PiQASso 2002

G. Attardi A. Cisternino F. Formica M. Simi A. Tommasi*

{attardi, cisterni, formicaf, simi, tommasi}@di.unipi.it
Dipartimento di Informatica
University of Pisa, Italy

Abstract

The University of Pisa participated to TREC 2002's QA track with PiQASso, a vertical QA system developed (except for some of the linguistic tools), entirely within our research group at the Computer Science department. The system features a filter-and-loop architecture in which non-promising paragraphs are ruled out basing on features ranging from keyword matching to complex semantic relation matching. The system also exploits the Web in order to get "hints" at what to look for in the internal collection. This article describes the system in its entire architecture, concentrating on the Web exploitation, providing figures of its efficacy.

1 Introduction

Last year's PiQASso system featured an architecture similar to that deployed by many systems [1]. Such architecture is organized in a series of subsequent filters that rule out paragraphs that are evaluated as not answering the question.

We kept preprocessing on the collection to a minimum, basing on the following considerations:

- pre-processing the whole collection is an expensive process;
- ideally, the collection to use as knowledge base to mine answers should be as large as possible: this makes systems that perform extensive processing on the data collection less scalable;
- the collection might even be changing dynamically, which in fact renders such systems unusable under this assumption;
- in an experimental setup, several different algorithms might be experimented, and re-processing the whole collection for each experiment is unfeasible;
- not assuming any preprocessing on the collection allows for immediate application to "foreign" collection, e.g. by querying a search engine.

Whenever the following holds:

- the collection is static;
- the set of operations to perform on it is "stable";
- the implementation of such operations is stable as well;
- the "enriched" collection size does not exceed disk space;

nothing prevents from performing all the necessary computing on the collection off-line, obtaining substantial speed-up.

As with most other systems, analysis of the question is performed as the first step during the question answering phase. Within PiQASso, the goals of such analysis are:

- identify the suitable keywords to perform IR-style search;
- classify the question onto the expected answer type (EAT) taxonomy;
- extract a set of "logical" relations connecting the question's words;

*Alessandro Tommasi would like to thank Microsoft Research, Cambridge, for supporting his PhD.

- assign a weight to each word, proportional to the "importance" of the word.

Each of these 4 characteristics of the question are used to rule out sentences from the set of candidate answers.

The overall system architecture can be sketched as follows. Each question undergoes question analysis. From the analysis, data is gathered to apply the following series of cascade filters to the collection:

retrieval IR performed by the keywords extracted from the query, and in proximity, returning paragraphs;

type filter paragraphs not containing entities of the expected answer type are discarded;

relation matching sets of logical relations between words are extracted from the paragraph and the question and a matching cost is computed deploying a measure based on *semantic distance* between words;

popularity ranking whenever multiple evidences of the same answer are collected, this boosts its confidence score.

If no answer makes it through this cascade of filters, the retrieval phase is re-executed by loosening the search, in a loop.

This article describes each of these steps. In addition it describes the use of the Web to narrow the IR phase, along with the results obtained over TREC 2002's question set.

2 Tools deployed

2.1 Indexing

This year's task was based on a different collection than 2001's: the AQUAINT collection. We indexed the collection similarly to last year: we deployed IXE [2], an efficient indexing and search engine based on template meta-programming and developed within our group[1], to build full-text indexes of the collection. Along with the full-text, we have stored paragraph splitting information, obtained by the processing of a Maximum Entropy-based sentence splitter. This way, by customizing IXE, we are able to issue queries and have the search engine to return results as paragraphs rather than documents. However, the indexes are still built over documents (i.e., we did not consider each paragraph as a separate document searchwise), allowing for proximity search to span over different adjacent paragraphs.

Just as it did for paragraph splitting information, IXE is able to store and handle additional information about the text being indexed. We are foreseeing moving part of the "inevitable" (e.g., NE tagging) work to the index phase, so that it could be exploited by issuing queries in a richer language (for instance, specifying the type of the entity sought for).

2.2 Parsing

To perform syntactic parsing of sentences we used Minipar [5], a dependency parser developed by D. Lin and free for non-commercial use. The parser has several additional capabilities that come in very handy for our task; in particular:

- it identifies logical relationships (subject, object, complement) rather than grammatical ones;
- it tags words with semantic tags, providing a form of named entity recognition;
- it partially solves coreference resolution problems;
- the dependency tree it returns turned out to be a good way of representing text.

