

# AUTOMATIC ONTOLOGY POPULATION BY GOOGLING

Gijs Geleijnse      Jan Korst

*Philips Research Laboratories  
Prof. Holstlaan 4, 5656 AA Eindhoven, the Netherlands*

## Abstract

We discuss a method to populate ontologies with the use of googled text fragments. We populate an ontology by the use of hand-crafted domain-specific relation patterns, which can be seen as a generalization of Hearst patterns. The algorithm described uses instances of some class returned by Google to find instances of other classes. A case study on populating an ontology on the movie domain is presented as an illustration of the method. We present the algorithm in detail and discuss the results of our work.

## 1 Introduction

Suppose we are interested in *the number of canals in Aalst, Dutch artists who scored a #1 hit in the US pop charts or all movies starring both Madonna and Sean Penn*. For such diverse information needs, the Internet in general and a search engine in particular can provide a solution. However, current search engines retrieve web pages, not the information itself<sup>1</sup>. We have to search within the search results in order to acquire the information. Moreover, we make implicit use of our knowledge (e.g. of the language and the domain), to interpret the web pages.

We are thus interested in a method to *automatically extract information* from the web.

In this paper, we present an algorithm that – given a domain of interest – extracts, structures and combines information from the web. With structured information available, we can easily find the information we are interested in. We present a generalization of Hearst patterns [9], viz. patterns that express arbitrary relations between instances of classes. We demonstrate how these patterns can be used as queries to Google.

### 1.1 Problem Definition

The semantic web (SW) community [3, 13] is providing standards for machine readable information on the web. SW-languages RDF(S) and OWL are developed for this purpose. Dedicated reasoners are created for ontology-based question-answering services. As such, these reasoners are able to provide answers to questions like the ones above, given a sufficiently populated ontology.

For our purposes we define an ontology as follows:

**Definitions.** A *reference ontology*  $O$  is a 4-tuple  $(C, I, P, T)$ , where

- $C$  =  $(c_0, c_1, \dots, c_{n-1})$ , an ordered set of  $n$  classes,
- $I$  =  $(I_0, I_1, \dots, I_{n-1})$ , with  $I_j$ ,  $0 \leq j < n$ ,  
the set of instances of class  $c_j \in C$ ,
- $P$  =  $(p_0, p_1, \dots, p_{m-1})$ , a set of  $m$  binary relations on the classes,  
with  $p_k = (c_{k,0}, c_{k,1}) \in C \times C$ ,  $0 \leq k < m$ , and
- $T$  =  $(T_0, T_1, \dots, T_{m-1})$ , with  $T_k$ ,  $0 \leq k < m$ ,  
the set of instances of relation  $p_k \in P$ ,  
with  $T_k = \{(s, o) \mid p_k(s, o)\}$ ,  $s \in I_{k,0}$  (an instance of  $c_{k,0}$ )  
and  $o \in I_{k,1}$  (instance of  $c_{k,1}$ ).

---

<sup>1</sup>The question-answering services of <http://www.google.com> and <http://www.askjeeves.com> do not provide answers to these (simple) questions.

A *partial ontology* of  $O$  is defined as  $O' = (C, I', P, T')$ , where

$$\begin{aligned} I'_j &\subseteq I_j && \text{for all } j, 0 \leq j < N, \\ T'_j &\subseteq T_j && \text{for all } j, 0 \leq j < M \text{ and} \\ (s, o) \in T'_k &\Rightarrow s \in I'_i \wedge o \in I'_j && \text{for some } i, j, k. \end{aligned} \quad \square$$

As an example, we give a partial ontology  $O'_{\text{soccer}}$  on soccer clubs, towns and players.  $O'_{\text{soccer}} = (C, I', P, T')$ , with

$$\begin{aligned} C &= ( \text{ Team, Town, Player} ) \\ I' &= ( \{ \text{ PSV, NAC Breda, RCS Anderlecht}, \\ &\quad \{ \text{ Eindhoven, Brussels, Breda, Neerpelt}, \\ &\quad \{ \text{ Van Hooijdonk, Goor} \} \} , \\ P &= ( \text{ hometown(Team, Town),} \\ &\quad \text{ plays\_with(Team, Player),} \\ &\quad \text{ was\_born\_in(Player, Town) } ) , \\ T' &= ( \{ (\text{PSV, Eindhoven}), (\text{RCS Anderlecht, Brussels}) \}, \\ &\quad \{ (\text{Van Hooijdonk, NAC Breda}), (\text{Goor, RCS Anderlecht}) \}, \\ &\quad \{ (\text{Goor, Neerpelt}) \} ) . \end{aligned}$$

This partial ontology thus contains three classes and three relations on the classes. Each of the classes and relations is populated by some instances.

