

Harvesting Relational and Structured Knowledge for Ontology Building in the *WPro* architecture

Daniele Bagni¹, Marco Cappella¹, Maria Teresa Pazienza¹, Marco Pennacchiotti²
Armando Stellato¹

¹ DISP, University of Rome “Tor Vergata”, Italy. {pazienza,stellato}@info.uniroma2.it
² Computational Linguistics, Saarland University, Germany. pennacchiotti@coli.uni-sb.de

Abstract. We present two algorithms for supporting semi-automatic ontology building, integrated in *WPro*, a new architecture for ontology learning from Web documents. The first algorithm automatically extracts ontological entities from tables, by using specific heuristics and WordNet-based analysis. The second algorithm harvests semantic relations from unstructured texts using Natural Language Processing techniques. The integration in *WPro* allows a friendly interaction with the user for validating and modifying the extracted knowledge, and for uploading it into an existing ontology. Both algorithms show promising performance in the extraction process, and offer a practical means to speed-up the overall ontology building process.

1 Introduction

Ontology learning from text [3] is today receiving growing attention, as a means to (semi-)automatically build ontologies from document collections. From the one side, the development of accurate and scalable Natural Language Processing (NLP) techniques for automatically harvesting different types of information, such as relations [12] and facts [5], has urged the creation of algorithms and applications for structuring and organizing this knowledge into formal ontological repositories. From the other side, the Semantic Web and the Knowledge Representation communities have reached a stage in which the formalisms (such as OWL) and the reasoning engines are ready to host and process the large amount of knowledge harvested by the NLP algorithms. As a further point, the impressive amount of unstructured (textual) and structured (tables and templates) information that we have at our disposal via the Web and other *e*-collections, indicate that the time is mature for integrating NLP techniques into Semantic Web oriented applications aimed at ontologically structuring the Web and other textual content. As a matter of fact, in recent years many tools have been created for semi-automatically supporting human experts in the task of ontology building from documents (e.g. KIM [10], Text-to-Onto [11], etc.). In this paper we present two algorithms for supporting the ontology building process from Web pages in a semi-automatic fashion. Specifically, we present an algorithm for automatically inducing from structured data (i.e. HTML tables) in an existing

ontology, new ontological entities (classes, instances, properties); and an algorithm for automatically extracting semantic relations between ontological instances from unstructured texts. Both algorithms are implemented as two independent modules into *WPro*, a new prototypical architecture for ontology building and engineering. By relying on their integration into the *WPro* graphical interface, the two algorithms offer a direct interaction with the user, for activating, managing and validating the extracted knowledge, and integrating it into an existing ontology.

The goal of *WPro* is to offer a means to support a user or an ontology engineer during the process of ontology building from the Web. It provides a friendly graphical interface divided in two main areas. A first area is the actual Firefox browser, which embodies the second area. This latter includes the ontology which the user is building, together with various tools for ontology engineering, i.e. to manually create and delete entities and to add new entities from the text of Web pages by simple drag-and-drop operations. The ontology is maintained and modified by relying on a background Protégé-based engine [7], which guarantees efficiency and robustness in the ontology engineering operations, and the possibility of producing a final ontology in different formalisms, such as OWL. The *WPro* architecture is implemented using various languages. It is based on XUL for the general integration in Mozilla Firefox, and interfaces to Protégé through the specific Protégé API. *WPro* also provides a set of APIs, which can be used to easily extend the architecture with further tools and modules to better automate and support the ontology building process. The two modules presented in the paper are directly integrated into *WPro* by using these APIs. In the rest of the paper we present our two novel algorithms: in Section 2 we introduce TOE, the *WPro* module implementing the algorithm for inducing ontological entities from HTML tables; in Section 3, we describe the *WPro* module for relation extraction. Both modules show promising and close to state of the art performance. Finally, in Section 4 we draw some final conclusions.