In figure 1 we see, for each word, linked to the head word, its relation to the father, its label, the part of speech, the morphological root, the semantic tags (in braces). Also, note, for the word "which", the "@5": it means that the pronoun refers to the word of label 5: "Messiah". Some of the tags (for instance, the "quote" tag) have been added by us to recognize particular entities, in this case quoted entities. For robustness, we have also added part of speech tags via an external part of speech tagger (TreeTagger [10]), in order to compare Minipar's output with TreeTagger's. This way we have greater confidence about the POS tag when both tools agree on it.

2.3 Named entity recognition

Just as we used an external POS tagger to verify Minipar's tagging, we also used an external NE-Tagger to

[1] Research supported by Ideare s.r.l.

```
_EMPTY ((null)  E2  (null)  (null)   (null) )
+--_EMPTY ((null)  E1  C  fin   (null) )
|     +--composed (i  2  V  compose   VBN )
|     |     +--Handel (s  1  N  Handel  {person} NN )
|     |     +--_EMPTY (subj  E3  N  Handel  {person}@1 (null) )
|     |     +--Messiah (obj  5  N  the Messiah  {quote} NP )
|     |     |     +--the (lex-mod  4  (null)  the  {quote} DT )
|     |     |     +--in (mod  7  Prep  in   IN )
|     |     |     |     +--1741 (pcomp-n  8  N  1741  {date} CD )
|     |     |     +--_EMPTY (rel  E0  C  fin   (null) )
|     |     |     |     +--which (whn  10  N  which  @5 WDT )
|     |     |     |     +--rearranged (i  13  V  rearrange   VBN )
|     |     |     |     |     +--was (be  11  be  be   VBD )
|     |     |     |     |     +--later (amod  12  A  later   RB )
|     |     |     |     |     +--_EMPTY (obj  E4  N  which  @5 (null) )
|     |     |     |     |     +--by (by-subj  14  Prep  by   IN )
|     |     |     |     |     |     +--Mozart (pcomp-n  15  N  Mozart  {person} NP )
```

Figure 1: **the tree returned by Minipar for the sentence:** ***Handel wrote "the Messiah" in 1741, which was later rearranged by Mozart.***

provide additional information about entities that Minipar did not recognize. The results of the NE-Tagger simply "enrich" the annotation provided by Minipar.

The NE-Tagger we chose is based on Maximum Entropy [9], whose accurary depends mostly on the quality of the training set used. The one we used is small and originally designed for speech recognition tasks. Also because of this, the entity taxonomy is small: entities can be: *person, organization, location, money, measure, cardinal, percent, duration, date, time*. These categories are however the standard ones as defined in the relative MUC task [3].

A mapping among the different taxonomies of the various tools (Minipar, the NE-Tagger and WordNet) is required.

2.4 WordNet

WordNet [8] has been used mainly for: classifying words (to determine the EAT) and measure the *semantic distance* of two words. This second activity has the goal of returning a score that is higher if the two words are semantically very close. This "closeness" refers to the interchangeability of the two words in a sentence. Consider for example the word "cat". The word "mammal" is definitely not a synonym of cat, yet it's a closer term, semantically, than "lizard". While this is obviously a context dependent property, it has been globally approximated.

The type of a word is simply the taxonomy the term belongs to according to WordNet. This yields to 24 categories, some of which can be mapped directly onto Minipar or the NE-Tagger categories, and others that constitute categories by themselves.

In order to evaluate the semantic distance or to chose the proper type of a word, the right *synset* (sense) of a word must be selected. This is, again, a property that should take the context into account. However, for several practical reasons (for instance, because the question provides too little a context), we have approximated it more pragmatically, exploiting WordNet's sorting of senses by frequency. For type identification, we return for word w, whose ordered senses in WordNet are $\{s_0, \ldots, s_n\}$, the category C that maximizes:

$$P(w, C) = k \sum_{j=0}^{n} \frac{n-j}{n} \gamma(s_j)$$

where

$$\gamma(s_j) = \begin{cases} 1 & \texttt{iff} \quad s_j \in C \\ 0 & \texttt{otherwise} \end{cases}$$

and k is a constant, presently set to 0.7.

