Query Expansion and Noise Treatment for Information Retrieval

Emerson L. dos Santos
Fabiano M. Hasegawa
Bráulio C. Ávila
Celso A. A. Kaestner

Pontifical Catholic University of Paraná — PUCPR
R. Imaculada Conceição, 1155 — 80.215-901 — Curitiba — PR — Brazil
{fmitsuo, emerson, avila, kaestner}@ppgia.pucpr.br

Abstract

Most of the search engines available over the Web are based on mathematical approaches — classical techniques in the Information Retrieval area. Thereby, they are suitable for the retrieval of documents containing some or all the terms of a query, though not to retrieve the documents containing the meaning those terms were intended to express. This paper presents some advantages obtained from query expansion with WordNet and noise treatment with knowledge on top of Paraconsistent Logic. Both methods are semantically driven, allowing the retrieval of documents which do not contain any term of the original query. Noise treatment results from the combination of a smooth term comparison with knowledge about term authentication based on behaviors of features in the collection. Although query expansion recurs for every query, noise treatment is part of the indexing mechanism, causing no overhead in queries. The domain is retrieval of ontologies represented in Resource Description Framework.

1 Introduction

Search engines are usually built according to three paradigms well defined in the Information Retrieval (IR) field: boolean, vector and probabilistic models. Those techniques were developed under assumptions concerning mathematical and statistical issues. The main problem with those techniques concerns the inability to deal with semantic issues, thus usually failing to retrieve some relevant documents and retrieving lots of irrelevant ones.

The collection chosen for this work was a set of documents represented in Resource Description Framework (RDF) — a framework designed for processing data describing Web resources [6]—, available at the site `http://www.daml.org/ontologies`. Documents from a domain are usually represented according to a vocabulary defined for that specific domain. In this collection, there are user-defined vocabularies covering some domains. Unfortunately, it is not clear which ontology should be used for an arbitrary domain. That requires one to search manually for a suitable ontology. This work presents a tool for ontology retrieval from an RDF ontology database, which implements the techniques described along the paper.

In this domain, each ontology is a document — because there is only one ontology a file. The program uses the same classical vector model acting upon a *tf-idf* index, such as other engines but with some important differences. Firstly, an *absolute frequency* index is constructed capturing the values of both properties which contain natural language in this domain: `label` and `comment` [6]. Secondly, the query is augmented using *WordNet* in order to generate more terms related to the original ones. Next, a smooth comparison of terms — based on semantic knowledge instead of lexical string approximation — is used to tolerate spelling mistakes in the documents. Afterwards, expert knowledge on top of Paraconsistent Logic along with automatic acquisition of features behaviors applies as an extension to the previous module in order not to allow unifications which are not supposed to occur — two different terms t_1 and t_2 which are the same from the point of view of the smooth unification cannot be unified if they are both authentic. Authentic terms are those ones which were really intended to be spelled the way they are; they are not spelling mistakes. Next, a new *absolute frequency* index is generated. In this index, only authentic terms are allowed; non-authentic terms are excluded and their frequencies are proportionally distributed among their possible intended forms. Finally, the *tf-idf* index required for the search engine is built.

There are two toplevel goals in this work which must be emphasized. Firstly, it is important to recall that semantic knowledge can be used to deal with natural language as a form to bring IR methods closer to cognitive processes. As people only understand natural language because of semantic knowledge stored in their memory structures, programs cannot be expected to deal properly with natural language without enough semantic knowledge. The motivation is that a bulk of AI research is concerned with domains that require semantic knowledge but only lexical, syntactical, mathematical, statistical and related approaches have been applied. Secondly, it must be asserted Paraconsistency is a suitable formalism to represent knowledge from multiple sources. Since contradictions can be identified easily, several opinions can be collected and amalgamated into one single set of rules and facts. In this case, the motivation is that more than one expert might contribute with their own opinions. Those opinions might fall onto contradictions.

In **Section 2**, *WordNet* is evaluated as a source of semantic relationships for query augmentation. **Section 3** shows how naïve string approximation methods for misspelling noise tolerance can be replaced by rules conveying semantic knowledge. Next, knowledge from experts is expressed in the form of paraconsistent rules in order to avoid inadequate unifications, reducing the impact of erroneous actions. The system architecture is presented in the fifth section. Arguments about some trouble in the implementation are exposed next. Finally, a conclusion is presented along with suggestions for future work. Examples concerning the work are supplied in the Appendices.