2 The TOE Module

Structured information, such as tables, lists and templates, offer a rich source of knowledge to enrich ontologies. Structured sources have the major advantage that the data they contain are structurally coherently organized, making their ontological interpretation easier with respect to unstructured documents. Also, structured information typically contain *dense* meaningful content which tends to be ontology-oriented, unlike unstructured text, where relevant information are scattered throughout sentences. Despite this, not much attention has been paid so far on the extraction of ontological information from tables. [14] propose *TARTAR*, a system for the automatic generation of semantic ontological frames in Frame Logic, from HTML, Excel, PDF tables and others. The methodology leverages the structural, functional and semantic aspect of tables. Our approach differs from *TARTAR*, both in the input considered (we focus on HTML), the final goal (we enrich an OWL based ontology, *TARTAR* builds independent frames) and in the adopted methodology (we rely on heuristics based on the content, structure and presentation, while *TARTAR* mostly focuses on functional aspects). [16] present *TANGO*, a system which automatically creates *mini-ontologies* from tables, that can be successfully integrated into larger *inter-ontologies*, using ontology mapping techniques. The approach is intended as a

process of reverse engineering from an actual table to a conceptual model. The output is a set of concepts and relationships among them. Unlike *TANGO* we have a more practical focus, as described hereafter.

The TOE module extracts ontological entities (classes, instances and properties) from this type of structured information. Specifically, TOE implements an algorithm for analyzing HTML tables embodied in Web pages, and for proposing a possible ontological interpretation to the user of the *WPro* application. Also, the user can accept, reject or modify TOE's interpretation through an easy to use graphical interface, and then start an automatic process to upload the new information into the existing *WPro* ontology. The TOE module is fully integrated in the *WPro* architecture, and can be activated over a Web page by a simple listener.

2.1 Tables' structure and ontological types

Many models have been proposed in the literature to describe tables (e.g. [9]). We here adopt a simple approach, specifically oriented to our task. In our framework, a table is intended as a matrix of cells (see Fig. 1), whose structure can be divided in four main areas, which in most cases contain different type of information: (1) *First*

<1,1>	<1,2>	<1,3>	...	<1,n>
<2,1>	<2,2>	<2,3>	...	<2,n>
...
<m,1>	<m,2>	<m,3>	...	<m,n>

Fig. 1. Structure of a $n \times m$ table

row (cells <1,2> ... <1,n>), which usually contains a *column header*, i.e. a short description of the information enclosed in each column. (2) *First column* (cells <2,1> ... <2,m>), typically containing a *row header*, describing the content of a single row. (3) *First cell* (cell <1,1>), sometimes used to give a short indication on the type of data contained in the table (*table header*); in other cases, it is part of the first row or the first column. (4) *Internal cells* (other cells), containing the actual *data* of the table, whose meaning is described by the related cells in the first row/column.

The above represents a typical structure, which is verified in most but not all cases. Our empirical investigation on a set of hundreds of tables revealed that most tables follow this structure, apart from two simple variants, in which the row or column headers are absent. TOE's heuristics have been then implemented to treat both the basic structure and these two variants.

The above structure indicates that from an information-content perspective, a table is at least *three-dimensional*, i.e. it can contain three types of information: *row header*, *column header* and *data*. *Table header* is not included, as it is a simple singleton value, which does not express an actual content. A table can also be *two-dimensional* (when either the row or column header is not present). Mono-dimensional tables are seldom, as the value of internal rows and columns must be somehow described by a header (these seldom cases assume that the description of the internal cells is given in the table caption). This observation implies that the ontological content of a table cannot be too complex. We can assume two basic facts: (a) in most cases tables contain simple *flat* non-hierarchical knowledge, i.e. they embody only knowledge describing entities and no hierarchical information; (b) a table cannot contain knowledge about more than one class. In facts, this would imply the table to be four-

dimensional, as it should represent class names, property names, instance names, property values. From these assumptions it follows that to leverage tables in the ontology building process we can classify them in three categories, based on the allowed types of ontological content:

Class Tables: these contain the definition of a class (the name of its properties) and a set of its instances. The information that must be enclosed in the table are: property names, instance names and property values. Potentially, the name of the class can be also represented. The table must then be three-dimensional, i.e. *it must have row header and column header*. Structurally, property names and instance names can be either reported in the row or column headers. Property values are reported in the internal cells, while the class name, if present, is typically in the table header. An example of class table is reported in Fig. 2: the class is “*Business District*”, with the property names indicated in the column header (e.g. “*Office Space*”), the instance names in the row header (e.g. “*The City*”), and the property values in the inner cells.

Instance Tables: these contain information about a single instance. The information enclosed in the table are property name and property value. *These tables are typically two dimensional and have either exactly two columns or rows*. The column (row) header indicates the name of the properties, while the second column contains the property values. A typical example is in Fig. 3: the table contains information for the instance “*London*” of a class *capital cities*, with property names in the first column, and values in the second.

