Google-based Information Extraction

Finding John Lennon and Being John Malkovich

Gijs Geleijnse Jan Korst Verus Pronk
Philips Research
Prof. Holstlaan 4
5656 AA, Eindhoven, the Netherlands
{gijs.geleijnse,jan.korst,verus.pronk}@philips.com

ABSTRACT
We discuss a method to extract information from Googled text fragments. We populate an ontology using hand-crafted domain-specific relation patterns and a rule-based approach to recognize instances of the classes. The algorithm uses the instances for one class found in the Google excerpts to find instances of other classes. The work is illustrated by two case studies. The first involves the population of an ontology on the movie domain. The second is a search for famous people and the collection of biographical entries such as nationality and profession.

Categories and Subject Descriptors
H.3.1 [**Content Analysis and Indexing**]: Linguistic Processing; H.3.3 [**Information Search and Retrieval**]: Query formulation

General Terms
Information extraction, Search engines, World Wide Web

1. INTRODUCTION
Suppose we are interested in *the countries where Burger King can be found, the Dutch cities with a technical university* or perhaps *the way to Amarillo*. For such diverse information needs, the World Wide Web in general and a search engine in particular can provide a solution. However, current search engines retrieve web pages, not the information itself[1]. 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.

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.

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

The extracted information can, for example, be used by recommender systems to acquire additional metadata to make meaningful recommendations for music or TV programs, by establishing links that would not have been able through the direct mapping of user preferences to the metadata provided with the content. For example, if the user has expressed a preference for TV programs relating to Italy, then by using extraction of information the recommender system will be able to recognize regions as Tuscany and Veneto and cities as Milano and Florence in the metadata of TV programs as relevant. Likewise, if the user has expressed a preference for TV programs relating to photography the system will be able to recognize the names of famous photographers as Cartier-Bresson and Moholy-Nagy.

This paper is organized as follows. After defining the problem and discussing related work in the next parts of this Section, we present an algorithm to populate an ontology in Section 2. Section 3 handles a case study on populating a movie ontology. In Section 4 we present work on finding famous people and their biographical data. Finally, Section 5 handles the conclusions and future work.

1.1 Problem definition
The semantic web community [2] is providing standards for machine readable information on the web. The languages RDF(S) and OWL are developed for this purpose by the World Wide Web Consortium[2]. Dedicated reasoners are created for ontology-based question-answering services. As such, these reasoners are able to provide answers to questions like the above, given a sufficiently populated ontology.

For our purposes we define an ontology as follows:

Definitions. Reference ontology O is a 4-tuple (C, I, P, T), where

C = $(c_0, c_1, ..., c_{N-1})$, an ordered set of N classes,
I = $(I_0,\ I_1,\ ...,\ I_{N-1})$, with I_j, $0 \le j < N$, the set of instances of class $c_j \in C$,
P = $(p_0, p_1, ..., p_{M-1})$, a set of M binary relations on the classes, with $p_i : c_{i,0} \times c_{i,1}$, $0 \le i < M$, and $c_{i,0}, c_{i,1} \in C$, and
T = $(T_0,\ T_1,\ ...,\ T_{M-1})$, is a set of instances of the relations in P, with $T_j = \{(s, o) \mid p_j(s, o)\}$ for each $j,\ 0 \le j < M$ and $s \in I_{j,0}$ (an instance of $c_{j,0}$) and $o \in I_{j,1}$ (instance of $c_{j,1}$).

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

[2]`http://w3c.org/`

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

We formulate the information extraction problem as an ontology population problem:

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

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

1.2 Related work

Information extraction and ontologies are two closely related fields. For reliable information extraction, we need background information, e.g. an ontology. On the other hand, we need information extraction to generate broad and highly usable ontologies. An overview on ontology construction and usage and ontology learning from structured and unstructured sources can be found in [16, 7].

Question Answering is a task in the Text Retrieval Conference, TREC [21]. The issue addressed here is, given a corpus, find answers to a set of questions. Such questions contain an instance and a relation, the problem is to find instances that are related to the instance in the question.

In [19], a method is described to extract information from the web. Ravichandran identifies patterns that express some relation. Using these patterns in queries to a search engine, he finds answers to questions in a question answering setting. Moreover, he discusses alternatives to pattern-based extraction, such as text clustering.