For semantic similarity the various senses of the two words are weighted, and contribute differently to the final score. Word similarity, is computed over nouns, verbs, adjectives and adverbs, while only the first two categories were treated last year.

3 Question analysis

The question is parsed. From then on, all further analysis takes place on the parse tree, which is possibly enriched by means of other tools.

3.1 Keyword extraction

The first important thing done on the question is the selection of those terms in the question that should be part of the IR-like query. A word is selected as a keyword if:

- it is not a "forbidden" word like wh-words;
- if it is an informative noun (you, me, something etc. are non-informative);
- it's a verb but it is not "to be";
- it is an adjective or an adverb.

The keyword set can be modified in 4 successive, progressively loosening steps in case no answer is found. First, morphological variations for each word are added. Second, synonyms (according to WordNet) are `OR`'ed to each keyword. Third, adverbs and first proper names are dropped. Fourth, verbs are dropped as well, and if more than 3 keywords are still there, all keywords whose father (according to the parse tree) is a keyword are dropped. The empirical justification is that in a dependency parse tree, the children of a node "modify" the node, that generally has a more central role in the sentence (the same rationale drove the word weight assignment, see 3.3).

3.2 Expected Answer Type

The EAT is the type of the entity that is asked by the question. The taxonomy for the EAT is not very uniform, for it reflects the natural taxonomy of questions. It is composed of the categories: `person`, `organization`, `location`, `time-date`, `quantity`, `quoted`, `language`, and those gathered from WordNet. The `person` category also features two subclasses, depending on whether the question expects a proper noun or rather a definition. The EAT can also be `person OR organization`, for questions of the kind "who did something". The category `quoted` is meant for titles (of books, songs, movies...), while the category `language` is for questions of the kind "what does something mean/stand for". The remaining categories (those referring to WordNet's taxonomy) are for the "what" kind of question, that usually provide a narrower description of the entity they're asking for by a focus noun such as "instrument" in "what instrument did Glenn Miller play?".

3.3 Weights

In our abstraction, words in the question express relations (verbs), attributes (adjectives and adverbs), concepts (common nouns) or entities (proper nouns) for which we expect to find a counterpart in an answering paragraph. However, not all the words in the question are expected to be found in the answer paragraph, nor all of them are sought with the same priority. Therefore, we assign a *weight* to each word in the question. This weight represents the relative importance of the concept associated with the word, and will influence the matching cost between question and answer by which candidate answers are sorted.

The weight is a number in the [0, 1] interval. Words that are not initially selected as keywords all have weight 0 (they may happily be missing from the answering paragraph). For all others, we visit the parse tree recursively. The first word has weight 1, and the weight halves as the depth of the node increases. However, nodes that are tagged with semantic features receive a 10% "bonus" weight for each feature they have.

The rationale of this heuristic is that nodes closer to the root (the main verb and its direct complements) express more "basic" properties of the required entity, while down deeper in the dependency tree are only additional, perhaps "optional" requirements that are less likely to appear in an answering paragraph.

4 Answer Selection

4.1 Retrieval: Proximity Search

We combine the keywords extracted from the question in a proximity query. The width of the proximity window is roughly estimated as being twice the number of nodes in the parse tree of the question. All paragraphs over which the resulting text window spans are retrieved as candidate answers. This allows for "elasticity": if a particular paragraph is retrieved, but it does not contain all the question keywords, we can rest assured that together with its close context (surrounding paragraphs) it did.

All paragraphs retrieved this way are however considered independently.

4.2 Type Filter

Of all the paragraphs returned by the retrieval phase, the top N (we have experimented with N's ranging from 400 to 1200) are considered. These paragraphs are parsed for further analysis, the first step of which is the type filter. The parse tree is visited, looking for at least a node that is tagged (or that is classifiable) in the EAT category of the question. Furthermore, it is required that such node did not occur in the question as well. Consider the question: "Who killed John Kennedy?", whose EAT is *person* OR *organization*. All paragraphs containing John Kennedy (and there will be plenty of these in the search result) contain a *person* node; for sure we are looking for a different person.

Paragraphs that do not contain an entity of the proper type are discarded.

4.3 Relation extraction

Once the candidate paragraph has passed the type filter, we want to make sure the paragraph actually *supports* its answer. For factual questions like TREC's, an answer is an entity that verifies the conditions imposed by the question. We have issued the following hypotheses:

- the conditions imposed by the question are expressed by the logical relations among words;
- such relations are found in answering paragraphs.