Empty tables: these are tables which do not contain any ontological interesting content. Typical cases of empty tables are those containing graphical elements.

2.1.1 Table extraction and selection. TOE takes as input the Web page currently displayed in the browser window. Once the user activates the module, it automatically extracts from the HTML source all the well-formed tables (a well-formed table is one which is compliant with the HTML 1.0 specifications), using a simple HTML parser. A special case regards nested tables, i.e. tables which recursively contain other tables. In that case, TOE applies the heuristic that the outermost table is that which is

Business District	Office Space (m ²)	Business Concentration
The City	7,740,000	finance, broking, insurance, legal
Westminster	5,780,000	head offices, real estate, private banking, hedge funds, government
Camden & Islington	2,294,000	creative industries, finance, design, art, fashion, architecture
Canary Wharf	2,120,000	banking, media, legal
Lambeth & Southwark	1,780,000	accountancy, consultancy, local government

Fig. 2 . A typical example of a *class table* representing an the class “*Business District*”

Sovereign state:	United Kingdom
Constituent country:	England
Region:	London
Regional authority:	Greater London Authority
Regional assembly:	London Assembly
HQ:	City Hall
Mayor:	Ken Livingstone

City of Rome	
Population by year	
350 BC	30,000
250 BC	100,000
100 BC	500,000
25 BC	1,000,000
120	1,650,000

Fig. 3 (left). A typical example of a *instance table* representing the instance “*London*”

Fig. 4 (right). An example of table with missing column header

informative, while the content of the inner tables is discarded. The motivation of this choice is that our empirical study on a set of table revealed that inner tables tend to contain in most cases graphical objects, or are used for a mere presentation purpose. Once all tables are extracted, the user selects in the graphical interface, the set of tables to analyse. TOE then starts the analysis of one table at time, as follows.

2.1.2 Identifying table type: header analysis. Given an input table, the goal of the first step of the TOE algorithm is to recognize if the table is an *instance table* or a *class table*. A first simple heuristic is applied, according to the assumption in Section 2.1: if a table has more than two column, it cannot be an *instance table*. In this case, TOE predicts that it is a *class table*¹ and goes directly to the ontological analysis step (Section **Errore. L'origine riferimento non è stata trovata.**). If a table has two columns, its type must be decided by verifying if it has a *column header*, as in the following.

From the assumptions in Section 2.1, a table without a header is two-dimensional, and is then an *instance table*. A table with both headers is three-dimensional, and then is a *class table*. Specifically, we here make the further assumption that a row header must be always present in a table. Otherwise, the meaning of the rows would be unknown. On the contrary, the column header can be missing in the case the description of the column is directly indicated in the table caption, as in Fig. 4. Again, this assumption comes from our empirical investigation. The header analysis step is then reduced to the problem of understanding if the table has or not a column header, by using the two following types of heuristics.

Style-based heuristics. The assumption of these heuristics is that a column header has a different style with respect to the rest of the table, e.g. a different background colour or a different font style (bold, italic, etc.). Both background colour and font-style are checked by relying on the information enclosed in related HTML tags (e.g. `` indicates a bold-style, and `<TR bgcolor="red">` indicates a row with a red background). Fig. 2 shows an example captured by these heuristics. A third heuristic looks at the content of the cell `<I,I>`. If it is empty, the first row is assumed to be a column header, otherwise the row would not be meaningful.

Value-based heuristics. If the style-based heuristics fail, a second (less certain) check is made on the values of the cells in the different rows. One of the basic assumption is that when the column header is present, its row is likely to contain cells of the same type, and that this data type is most likely `string` (as it is the most descriptive, as required in a header). Different heuristics are implemented to treat different cases. For example, if rows $2...m$ contain all numeric values (excluding the first column, which represents the row header) and also row 1 contains the same type of numeric values, then the column header is likely to be absent. As another example, if rows $2...m$ contain numeric values and row 1 is formed by `string`, then this latter is likely to be a header. Once the presence of the column header has been verified, the first step decides the table type according to the following heuristic: if the column header is present then the table is a *class table*, otherwise it is an *instance table*. This information is then given to the second step, together with the table itself.

2.1.3 Ontological analysis. on a second step TOE identifies what ontological entity each cell contains. The analysis varies according to the table type.

¹ The decision if a table is an *empty table* is left to the user in the later validation phase. TOE only discards as empty, those table which contains only figures and symbols.