2 Improving Recall of Queries

WordNet is a lexical database which provides several sorts of relationships among concepts and their respective signs [4]. However, IR deals only with signs rather than concepts. Thus, although *WordNet* provides relationships among concepts, without the meaning of a term what one gets is just relationships among signs.

Anyway, *WordNet* can still increases the *recall* of a given query. By acting as an auxiliary tool for query expansion [5], *WordNet* makes the query able to retrieve documents where the terms of the original query did no occur related ones did. There is a significative chance relevant documents may contain those related words instead of the original words. Nonetheless, the amount of documents retrieved may grow out of control, jeopardising *precision*.

Two *WordNet* relationships were chosen: *synonym* — two words w_1 and w_2 are synonyms

Query	Regular		Expanded	
	Precision	Recall	Precision	Recall
q_1	0.66	1	0.5	1
q_2	0.41	0.24	0.4	0.31
q_3	0.44	0.11	0.46	0.17
q_4	0.25	0.08	0.14	0.16
q_5	0	0	0	0
q_6	0.27	0.1	0.23	0.1
q_7	0	0	0.04	0.33
q_8	0.18	0.5	0.06	0.75
q_9	0.14	0.25	0.1	0.25
q_{10}	0	0	0	0
q_{11}	0.1	0.31	0.16	0.75
$\overline{q}_i$	0.22	0.24	0.19	0.35

Table 1: Comparison between regular and expanded search.

if and only if w_1 and w_2 can express the same concept — and *hyponym* — a word w_1 is an hyponym of another word w_2 if and only if w_1 expresses a more specific type of the concept that w_2 expresses.

Thereby, the query is augmented by collecting the whole set of synonyms and hyponyms of each term of the query and adding them to the original terms. The search is then initiated with all those terms.

In **Table 1**, it is remarkable the improvement on *recall*. Four out of eleven queries had its *recall* increased; on average, a increase of 33% on *recall* and a decrease of less than 9% on *precision*.

Although *recall* was indeed increased, **Table 1** also shows *precision* was decreased: as the search engine do not know the meanings of the terms, it cannot be supposed to notice inadequate meanings in an arbitrary document and those might be lots due to the amount of new terms which were added; however, this occurs even for the original words of the query and is a known trade-off in the IR community. Anyway, the gain with *recall* seems to enough to compensate what was lost in *precision*.

3 Noise Tolerance with Smooth Unification

As it is common in real world data, the RDF collection contained several typing errors. If no tolerance were applied, the search engine would not be able to identify *approximmation* was, in fact, intended to be *approximation*, for instance. There are some algorithms for approximate string unification [2]. Yet, those algorithms do not take into account knowledge about which kinds of mistakes can be allowed; they care about letters rather than types of mistakes. Human beings tend to analyse two terms as a whole when judging whether one of them is, actually, just the other one misspelled. They use to allow certain sorts of mistakes in some cases and not in others. They care about the size of a string as well as the relapse of mistakes. In other words, people decide whether two almost equal strings should be, indeed, totally equal taking into account knowledge about rules derived from cases they have already seen instead of a rigid and nonsense letter-oriented algorithm.

Query	Regular		Smooth	
	Precision	Recall	Precision	Recall
q_1	0.66	1	0.16	1
q_2	0.41	0.24	0.38	0.24
q_3	0.44	0.11	0.45	0.14
q_4	0.25	0.08	0.25	0.08
q_5	0	0	0	0
q_6	0.27	0.1	0.25	0.1
q_7	0	0	0	0
q_8	0.18	0.5	0.12	0.5
q_9	0.14	0.25	0.14	0.25
q_{10}	0	0	0	0
q_{11}	0.1	0.31	0.1	0.31
$\overline{q}_i$	0.22	0.24	0.17	0.24

Table 2: Comparison between regular and smooth search.