In [14], the use of patterns to discover relations was introduced. Hearst presents a number of phrases that are natural text formulations of the hyponym relation. By extracting the terms surrounding such phrases, hyponym pairs can be extracted. For example, from the phrase ‘artists, like Michael Jackson and David Bowie’ the relations (Michael Jackson, artist) and (David Bowie, artist) can be extracted.

Brin identifies the use of patterns in the discovery of relations on the web [5]. He describes a website-dependent approach to identify hypertext patterns that express some relation. For each web site, such patterns are learned and explored to identify instances that are similarly related. In [1], the system described in [5] is combined with a named-entity recognizer. This Snowball-system also identifies instances with the use of the named-entity recognizer.

Cimiano and Staab [9] use Google to identify relations between concepts. They test a hypothesis rather than to extract new knowledge. For example, they test whether the phrase ‘the nile is a river’ returns enough Google hits to accept the relation is_a(Nile, river).

In the nineties, the Message Understanding Conferences focussed on the recognition of named entities (such as names of persons and organizations) in a text [8]. This work is mostly based on rules on the syntax and context of such named entities. For example, two capitalized words preceded by ‘mr.’ will denote the name of a male person. Research on named entity recognition is continued for example in [3, 6].

Automated part of speech tagging [4] is a useful technique in term extraction [11], a domain closely related to named-entity recognition. Here, terms are extracted with a predefined part-of-speech structure, e.g. an adjective-noun combination. In [18], methods are discussed to extract information from natural language texts with the use of both part of speech tags and Hearst patterns.

2. SOLUTION APPROACH

A straightforward method to extract information from the web, is to implement a wrapper algorithm [10]. Such an algorithm crawls a number of large websites and makes use of the homogeneous presentation of the information on the pages of a website. When instances are denoted on exactly the same place on each 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 one of the website changes.

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[3]. We use the homogeneous presentation of the Google search results pages to identify the excerpts from the pages found. Our approach is thus only dependent on the mark-up of one website, namely Google.

Our research can be separated into three issues.

1. To *identify natural language formulations of the relations in* O'.

   Given some relation p_i in O', we are interested in text patterns that express this relation. For example, given the relation ‘has capital’ between ‘country’ and ‘city’, the pattern ‘is the capital of’ is a good candidate to express this relation. In this work, we manually select the text patterns expressing the relations p_i in the ontology.

2. To *identify instances of the classes in* O'.

   Within the excerpts produced by Google after submitting a query, we want to identify instances of the classes that are not yet in I'. Per class we combine a number of heuristics to do so.

3. 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 other instances and to find instances of relations. For example, given the instance ‘Johannes Vermeer’, we can use this instance in the query ‘Johannes Vermeer was born in’ in order to retrieve a place in general and Vermeer’s place of birth in particular. The place of birth, Delft, can be extracted from the retrieved documents. Now, ‘Delft’ can be used in the query ‘was born in Delft’ to discover other (Delft-born) persons such as Antony van Leeuwenhoek and Hugo de Groot.

   We thus use both an instance $I'_{k,0}$ and a natural language formulation of a relation p_k in our Google queries. Subsequently, the Google excerpts are scanned for instances of class $c_{k,1}$ and instance-pairs of p_k.

[3]`http://www.google.com`

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 a disadvantage, since the creator of the ontology is expected to have some knowledge of the field.

In Sections 2.1, 2.2 and 2.3, we will discuss these three issues in more detail. We combine these strategies into the ontology population algorithm as found in Section 2.4.

2.1 Identifying relation patterns

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 $[c_{k,1}]$" [4], 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 for(player, team)*, we can identify this set of patterns: $\mathcal{P}_{plays\_for} = \{$ "[*team*]-*player* [*player*]", "[*player*] ([*team*])", "[*player*] *signed for* [*team*]", "[*team*] *substituted* [*player*] *for* [*player*]" $\}$.

In this work, we select the relation patterns manually. However, work has been done to automatically select precise patterns [20]. Current work involves the automatic identification of effective patterns [12], which are patterns that are likely to give useful results when using them as queries.

2.2 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 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 expressions 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 terms 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.