Class table analysis. Class tables describe instances of a given class, i.e. they must contain the property names of the class, the instances names and the related property values. Instance and property names can be alternatively coded in the row or in the column headers, while the property values are in the internal cells.

The only problem is then to understand which header contains which name. For this purpose, the assumption is that an ontological property (either datatype- or object-property) has always the same range. Consequently, if the column header contains the property names, the first element of a column (property name) must be followed by cells of the same data type (property values). The same observation stands for the row header. An example is reported in Fig. 5. In the case that all internal cells are of the same data type, the system simply guesses as default that the column header represents the property names.

In class tables, cell $\langle I, I \rangle$ represents the table header. This cell can either be empty, contain the class name to which the table refers, or contain additional information not interesting for the ontology. The treatment of the table header is left to the user during the validation step.

1971-2000	Jan	Feb	Mar	Apr	May	June	July	Ago	Sep	Oct	Nov	Dec	Total
Maximum temperature (°C)	9,7	12,0	15,7	17,5	21,4	26,9	31,2	30,7	26,0	19,0	13,4	10,1	19,4
Minimum temperature (°C)	2,6	3,7	5,6	7,2	10,7	15,1	18,4	18,2	15,0	10,2	6,0	3,8	9,7
Rainfall (mm)	37	35	26	47	52	25	15	10	28	49	56	56	436

Fig. 5. An example of system complete ontological interpretation for a *class table* “Monthly Weather”. In the column header (*Jan...Dec*) are reported instance names; in the row header (first column) are property names; other cells are property values.

Instance table analysis. Instance tables describe a single instance, and are assumed to be formed by two columns: the left columns (row header) containing property names, and the right column containing the properties’ values (internal cells). Then, in this case TOE does not have to perform any specific operation.

At the end of the ontological analysis step, TOE returns an ontological interpretation of the table, by indicating the ontological type of each cell (property name or property value). This interpretation is then reported to the user for validation.

2.1.4 Validation and Ontology Uploading. In this phase, TOE shows to the user the results of the ontological analysis (see Fig.6). The user can then decide either to reject the table as not interesting (*empty table*) to accept completely TOE’s interpretation, or to modify it. In the latter case, the user is provided with different tools to change the ontological type of cells. Once the correct ontological interpretation of the table is decided, the information is automatically uploaded in the ontology. Yet, the user has to manually specify some information:

- *Class/Instance name*, for *class/instance tables*. In the case of class tables, TOE gives as suggested name the value in the table header (cell $\langle I, I \rangle$), or a name induced by a lexical-semantic analysis of the table (see Section **Errore. L'origine riferimento non è stata trovata.**).
- *Ontological Attachment*: the user must specify where in the existing *WPro* ontology the information must be added. For *class table* it must be indicated the parent class of the new class, for *instance table* which is its class in the ontology.
- *Properties range*: the user has to indicate the range of each property.

Before information are finally uploaded in the ontology, TOE performs correctness checks on all entities, and consistency checks on the (possible) new properties and on the (possibly) discrepancies in the ranges of existing ones. In case of incorrect data, the user is requested to perform the needed correction.



Fig. 6. Validation interface of the TOE module. Different cells' colors indicate different ontological entities (*blue*, row header: property names ; *green*, column header: instance names; *white*, internal cells: property values; *gray*, cell <1,1>: table header).

2.1.2 Naming classes: WordNet semantic analysis. If requested, TOE can optionally propose a name for the new class of a *class table*. The name is induced by a lexical-semantic analysis of the table, carried out using WordNet [6] as support. The intuition is that a class name is likely to be a common ancestor of the instances' names in the WordNet hierarchy. Then, for each cell k containing an instance name, TOE extracts its content W^k , which can be formed by one or more word $w^k_i \in W^k$. It then finds the *least common subsumer* [17] of at least one word w^k_i in each of the k cells, up to a given degree of generalization chosen by the user through the graphical interface. In order to cope with acronyms and abbreviations (which are in most cases not present in WordNet), TOE uses as support a gazetteer of acronyms and abbreviations, which is used to expand them into their original form (e.g. "fifa" \rightarrow "international football federation").