While verbs are usually considered as relations among entities expressed by nouns, we have reified them, and considered them as "concepts" too. This way, the number of relations is considerably cut down to syntactical ones, like "subj, obj, p-comp" etc., as output by Minipar.

To achieve better flexibility with respect to the various different expressions of the same concepts, we must perform syntactical normalization. John's ownership of a car is expressed by both "John has a car" and "The car of John". To make sure we are able to understand that sentences like these convey the same meaning, we need to perform normalization, for which we have flattened the dependency tree to a set of relations (head word, relation, modifier word), which is less strict, and allows us to insert new relations without messing up the structure.

First, all relations found in Minipar are inserted in the relation set. From then on, a rule system is applied to the set, extending it, until the fix-point is reached (no more relations are added). Rules can be very general, but are usually in the form: "if A is in relation R_1 to B, and B is in relation R_2 to C, then add the relation (A, R_3, C)". With respect to the example above, a rule for "to have" says that: "if A is the subject of the verb to have, and C is the object, then A is a specifier of C". This way, both example text snippets would "agree" on the relation (C, specifier, A).

In the current system, we have 19 such rules, all oriented to syntactical normalization.

In a domain as open as TREC's, it is hard to imagine how semantic rules of the same kind could be added. However, in principle nothing keeps from adding more rules for specific domains. In our current system, only three semantic rules have been added, mainly for testing purposes: "if A is known as B, then A is B"; "if A directed, composed, wrote, manufactured, produced or invented B, then it A is a specifier of B", "if A means/stands for B, than A is B". These rules are easy to write, and all they do is enriching the relation set relative to a paragraph. However, it is impractical to think we could write rules for all common sense facts.

Relation sets are extracted both from candidate paragraphs and from the the question, for which we also set an empty slot, a node that is missing in the question, and yet it's in relation with question words. This empty slot represent the answering entity: the node with similar relations to this one in the answering paragraph is considered to be the had of the answer. The position of the empty slot in the parse tree (and therefore the relations it has with other words) is usually the position of the first "missing" node; we look for the first verb whose subject or object is missing. Special cases correct "when" and "where" questions, for which such node is placed as a modifier of the main verb.

4.4 Relation matching

Once the relation sets for question and candidate answer are determined, we use a matching function to determine how well an entity in the paragraph matches the

Q: *Who was the first person to reach the south pole?*
A: *Any bright schoolchild can tell you that Roald Amundsen was the first man to reach the South Pole, but can anybody name the first baseball fan to brave Candlestick?*

`reach, Amundsen (subj) <-> reach, ANODE ((null))`
`VROOT, first (post) <-> VROOT first (lex-mod)`
`reach, Pole (obj) <-> reach, pole (obj)`
`Pole, South (lex-mod) <-> pole, south (lex-mod)`

Figure 2: **an interesting case, in which the answer is provided by a question. We show the matching relations (on the left of `<->`, the answer relations, on the right, the question ones). ANODE is the empty slot in the question. Notice how the subject of "reach" in the answer is Amundsen, while the word "first" cannot be attached to it, and is therefore "pending" (attached to a virtual root VROOT).**

question's requirements, allowing for sorting answering paragraphs. All paragraphs that are more distant than a threshold are considered not answering.

The matching algorithm begins by matching the question's empty slot against a node of the required type in the answer (*answering node*). The set of nodes in relation with the empty slot is then identified. For each such node, a node in relation with the answering node is looked for as "partner". The algorithm then proceeds by trying to match nodes of distance 2 from the empty slot to nodes of distance 2 from the answering node, and so on. We find the optimal solution, choosing for each node in the question a partner in the answer so that the global matching cost is minimal.

The global matching cost is defined in terms of several properties.

- Nodes in the question that do not have a match in the answer contribute to the cost proportionally to their weight (question requirements missing in the answer);
- For each question node/answer node association, a contribution to the cost is added proportionally to the weight of the question node and the distance of it from the answer node.

The distance from a question node to the corresponding answer node is evaluated basing on the similarity measure defined on WordNet, and on other properties, such as:

- the number of semantic features "missing" in the answer node;
- the difference of modifiers;
- different relation with the father.

An example of the matching returned for a question and a candidate answer is shown in figure 2, in which the matching is more compactly shown as associations between relations extracted from the question and the answer.