Given the definitions of reference ontology and partial ontology, we formulate the information extraction problem as an ontology population problem<sup>2</sup>:

**Problem.** Given a partial ontology  $O'$ , extend  $O'$  to some  $O''$  that maximizes the precision and/or recall. □

We define *precision* and *recall* as measures of a class  $c_i \in C$ :  $\text{precision}(c_i) = \frac{|I_i \cap I''_i|}{|I''_i|}$  and  $\text{recall}(c_i) = \frac{|I_i \cap I''_i|}{|I_i|}$ . Similar measures can be formulated for relations  $p_j$ .

## 1.2 Related Work

An overview of ontologies and ontology learning and construction can be found in [12]. In [2] the reader can find more on ontology mapping and population.

Work on information extraction from large websites, so-called wrapper algorithms, can be found in [7]. Hearst [9] identifies a method to extract hyponym relations from texts. Cimiano and Staab [6] use Google to identify such patterns to populate ontologies. Brin [5] combines techniques from pattern extraction and wrapper algorithms to extract instances of relations from web sites.

Automated part of speech tagging [4] is a useful technique in term extraction, applied for example in [8]. Here, terms are extracted with a predefined part-of-speech structure, e.g. an adjective-noun combination. In [11], methods are discussed to extract information from natural language texts with the use of both part of speech and Hearst patterns.

For literature on state-of-the-art question-answering systems we refer to the proceedings of the TREC conference [1].

## 2 Global Approach

A straightforward method to extract information from the Internet, is to implement a wrapper algorithm [7]. Such an algorithm crawls a large website and makes use of the homogeneous presentation of the information on its pages. When instances are denoted on exactly the same place on each similar page within a website, it is easy to extract them.

When we use such algorithms, we limit ourselves to the information contained on these websites. Moreover, the algorithm has to be adapted each time the design of the website changes.

<sup>2</sup>Readers more familiar with databases, may view of this as a relational database population problem as well.

We are interested in a technique that is both domain and website independent. We therefore choose to extract information from arbitrary web sites. To find relevant web sites – and thus relevant information – we use the Google search engine<sup>3</sup>.

Our research can be separated into two issues: on the one hand the search for relevant (hyper)texts on web pages, on the other hand ontology population using these relevant (hyper)texts.

We want to formulate precise and specific queries, in order to generate Google-query-results with high information density for our purposes. We therefore choose to use instances in our queries to simultaneously find instances of classes and instances of relations. For example, given the instance ‘Johan Cruijff’, we can use this instance in the query ‘Johan Cruijff was born in’ in order to retrieve a place in general and Cruijff’s place of birth in particular. The place of birth, Amsterdam, can be extracted from the retrieved documents. Now, ‘Amsterdam’ can be used in the query ‘was born in Amsterdam’ to discover other (Amsterdam-born) persons, like Willem Frederik Hermans and Paul Verhoeven.

For a given relation  $p_k$ , we thus use both an instance  $I'_{k,0}$  and a natural language formulation of the relation in our Google queries. Subsequently, the Google excerpts are scanned for instances of  $I'_{k,1}$  and  $(I'_{k,0}, I'_{k,1}) \in T'_k$ .

### 3 Detailed Outline

Our method assumes a (hand-crafted) partial ontology  $O'$  of an arbitrary knowledge domain. Since we use an instance each time we query Google, initially at least one of the sets  $I'_j$  must be non-empty. We do not consider this as a disadvantage, since the creator of the ontology is expected to have some knowledge of the field.