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

When we use such a 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.

2.3 Formulation of Google-queries

When we have chosen the sets of relation 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.

2.4 Sketch of algorithm

Per relation, we maintain a list of instances that already have been used in a query in combination with the patterns expressing this relation. Initially, these lists are thus empty.

The following steps of the algorithm are performed until either some stop criterion is reached, or until new instances and instance-pairs no longer can be found.

- **Step 1:** Select a relation p_k on $c_i \times c_j$, and an instance v from either I_i or I_j we have not yet used in a query.
- **Step 2:** Combine the patterns expressing $\mathcal{P}_k$ with v and send these queries to Google.
- **Step 3:** Extract instances from the excerpts using the instance identification rules for the class of v.
- **Step 4:** Add the newly found instances to the corresponding instance set and add the instance-pairs found (thus with v) to $T'_{(i,j)}$.
- **Step 5:** If there exists an instance that we can use to formulate new queries, then repeat the procedure.

Note that instances of class c_i learned using the algorithm applied on relation p_k on $c_i \times c_j$ can be used as input for the algorithm applied to some relation p_l on $c_i \times c_h$ to populate the sets I'_h and $T'_{(i,h)}$.

3. POPULATING A MOVIE ONTOLOGY

For our first case study, we have constructed a small partial ontology on the movie domain. It is defined as

$$
\begin{aligned}
O'_{movie} = (\;\; & (\textit{Director, Actor, Movie}\,)\,,\\
& (\,\{\,\textit{Steven Spielberg}\,,\\
& \quad \textit{Francis Ford Coppola}\,\},\ \emptyset, \emptyset)\,,\\
& (\,\textit{acts in(Movie,Actor)},\\
& \;\; \textit{director of(Movie,Director)}\,)\,,\\
& (\emptyset, \emptyset)\;\;).
\end{aligned}
$$

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

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.

Formulation of Google-queries. The relation patterns lead to the following set of Google-queries: $\{$"[*Director*]*'s*", "[*Movie*] *starring*", "[*Movie*] *director*" , "*starring* [*Actor*]"$\}$. We have analyzed the first 100 excerpts returned by Google after querying a pattern in combination with an instance.

3.1 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 of directors who are 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)[5] 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 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.

We identify three categories of incorrect instances in the ontology:

1. Different formulations and misspellings of an instance, e.g. *the Karate Kid* vs. *Karate Kid*, and *Leo DiCaprio* vs. *Leonardo DiCaprio*.
2. Different titles for the same movie (e.g. *Hable con Ella* vs. *Talk to Her*). Often, both titles are mentioned on web sites. When two movie titles share the same director and set of actors, a check can be done to find the relationship between the two titles.
3. False Instances. On each candidate instance we perform a check. However, terms like *James Bond* and *Mickey Mouse* slip through this check and are identified as instances.

Likewise, we have also analyzed the precision of the relations, we estimate the precision of the relation between movie and director around 85 %, and 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. First, we made lists of all Academy Award winners (1927-2005) in a number of relevant categories, and checked the recall (Table 1).

CATEGORY	RECALL
Best Actor	96%
Best Actress	94%
Best Director	98%
Best Picture	87%

Table 1: Recall of Academy Award Winners

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.

4. EXTRACTING INFORMATION ON FAMOUS PEOPLE

The second case study aims at extracting a long list of famous persons and in addition extracting for each of them biographical information such as nationality, period of life, and profession. Using this

[5] `http://www.imdb.com`

additional information, we can create sublists of e.g. 17th-century Dutch painters. The information extraction is carried out in two phases. First a long list of famous persons is extracted, and secondly, additional information on these persons is gathered.

4.1 Relation patterns and query formulation

It has been observed by e.g. [20] that a surface pattern as "*Wolfgang Amadeus Mozart (*" is very successful to determine the year of birth of in this case Mozart, as the open bracket will be often followed by the period of life of the person (in this case: 1756-1791). We decided to use this observation but in a different fashion. Instead of looking for the year of birth of a given person, we use year intervals that possibly relate to the lifetime of a person to find famous persons. More precisely, we issued all year intervals "(y_1 - y_2)" as queries to Google, with $y_1 \in [1000..1990]$, $y_2 - y_1 \in [15..110]$ and $y_2 \leq 2005$. In other words, we search for persons who were born during the last millenium and who died at an age between 15 and 110. Note that, in this way, we will only find persons that already passed away.