5 Exploiting the Web

5.1 Narrowing the search for the answer

While the type filter and the relation matching phase are highly semantic filters, the most selective, the one that reduces the number of paragraphs from the millions to a few hundreds, is the IR phase. If questions like "What is an atom?" leave little choice as to what to deploy as keywords, in several other cases, for more complex questions, the choice of keywords plays a crucial role. Several times, morphological variations or adding synonyms does not help: a honeymoon implies a wedding, but it is not a synonym.

The difficulty of finding the proper paragraphs for complex questions is evident from figure 3, in which it is shown that the system performs considerably better for short questions. Moreover, keeping at most the top 400 results of every query, over the 500 TREC2002 questions we retrieved in total about 180.000 paragraphs of which only about 1 every 30 matched the corresponding answer string (lists of regular expressions matching correct answers to each question, and kindly provided by Ken Litkowski [6] to the QA-track mailing list).

Figure 4 shows that the number of questions for which no paragraph retrieved contained an answer is indeed high.

Average words in question: 6.3	
Longer questions: 215	Shorter questions: 285
Correct: 44	Correct: 123

Figure 3: **a comparison between "long" and "short" questions. Short ones appear to be easier, mainly because for long ones, it is more difficult to select the right keywords (TREC 2001 question set, our best submitted run). The TREC 2002 question set amplifies this problem, for the average length of a question increased to 7.6 words per question, still with 285 questions under the average.**

	# questions	# correct NILs
0 par. returned	37	14
0 ans. within par.	180	37
1+ ans. within par.	173	//

Figure 4: **numer of questions for which: no paragraphs were returned at all, some paragraphs were returned but none of them contained an answer matching Litkowski's patterns, some paragraphs matching the patterns was retrieved. On the right column, the number of questions for which the correct answer was NIL (TREC 2002 set).**

Hypothesis: the Web features such a high redundancy, that it is likely to find a fact stated in a particular way.

Assuming this fact, we have decided to issue a very narrow query to a search engine, to see if an answer could be found on the Web. Such answer (that we call "Web suggestion") can then be searched for directly on the internal collection

5.2 Answer template

During the question analysis, an *answer template* is constructed: a text fragment obtained by turning the question into a direct form. Examples of pairs of question and answer template are provided in figure 5.

Templates are generated simply by turning the main verb into the direct form, and adjusting its conjugation (did originate → originated, does have → has).

Additionally, looser templates are obtained from these ones by simply removing verbs (and therefore breaking the template in two or more chunks). This is done in case no suggestion is found on the Web using the stricter template.

5.3 Google candidates

By means of Google's API, we issue the answer template as query to Google, and isolate the resulting excerpts. By the way Google works, such excerpts contain the text in the document retrieved that matched the query. In that respect, it is not dissimilar from the paragraphs we return by our search engine. Also, because the query asks for adjacent words, we know that the relative excerpts must contain a text fragment that is (almost) well constructed, and therefore parsable.

Some examples of excerpts are shown in figure 6.

We apply type filtering and relation matching to these excerpts, in order to obtain actual verified answers out of the excerpts. Because the search string was the question turned into direct form, it is usually easier to verify its correctness.

5.4 Justifying Web suggestions in the collection

Even if the answers gathered from the Web are justified by their exceprt, it is the case that such excerpts might indeed be wrong, or that an answer must be found on an internal, trusted collection.

To do so, and bearing in mind that several irrelevant paragraphs were returned by the "standard" search strategy, we have "doped" the keyword set by adding the answers found on the Web. This in fact renders the search much narrower, and hopefully, more likely to find answers. In fact, we are searching the question's keywords along with answers to the same question as found elsewhere.

In figure 7 some question/suggestions pairs are shown. From the figure we see that these candidates have a high rate of correctness, mainly due to the strict template they were gathered from, and to the answer validation that is applied to them. In fact, while the template is often very good by itself in selecting good answers, validation can rule out cases such as "Galileo was very nice", or "Wilt Chamberlain scored 100 points and then showered", which match the template but do not contain answers.

In the same figure we see another fact that is important to consider when looking for answers in the Web.