#### 3.1 Patterns identifying relations

For relation  $p_k$ , defined on  $(c_{k,0}, c_{k,1})$ , in the partial ontology  $O'$ , we have to identify explicit natural language formulations of this relation. We are thus interested in patterns  $\mathcal{P}_k$  of the form “[ $c_{k,0}$ ] expression [ $c_{k,1}$ ]”<sup>4</sup>, that express the relation  $p_k$  in natural language. Such patterns have to meet two criteria:

(*Precision.*) Preferably, the phrase is unambiguous, i.e. the probability that the terms found do not belong to the intended class must be small. For example, consider the relation `place_of_birth(Person, City)`. The pattern ‘[*Person*] was born in [*City*]’ is not an unambiguous representation of this relation, since ‘[*Person*] was born in’ can precede a date or the name of a country as well.

(*Recall.*) The pattern must frequently occur to allow high recall.

Suitable formulations can be found by observing how instances of the related classes are connected in natural language texts. For example, if we are interested in populating `plays_with(player, team)`, we can identify this set of patterns:  $\mathcal{P}_{plays\_for} = \{ \text{“[team]-player [player]”, “[player] ([team])”, “[player] signed for [team]”, “[team] substituted [player] for [player]” } .$

#### 3.2 Formulation of Google-queries

When we have chosen the sets of relation  $\mathcal{P}_k$ , we can use these to create Google queries. For each “[ $c_{k,0}$ ] expression [ $c_{k,1}$ ]” pattern, we can formulate two Google queries: “[ $c_{k,0}$ ] expression” and “expression [ $c_{k,1}$ ]”. For example, with the relation `was_born_in` and instances `Amsterdam` and `Spinoza`, we can formulate the queries “Spinoza was born in” and “was born in Amsterdam”.

This technique thus allows us to formulate queries with instances that have been found in results of prior queries.

#### 3.3 Instance identification

A separate problem is the identification of terms in the text. An advantage is that we know the place in the text by construction (i.e. either preceding or following the queried expression). A disadvantage is that each class requires a different technique to identify its instances. Especially

<sup>3</sup><http://www.google.com>

<sup>4</sup>We use the [ $c_i$ ] notation to denote a variable instance of class  $c_i$

---

```

identify patterns  $\mathcal{P}_k$  for all  $p_k$ .
; identify recognition functions  $f_i$  for all  $c_i$ .
; identify check functions  $check_i$  for all  $c_i$ .
;  $O'' := O'$ 
; for all  $i$  add all instances in  $I''_i$  to  $U_i$ 
; while  $\exists_i(U_i \neq \emptyset)$  do
  select  $h \in U_i$ 
  ; for all  $k, l$  such that  $p_k = (c_i, c_l) \vee p_k = (c_l, c_i)$ :
    query  $\mathcal{P}_k$  with  $h$  substituted for  $c_i$ 
    ; extract all  $j$  such that  $f_k(j) \wedge check_k(j)$ 
    ; add all  $j$  and  $(h, j)$  to  $O''$ 
    ; add all  $j$  to  $U_l$ 
  ; remove  $h$  from  $U_i$ 
od

```

---

Table 1: pseudo-code of the ontology population algorithm

terms with a less determined format, such as movie titles, are hard to identify. We therefore design recognition functions  $f_i$  for each class.

For these functions  $f_i$ , we can adopt various techniques from the fields of (statistical) natural language processing, information retrieval and information extraction. A regular expression that describes the instances of class  $c_i$  can for example be a part of the function  $f_i$ . The user may also think of the use of part of speech tagging [4]. We note that the HTML-markup can be of use as well, since some concepts (such as named entities) tend to be emphasized, or made ‘clickable’.

After extracting a term, we can perform a *check* to find out whether the extracted term is really an instance of the concerning class. We perform this check with the use of Google. We google phrases that express the term-class relation. Again, these phrases can be constructed semi-automatically. Hearst-patterns are candidates as well for this purpose. A term is to be accepted as instance, when the number of hits of the queried phrase is at least a certain threshold.

When we use such an additional check function, we can allow ourselves to formulate less strict recognition functions  $f_i$ . That is, false instances that are accepted by  $f_i$ , are still rejected as an instance by the use of the check function.