2.2 Experimental investigation

The goal of the experiment is to verify the TOE's accuracy in proposing the correct ontological interpretation of a table. We chose as corpus a set of 100 Web pages of European and Asian capitals taken from *Wikipedia*. These pages contain in all 207 tables. We evaluated TOE on three tasks: **(1) table type identification:** the accuracy a_{ident} (fraction of correct predictions) in predicting the table type (i.e. *class* or *instance table*). **(2) ontological interpretation:** the accuracy a_{ont} in predicting a completely correct interpretation of the table (in all cells). **(3) WordNet analysis:** the accuracy a_{WN} of WordNet in predicting a correct name for class tables. The performance evaluation has been carried out by an expert ontology engineer². Results, reported in Table 1, show that TOE guarantees a high accuracy in both tasks (1) and (2). In particular, TOE is highly accurate in predicting the correct table type, revealing that the simple heuristics implemented are effective. Yet, the accuracy in the WordNet

² Table recognized as *empty tables* are not considered in the evaluation

prediction is low. This indicates that in general the use of linguistic analysis in tables is not particularly effective, as the linguistic content is very limited. Also, in many cases we verified problems of coverage: authors of tables often tend to use atypical word abbreviation and truncations in order to limit the cell content. This causes WordNet and the gazetteer do fail in interpreting many expressions. In the future we plan to expand our evaluation to more heterogeneous and large sets of Web pages.

Table 1. Accuracy of TOE over a corpus of 100 Web pages (207 tables).

<i>task</i>	(1)	(2)	(3)
<i>accuracy</i>	0.91	0.77	0.25

Even if not directly comparable, as they aim at extracting different knowledge, both TOE and the TARTAR systems show good performance (TARTAR has an accuracy of 0.85 on the construction of frames over a corpus of 158 tables), which indicate that the extraction of ontological information from table is valuable, feasible and effective.

3 The Relation Extraction Module

Relation extraction is generically intended as the task of extracting generic and specific binary semantic relations between entities from textual corpora, such as *is-a(bachelor, man)* and *capital-of(Roma, Italy)*. It plays a key role in ontology learning from text, and in general in NLP applications which leverage ontological information, such as Question Answering (QA) and Textual Entailment. Consider for example a web portal application supporting on-line questions regarding geographical knowledge. It could answer questions such as: “*Where is Cape Agulhas located?*” or “*What is the longest river of Mozambico?*” by relying on a domain ontology built using a relation extraction engine applied to the relations *located-in(x,y)* and *longest-river-of(x,y)*. It is here important to note that many applications which leverage domain ontologies require information which are in most cases relational, or that can be reduced to a relational format. Relation extraction is then often a key issue.

Most approaches to relation extraction are either pattern- or clustering-based. *Pattern-based approaches* are the most used: [8] pioneered, using patterns to extract hyponym (*is-a*) relations. Manually building three lexico-syntactic patterns, [8] sketched a bootstrapping algorithm to learn more patterns from instances, which has served as the model for most subsequent pattern-based algorithms, such as [1] for the *part-of* relation. [15] focused on scaling relation extraction to the Web, proposing a simple and effective algorithm to discover surface patterns from a small set of seeds. [12] propose an approach inspired by [8] to infer patterns, making use of generic patterns and applying refining techniques to deal with wide variety of relations and principled reliability measures for patterns and instances. Our approach uses a technique similar to [12]: yet, it uses a dependency parser to extract patterns, allowing the exploitation of long distance dependencies. *Clustering approaches* have so far been applied only to *is-a* extraction (e.g. [4],[13]): they use clustering to group words, label the clusters using their members’ lexical or syntactic dependencies, and extract an *is-a* relation between cluster members and labels. These approaches fail to produce coherent clusters from less than 100 million words, being then unreliable for small corpora.

In the framework of the *WPro* environment, which aims at supporting the creation of domain ontologies from the Web, the relation extraction module plays a primary role. Suppose for example that the *WPro* ontology contains the entities *Madrid* and *Spain*,