4.2 Instance identification

For each of these issued queries, we scanned the at most 1000 excerpts that Google returned. In each of these excerpts, we determined the first occurence of the queried pair of numbers. Since Google ignores non-alphanumeric characters, the queried pair of numbers may also occur as y_1, y_2 or as y_1/y_2. If the queried pair of numbers is in the intended context $(y_1 - y_2)$, i.e. if they are surrounded by brackets and seperated by a hyphen, then the words directly preceding this first occurrence are stored for later analysis, to a maximum of six words. In this way, we obtain for each queried pair of numbers up to 1000 short text fragments that potentially contain person names. In addition, for each of the stored text fragments, we remove potential pre- and suffixes that normally cannot be part of a name. For example, we delete all words that precede a full stop (except when preceded by a single capital letter), a colon, or a semicolon. In addition, of words consisting of upper-case letters only we transform the upper-case into lower-case letters, except for the first one (with some specific exceptions concerning ordinal numbers of kings, queens, etc., composite names including hyphens or apostrophes, and Scottish and Irish names). This results in a set of candidate names.

The *check* phase consists of two filtering steps: one to filter out non-person names and one to filter out multiple variants of a single person name. These steps are next discussed in more detail.

Not all text fragments we have found in the extraction phase will be person names. Typically, historic periods, art styles, geographic names, etc. can also directly precede a time interval. Table 2 illustrates the difficulties in discriminating between person names and other text fragments. We note that *West Mae* is an inversion of the person name *Mae West* and that *Napoleon Hill* refers to a person as well as to a geographic location in the state Idaho (USA).

To filter out non-person names, we first constructed from dedicated websites a long list of the most common first names (boy's and girl's names). If a text fragment starts with such a name, then this is a strong indication that the text fragment is a person name. In addition, we constructed a long list of suspect words that typically do not occur in person names, as follows. From the many excerpts that we gathered with the year interval queries we extracted all words, counting how often they occur with a capital and without a capital. If a word occurs most often without a capital, and it is not a special

PERSON NAME	NON-PERSON NAMES
Art Blakey	*Art Deco*
West Mae	*West Virginia*
Amy Beach	*Miami Beach*
HP Lovecraft	*HP Inkjet*
Napoleon Hill	*Napoleon Hill*

Table 2: Some examples to illustrate the difficulties in discriminating between persons names and other text fragments.

word as 'van', 'de', or 'la', then it is added to the long list of suspect words. We next apply a rule-based approach using these lists of first names and suspect words to filter out text fragments that probably do not relate to person names.

In addition to filtering out non-person names, we also want to filter out multiple occurrences of the same person name. These occurrences are caused by variations in spelling of names and errors in the lifetimes. To this end, we carried out the following filtering steps.

1. *Keeping only the last name/lifetime variants that occur most often*. For each last name/lifetime combination, we often find different variants of first names preceding it. For example, *Bach (1685 - 1750)* is preceded by, e.g., *Johann Sebastian*, *JS*, and *Johann S.* Of all these variants we only store the one that is found most often, i.e., the variant that occurs most often in the text fragments we found in the 1000 excerpts that Google returned on query *"(1685 - 1750)"*.

2. *Filtering out small variations in name*. If two names have exactly the same lifetime and the edit distance [17, 13] between these full names is less than a given threshold, then only the variant that is found most often is kept. As threshold we use an edit distance of two.

3. *Filtering out single errors in lifetimes*. If two names are completely identical but their lifetimes differ in only the year of birth *or* the year of death, then only the variant that is found most often is kept.

Experiments indicate that in this step we reduce the candidate set of names by approximately 25%.

4.3 Ordering persons by fame

To order the persons by fame, we use Google page count (GPC) as our measure of fame. Now, the question is which query we should issue to Google to determine the GPC of a person. The query should be neither too general nor too specific.