The tolerant unification was implemented according to some heuristics which try to represent the way human beings reason when they must decide whether two strings should or should not be equal. The heuristics were expressed as rules and concern the following ideas:

- Two strings are equal if a mistake does not occur more than once before it is forgotten;
- A mistake is forgotten if it was not repeated within the word in the last θ characters [1];
- There are three kinds of mistakes: *transposition* mistake — which is the reversal of two letters —, *typing* mistake — when a different letter is found instead of the expected one — and *missing* mistake — when a letter is missing[2].

It can be noticed in **Table 2** *recall* was not affected by the smooth unification. Probably, there was not a significative number of mistakes on the documents. So allowing smooth matching was useless for *recall*. Nonetheless, its effects for *precision* were extremely bad: lots of inadequate unifications were allowed.

In principle, this approach would be used in the search. Every term in the index which matched the terms on the query would be used to retrieve the related documents. However, this technique proved not to be suitable because of inadequate unifications, according to the results provided in **Table 2**. Actually, the way it was being dealt with was completely incorrect. Consequently, a new strategy turned out to be required: it is not the possible matchings that should be recognized in the search; search must look for exact words. Instead, the index must be properly constructed in order to incorporate misspelling issues. That is explained in the next section.

Briefly, the trouble with using a smooth search process is that it is not taken into account knowledge about authenticity of a word: is a word really "that" word? If it is an authentic word, it must not match with other authentic words; if it is not authentic, it should not exist in the index: it is a misspelling. A smooth unification would recognize similar words. This is not the point: only the intended word and its misspellings should be recognized.

[1]In this work, the value of θ was 3.

[2]Additional letters are treated as missed letters, as none of the terms is considered to be right *a priori*.

4 Paraconsistent Logic as a Foundation for Term Authentication

Although noise tolerance allowed the search engine to recognize misspelled words, the results were quite bad, as shown in **Section 3**. The trouble arises when an arbitrary term is compared with similar authentic terms[3]. The tolerant unification was designed not to care about some differences when comparing two terms in order to allow partial matchings. Unfortunately, the tolerant unification also matches two authentic terms which are alike spelled but have distinct meanings. The tolerant unification cannot realize when two similar terms are both authentic because it does not have any knowledge concerning this issue.

Checking whether a term exists in a thesaurus is not enough. Normally, a misspelled term is not found in a dictionary; however, the mistake term might have been spelled in such a way that it still exists in a thesaurus. Plainly, authenticity of a term is not just the same as its existence in a dictionary. Since there is no contextual aid due to defficiencies of the paradigm, the terms of the indices and their respective frequencies in the documents are all the available information to decide whether a term t_2 is either just the authentic term t_1 misspelled or, indeed, another authentic term. If it is a misspelling, its frequency should be added to t_1's and its entry removed from the index.

By analogy, it is possible to raise some craw issues about what would make someone recognize misspellings and authentic words properly. Simple heuristics like term frequency, document frequency and knowledge concerning the existence of that word can evolve to more complex heuristics. Alternatively, different views might be collected and the whole bag of heuristics used together. A crucial point must be observed here: how to eliminate the inconsistency from the knowledge base? An elegant and light strategy is, certainly, not to worry about the inconsistent cases and to query the knowledge base through an intermediate module in such a way that an answer could recognize inconsistencies. Thus, the problem of inconsistency would be postponed and the decision making process could predict actions for each case directly in the application. *ParaLog* [1] is supposed to be such a module.

4.1 Notes on Paraconsistent Logics

ParaLog is a reasoner based on Paraconsistent Logics [3] implemented on Prolog. Thus, *ParaLog* is able to represent explicitly a negation, instead of just supressing it[4] — like Prolog does — and to identify inconsistencies. There are several Paraconsistent Logics; the interesting one here, which is the basis for *ParaLog*, is the *Infinitely Valued Paraconsistent Logics*, whose lattice is showed in **Equation 1**.

$$|\tau| \ = \ [0,\ 1] \ \times \ [0,\ 1] \tag{1}$$

The elements situated on the corners of the lattice τ defined in **Equation 1** are particularly interesting for the context of this work because together the meanings of their annotations constitute the set {`indeterminate`, `false`, `true`, `inconsistent`}, which is also represented by $\{\bot,\ f,\ t,\ \top\}$. The infimum [0, 0] and the supremum [1, 1] of the lattice τ denote the values `indeterminate` and `inconsistent`, respectively. Likewise, the elements [0, 1] and

[3]A term is considered authentic if it is not a spelling mistake.