related by the property *capital-of*. It can then be assumed that the semantic relation *capital-of* between *cities* and *nations* is interesting for the ontology. The module should then allow the *WPro* user to extract automatically other instances of *capital-of* from the web collection. Our relation extraction module aims at providing such a service. Given a pair of entities related by an ontological property, the module provides the user a dedicated interface to start a *relation extraction engine* for the relation. The engine implements an algorithm based on the framework adopted in [8]. It uses the pair of entities as seeds (e.g. (*Madrid,Spain*)) and starts the extraction of other pairs implementing the following steps. First, it collects from the corpus of Web pages all sentences containing the seed (e.g. “*Madrid is a beautiful city and is the capital of Spain*”). Secondly, the syntactic patterns connecting the entities in the seed are extracted from the sentences (e.g. *Madrid-is-the-capital-of-Spain*). Then, patterns are used to extract new instances from the corpus (e.g. (*Roma,Italy*) is extracted from the snippet “*Roma has been the capital of Italy since 1870*”). Finally, the engine returns to the user a list of extracted instances, ranked according to a reliability measure; the user can then validate the instances using an interface and upload them in the ontology. The module guarantees: (1) *minimal supervision*, by using as input one of few instance(s) already in the ontology, and by presenting as output a list of ranked instances which can be easily validated and uploaded; (2) *high accuracy*, by adopting a dedicated reliability measure to weight the extracted instances; (3) *easing data sparseness*: as *WPro* must efficiently work in small domain corpora, our algorithm implements specific techniques, such as syntactic expansion and the use of dependency parsing to exploit long distance dependencies; (4) *generality*, as it is applicable to a wide variety of relations; (5) *WPro integration*, being explicitly integrated in the *WPro* architecture, leveraging the APIs and dedicated user interfaces.

3.1 Module Components and Algorithm

Hereafter we present the different components of the module.

Input Interface. This provides to the user the functionalities needed for starting the relation extraction process, by allowing to select a seed pair $s=(x_s,y_s)$ for a given relation. The pair consists in two entities of the *WPro* ontology, related by an object property. Fig.7 shows an example, in which the user selects as seed words the pair $s=(\textit{Madrid,Spain})$. The extraction process is then executed as follows³.

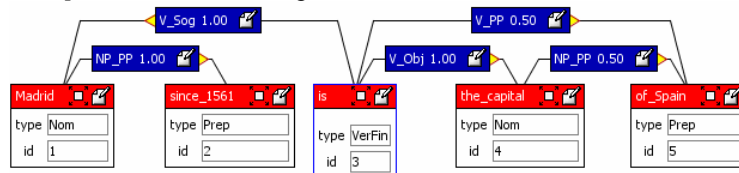


Fig.7. A dependency graph output by *Chaos*. Lower boxes are constituents. Upper boxes are dependencies, with the related *plausibility*, i.e. a representation of ambiguous

Pattern Induction and Expansion. Given an input seed instance s , the algorithm looks in the corpus for all sentences containing the two terms. These sentences are

³ In the present implementation the system carries out the following phases only once. We also experimented bootstrapping iterative calls over these phase, as suggested in [12]. In Section 3.2 we report early results.

parsed by the *Chaos* constituent-dependency parser [1]. Then, all dependency paths connecting the seed words are extracted as patterns P . The use of *Chaos* guarantees two main advantages with respect to simple surface approaches such as [15]. First, the use of dependency information allows to extract more interesting patterns. Second, *Chaos* explicitly represents ambiguous relations between constituents, allowing to infer patterns also when the syntactic interpretation is not complete. Fig. 7 shows a parsed sentence connecting the seeds *Madrid* and *Spain*. The algorithm extracts as patterns the paths: “ X is the capital of Y ” and “ X is of Y ”, which connect the two words⁴. Notice that a simple surface approach would have extracted the only irrelevant pattern “ X since 1561 is the capital of Y ”. The dependency analysis thus allows our algorithm to extract more useful patterns, helping to deal with data sparseness, i.e. cases in which only few sentences are retrieved for a given seed, and then patterns must be carefully created. Yet, the algorithm is prone to capture too generic patterns such as “ X of Y ”: a reliability measure is applied to cope with this problem. Also, to further deal with data sparseness, the algorithm expands the patterns P in a bigger set P' , by including different morphological variations of the main verb (e.g. “ X being the capital of Y ”, “ X was the capital of Y ”, etc.).

Instance Induction and Reliability Ranking. Given the set P' , the algorithm retrieves all the sentences containing the words of any $p \in P'$. Each sentence is then parsed by *Chaos*. The constituents connected by a dependency path corresponding to a pattern in P' are then extracted as new instances I . For example from the sentence “*New Delhi being the capital of India, is an important financial market*”, it is extracted the new instance (*New Delhi, India*).