Question	Template
In what country did the game of croquet originate?	The-game-of-croquet-originated
What year did Wilt Chamberlain score 100 points?	Wilt-Chamberlain-scored-100-points
How many chromosomes does a human zygote have?	A-human-zigote-has
What lays blue eggs?	Lays-blue-eggs
Which vintage rock and roll singer was known as "The Killer"?	Was-known-as-The-Killer

Figure 5: **some questions with the relative answer template. A hyphen between words means that they must be adjacent (Google syntax).**

Google Excerpt
... AP. The following is The Associated Press story filed March 2, 1962, the night Wilt Chamberlain scored 100 points in a game against the New York Knicks. ...
... March 2 came and went quietly, but was a noteworthy date in sports history ? Saturday was the 40th anniversary of the day Wilt Chamberlain scored 100 points ...
... 28. The ball was supposedly the one that 7'1" Wilt Chamberlain scored 100 points with on March 2, 1962, in Hershey, Pennsylvania. ...
... The Associated Press. After Wilt Chamberlain scored 100 points in a game in Hershey, Penn., Kerry Ryman snatched the ball after the contest. ...

Figure 6: **some of the excerpts returned in the first result page of Google for the query "Wilt-Chamberlain-scored-100-points".**

Question	Web Suggestions
Who was Galileo?	astronomer, scientist, philosopher, son, physicist, professor, pioneer, egoist
What is an atom?	quantum, block, Raspberry, Lies, part, Friend, thing, hero, particle, helium, Port, member, vehicle, constituent, piece, organization, proof
What year did Wilt Chamberlain score 100 points?	1962, 1984
Where did the game of croquet originated?	France
When did Bob Marley die?	1981, May, 1980, 1982

Figure 7: **questions and relative candidates, obtained by Google issuing the answer template of each question as query.**

The author of Web pages can be wrong when he assesses something, so for instance he might believe that Bob Marley died in 1980 (and write so), while he did in 1981. Sorting suggestions by popularity may help in these cases.

Web suggestions are treated as additional keywords to the IR-like query. In case the collection does not "agree" with the Web, this will result in a query that returns no candidate paragraphs. In such cases, we can remove the Web suggestion from the query, and proceed with the default behavior.

In figure 8, we see some examples of the generated queries. In bold face, Web suggestions. As it's easy to notice, for long queries it is likely that the addition of a keyword is irrelevant, because few (if any at all) paragraphs are already returned. However, the addition of the Web suggestion is very effective for short questions, where it can narrow and direct the search. After the first expansion loop, that is if no answer is found, the system is given the chance to "prove the Web wrong":

Run	CWS
pqas21	0.357
pqas22	0.358
pqas23	0.354
resorted	0.438

Figure 9: **results for the TREC 2002 answer set. The last run was not submitted, but has still been obtained from NIST judgments.**

	# questions	# correct NILs
0 par. returned	30	13
0 ans. within par.	110	38
1+ ans. within par.	243	//

Figure 10: **same as figure 4, this time using Web suggestions in the query.**

by removing the Web suggestion, we let open the chance of finding a different answer. However, we re-insert the Web suggestion at the last expansion phase, at which, by experience, usually so many paragraphs match, that hardly anything useful is ever returned (only the first 400 paragraphs returned by the query are analyzed, in order to save time). Finally, there is a last "desperate" query that we called *last resort*, in which only the Web suggestion is looked for in the collection.

6 Results

We submitted three runs at this year's TREC (see figure 9). The difference among the three runs laid in different settings for constants and weights that showed little effect.

However, with respect to these three runs, a big mistake was made in sorting the answers. The system is biased toward precision, therefore every NIL answer (there are still about 210 NILs!) should be considered as highly uncertain. Re-sorting the answers so that all the NILs come later (and keeping the relative sorting), still basing on NIST judgments, yielded a considerably better result: a CWS of 0.438.

Interesting results regard the use or not of the Web to narrow the search during the IR phase. Thanks to the doping of the search expression by Web suggestions, the searches for TREC 2002 questions returned about 210.000 paragraphs, of which 1 every 15 in average contained the proper answer string. This is an indication that Web suggestions were, in most cases, correct.

Particularly relevant is figure 10, which is the same as figure 4, but modifying the keyword set by adding the Web suggestions. The figure shows how using Web suggestions we had the chance to answer correctly to 70 more questions than we did without the use of the Web. Unfortunately, not all of the paragraphs with the right answer to these 70 questions made it through all the remaining filters, and the final result (the unofficial 0.438) is not extremely higher than what we'd obtain without using the Web (about 0.40).