[4]Actually, a supressed information in *ParaLog* is an unknown information — nothing was declared about it. This is native from Paraconsistent Logic: no one has said it is true, as well as no one has said it is not. Thus, at least for a while, it is neither `true` nor `false`: it is `indeterminate`.

CERTAINTY DEGREE	CONFIDENCE
$G_1 \geq 0.5$	The *confidence* of p_1 is assumed to be `true`.
$G_1 \leq -0.5$	The *confidence* of p_1 is assumed to be `false`.
$-0.5 < G_1 < 0.5$	p_1 cannot be stood on safely.

Table 3: Conversion of *certainty degree* into *confidence.*

[1, 0] correspond to the values `false` and `true`. Each proposition in *ParaLog* is qualified with an annotation of the lattice τ. The values which compose an annotation are called evidences. The first evidence favours the proposition and is named *belief*; the other, named *disbelief*, is contrary to the proposition. Both evidences may be used to determine a single discrete value of *confidence* on a proposition, from the set {`false`, `uncertain`, `true`}, using the *certainty degree* G, as defined in [3], showed in **Equation 2**.

$$G = \mu - \rho \tag{2}$$

Let p_1 be a proposition whose annotation $[\mu_1, \rho_1]$ yields a *certainty degree* G_1. So with a *certainty degree* threshold $\phi = 0.5$, for example, the assertions in **Table 3** hold. According to **Table 3**, if the *confidence* of a proposition p is `uncertain`, it is unwise to assume either `true` or `false` as a default value. An alternative would be to try to find more information about p and reevaluate it until its *confidence* become either `true` or `false`.

Thus far, the basis for knowledge representation of term authenticity in this paper has been defined. The process which determines whether a term either is or is not authentic is described next.

4.2 Heuristics for Term Authentication

Firstly, some craw heuristics related to term, based on the supposed relations between *confidence* on the authenticity of an arbitrary term and behaviour of some features noticed in collections, were defined. the higher the value of a certain feature is, the more the *confidence* on the authenticity of the referred term increases or decreases, except for the feature *Existence in WordNet* which indicates that if a term exists in the *WordNet*, then the *confidence* on the authenticity of it increases.

However, continuous values seem to be at least a little bit harder to be dealt with than discrete values — such as the value of the feature *Existence in WordNet*. So the behaviours of the features were converted from continuous to discrete for simplicity. That allowed the use of facts with annotations corresponding only to values in the subset {`false`, `true`} of the lattice τ. The resulting heuristics are presented in **Table 4**. *Opponents* of a term t are all the terms in the collection which are similar to t according to the smooth term unification. An arbitrary *tf* is always discriminated by a term and a document, while *document frequency* (*df*) is only

EXISTENCE OF	PRESENT	AUTHENTICITY
At least one high *tf*	$+$	$\upuparrows$
High *df*	$+$	$\upuparrows$
High frequency of high-*tf*s	$+$	$\upuparrows$
High frequency of opponents	$+$	$\upuparrows$
Referred term in *WordNet*	$+$	$\upuparrows$
High frequency of high-*tf* opponents	$+$	$\downdownarrows$
At least one high-*tf* opponent	$+$	$\downdownarrows$

Table 4: Discrete heuristics to determine the authenticity of a word.

discriminated by a term. Thereby, an index of *absolute frequency* representing *term frequency* (*tf*) is required.

The developer must define the thresholds for a decision between which of the elements of the set {`false`, `true`} — that, actually, denote {$\neg$ `high`, `high`} — should be attributed to a term in order to represent the corresponding fact with that information along with the proper annotation. In the tests presented in this paper, dynamic thresholds λ_k were obtained as indicated in **Equation 3** for each of the k features evaluated.

$$\lambda_k = \overline{f}_k \tag{3}$$

In **Equation 3**, $\overline{f}_k$ means the global *average* of the values of the feature k[5]. In this approach, each term t is evaluated according to each of the features k. Thereby, the result of the procedure applied in this section is a vector $V = \{v_1^1, v_1^2, \cdots, v_1^k,\ v_2^1, v_2^2, \cdots, v_2^k,\ \ldots,\ v_n^1, v_n^2, \cdots, v_n^k\}$, where v_i^k is internally represented as a fact:

(a) either $feature\_k(t_i)^{[0,\ 1]}$ if $f_k^i < \lambda_k$;

(b) or $feature\_k(t_i)^{[1,\ 0]}$ otherwise ($f_k^i \geq \lambda_k$).