Each instance $i=(x,y) \in I$ is then assigned a *reliability score*, according to a measure $R(i)$, which accounts for the intuition that an instance is reliable, i.e. it is likely to be correct, if: (1) it is activated by many patterns; (2) the Part-of-Speech (PoS) of the instance and of the seed s are the same; (3) the semantic class of x and y are respectively similar to those of x_s and y_s . The measure is then:

$$R(i) = \alpha \frac{|P_i|}{|I|} + \beta POS_i + \gamma \left(\frac{1}{k} + \frac{1}{j} \right)$$

where P_i are the patterns activating i ; POS_i is a binary value which is 1 if the PoS of i and s are the same, 0 otherwise; k and j are the depths of the *least common subsumer* [17] respectively between x and x_s and between y and y_s in the WordNet [6] hyperonymy hierarchy. α , β , and γ parameters sum to 1, and weight the contribution of respectively point (1), (2) and (3). The value of the parameters can be manually set, or induced using exploratory experiments.

Validation Interface and Ontology Uploading. The ranked list of extracted instances I according to $R(i)$, is returned to the user, via a validation interface. The user can select the instances to be uploaded in the ontology by a simple click. To support the user and speed-up the validation, instances with different degrees of reliability are displayed in the interface with different colors. Once the validation is finished, selected instances are uploaded in the *WPro* ontology: x and y are inserted in the same ontology class of x_s and y_s and the related object property is activated. For example, if $x_s=Madrid$ is an instance of the ontology class *city* and $y_s=Spain$ is instance of *nation*, then the new ontological entities *New Delhi* and *India* are added as instances of the class *city* and *nation*, and related by the object property *capital-of*.

⁴ For clarity, we here do not report the grammatical dependencies of the pattern, but only its surface form.

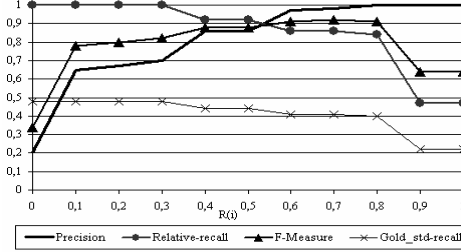


Fig. 8. Precision, Relative-Recall, F-measure and GoldStd-Recall at different levels of τ for the relation *capital-of*.

3.2 Experimental Investigation

The goal of the experiments is to verify the accuracy of the relation extraction algorithm in harvesting instances in a typical *WPro* framework, i.e. a small domain specific corpus of Wikipedia documents, and the extraction of domain related relations. We chose as domain of interest the domain of *European and Asian cities*, and *capital-of* and *located-in* as target relations. The corpus is composed of 80 Wikipedia pages about capitals, consisting in 207,555 tokens. The seed instance for the algorithm is $s=(Madrid,Spain)$, and the parameters α , β , and γ are set to 0.05, 0.25 and 0.75, by using a small annotated development corpus of 10 pages. As gold standard we use a list of instances I_{gs} manually extracted from the corpus. As the goal of the experiment is also to evaluate the effectiveness of the reliability measure $R(i)$, we measure performance in term of *precision* P , *relative-recall* R , *F-measure*, and *goldStd-recall* G at different levels of a threshold τ . The set of instances $I_{\tau} \in I$ which have a score $R(i)$ above the threshold are taken as accepted by the system. At each level of τ , P_{τ} and R_{τ} , F_{τ} and G_{τ} are then defined as follows:

$$P_{\tau} = \frac{|I_{\tau} \cap I_{gs}|}{|I_{\tau}|} \quad R_{\tau} = \frac{|I_{\tau} \cap I_{gs}|}{|I_{gs}|} \quad F_{\tau} = 2 \frac{P_{\tau} * R_{\tau}}{P_{\tau} + R_{\tau}} \quad G_{\tau} = \frac{|I_{\tau} \cap I_{gs}|}{|I_{gs}|}$$

GoldStd-recall is intended to capture the recall over the gold standard, while relative-recall captures recall at a given threshold over all extracted instances. Results for the *capital-of* relation are reported in Figs. 8-9. They reveal that in general our algorithm is able to extract instances with high precision and recall. For example, at $\tau=0.5$, precision is high (almost 0.90) while goldStd recall is still acceptable, about 0.45. Precision is in general comparable to that obtained by state of the art algorithms: for example [12] obtain 0.91 on a chemistry corpus of the same size as our for the *reaction* relation. Yet, our recall is lower, as we do not exploit generic patterns. Figures indicates that according to the intuition, as the threshold grows, precision improves, while recall decreases. This indicate that our reliability measure is coherent and can be effectively used to select correct/incorrect instances ad different levels of reliability. Fig.9 reports some of the best scoring instances extracted by the algorithm for both *capital-of* and *located-in* relations (in all, the algorithm extracted around 50 instances for both relations). From a qualitative perspective, most of the erroneous extracted instances correspond to parsing errors or to the induction of wrong patterns (e.g. the incorrect instance (*Antananarivo, University*) for the *capital-of* relation is fired by the wrong pattern “*X is home of Y*”). As a further exploratory study we experimented our algorithm using two iterations over the loop *pattern induction – instance extraction*, hoping to improve recall. Results show that we extract more