A single person is often identified in different ways, e.g. *Johann Sebastian Bach*, *JS Bach*, *JOHANN SEBASTIAN BACH* and *Bach, Johann Sebastian* all refer to the same person. The last variant is called an *inversion*. The latter two variants can be transformed into the first variant by substituting upper-case characters by lower-case ones and by adjusting the order of first and last names. Complicating factors in the identification of inversions are (i) that a comma between last name and first names is sometimes omitted and (ii)

that many first names also occur as last names. An additional complication is that the first names of a name sometimes vary per language. To achieve that we are less sensitive to these variants, we use the following query to determine the GPC:

"[*last name*] ([*year of birth*] - [*year of death*])"

For kings, queens, popes, etc., we use the Latin ordinal number as last name. In this way *Charles V (1500 - 1558)*, *Carlos V (1500 - 1558)*, and *Karel V (1500 - 1558)* are all covered by query *"V (1500 - 1558)"*. Note that we assume the combination of last name and lifetime to be specific enough to uniquely identify famous persons.

4.4 Extracting additional information

The first phase, described above, resulted in a large list of famous persons that was ordered using GPC as measure of fame. For further details on this list we refer to [15]. In the next phase, we extracted additional information, such as gender, nationality, and professions. Also, we tried to retrieve related images and a few one-liners that already give a brief impression of how the person gathered fame. We extracted additional information for the top 10,000 of the list of famous persons that we obtained in the first phase. We next briefly describe how we gathered this additional material.

To acquire additional information, we again issued queries to Google of the type "*Albert Einstein was*", i.e., we used the full name of a person followed by the word *was*, where we restrict ourselves to English language pages. From the excerpts that Google returns, we extracted complete sentences that contain the query. Hence, if only a fraction of a sentence was given in an excerpt, then this fraction was simply ignored. These sentences were next used to identify specific words that indicate gender, nationality and professions.

Determining gender. We simply counted words that refer to the male gender, namely the words *he*, s his, *son of*, *brother of*, *father of*, *man* and *men*. Likewise, we counted words that refer to the female gender, namely the words *she*, *her*, *daughter of*, *sister of*, *mother of*, *woman*, and *women*. We simply assigned the gender with the highest count.

Determining nationality. We extracted for each country from the CIA World Factbook website the country name (in conventional short form) and the corresponding adjective that indicates nationality, e.g. 'Belgium' and 'Belgian'. In addition, for some countries we added a number of additional terms relating to parts of the country, such as 'Flemish' for Belgium and 'English', 'Scottish', and 'Welsh' for the United Kingdom. To determine the nationality, we count for each country the number of word occurrences in the set of sentences, and simply assign the nationality with the highest count. So far, we did not consider country names of countries that do no longer exist, such as Prussia.

Determining professions. To determine in a similar fashion the professions of a given person, we first have to construct a list of potential professions. This list is generated as follows. We started with a hand-made list of 40 professions, that we extended automatically as follows. Using the nationalities and short list of professions we constructed queries of the form

"[*nationality*] [*profession*] and"

such as e.g. "*Dutch astronomer and*". These were issued to Google, and the resulting excerpts were analysed for additional profession names. More precisely, if in a resulting excerpt the query was succeeded by up to three words without a capital followed by a word with a capital, then these one to three words without a capital are added to the list of potential professions. This resulted in a list of approximately 2500 potential professions. This list includes qualifications that are not professions in the strict sense but are used to characterize persons, such as *free thinker*, *sex symbol*, and *feminist*. Table 3 give the top-40 of the professions found, ranked by the number of times that these professions were found in the excerpts.

PROFESSIONS			
philosopher	1275	designer	222
composer	804	scientist	215
mathematician	773	musician	213
poet	668	historian	210
physicist	501	inventor	208
writer	478	essayist	201
playwright	469	engineer	199
novelist	429	singer	198
sculptor	362	dramatist	186
author	352	theorist	175
critic	346	illustrator	171
astronomer	343	journalist	166
painter	329	statesman	138
politician	323	teacher	138
artist	286	mystic	133
architect	284	educator	132
director	270	theologian	127
conductor	267	physician	125
actor	261	printmaker	124
pianist	224	scholar	112

Table 3: The professions that were found most often.