### 3.4 Sketch of algorithm

In Table 1, we give the algorithm, as described in this section, in pseudo-code. An instance  $h$  from some set  $U_i$  is selected that has not yet been queried. All relations that are associated with the class of  $h$  are selected and queries are performed with the corresponding patterns. From the Google query-results, instances of the relations  $(h, j) \in T'_k$  or  $(j, h) \in T'_k$  and – simultaneously –  $j \in I'_l$  are extracted with the use of the recognition and check functions.

The algorithm takes instances found as its input again. We use the sets  $U_i$  to identify the set of instances which have not been used for this purpose. The algorithm terminates when no new instances can be found.

The first three steps – the identification of patterns and the recognition and check functions – can be seen as preprocessing.

## 4 A Case Study

We illustrate our work with a case-study on the population of an ontology with actors, directors and movies. We have chosen this case, because it is a domain on which much structured and unstructured data is available. It is thus interesting to examine how much of this information we can retrieve. The structured databases on the internet on this domain facilitate easy verification of the results of the algorithm.

We have selected a small partial ontology on the movie domain. It is defined as

$$O'_{movie} = ( ( Director, Actor, Movie ) , ( \{ Steven Spielberg, Francis Ford Coppola \}, \emptyset, \emptyset ) , ( acts\_in(Movie, Actor ) , director\_of(Movie, Director) ) , ( \emptyset, \emptyset ) ).$$

We thus only identify three classes, of which only the class *Director* has instances. Using our method, we want to find movies directed by these directors. The movies found are used to find starring actors, where those actors are the basis of the search for other movies in which they played, etc. The process continues until no new instances can be found.

**Relation patterns.** This small ontology contains two relations, *acts\_in* and *director\_of*. For these relations, we have manually selected the sets of patterns:

$$\mathcal{P}_{acts\_in} = \{ "[Movie] \text{ starring } [Actor], [Actor] \text{ and } [Actor]" \} \text{ and}$$

$$\mathcal{P}_{director\_of} = \{ "[Director]'s [Movie]", "[Movie], \text{ director : } [Director]" \}.$$

**Formulation of Google-queries.** These patterns lead to the following set of Google-queries:  $\{ "[Movie] \text{ starring}", "starring [Actor]", "[Director]'s", "[Movie] \text{ director}" \}$ . We have analyzed the first 100 excerpts returned by Google.

**Instance identification.** We identify a term as a *Movie* title, if it is placed in a text between quotation marks. Although this may seem a severe restriction, in practice we can permit to loose information contained in other formulations since each Google query-result gives much redundant information. So, if a movie title is placed between quotation marks just once in the Google results, we are able to recognize it.

A person's name (instances of the classes *Director* and *Actor*) is to be recognized as either two or three words each starting with a capital.

Another feature of the recognition function is the use of lists with tabu words. If a tabu word is contained in an expression, we ignore it. We use a list of about 90 tabu words for the person names (containing words like 'DVD' and 'Biography'). For the movie titles we use a much shorter list, since movie titles can be much more diverse. We have constructed the tabu word lists based on the output of a first run of the algorithm.

We *check* each of the extracted candidate instances with the use of one of the following Google-queries: "The movie *[Movie]*", "*[Actor]* plays", or "*[Director]* directed". A candidate is accepted, if the number of Google-results to the query exceeds a threshold. After some tests we choose 5 as a threshold value, since this threshold filtered out not only false instances but most of the common spelling errors in true instances as well.

For a description of another case study on this subject, we refer to [10]. Here, we compose a ranking of famous people based on their presence on the World Wide Web.

## 4.1 Results

We first ran the algorithm with the names of two (well-known) directors as input: Francis Ford Coppola and Steven Spielberg. Afterwards, we experimented with larger sets of directors and small sets with directors who were unknown to us as input.

An interesting observation is that the outputs are relatively independent of the input sets. That is, when we take a subset of the output of an experiment as the input of another experiment, the outputs are roughly the same. The small differences between the outputs can be explained by the changes in the Google query results over time.

We have found 7,000 instances of the class *Actor*, 3,300 of *Director* and 12,000 of *Movie*. The number of retrieved instances increases, about 7%, when 500 query results are used instead of 100.

