paper, we have presented SPIDER, an approach to **automatic extraction of binary relationships from large text corpora**. The main advantage of our system is to guarantee both **a better quality** and **a strong improvement in terms of performance** over similar approaches, thus providing new opportunities for discovering relationships at large scale. Finally, we have demonstrated **the feasibility of SPIDER** at Web-scale.

## References


1. E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Proc. of DL*, pages 85–94. ACM, 2000.
2. M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *Proc. of IJCAI*, pages 2670–2676. Morgan Kaufmann, 2007.
3. S. Brin. Extracting patterns and relations from the world wide web. In *Proc. of WebDB*, pages 172–183. Springer, 1998.
4. A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. Hruschka Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In *Proc. of AAAI*. AAAI Press, 2010.
5. O. Etzioni, M. Banko, S. Soderland, and D. S. Weld. Open information extraction from the web. *Communication of ACM*, 51:68–74, December 2008.
6. A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *Proc. of EMNLP*, pages 1535–1545. ACL, 2011.
7. V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Journal of Soviet Physics Doklady*, 10:707, 1966.
8. T. R. Lynam, G. V. Cormack, and D. R. Cheriton. On-line spam filter fusion. In *Proc. of SIGIR*, pages 123–130. ACM, 2006.
9. Mausam, M. Schmitz, S. Soderland, R. Bart, and O. Etzioni. Open language learning for information extraction. In *Proc. of EMNLP*, pages 523–534. ACL, 2012.
10. N. Nakashole, M. Theobald, and G. Weikum. Scalable knowledge harvesting with high precision and high recall. In *Proc. of WSDM*, pages 227–236. ACM, 2011.
11. P. Pantel and M. Pennacchiotti. Espresso: leveraging generic patterns for automatically harvesting semantic relations. In *Proc. of ACL*, pages 113–120. ACL, 2006.
12. A. Parameswaran, H. Garcia-Molina, and A. Rajaraman. Towards the web of concepts: extracting concepts from large datasets. *VLDB Endowment*, 3:566–577, September 2010.
13. P. Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research*, 11:95–130, 1999.
14. N. Takhirov, F. Duchateau, and T. Aalberg. An evidence based verification approach to extract entities and relations for knowledge base population. In *Proc. of ISWC*, pages 575–590. Springer, 2012.