As for gender and nationality, we now simply count how often each of these profession names occur in the sentences. However, instead of only selecting the one with the highest count, we here want to be able to retain multiple professions. For that reason, we select the ones that have at least a count of $0.5 \cdot c_{\max}$, where $c_{\max}$ is the score of the highest scoring profession, ordered by decreasing count.

4.5 Results

To give an impression of the results that we obtained in this case study, we present three tables. Table 4 gives the top of the persons born in the period $[1880..1889]$, Table 5 gives the top of the persons that has as their highest scoring profession either artist or painter, and Table 6 gives the top of the persons that were identified as Dutch.

Recall. To get an impression of the performance of our algorithm, we estimate the recall by choosing a diverse set of six books containing short biographies of persons whom we would expect to find in our list. For each of these books, we determined for the persons that could potentially be found by our algorithm (i.e., the persons who are born in the intended time period and have died). Of these 1049 persons, 1033 were present in our list, which is a fraction of 0.98. For further details on the chosen books we refer to [15]. We observe that the recall is close to one, for each of the six books, even for a more specialized topic as 17th century Dutch painters. Of the total 108 of these painters mentioned in one of the books, 106 were found. We note that of the 16 persons that did not appear in our list, there were 4 persons for which the books could not provide the lifetime.

BORN IN [1880, 1889]		
James Joyce (1882-1941)	Ireland	author
Bela Bartok (1881-1945)	Hungary	composer
Pablo Picasso (1881-1973)	Spain	artist
Anton Webern (1883-1945)	Austria	musician, composer
HL Mencken (1880-1956)	United States	author, journalist
Niels Bohr (1885-1962)	Denmark	scientist, physicist
Adolf Hitler (1889-1945)	Germany	leader
Amedeo Modigliani (1884-1920)	Italy	artist, painter
Agustin Barrios (1885-1944)	Paraguay	musician, composer
Le Corbusier (1887-1965)	Switzerland	architect
John Maynard Keynes (1883-1946)	United Kingdom	economist
Ludwig Wittgenstein (1889-1951)	Austria	philosopher
Igor Stravinsky (1882-1971)	Russia	composer
TS Eliot (1888-1965)	United Kingdom	poet
Franz Kafka (1883-1924)	Czech Republic	author
Franklin D. Roosevelt (1882-1945)	United States	president
Marc Chagall (1887-1985)	Russia	painter, artist
Martin Heidegger (1889-1976)	Germany	philosopher
Kahlil Gibran (1883-1931)	Lebanon	poet, philosopher,...
Heitor Villa-Lobos (1887-1959)	Brazil	composer

Table 4: The 20 persons born between 1880 and 1889 with the highest GPC.

For the recall of the additional information, we observe that for the 10,000 persons that we considered all were given a gender, 77% were given a nationality, and 95% were given one or more professions.

Precision. All kinds of imperfections can still be observed in our list of famous persons, such as remaining inversions, missing parts of a name, and errors in lifetimes, although each of these occurs relatively infrequently. We concentrate on estimating the fraction of names that do not relate to persons. The corresponding precision that is obtained by the algorithm has been estimated as follows. We selected three decennia, namely 1220-1229, 1550-1559 and 1880-1889, and analyzed for each the candidate persons that were 'born' in this decennium. For the first two decennia we analyzed the complete list, for decennium 1880-1889 we analyzed only the first 1000 as well as the last 1000 names. This resulted in a precision of 0.94, 0.95, and 0.98, respectively. As the decennium of 1880-1889 resulted in considerably more names, we take a weighted average of these results. This yields an estimated precision for the complete list of 0.98 [15].

Regarding the precision of the additional information, we make the following observations. The algorithm will find at most one nationality. For persons that migrated during their lives this poses a problem. Very often, sentences with the pattern "*was born in*" occur so frequent that the country of birth determines the nationality found, such as for Henri Michaux. Adolf Hitler forms an exception on this rule.