4.3 The Process of Term Authentication

Thus far, every term was represented by a set of facts concerning the presence of features observed in the collection, as defined in the previous subsection. In other words, there is more explicit information about terms of the collection than before. Moreover, the different sorts of information somehow relate to each other.

The specific kinds of information about terms allow the construction of rules based on domain knowledge representing particular opinions about authenticity of a term, as there are

[5]Notice there is exactly one value of each feature for each term.

		Real Class	
		+	−
Predicted Class	+	4165	25
	−	27	59

Table 5: Predicted Results × Manual Reference.

relationships among types of information. These opinions were implemented as clauses named `authenticity` and annotated with any element of the lattice τ and are formed by conjunctions of queries concerning the information in **Table 4** annotated only with elements of the subset $\{$`false`, `true`$\}$ of the lattice τ. The high number of possible combinations permits inconsistent results. Therefore, *ParaLog* is used as a query module to retrieve the evidences related to the authenticity of a given term.

Finally, both evidences — *belief* and *disbelief* — are used to get the *certainty degree* of authenticity of that term as showed in **Equation 2**. Consequently, the *confidence* of a term can be obtained as in **Table 3**.

With knowledge of authenticity of terms available, it is possible to create a new index of *absolute frequency* corrected. Let I be the index of *absolute frequency* of all terms in the collection, where rows are terms and columns are documents. Each element $e_{i,\ j}$ contains the frequency of the term t_i in the document d_j. The confidence of each term t_i is evaluated: if the term t_i is authentic, then its entry in the index I is conserved; otherwise, each of its frequencies F_i^j is equally divided among all their authentic opponents O_i, increasing the frequencies $F_{O_i}^j$, and its entry is removed from the index I.

The argument of this paper about the process adopted here is that a term which is not believed to be authentic should not exist in the index; so its entry should be removed. Nonetheless, it might be a misspelling of a term similar to it. As it is not clear which of their authentic opponents it could be, dividing the frequency of the fake term among their authentic opponents is a valid heuristic because if the entry were just removed, the frequency of the fake term would be lost. In **Subsection 4.4**, the accuracy of the authentication process is measured, taking as reference a manual authentication.

4.4 Evaluation of the Authentication Process

The results obtained with the automatic authentication of terms were compared with the results expected according to a human-made reference. **Table 5** contrasts those results.

The high value of real positives mistakenly predicted as negatives are due to the high number of acronyms. Acronyms are difficult to be identified without context. Thus, they are likely to be wrongly classified as not authentic if they have a low frequency.

Words misspelled frequently may be motivated to be considered authentic. On the other hand, if a term is misspelled and there are not enough ocurrences of its correct form to allow the identification of the authentic form, it is almost impossible to figure out the right form if it does not appear in *WordNet*.

More than one mistake inside the interval delimited by the smooth unification threshold forbids the identification of misspellings. To cope with that, a more robust method is needed.

Table 6 makes a comparison of the accuracy discriminated by class. The remarkable difference between the positive and the negative classes enhances the ease to identify authentic words with the use of *tf*, *df* and *WordNet*. The main exceptions are the acronyms. Whereas

Class	Accuracy Percentages
Positive	0.9935
Negative	0.7023
Balanced General	0.8353

Table 6: Automatic Authentication Accuracy Percentages Discriminated by Class.