**Precision.** When we analyze the precision of the results, we use the data from the Internet Movie Database (IMDb)<sup>5</sup> as a reference. An entry in our ontology is accepted as a correct one, if it can be found in IMDb. We have manually checked three sequences of 100 instances (at the beginning, middle and end of the generated file) of each class. We estimate a precision of 78 %. Most misclassified instances were misspellings or different formulations of the same entity (e.g. "Leo DiCaprio" and "Leonardo DiCaprio"). In the future, we plan to add postprocessing to recognize

<sup>5</sup><http://www.imdb.com>

these flaws. We can analyze the context (e.g. when 2 actors act in the same set of movies) and use approximate string matching techniques to match these cases.

Likewise, we have also analyzed the precision on the relationships, we estimate the precision of the relations between movie and director is around 85 %, between movie and actor around 90%.

**Recall.** The number of entries in IMDb exceeds our ontology by far. Although our algorithm performs especially well on recent productions, we are interested how well it performs on ‘significant’ movies, actors and directors. Firstly, we made lists of all Academy Award winners (1927-2005) in a number of relevant categories, and checked the recall:

category	recall
Best Actor	96%
Best Actress	94%
Best Director	98%
Best Picture	87%

IMDb has a top 250 of best movies ever. The algorithm found 85% of them. We observe that results are strongly oriented towards Hollywood productions. We also made a list of all winners of the Cannes Film Festival, the ‘Palme d’Or’. Alas, our algorithm only found 26 of the 58 winning movies in this category.

## 5 Conclusions and Future Work

We have presented a framework for ontology population using googled expressions. Inspired by Hearst [9], we use patterns that express arbitrary relations between instances of classes.

The method is based on hand-crafted patterns which are tailor-made for the classes and relations considered. These patterns are queried to Google, where the results are scanned for new instances. Instances found can be used within these patterns as well, so the algorithm can populate an ontology based on a few instances in a given partial ontology. Although our examples focussed on the English language, the method itself is language independent.

The results of the first experiments with this approach are encouraging. We used simple patterns, recognition functions and checks that proved to be successful. With a starting set of only two directors, we have found a well-populated ontology with not only directors, but also actors and movie titles.

In future work, we want to investigate methods to (semi)-automatically find patterns, recognition functions and checks. By searching the web, an application should observe frequently used relation patterns and the structure of the instances of the class. These observations can be used as a decision support for the user to choose appropriate patterns and functions.

We expect improvement of the performance when we add patterns, also in other languages. Performance may also improve when we use HTML formatting in our patterns, instead of the (poorer) plain text patterns we have used so far. Moreover, postprocessing is necessary to observe different formulations of the same entity.

## References

- [1] *Proceedings of the 13th Text Retrieval Conference (TREC’04)*. Gaithersburg, Maryland.
- [2] *Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web*. Dagstuhl, Germany, February 2005.
- [3] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. In *Scientific American*, 2001.
- [4] E. Brill. A simple rule-based part-of-speech tagger. In *Proceedings of the third Conference on Applied Natural Language Processing (ANLP’92)*, pages 152–155, Trento, Italy, 1992.
- [5] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB Workshop at sixth International Conference on Extending Database Technology (EDBT’98)*, 1998.

- [6] P. Cimiano and S. Staab. Learning by googling. *SIGKDD Explorations Newsletter*, 6(2):24–33, 2004.
- [7] V. Crescenzi and G. Mecca. Automatic information extraction from large websites. *Journal of the ACM*, 51(5):731–779, 2004.
- [8] K. Frantzi, S. Ananiado, and H. Mima. Automatic recognition of multi-word terms: the c-value/nc-value method. *International Journal on Digital Libraries*, 3:115–130, 2000.
- [9] M. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics*, pages 539–545, Morristown, NJ, USA, 1992.
- [10] J. Korst, G. Geleijnse, N. de Jong, and M. Verschoor. Ontology-based extraction of information from the World Wide Web. In *Intelligent Algorithms* (to appear in the Philips Research Book Series). Springer, 2005.
- [11] G. Nenadić, I. Spasić, and S. Ananiadou. Automatic discovery of term similarities using pattern mining. In *Proceedings of the second international workshop on Computational Terminology (CompuTerm'02)*, Taipei, Taiwan, 2002.
- [12] S. Staab and R. Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.
- [13] The World Wide Web Consortium (W3C) [www.w3c.org](http://www.w3c.org), 2005.