Regarding the professions found, we observe that the results are usually quite accurate, even if persons have performed diverse things in life. For example, Leonardo da Vinci has been given the professions artist, scientist, and inventor, and Benjamin Franklin the professions inventor, scientist, statesman, and author. Some of the profession are ambiguous words such as *general*, *director* and *judge*. Such professions lead to less precise results. Also, *king*, *queen*, and *saint* are often used in a metaphorical sense. Errors also occur due to the mentioning of the parents' professions. For example, Edgar Degas was given *banker* as his third profession, because he was

ARTISTS/PAINTERS		
Leonardo da Vinci (1452 - 1519)	Italy	artist, scientist,...
Pablo Picasso (1881 - 1973)	Spain	artist
Vincent van Gogh (1853 - 1890)	Netherlands	artist, painter
Claude Monet (1840 - 1926)	France	artist, painter,...
Pierre-Auguste Renoir (1841 - 1919)	France	painter
Paul Gauguin (1848 - 1903)	France	painter
Edgar Degas (1834 - 1917)	France	artist, painter,...
Paul Cezanne (1839 - 1906)	France	painter, artist
Salvador Dali (1904 - 1989)	Spain	artist
Henri Michaux (1899 - 1984)	Belgium	artist, poet
Gustav Klimt (1862 - 1918)	Austria	painter, artist
Peter Paul Rubens (1577 - 1640)	Belgium	artist, painter
Katsushika Hokusai (1760 - 1849)	Japan	painter
Amedeo Modigliani (1884 - 1920)	Italy	artist, painter
JMW Turner (1775 - 1851)	United Kingdom	artist, painter
James Mcneill Whistler (1834 - 1903)	United States	artist
Rene Magritte (1898 - 1967)	Belgium	artist, painter
Henri Matisse (1869 - 1954)	France	artist
Rembrandt van Rijn (1606 - 1669)	Netherlands	artist, painter
Edouard Manet (1832 - 1883)	France	artist, painter
Herm Albright (1876 - 1944)	-	artist, engraver,...
Marc Chagall (1887 - 1985)	Russia	painter, artist
Edvard Munch (1863 - 1944)	Norway	painter, artist
Wassily Kandinsky (1866 - 1944)	Russia	artist, painter
Francisco Goya (1746 - 1828)	Spain	artist, painter

Table 5: The 25 artists/painters with the highest GPC.

born to a banking family. Some persons remain difficult the characterize, however. Calamity Jane was given a long list of professions: *actress*, *horsewoman*, *prostitute*, *musician*, *entertainer*, *dancer*, and *hunter*. As wife of Franklin Delano Roosevelt, Eleanor Roosevelt is given the profession *president*.

Biographical entries. To get a better impression of the quality of the biographical entries, we manually checked 50 persons, evenly distributed in the top-2500. Of these 50 persons, we observed that gender, nationality and professions were all correct for 38 persons. No errors in gender were detected in any of the 50 persons. For three persons the nationality was not found. All nationalities found proved to be correct. For two persons, all given professions were wrong. For eight others, one or more given professions were incorrect, but usually the professions with the highest count were correct. In the final paper, results of a more extensive analysis will be provided

5. CONCLUSION AND FUTURE WORK

We have presented a framework algorithm for ontology population using Googled expressions. We combine patterns expressing relations and an instance of a class into queries to generate highly usable Google excerpts. From these excerpts we simultaneously extract instances of the classes and instance pairs of the relations.

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.

The results of the experiments are encouraging. We used simple patterns, recognition functions and checks that proved to be successful.