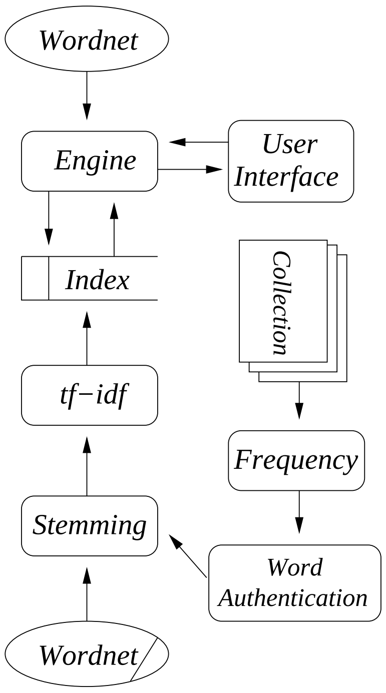

Figure 1: The system architecture.

the noise is small, the classes are not balanced — so the total accuracy rate needs to take into account that information.

The result of the process described in this subsection is a new *absolute frequency* index derived from the original one, where not all the terms encountered in the collection are present, because some of them were detected as misspellings and their correspondent frequencies were distributed among the possible intended terms.

5 The System Architecture

This section presents the architecture of the program which implements the issues discussed along the paper. The architecture is depicted in **Figure 1** and the system is explained step by step.

5.1 Indices Generation

The collection utilized in the whole set of experiments was obtained from the DAML ontologies repository at `http://www.daml.org/ontologies`. This repository is composed of ontologies constructed to serve as vocabularies for exchanging of information in several domains. Each ontology is stored in a file, so an ontology is a document. The ontologies are represented in the XML syntax but following RDF and DAML+OIL vocabularies. DAM+OIL is an expansion of RDF, adding several knowledge representation functionalities [9]. Although RDF has a lot of tags representing properties, only two are interesting for this work: `label`, which conveys the name of a resource in natural language, and `comment`, which provides a description of a resource. Thus, only the values of those two properties are used to index a document. Ideally, each resource should contain those properties; unfortunately, the real world does not use to collaborate in this case; some documents do not even qualify one resource with a `label` or `comment` property, becoming useless.

Once the collection had been downloaded — not the whole collection, because some links did not work —, the first step was to generate indices. A list of *stop words* helped to select the words to include in the indices. Next, and index of *absolute frequency* was generated.

Secondly, using the paraconsistent knowledge rules about authenticity of words presented in the **Appendix A**, a new index was obtained according to the procedures described in **Subsection 4.3**, reducing noise due to misspelled words. Example facts concerning the extra information automatically acquired about terms are provided in the **Appendix B**.

Morphological *WordNet* operations were then used to reduce this new index to a normalized form, where only primitive words exist. Two terms which have the same stem are counted onto the same entry of the index, regarding the documents which they belong to. That was the only kind of *stemming* applied on the terms along the whole process. Words which did not have a normal form in *WordNet* entered the index under the form encountered.

Finally, that last index generated so far is used to create the ultimate index: a *tf-idf* index. This is the ideal index upon which the search will be carried out. Recall that this index is normalized — only primitive forms — and noise due to misspelling errors is inhibited. The use of a classical measure was also a primary goal of this work, because it permits comparative analysis, facilitating, hence, comprehension.

5.2 The Search Mechanism

Once the desired keys are provided, they are passed to the *WordNet* module. The set of keys is then normalized according to the *stemming* described in **Subsection 5.1** — each not primitive key is substituted by its primitive form — and query expansion is executed, as described in **Section 2**. The result is an augmented query in a normalized form.

This resulting query is then passed to the vector model search engine. This engine uses the *tf-idf* index to retrieve the supposed relevant documents. Documents with *similarity* $\delta \geq 0.3$ are retrieved in a ranked fashion[6].

6 Related Work and Remarks on this Work

The Agent Semantic Communication Service (ASCS) [7] is a search engine where agents perform search based on DAML annotated documents. The developers argue their tool is very accurate since the search is semantic oriented — based on DAML tags.

Relationships among words are discovered in [8] through *WordNet*. It finds related words by looking for candidate words in the definition of a concept. If the candidate word exists in

[6]The *similarity* measure is defined as the classical one for vector model search.

the definition of the referred concept, then it is assumed to be a related word.

A lot of work on IR can be found in [2]. Nonetheless, all the approaches are strict knowledge-free IR techniques with some customizations.