7 Conclusions and further work

2002's TREC featured plenty of systems that used the Web in peculiar ways. PiQASso used it to look for possible answers, in order to find "confirmation" for them in the internal collection. Magnini et al. [7] used it the other way around: to confirm correctness of answers gathered from the internal collection. Yang and Chua [11] used it, more similarly to us, to gather additional keywords to include in the IR-style search.

While we also feature complex semantic answer justification (not unlikely Harabagiu et al. [4]), it turns out our cascade structure of the filters is way too strict, at the point that for more than 200 questions we returned NIL. This is mostly due to the relation matching filter being too "strict" (too syntax-concerned). In that respect, techniques like Harabagiu et al.'s lexical chains seem to be able to lead to significant improvement.

However, as we feel we have improved the search phase, and further improvements will concern, for instance, the deploying of NE-tagging-aware indexing, the quality of the semantic filter is on a completely different dimension, and can therefore be developed independently from the search phase.

Moreover, since Google's excerpts undergo the same filters as the paragraphs found in the internal collection, failures of the relation matching filter affect negatively also the effectiveness of the Web search. That is, very often the answer would actually be within the excerpts returned by Google, but it would be filtered out by either the type or semantic filters. We therefore expect great improvements out of the tuning of the last filter, that can be obtained mainly by better exploitation of WordNet (à

Questions	**Queries**
Where did the game of croquet originate?	- ((**France**) & (originate) & (game) & (croquet)) - ((game) & (croquet croquets)) - ((game biz) & (croquet croquets)) - ((**France**) & (game biz) & (croquet croquets)) - ((**France**))
Who was the first person to run the mile in less than four minutes?	- ((**Bannister**) & (person) & (run) & (mile) & (minutes) & (less) & (four) & (first)) - ((person) & (run ran run running) & (mile) & (minutes) & (less) & (four) & (first)) - ((**Bannister**) & (mile knot mi) & (minutes) & (first)) - ((**Bannister**))
Who is the Governor of Tennessee?	- ((**Alexander Johnson Menkov Pearsall Ronnie War Hilleary McWhorter**) & (governor) & (Tennessee)) - ((governor governors) & (Tennessee)) - ((**Alexander Johnson Menkov Pearsall Ronnie War Hilleary McWhorter**)

Figure 8: **examples of question/queries pair, at various expansion levels. Words in the inner parenthesis are in or, otherwise in and. The query is a proximity query.**

la lexical chains).

References

[1] G. Attardi et al., *PiQASso: Pisa Question Answering System*, proc. TREC 2001 Conference, Columbia, MD, November 2001

[2] G. Attardi, A. Cisternino, *Reflection support by means of template meta-programming*, proc. of Third International Conference on Generative and Component-based Software Engineering, LNCS, Springer-Verlag, Berlin, 2001

[3] R. Grisham and B. Sundheim, *Design of the MUC-6 evaluation*, proc. of the MUC-6 conference, Columbia, MD, 1995

[4] S. Harabagiu, Moldovan et al., *LCC Tools for Question Answering*, preliminary proc. of TREC 2002 conference, Columbia, MD, November 2002

[5] D. Lin, *LaTaT: Language and Text Analysis Tools*, proc. Human Language Technology Conference, San Diego, California, March 2001

[6] K. C. Litkowsi, *Question Answering Using XML-Tagged Documents* preliminary proc. of TREC 2002 conference, Columbia, MD, November 2002

[7] B. Magnini et al., *Mining Knowledge from Repeated Co-occurrences: DIOGENE at TREC-2002*, preliminary proc. of TREC 2002 conference, Columbia, MD, November 2002

[8] G. Miller, *Five papers on WordNet*, special issue of International Lexicography 3(4), 1990.

[9] J. C. Reyner and A. Ratnaparkhi, *A Maximum Entropy Approach to Identify Sentence Boundaries*, Computational Language, 1997.

[10] G. Schmid, *TreeTagger – a language independent part-of-speech tagger*, 1994. Available: http://www.ims.uni-stuttgart.de/Tools/DecisionTreeTagger.html

[11] H. Yang and T. Chua, *The Integration of Lexical knowledge and External Resources for Question Answering*, preliminary proc. of TREC 2002 conference, Columbia, MD, November 2002