Belgian/Dutch	
Cesar Franck (1822 - 1890, B)	organist, composer, pianist
Vincent van Gogh (1853 - 1890, NL)	artist, painter
Roland de Lassus (1532 - 1594, B)	composer
Abraham Kuyper (1837 - 1920, NL)	theologian, politician
Henri Michaux (1899 - 1984, B)	artist, poet
Peter Paul Rubens (1577 - 1640, B)	artist, painter
Baruch Spinoza (1632 - 1677, NL)	philosopher
Rene Magritte (1898 - 1967, B)	artist, painter
Christiaan Huygens (1629 - 1695, NL)	astronomer, scientist,...
Rembrandt van Rijn (1606 - 1669, NL)	artist, painter
Johannes Vermeer (1632 - 1675, NL)	painter, artist
Edsger Wybe Dijkstra (1930 - 2002, NL)	computer scientist
Anthony van Dyck (1599 - 1641, B)	painter
MC Escher (1898 - 1972, NL)	artist
Antony van Leeuwenhoek (1632 - 1723, NL)	scientist
Piet Mondrian (1872 - 1944, NL)	artist, painter
Hugo Grotius (1583 - 1645, NL)	lawyer, philosopher,...
Jan Pieterszoon Sweelinck (1562 - 1621, NL)	composer, organist,...
Andreas Vesalius (1514 - 1564, B)	physician
Hieronymus Bosch (1450 - 1516, NL)	painter
Audrey Hepburn (1929 - 1993, B)	actress, princess
Ferdinand Verbiest (1623 - 1688, B)	astronomer
Desiderius Erasmus (1466 - 1536, NL)	philosopher, reformer,...
Theo van Gogh (1957 - 2004, NL)	judge, artist
Gerard Dou (1613 - 1675, NL)	painter, artist
Nicolaas Beets (1814 - 1903, NL)	king, poet, writer
Carel Fabritius (1622 - 1654, NL)	painter
Georges Simenon (1903 - 1989, B)	author
Kees van Dongen (1877 - 1968, NL)	painter
Gerardus Mercator (1512 - 1594, B)	cartographer
Emile Verhaeren (1855 - 1916, B)	poet, dramatist
Abel Janszoon Tasman (1603 - 1659, NL)	explorer
Pieter de Hooch (1629 - 1684, NL)	painter
Jan van Goyen (1596 - 1656, NL)	artist
Hendrick Goltzius (1558 - 1617, NL)	artist
Simon Stevin (1548 - 1620, NL)	mathematician
Jacob Jordaens (1593 - 1678, B)	artist, painter
Jan Steen (1626 - 1679, NL)	artist, painter,...
Jacobus Arminius (1560 - 1609, NL)	theologian
Guillaume Dufay (1400 - 1474, B)	composer

Table 6: The 40 Belgian/Dutch persons with the highest GPC.

The current algorithms contain two hand-crafted elements: the construction of the patterns and the identification of the instances. In future, we plan to investigate methods to (semi-) automate these steps.

6. REFERENCES

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM International Conference on Digital Libraries*, 2000.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. In *Scientific American*, May 2001.

[3] T. Bogers. Dutch named entity recognition: Optimizing features, algorithms and output. Master's thesis, Tilburg University, 2004.

[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] A. Brothwick. *A Maximum Entropy Approach to Named Entity Recognition*. PhD thesis, New York University, 1999.

[7] P. Buitelaar, P. Cimiano, and B. Magnini, editors. *Ontology Learning from Text: Methods, Evaluation and Applications*, volume 123 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2005.

[8] N. A. Chinchor, editor. *Proceedings of the Seventh Message Understanding Conference (MUC-7)*. Morgan Kaufmann, Fairfax, Virginia, 1998.

[9] P. Cimiano and S. Staab. Learning by googling. *SIGKDD Explorations Newsletter*, 6(2):24–33, 2004.

[10] V. Crescenzi and G. Mecca. Automatic information extraction from large websites. *Journal of the ACM*, 51(5):731–779, 2004.

[11] 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.

[12] G. Geleijnse and J. Korst. Learning effective surface text patterns for information extraction. Submitted to: *the EACL 2006 workshop on Adaptive Text Extraction and Mining (ATEM 2006)*, 2006.

[13] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, Cambridge, UK, 1997.

[14] 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.

[15] J. Korst, G. Geleijnse, N. de Jong, and M. Verschoor. Ontology-based extraction of information from the World Wide Web. In *Intelligent Algorithms* (Philips Research Book Series, to appear.). Springer, 2006.

[16] N. Kushmerick, F. Ciravegna, A. Doan, C. Knoblock, and S. Staab, editors. *Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web*. Dagstuhl, Germany, February 2005.

[17] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics*, 10:707–710, 1966.

[18] 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.

[19] D. Ravichandran. *Terascale Knowledge Acquisition*. PhD thesis, University of Southern California, 2005.

[20] D. Ravichandran and E. Hovy. Learning surface text patterns for a question answering system. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, pages 41–47, Philadelphia, PA, 2002.

[21] E. Voorhees. Overview of the trec 2004 question answering track. In *Proceedings of the 13th Text Retrieval Conference (TREC 2004)*, Gaithersburg, Maryland, 2004.