Several documents of the collection did not have `label` and `comment` properties. Also, some of the links in the ontologies repository were no longer active, making it impossible to fetch the correspondent ontologies. All those problems were harmful for the search.

7 Conclusion and Suggestions for Future Work

It seems fairly clear knowledge is required in IR. The targets of this community are too much related to cognitive processes because, in essence, they want to do something people use knowledge to do — memory is the basis for every cognitive process in human beings.

New issues about authenticity of terms are as well important. As the Internet is filled of mistakes, they cannot be left out.

AA important point to discuss is the use of Data Mining techniques in the IR field. In Data Mining, there is usually no or little knowledge about the database. The aim is to mine the database in order to discover that knowledge. It seems weird to use Data Mining techniques in IR: the goal is not to discover knowledge; the goal is to understand texts. Experts have that knowledge. Thereby, expert knowledge should be used to understand text. In short, there is no need to look for knowledge because it is already known.

All the extra processing time is spent in indices generation. Thus, there is almost no extra time for search but the query expansion.

It was not taken advantage of knowledge concerning the documents in which the terms appear, except by the *df* measure. More specific paraconsistent rules might have been build as well as improvements in the process of word authentication as a whole might have been added.

References

[1] Bráulio Coelho Ávila. *Uma Abordagem Paraconsistente Baseada em Lógica Evidencial para Tratar Exceções em Sistemas de Frames com Múltipla Herança.* PhD thesis, Escola Politécnica da Universidade de São Paulo, São Paulo, 1996.

[2] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* ACM Press, New York, 1999.

[3] Newton C. A. da Costa et al. *Lógica Paraconsistente Aplicada.* Atlas, São Paulo, 1999.

[4] G. A. Miller et al. Five Papers on *WordNet.* CLS Report 43, Cognitive Science Laboratory, Princeton University, 1990.

[5] Christiane Fellbaum, editor. *WordNet: an electronic lexical database.* The MIT Press, Cambridge, Massachusetts, 1998.

[6] Ora Lassila and Ralph R. Swick. *Resource Description Framework (RDF) Model and Syntax.* W3C World Wide Web Consortium, February 1999. W3C Recommendation.

[7] John Li, Adam Pease, and Christopher Barbee. Experimenting with ASCS Semantic Search. Teknowledge Corporation, Palo Alto, CA. Available on the Internet at `http://reliant.teknowledge.com/DAML/DAML.ps` on Feb 19th 2002.

[8] Nitish Manocha, Diane J. Cook, and Lawrence B. Holder. Structural Web Search Using a Graph-Based Discovery System. *intelligence: New Visions of AI in Practice*, 12(1):20–29, 2001.

[9] Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. *Reference Description of the DAML+OIL (March 2001) — Ontology Markup Language*. Available on the Internet at `http://www.daml.org/2001/03/reference.html` on Feb 19th 2002, March 2001. Work in progress.

A Paraconsistent Rules used in Word Validation

Here are some of the rules upon which a *ParaLog* query is triggered. The predicate `authentic` is used to query the knowledge base. Values between square brackets are the evidences attributed to the respective clause. The predicates which constitute the body of each clause express the features involved. The names of the predicates are significative for each feature, so there is no need for captions.

```
...
authentic(T):[0.8, 0] <--
        high_tf(T):[1,0] &
        high_freq_opponents(T):[1,0].
authentic(T):[1, 0] <--
        high_freq_high_tf(T):[1, 0].
authentic(T):[0.7, 0] <--
        high_tf(T):[1, 0] &
        high_df(T):[1, 0].
authentic(T):[0, 0.7] <--
        high_tf_opponent(T):[1, 0] &
        high_tf(T):[0, 1] &
        high_df(T):[0, 1].
...
```

B A Piece of the Information Automatically Acquired

Some facts representing the information automatically acquired about terms of the collection are supplied below as examples. There is a clause for each pair termfeature.

```
...
high_tf(person):[0, 1].
high_df(person):[1, 0].
high_freq_high_tf(person):[0, 1].
high_tf_opponent(person):[0, 1].
high_freq_opponents(person):[1, 0].
high_freq_high_tf_opponents(person):[0, 1].
wordnet(person):[1, 0].
...
```