correct instances, without losing much precision. It is interesting to notice that the best choice of α , β , and γ values indicates that the semantic information conveyed by the WordNet measure is the most important to select correct instances (as $\gamma=0.75$).

4 Conclusions

We presented two modules for supporting the semi-automatic enrichment and building of ontologies from Web pages, in the framework of *WPro*. Both modules guarantee a significant speed-up in the construction process, by automatically extracting information that otherwise would require time consuming manual analysis. The modules guarantee high level of precision and recall. In the future, we plan to integrate other linguistically-based modules in the *WPro* architecture, such as terminology and event extractors; and to improve the performance of TOE and the relation extractor, by using new heuristics for pattern induction. We will expand and improve the functionalities of *WPro*, to guarantee more usability and robustness in the architecture, which at the moment is intended as a prototypical framework.

References

1. Basili, R. and Zanzotto, F.M. 2002. Parsing engineering and empirical robustness. In *Natural Language Engineering*, 8/2-3, pp. 1245-1262.
2. Berland, M. and E. Charniak, 1999. Finding parts in very large corpora. In *Proceedings of ACL-1999*, pp. 57-64. College Park, MD.
3. Buitelaar, P., Cimiano, P. and Magnini, B. 2005. *Ontology learning from texts: methods, evaluation and applications*. IOS Press.
4. Caraballo, S. 1999. Automatic acquisition of a hypernym labeled noun hierarchy from text. In *Proceedings of ACL-99*, pp 120-126, Baltimore, MD.
5. Etzioni, O., Cagarella, M.J., Downey, D., Popescu, A.M., Shaked, T., Soderland, S., Weld, D.S. and Yates, A.. 2002. Unsupervised named entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91-143.
6. Fellbaum Christiane, editor. 1998. *WordNet: Electronic Lexical Database*. MIT Press.
7. Grosse, W. E., Eriksson, H., Fergerson, R. W., Gennari, J. H., Tu, S. W. and Musen, M. A. 1999. *Knowledge Modeling at the Millennium*.
8. Hearst, M. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of COLING-92*, pp.539-545. Nantes, France.
9. Hurst, M.. *Layout and language: Beyond simple text for information interaction-modelling the table*. In *Proceedings of the 2nd ICMI*, Hong Kong, 1999.
10. Jüling, W., Maurer, A. *Karlsruher Integriertes InformationsManagement*. 2005. in PIK 28.
11. Maedche and Volz. 2001. *The Text-To-Onto Ontology Extraction and Maintenance Environment*. In *Proceedings of the ICDM Workshop on integrating data mining and knowledge management*, San Jose, California, USA.
12. Pantel, P and Pennacchiotti, M. 2006. Espresso: A Bootstrapping Algorithm for Automatically Harvesting Semantic Relations. In *Proceedings of COLING/ACL-06*. Sydney, Australia.
13. Pantel, P. and Ravichandran, D. 2004. Automatically labeling semantic classes. In *Proceedings of HLT/NAACL-04*, pp. 321-328. Boston, MA.
14. Pivk, A., Cimiano P., Sure, Y., Gams, M., Rajković, V., Studer, R. 2007. Transforming arbitrary tables into logical form with TARTAR. *Data & Knowledge Engineering*, 60:3.
15. Ravichandran, D. and Hovy, E.H. 2002. Learning surface text patterns for a question answering system. In *Proceedings of ACL-2002*, pp. 41-47. Philadelphia, PA.
16. Tiberino, A.J., Embley, D.W., Lonsdale, D.W., Ding, Y., Nagy, G. 2005. Towards Ontology Generation from Tables. *World Wide Web: Internet and Web Information Systems*, 8, 261-285.
17. Wu, Z. and Palmer, M.. 1994. Verb semantics and lexical selection.